**Maven POM File**

A Maven POM file (Project Object Model) is an XML file that describes the resources of the project. This includes the directories where the source code, test source etc. is located in, what external dependencies (JAR files) your projects has etc.

The POM file describes *what* to build, but most often not *how* to build it. How to build it is up to the Maven build phases and goals. You can insert custom actions (goals) into the Maven build phase if you need to, though.

Each project has a POM file. The POM file is named pom.xml and should be located in the root directory of your project. A project divided into subprojects will typically have one POM file for the parent project, and one POM file for each subproject. This structure allows both the total project to be built in one step, or any of the subprojects to be built separately.

For a full reference of the POM file, see the [**Maven POM Reference**](http://maven.apache.org/pom.html).

Here is a minimal POM file:

<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>java-web-crawler</artifactId>

<version>1.0.0</version>

</project>

The ***modelVersion*** element sets what version of the POM model you are using. Use the one matching the Maven version you are using. Version 4.0.0 matches Maven versions 2 and 3.

The ***groupId*** element is a unique ID for an organization, or a project (an open source project, for instance). Most often you will use a group ID that is similar to the root Java package name of the project. For instance, for my Java Web Crawler project, I may choose the group ID com.jenkov. If the project was an open source project with many independent contributors, perhaps it would make more sense to use a group ID related to the project than an a group ID related to my company. Thus, com.javawebcrawler could be used.

The group ID does not have to be a Java package name, and does not need to use the . notation (dot notation) for separating words in the ID. But, if you do, the project will be located in the Maven repository under a directory structure matching the group ID. Each **.** is replaced with a directory separator, and each word thus represents a directory. The group ID **com.jenkov** would then be located in a directory called **MAVEN\_REPO/com/jenkov**. The **MAVEN\_REPO** part of the directory name will be **replaced with the directory path of the Maven repository(.m2)**.

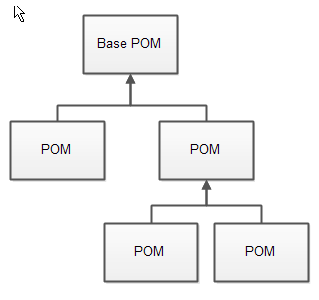
The ***artifactId*** element contains the name of the project you are building. In the case of my Java Web Crawler project, the artifact ID would be java-web-crawler. The artifact ID is used as a name for a subdirectory under the group ID directory in the Maven repository. The artifact ID is also used as part of the name of the JAR file produced when building the project. The output of the build process, the build result that is, is called an artifact in Maven. Most often it is a JAR, WAR or EAR file, but it could also be something else.

The ***versionId*** element contains the version number of the project. If your project has been released in different versions, for instance an open source API, then it is useful to version the builds. That way users of your project can refer to a specific version of your project. The version number is used as a name for a subdirectory under the artifact ID directory. The version number is also used as part of the name of the artifact built.

The above *groupId*, *artifactId* and *version* elements would result in a JAR file being built and put into the local Maven repository at the following path (directory and file name):

**MAVEN\_REPO/com/jenkov/java-web-crawler/1.0.0/java-web-crawler-1.0.0.jar**

**Super POM**

All Maven POM files inherit from a super POM. If no super POM is specified, the POM file inherits from the base POM. Here is a diagram illustrating that:

You can make a POM file explicitly inherit from another POM file. That way you can change the settings across all inheriting POM's via their common super POM. You specify the super POM at the top of a POM file like this:

<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>

<parent>

<groupId>org.codehaus.mojo</groupId>

<artifactId>my-parent</artifactId>

<version>2.0</version>

<relativePath>../my-parent</relativePath>

</parent>

<artifactId>my-project</artifactId>

...

</project>

An inheriting POM file may override settings from a super POM. Just specify new settings in the inheriting POM file.

**Effective POM**

With all this POM inheritance it may be hard to know what the total POM file looks like when Maven executes. The total POM file (result of all inheritance) is called the *effective POM*. You can get Maven to show you the effective POM using this command:

mvn help:effective-pom

This command will make Maven write out the effective POM to the command line prompt.

**Maven Settings File**

Maven has two settings files. In the settings files you can configure settings for Maven across all Maven POM files. For instance, you can configure:

* Location of local repository
* Active build profile
* Etc.

The settings files are called settings.xml. The two settings files are located at:

* The Maven installation directory: $M2\_HOME/conf/settings.xml
* The user's home directory: ${user.home}/.m2/settings.xml

Both files are optional. If both files are present, the values in the user home settings file overrides the values in the Maven installation settings file.

**Running Maven**

When you have [**installed Maven**](https://jenkov.com/tutorials/maven/maven-tutorial.html#installing-maven) and have created a [**POM file**](https://jenkov.com/tutorials/maven/maven-tutorial.html#maven-pom-files) and put the POM file in the root directory of your project, you can run Maven on your project.

Running Maven is done by executing the **mvn** command from a command prompt. When executing the mvn command you pass the name of a [**build life cycle, phase or goal**](https://jenkov.com/tutorials/maven/maven-tutorial.html#maven-build-life-cycles-phases-and-goals) to it, which Maven then executes. Here is an example:

**mvn install**

This command executes the build phase called install (**part of the default build life cycle**), which builds the project and copies the packaged JAR file into the local Maven repository. Actually, this command executes ***all build phases before install in the build phase sequence***, before executing the install build phase.

You can execute multiple build life cycles or phases by passing more than one argument to the mvn command. Here is an example:

**mvn clean install**

This command first executes the clean build life cycle, which removes compiled classes from the Maven output directory, and then it executes the install build phase.

**Maven Directory Structure**

Maven has a standard directory structure. If you follow that directory structure for your project, you do not need to specify the directories of your source code, test code etc. in your POM file.

I have covered the Maven directory structure in more detail here: [**Maven Directory Structure**](https://jenkov.com/tutorials/maven/directory-structure.html).

Here are the most important directories:

- src

- main

- java

- resources

- webapp

- test

- java

- resources

- target

The ***src*** directory is the root directory of your source code and test code. The **main** directory is the root directory for source code related to the application itself (not test code).

The **test** directory contains the test source code. The java directories under main and test contain the Java code for the application itself (under main) and the Java code for the tests (under test).

The **resources** directory contains other resources needed by your project. These could be property files used for the internationalization of an application, or something else.

The **webapp** directory contains your Java web application, if your project is a web application. The webapp directory will then be the root directory of the web application. Thus the webapp directory contains the WEB-INF directory etc.

The target directory is created by Maven. It contains all the compiled classes, JAR files etc. produced by Maven. When executing the clean build phase, it is the target directory that is cleaned.

**Project Dependencies …….**