Maven

Maven is a software tool that helps you manage Java projects, and automate application builds. IntelliJ IDEA fully integrates with Maven version 2.2 and later versions, allowing you to create or import Maven modules, download artifacts and perform the goals of the build lifecycle and plugins.

In this section:

Maven support

Maven integration is shipped with IntelliJ IDEA, and you do not need to perform any additional actions to install it. You can start using it straight away for importing Maven projects and working with them. However, for running Maven goals, you have to install Maven on your computer.

Maven support in IntelliJ IDEA includes the following features:

- Dedicated module type
- Maven repositories support
- Full editing support for pom.xml file
- Possibility to import Maven projects
- Running and debugging Maven goals
- Compiling

- WAR overlays
- Dependency Graph

Maven module

For the Maven projects IntelliJ IDEA provides a dedicated module type. For each *Maven Module*, IntelliJ IDEA creates a `pom.xml` file. So doing, a Maven Module can be created either with the basic `pom.xml` file, or from a certain pattern called *Maven archetype*.

The dedicated module type allows creating Maven projects that have parent and aggregator Maven projects.

Maven repositories

IntelliJ IDEA enables communication with remote Maven repositories, and maintaining the local ones.
When Maven goals are executed in the IDE, IntelliJ IDEA is aware of all downloads and artifacts. However, if you launch Maven from command line, the artifacts produced come unnoticed, and you have to make IntelliJ IDEA search for updates. For this purpose, IntelliJ IDEA suggests a quick fix to update indices, and a node in the Settings dialog (Maven | Repository Indices).

In particular, updating indices helps keep the list of available Maven archetypes up to date.

pom.xml

Maven works with `pom.xml` files to build projects. At minimum, a `pom.xml` file should contain a root element `<project>`, and identifiers of the project group, artifact and version.

IntelliJ IDEA supports syntax of the `pom.xml` files. When editing `pom.xml` files, you can enjoy the following advanced editing features:

- Syntax highlighting.
- Maven dependencies and parent generation using **`Alt+Insert`**.
- Quick fixes for adding dependencies and updating Maven indices.
- Code completion.
- Navigation between modules and `pom.xml` dependencies (**`Ctrl+B`**, **`Ctrl+Button1 Click`** or **`Button2 Click`**), **Navigate | Declaration** on a property from `pom.xml` files, `settings.xml` and `profiles.xml` files, system and environment properties, and properties defined in custom Maven filters.
- Structure view.
- Find Usages (**`Ctrl+F7`**).
- Code folding.
- Reformatting.
- Rename refactoring for properties defined in Maven project and custom filters files.
- Validation.
- Viewing parameter information **`Ctrl+P`**.
- Viewing quick info **`GuiDesigner.QuickJavadoc`**.

Import of maven projects

If you want to use an existing Maven project, you can import it directly by opening its `pom.xml` file. When a Maven project is imported, it maps to an IntelliJ IDEA module with the name, which is equal to the Maven project's `artifactId`. Dependencies between the Maven projects map to the dependencies of IntelliJ IDEA modules from the libraries and other modules. IntelliJ IDEA analyzes the `pom.xml` file and automatically downloads the necessary dependencies.

Maven projects tool window

The dedicated Maven Projects tool window allows you to manage Maven projects, configure preferences for the current Maven project and the defaults for the new ones, and execute Maven goals. Results are displayed in the Maven Build Output window.

Running and debugging Maven goals

IntelliJ IDEA provides two ways of running the Maven goals:

- Create run/debug configuration and launch it.
- Use the Run Maven Build command in the Maven Projects tool window. This way doesn't require any run/debug configuration.

Compiling

IntelliJ IDEA's Make features are capable of filtering Maven resources. However, IntelliJ IDEA yet does not support filtering web resources.
Details on configuring filter options within the `pom.xml` can be found at http://maven.apache.org/plugins/maven-resources-plugin/.

Note, that classpath for Maven-based projects is built following the Maven rules. IntelliJ IDEA supports *compile*, *test* and *run-time* dependency scopes.

WAR overlays

IntelliJ IDEA Maven integration correctly handles WAR overlays, which is important for the Web projects that use common resources defined in a WAR module. IntelliJ IDEA unpacks such WAR to the `overlays` directory under the content root of the dependent Maven Module.

Dependency graph

IntelliJ IDEA provides you with a handy graph-view of Maven dependencies available from the context menu in the **Maven Projects** tool window.

See Also

Procedures:

- Creating a Project from Scratch
- Importing Project from Maven Model
- Creating a Maven Module

Reference:

- Maven Projects Tool Window

External Links:

- Creating and importing Maven projects

Web Resources:

- Developer Community