POM stands for Project Object Model. It is fundamental unit of work in Maven. It is an XML file that resides in the base directory of the project as pom.xml.

The POM contains information about the project and various configuration detail used by Maven to build the project(s).

POM also contains the goals and plugins. While executing a task or goal, Maven looks for the POM in the current directory. It reads the POM, gets the needed configuration information, and then executes the goal. Some of the configuration that can be specified in the POM are following −

* project dependencies
* plugins
* goals
* build profiles
* project version
* developers
* mailing list

Before creating a POM, we should first decide the project **group** (groupId), its **name** (artifactId) and its version as these attributes help in uniquely identifying the project in repository.

POM Example

<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.companyname.project-group</groupId>

<artifactId>project</artifactId>

<version>1.0</version>

</project>

It should be noted that there should be a single POM file for each project.

* All POM files require the **project** element and three mandatory fields: **groupId, artifactId, version**.
* Projects notation in repository is **groupId:artifactId:version**.
* Minimal requirements for a POM −

|  |
| --- |
|  |
| **Sr.No.** | **Node & Description** |
| 1 | **Project root**  This is project root tag. You need to specify the basic schema settings such as apache schema and w3.org specification. |
| 2 | **Model version**  Model version should be 4.0.0. |
| 3 | **groupId**  This is an Id of project's group. This is generally unique amongst an organization or a project. For example, a banking group com.company.bank has all bank related projects. |
| 4 | **artifactId**  This is an Id of the project. This is generally name of the project. For example, consumer-banking. Along with the groupId, the artifactId defines the artifact's location within the repository. |
| 5 | **version**  This is the version of the project. Along with the groupId, It is used within an artifact's repository to separate versions from each other. For example −  **com.company.bank:consumer-banking:1.0**  **com.company.bank:consumer-banking:1.1.** |

Super POM

The Super POM is Maven’s default POM. All POMs inherit from a parent or default (despite explicitly defined or not). This base POM is known as the **Super POM**, and contains values inherited by default.

Maven use the effective POM (configuration from super pom plus project configuration) to execute relevant goal. It helps developers to specify minimum configuration detail in his/her pom.xml. Although configurations can be overridden easily.

An easy way to look at the default configurations of the super POM is by running the following command: **mvn help:effective-pom**

Create a pom.xml in any directory on your computer.Use the content of above mentioned example pom.

In example below, We've created a pom.xml in C:\MVN\project folder.

Now open command console, go the folder containing pom.xml and execute the following **mvn** command.

C:\MVN\project>mvn help:effective-pom

Maven will start processing and display the effective-pom.

[INFO] Scanning for projects...

[INFO] Searching repository for plugin with prefix: 'help'.

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

[INFO] Building Unnamed - com.companyname.project-group:project-name:jar:1.0

[INFO] task-segment: [help:effective-pom] (aggregator-style)

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

[INFO] [help:effective-pom {execution: default-cli}]

[INFO]

.....

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

[INFO] BUILD SUCCESSFUL

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

[INFO] Total time: < 1 second

[INFO] Finished at: Thu Jul 05 11:41:51 IST 2012

[INFO] Final Memory: 6M/15M

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

Effective POM displayed as result in console, after inheritance, interpolation, and profiles are applied.

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

<!-- ============================================== -->

<!-- -->

<!-- Generated by Maven Help Plugin on 2015-04-09T11:41:51 -->

<!-- See: http://maven.apache.org/plugins/maven-help-plugin/ -->

<!-- -->

<!-- ==============================================-->

<!-- ==============================================-->

<!-- -->

<!-- Effective POM for project -->

<!-- 'com.companyname.project-group:project-name:jar:1.0' -->

<!-- -->

<!-- ============================================== -->

<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.companyname.project-group</groupId>

