**Installing Maven**

To use Maven your must first make sure that you have installed Maven on your computer. The first page in this Maven tutorial covers how to [**install Maven**](http://tutorials.jenkov.com/maven/maven-tutorial.html#installing-maven).

**Creating the Project Directory**

Once you have assured that Maven is installed, create a new directory somewhere on your hard disk. This directory will be the root directory for your first Maven project.

**Creating the POM File**

Once you have created the project root directory, create a file called pom.xml inside the directory.

Inside the pom.xml file you put the following XML:

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.jenkov</groupId>

<artifactId>hello-world</artifactId>

<version>1.0.0</version>

</project>

This is a minimal pom.xml file.

The groupId identifies your organization.

The artifactId identifies the project. More specifically, it identifies the artifact built from the project, like for instance a JAR file.

The version identifies the version of the artifact which the POM file builds. When you evolve the project and you are ready to release, remember to update the version number.

Other projects that need to use your artifact will refer to it using the groupId, artifactId and version, so make sure to set these to some sensible values.

**Testing the POM File**

When you have created the pom.xml file inside the project root directory it is a good idea to just test that Maven works, and that Maven understands the pom.xml file.

To test the pom.xml file, open a command prompt and change directory (cd) into the project root directory. Then execute this command:

mvn clean

The mvn clean command will clean the project directory for any previous temporary build files. Since the project is all new, there will be no previous build files to delete. The command will thus succeed.

You will see that Maven writes what project it has found. It will output that to the command prompt. This is a sign that Maven understands your POM. Here is an example of what Maven could output:

D:\data\projects\my-first-maven-project>mvn clean

[INFO] Scanning for projects...

[INFO]

[INFO] ------------------------------------------------------------------------

[INFO] Building hello-world 1.0.0

[INFO] ------------------------------------------------------------------------

[INFO]

[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ hello-world ---

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 0.873 s

[INFO] Finished at: 2015-07-05T14:57:00+02:00

[INFO] Final Memory: 4M/15M

[INFO] ------------------------------------------------------------------------

## Creating a Java Source Directory

src

main

java

That means, a src directory inside the project root directory. Inside the src directory you create a maindirectory. Inside the main directory you create a java directory. The java directory is the root directory for your Java source code.

**Creating a Java Source File**

Inside the Java root source directory (src/main/java) create a new directory (java package) calledhelloworld.

Inside the helloworld directory (java package) insert a file named HelloWorld.java. Inside the HelloWorld.java file you put the following Java code:

package helloworld;

public class HelloWorld {

public static void main(String args[]){

System.out.println("Hello World, Maven");

}

}

Save the file.

**Building the Project**

When you have created the Java source file, open a command prompt and change directory into the project root directory. Then execute this command:

mvn package

The mvn package command instructs Maven to run the package build phase which is part of the default build life cycle

.

Maven should now run. Maven will compile the Java source file and create a JAR file containing the compiled Java class.

Maven creates a target subdirectory inside the project root directory. Inside the target directory you will find the finished JAR file, as well as lots of temporary files (e.g. a classes directory containing all the compiled classes).

The finished JAR file will be named after this pattern:

artifactId-version

So, based on the POM shown earlier in this tutorial, the JAR file will be named:

hello-world-1.0.0.jar

## Maven Command Structure

A Maven command consists of two elements:

* mvn
* One or more build life cycles, build phases or build goals

Here is a Maven command example:

mvn clean

This command consists of the mvn command which executes Maven, and the build life cycle named clean.

Here is another Maven command example:

mvn clean install

This maven command executes the clean build life cycle and the install build phase in the default build life cycle.

You might wonder how you see the difference between a build life cycle, build phase and build goal. I will get back to that later.

## Build Life Cycles, Phases and Goals

* clean
* default
* site

Inside each build life cycle there are build phases, and inside each build phase there are build goals.

You can execute either a build life cycle, build phase or build goal. When executing a build life cycle you execute all build phases (and thus build goals) inside that build life cycle.

When executing a build phase you execute all build goals within that build phase. Maven also executes all build phases earlier in the build life cycle of the desired build phase.

Buid goals are assigned to one or more buid phases. When the build phases are executed, so are all the goals in that build phase. You can also execute a build goal directly.

**Executing Build Life Cycles, Phases and Goals**

When you run the mvn command you pass one or more arguments to it. These arguments specify either a build life cycle, build phase or build goal. For instance to execute the clean build life cycle you execute this command:

mvn clean

To execute the site build life cycle you execute this command:

mvn site

**Executing the Default Life Cycle**

The default life cycle is the build life cycle which generates, compiles, packages etc. your source code.

