Apache Maven

The local repository is found in the *.m2/repository* folder of the users home directory, Maven uses this directory to cache downloaded artifacts.

**Proxy Settings**  
  
To configure you proxy, add a file named *.m2/settings.xml* to your user home with the proxy settings, similar to the following:

**<settings>**

**<proxies>**

**<proxy>**

**<id>**example-proxy**</id>**

**<active>**true**</active>**

**<protocol>**http**</protocol>**

**<host>**proxy.example.com**</host>**

**<port>**8080**</port>**

**<username>**proxyuser**</username>** **<password>**somepassword**</password>** **<nonProxyHosts>**www.google.com|\*.example.com**</nonProxyHosts>**

**</proxy>**

**</proxies>**

**</settings>**

By default, the configuration of a Maven project is done via at least one pom.xml configuration file. This name *pom* file is derived from the term ***Project Object Model*.**

The result of a build is called *artifact* and can be, for example, an executable JAR file or zip file.

| **Name** | **Description** |
| --- | --- |
| groupId | Defines a unique base name of the organization or group that created the project. **This is normally a reverse domain name** or the name of an open source project. For the generation of new projects, **the groupId also defines the package of the main class**. |
| artifactId | Defines the unique name of the project within the groupId. If you generate a new project via Maven **this is also used as root folder for the project**. |
| version | This defines the version of the project. If a new version of the project is build, this version should change so that consumers can see that a different version is used. |
| packaging | Defines the packaging method. This could be e.g. a jar, war or ear file. If the packaging type is pom, Maven does not create anything for this project, it is just meta-data. |

**Note:** Each pom file inherits from a parent (super) pom. If no explicit parent pom is defined it inherits from the base pom of Maven.

**Snapshot dependencies**  
  
These are libraries which are constantly under development. Maven downloads these snapshots on every build, even if a matching version is already available in the local repository.

**Excluding transitive dependencies**Sometimes your dependencies have conflicting transitive dependencies. For example, library A requires library C in version 1.0.0 and library B requires it in version 1.1.0. To use library C in version 1.0.0 you can exclude C from the dependencies of A.  
  
Example of excluding:

**<dependency>**

**<groupId>**com.vogella.group**</groupId>** **<artifactId>**com.vogella.example**</artifactId>**

**<version>**1.0**</version>**

**<scope>**compile**</scope>**

**<exclusions>**

**<exclusion>**

**<groupId>**c.groupId**</groupId>**

**<artifactId>**c.artifactId**</artifactId>**

**</exclusion>**

**</exclusions>**

**</dependency>**

**Build Phases**

It is possible for Maven to only execute tests, package the artifact, install the result on local repo or even deploy to a target (i.e. remote repository).

**Maven & Version Control Systems**  
  
Maven generates its output into the ***target*** folder of each project. This build output should not get included into your version control system.

**Dealing with Build failure**  
  
If you are running a complex multi-module project build, you can define how the Maven build system should react to errors in one module.

* -fae, --fail-at-end - fails the build after all modules are build; allow all non-impacted builds to continue
* -ff, --fail-fast - Stop at first module build failure
* -fn, --fail-never - NEVER fail the build, regardless of module build result

The -fn and -fae options are useful to verify builds that are running within a continuous integration tool like Jenkins and to see all errors in the build.

**Scaffolding a project with Maven**You can create a project by executing the generation *goal* on the archetype plugin via the following command: ***mvn archetype:generate***.

For example:

mvn archetype:generate -DgroupId=com.vogella.maven.first \ -DartifactId=com.vogella.maven.first \ -DarchetypeArtifactId=maven-archetype-quickstart \ -DarchetypeVersion=1.4 \ -DinteractiveMode=false

**Specifying the main class in pom file (JAR artifact)**In order to execute the jar, we must specify what the main class is. It can be done as follows:

<?xml version="1.0" encoding="UTF-8"?>

<build>

<plugins> <!-- all the existing build entries-->

<plugin>

<groupId>org.apache.maven.plugins</groupId> <artifactId>maven-jar-plugin</artifactId> <version>3.2.0</version> <configuration> <archive> <manifest> <mainClass>com.vogella.maven.first.App</mainClass> </manifest> </archive> </configuration>

</plugin>

</plugins>

</build>

**Dependencies tree**  
  
Showing the dependencies tree and saving its output to a file:

mvn dependency:tree -Dverbose -DoutputFile=/path/to/file

**Maven vs other build tools**  
  
**Maven vs Ant**  
Ant uses an imperative style of declaring the actions the Ant build should do. Maven is declarative.  
Ant has no dependency management tool by default, other tools need to be used for that.  
  
**Maven vs Gradle**  
Gradle provides the same functionality as Maven but uses a concise syntax.

Gradle supports declarative & imperative styles.  
Maven is maintained by a group of volunteers whereas Gradle is a capital founded company.  
Gradle has more feature which allows a better performance.  
(Gradle is based on a graph of task dependencies – in which tasks are the things that do the work – while Maven is based on a fixed and linear model of phases).