<artifactId>project</artifactId>

<version>1.0</version>

<build>

<sourceDirectory>C:\MVN\project\src\main\java</sourceDirectory>

<scriptSourceDirectory>src/main/scripts</scriptSourceDirectory>

<testSourceDirectory>C:\MVN\project\src\test\java</testSourceDirectory>

<outputDirectory>C:\MVN\project\target\classes</outputDirectory>

<testOutputDirectory>C:\MVN\project\target\test-classes</testOutputDirectory>

<resources>

<resource>

<mergeId>resource-0</mergeId>

<directory>C:\MVN\project\src\main\resources</directory>

</resource>

</resources>

<testResources>

<testResource>

<mergeId>resource-1</mergeId>

<directory>C:\MVN\project\src\test\resources</directory>

</testResource>

</testResources>

<directory>C:\MVN\project\target</directory>

<finalName>project-1.0</finalName>

<pluginManagement>

<plugins>

<plugin>

<artifactId>maven-antrun-plugin</artifactId>

<version>1.3</version>

</plugin>

<plugin>

<artifactId>maven-assembly-plugin< /artifactId>

<version>2.2-beta-2</version>

</plugin>

<plugin>

<artifactId>maven-clean-plugin< /artifactId>

<version>2.2</version>

</plugin>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>2.0.2</version>

</plugin>

<plugin>

<artifactId>maven-dependency-plugin</artifactId>

<version>2.0</version>

</plugin>

<plugin>

<artifactId>maven-deploy-plugin</artifactId>

<version>2.4</version>

</plugin>

<plugin>

<artifactId>maven-ear-plugin</artifactId>

<version>2.3.1</version>

</plugin>

<plugin>

<artifactId>maven-ejb-plugin</artifactId>

<version>2.1</version>

</plugin>

<plugin>

<artifactId>maven-install-plugin</artifactId>

<version>2.2</version>

</plugin>

<plugin>

<artifactId>maven-jar-plugin</artifactId>

<version>2.2</version>

</plugin>

<plugin>

<artifactId>maven-javadoc-plugin</artifactId>

<version>2.5</version>

</plugin>

<plugin>

<artifactId>maven-plugin-plugin</artifactId>

<version>2.4.3</version>

</plugin>

<plugin>

<artifactId>maven-rar-plugin</artifactId>

<version>2.2</version>

</plugin>

<plugin>

<artifactId>maven-release-plugin</artifactId>

<version>2.0-beta-8</version>

</plugin>

<plugin>

<artifactId>maven-resources-plugin</artifactId>

<version>2.3</version>

</plugin>

<plugin>

<artifactId>maven-site-plugin</artifactId>

<version>2.0-beta-7</version>

</plugin>

<plugin>

<artifactId>maven-source-plugin</artifactId>

<version>2.0.4</version>

</plugin>

<plugin>

<artifactId>maven-surefire-plugin</artifactId>

<version>2.4.3</version>

</plugin>

<plugin>

<artifactId>maven-war-plugin</artifactId>

<version>2.1-alpha-2</version>

</plugin>

</plugins>

</pluginManagement>

<plugins>

<plugin>

<artifactId>maven-help-plugin</artifactId>

<version>2.1.1</version>

</plugin>

</plugins>

</build>

<repositories>

<repository>

<snapshots>

<enabled>false</enabled>

</snapshots>

<id>central</id>

<name>Maven Repository Switchboard</name>

<url>http://repo1.maven.org/maven2</url>

</repository>

</repositories>

<pluginRepositories>

<pluginRepository>

<releases>

<updatePolicy>never</updatePolicy>

</releases>

<snapshots>

<enabled>false</enabled>

</snapshots>

<id>central</id>

<name>Maven Plugin Repository</name>

<url>http://repo1.maven.org/maven2</url>

</pluginRepository>

</pluginRepositories>

<reporting>

<outputDirectory>C:\MVN\project\target/site</outputDirectory>

</reporting>

</project>

In above pom.xml, you can see the default project source folders structure, output directory, plug-ins required, repositories, reporting directory, which Maven will be using while executing the desired goals.

Maven pom.xml is also not required to be written manually. Maven provides numerous archetype plugins to create projects, which in order, create the project structure and pom.xml

What is Build Profile?

A Build profile is a set of configuration values, which can be used to set or override default values of Maven build. Using a build profile, you can customize build for different environments such as Production v/s Development environments.

Profiles are specified in pom.xml file using its activeProfiles/profiles elements and are triggered in variety of ways. Profiles modify the POM at build time, and are used to give parameters different target environments (for example, the path of the database server in the development, testing, and production environments).

Types of Build Profile

Build profiles are majorly of three types.

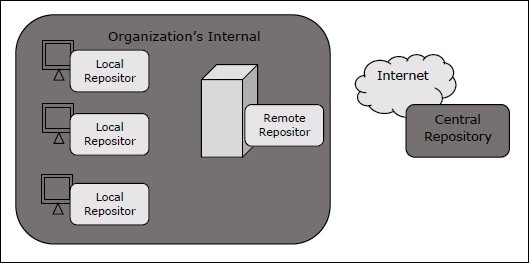
|  |  |
| --- | --- |
| **Type** | **Where it is defined** |
| Per Project | Defined in the project POM file, pom.xml |
| Per User | Defined in Maven settings xml file (%USER\_HOME%/.m2/settings.xml) |
| Global | Defined in Maven global settings xml file (%M2\_HOME%/conf/settings.xml) |

What is a Maven Repository?