You cannot execute the default build life cycle directly, as is possible with the clean and site. Instead you have to execute a specific build phase within the default build life cycle.

The most commonly used build phases in the default build life cycle are:

|  |  |
| --- | --- |
| Build Phase | Description |
| validate | Validates that the project is correct and all necessary information is available. This also makes sure the dependencies are downloaded. |
| compile | Compiles the source code of the project. |
| test | Runs the tests against the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed. |
| package | Packs the compiled code in its distributable format, such as a JAR. |
| install | Install the package into the local repository, for use as a dependency in other projects locally. |
| deploy | Copies the final package to the remote repository for sharing with other developers and projects. |

Executing one of these build phases is done by simply adding the build phase after the mvn command, like this:

mvn compile

This example Maven command executes the compile build phase of the default build life cycle. This Maven command also executes all earlier build phases in the default build life cycle, meaning the validatebuild phase.

**Executing Build Phases**

You can execute a build phase located inside a build life cycle by passing the name of the build phase to the Maven command. Here are a few build phase command examples:

mvn pre-clean

mvn compile

mvn package

Maven will find out what build life cycle the specified build phase belongs to, so you don't need to explicitly specify which build life cyle the build phase belongs to.

## Available Maven Archetypes

Maven contains a lot of archetypes, so this Maven archetype tutorial will just show you some of the most commonly used archetypes. To see a full list of Maven archetypes, simply run this command:

mvn archetype:generate

This command actually intends to generate a Maven archetype for you, but since you have not specified in the command which archetype to build, Maven will output all its available archetypes to the command prompt. At the end Maven will ask you which Maven archetype to generate. If you know the number of the archetype to generate, you can type in the number in the command prompt and press enter.

The list contains more than 1.300 Maven archetypes, so it is not really that easy to find the archetype you need. Too look at the list of available Maven archetypes, you can pipe the output into a file, and open that file in e.g. Notepad++ or so. You pipe the available Maven archetypes into a file using this Maven command:

mvn archetype:generate > archetypes.txt

You may have to cancel the command at the point where it asks you to enter the archetype number. You can do so on Windows with CTRL-C. The archetypes will still be written into the file.

## Named Archetypes

Maven contains a set of named archetypes which you can create. I will list a few of these archetypes in the following sections.

### Eclipse

There is a Maven archetype which can generate a new Java project including files for the Eclipse IDE. You can generate that archetype using this Maven command:

mvn eclipse:eclipse

Before you can generate this Maven archetype though, you need to have a POM file in the project root directory into which you want to generate the archetype.

### IDEA Archetype

Similar to the Eclipse archetype, Maven contains an IntelliJ IDEA archetype. You can generate the IDEA archetype using this Maven command:

mvn idea:idea

# Maven Unit Test Report

Maven can generate unit test reports in HTML. Maven does so by running the unit tests and recording the results of the unit tests. Maven then generates an HTML report from the unit test results.

Using Maven to generate unit test reports can be useful to see what unit tests fails during the build. Especially if the build is big and takes a long time, because then you might not run all the unit tests from inside your IDE during development. You might only run the unit tests relevant to the changes you make. Then, to check that everything in the project still works, you can tell Maven to run the unit tests and generate a unit test report. You can then read the report to see if any of the unit test have failed during the build.

The Maven unit test report is an HTML file. That means that it can be copied to a server if you need to share it with other developers.

## The Maven Surefire Plugin

The Maven unit test reports are generated by the Maven Surefire plugin. Therefore a unit test report is also some times referred to as *surefire report*.

## Generating a Unit Test Report

You generate a Maven unit test report (Surefire unit test report) using the following Maven command:

mvn surefire-report:report

For more information about how the Maven command structure looks, see my [**Maven commands tutorial**](http://tutorials.jenkov.com/maven/maven-commands.html).

The generated unit test report can be found in the target/site directory. The unit test report is named surefire-report.html. The path to the unit test report is thus:

your-project/target/site/surefire-report.html

## Skipping the Tests

Sometimes you might want Maven to generate a unit test report without running all the unit tests again. You might just want to use the results from the last run. You can get Maven to just generate the unit test report without rerunning the unit tests using this command:

mvn surefire-report:report-only

This command will only generate the unit test report.