In Maven terminology, a repository is a directory where all the project jars, library jar, plugins or any other project specific artifacts are stored and can be used by Maven easily.

Maven repository are of three types. The following illustration will give an idea regarding these three types.

* local
* central
* remote



Local Repository

Maven local repository is a folder location on your machine. It gets created when you run any maven command for the first time.

Maven local repository keeps your project's all dependencies (library jars, plugin jars etc.). When you run a Maven build, then Maven automatically downloads all the dependency jars into the local repository. It helps to avoid references to dependencies stored on remote machine every time a project is build.

Maven local repository by default get created by Maven in %USER\_HOME% directory. To override the default location, mention another path in Maven settings.xml file available at %M2\_HOME%\conf directory.

<settings xmlns = "http://maven.apache.org/SETTINGS/1.0.0"

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

xsi:schemaLocation = "http://maven.apache.org/SETTINGS/1.0.0

http://maven.apache.org/xsd/settings-1.0.0.xsd">

<localRepository>C:/MyLocalRepository</localRepository>

</settings>

When you run Maven command, Maven will download dependencies to your custom path.

Central Repository

Maven central repository is repository provided by Maven community. It contains a large number of commonly used libraries.

When Maven does not find any dependency in local repository, it starts searching in central repository using following URL − <https://repo1.maven.org/maven2/>

Key concepts of Central repository are as follows −

* This repository is managed by Maven community.
* It is not required to be configured.
* It requires internet access to be searched.

To browse the content of central maven repository, maven community has provided a URL − <https://search.maven.org/#browse>. Using this library, a developer can search all the available libraries in central repository.

Remote Repository

Sometimes, Maven does not find a mentioned dependency in central repository as well. It then stops the build process and output error message to console. To prevent such situation, Maven provides concept of **Remote Repository**, which is developer's own custom repository containing required libraries or other project jars.

For example, using below mentioned POM.xml, Maven will download dependency (not available in central repository) from Remote Repositories mentioned in the same pom.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.companyname.projectgroup</groupId>

<artifactId>project</artifactId>

<version>1.0</version>

<dependencies>

<dependency>

<groupId>com.companyname.common-lib</groupId>

<artifactId>common-lib</artifactId>

<version>1.0.0</version>

</dependency>

<dependencies>

<repositories>

<repository>

<id>companyname.lib1</id>

<url>http://download.companyname.org/maven2/lib1</url>

</repository>

<repository>

<id>companyname.lib2</id>

<url>http://download.companyname.org/maven2/lib2</url>

</repository>

</repositories>

</project>

Maven Dependency Search Sequence

When we execute Maven build commands, Maven starts looking for dependency libraries in the following sequence −

* **Step 1** − Search dependency in local repository, if not found, move to step 2 else perform the further processing.
* **Step 2** − Search dependency in central repository, if not found and remote repository/repositories is/are mentioned then move to step 4. Else it is downloaded to local repository for future reference.
* **Step 3** − If a remote repository has not been mentioned, Maven simply stops the processing and throws error (Unable to find dependency).
* **Step 4** − Search dependency in remote repository or repositories, if found then it is downloaded to local repository for future reference. Otherwise, Maven stops processing and throws error (Unable to find dependency).

What are Maven Plugins?

Maven is actually a plugin execution framework where every task is actually done by plugins. Maven Plugins are generally used to −

* create jar file
* create war file
* compile code files
* unit testing of code
* create project documentation
* create project reports

A plugin generally provides a set of goals, which can be executed using the following syntax −

mvn [plugin-name]:[goal-name]

For example, a Java project can be compiled with the maven-compiler-plugin's compile-goal by running the following command.

mvn compiler:compile

Plugin Types

Maven provided the following two types of Plugins −

|  |  |
| --- | --- |
| **Sr.No.** | **Type & Description** |
| 1 | **Build plugins**  They execute during the build process and should be configured in the <build/> element of pom.xml. |
| 2 | **Reporting plugins**  They execute during the site generation process and they should be configured in the <reporting/> element of the pom.xml. |

Following is the list of few common plugins −

|  |  |
| --- | --- |
| **Sr.No.** | **Plugin & Description** |
| 1 | **clean**  Cleans up target after the build. Deletes the target directory. |
| 2 | **compiler**  Compiles Java source files. |
| 3 | **surefire**  Runs the JUnit unit tests. Creates test reports. |
| 4 | **jar**  Builds a JAR file from the current project. |
| 5 | **war**  Builds a WAR file from the current project. |
| 6 | **javadoc**  Generates Javadoc for the project. |
| 7 | **antrun**  Runs a set of ant tasks from any phase mentioned of the build. |