POM IN MAVEN

A Project Object Model or POM is the fundamental unit of work in Maven. It is an XML file that contains information about the project and configuration details used by Maven to build the project. It contains default values for most projects.

Examples for this is the build directory, which is target; the source directory, which is src/main/java; the test source directory, which is src/test/java; and so on.

When executing a task or goal, Maven looks for the POM in the current directory. It reads the POM, gets the needed configuration information, then executes the goal.

Some of configuration that can be specified in the POM are the project dependencies, the plugins or goals that can be executed, the build profiles, and so on. Other information such as the project version, description, developers, mailing lists and such can also be specified.

The minimum requirement for a POM are the following:

* project root
* modelVersion - should be set to 4.0.0
* groupId - the id of the project's group.
* artifactId - the id of the artifact (project)
* version - the version of the artifact under the specified group

Here's an example:

1. <project>
2. <modelVersion>4.0.0</modelVersion>
4. <groupId>com.mycompany.app</groupId>
5. <artifactId>my-app</artifactId>
6. <version>1</version>
7. </project>

**What are the dependencies u added for your project in the POM file?**

|  |  |
| --- | --- |
| **groupId** | **artifactId** |
| **io.github.sukgu** | **automation** |
| **Com.aventstack** | **extentreports** |
| **commons-io** | **commons-io** |
| **io.cucumber** | **cucumber-java** |
| **io.cucumber** | **cucumber-testng** |
| **org.apache.poi** | **poi-ooxml** |
| **org.apache.poi** | **poi** |
| **org.testng** | **testng** |
| **org.seleniumhq.selenium** | **selenium-java** |
| **io.github.bonigarcia** | **webdrivermanager** |
| **test** | **seleniumproject** |

**What do you mean by Maven?**

**Answer**: Maven is a project management tool (introduced by Apache Software Foundation) that provides an entire framework for the build cycle. It is open-source and is mainly used for the project developed in Java.

Maven is driven by a project object model popularly known as POM. It is the central repository for all the dependencies. It maintains the same folder convention across organizations and can be easily integrated with continuous integration tools like Jenkins.

Besides, it takes care of the reporting and documentation of the project. It is developed in Java language.

**What are the features or advantages of Maven?**

**Answer**:

**The features or advantages of Maven are as follows:**

* Not required to manually add jars for the project. The updates to the project dependencies and transitive dependencies are carried out automatically by Maven.
* Maven maintains a uniform directory structure across the organization.
* Both the deployment and build activities are taken care by Maven.
* Maven is simple, easy to set up and utilize in the projects as it is driven by the POM file.
* Maven contains a large number of libraries that can be used for multiple projects at a time.

**Various other build tools that you are heard of?**

Apart from Apache Maven, I have heard of:

* Gradle
* Apache Ant
* Terraform
* SBT(Scala Oriented Build Tool)
* Apache Buildr

**What Plugins used in Maven and why it is used?**

Maven-compiler-plugin (To compile)

Maven-surefire-plugin (To Execute)

Maven Plugins are used to:

Create a JAR file.

Create a WAR file.

Compile code files.

Unit testing of code.

Create project documentation.

Create project reports.

How do you know the version of Maven being used?

mvn –version is used to check the version of Maven present in the system.

 What is Clean, Default, and Site in Maven?

There are three built-in build life cycles:

* Clean: The clean lifecycle looks after project cleaning.
* Default: The default lifecycle handles the project deployment.
* Site: The site lifecycle refers to the creation of the project’s site documentation.

### What is a Maven Repository?

Maven repositories refer to the directories of packaged JAR files that contain metadata. The metadata refers to the POM files relevant to each project. This metadata is what allows Maven to download dependencies.

There are three types of Maven repositories:

#### **1. Local Repository:**

* Local repository refers to the machine of the developer where all the project material is saved.
* The local repository contains all the dependency jars.
* Maven downloads all the required dependencies and store it in local repository called m2(.m2)

#### **2. Remote Repository:**

* The remote repository refers to the repository present on a server that is used when Maven needs to download dependencies.
* Whenever anything is required from the remote repository, it is first downloaded to the local repository, and then it is used.

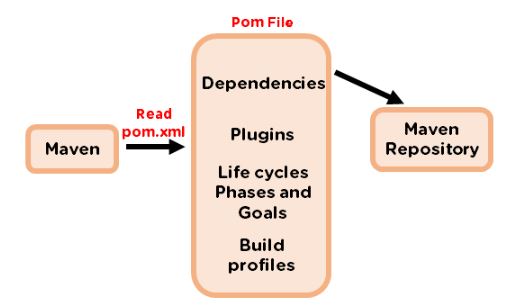
#### **3. Central Repository:**

* Central repository refers to the Maven community that comes into action when there is a need for dependencies, and those dependencies cannot be found in the local repository.
* Maven downloads the dependencies from here in the local repository whenever needed.

How does Maven Architecture work?

Maven architecture works in three steps, which are as follows:

* The first step is to read the pom.xml file.
* Then, it downloads the dependencies defined in pom.xml into the local repository from the central repository.
* Lastly, it creates and generates a report according to the requirements, and executes life-cycles, phases, goals, plugins, etc.



### What are the different phases in the Maven Build Lifecycle?

### //For your reference:

### Clean: to clean the project after its lifecycle

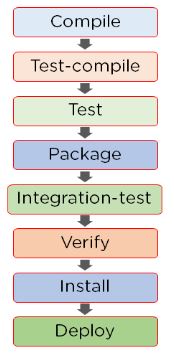
### Compile: used to compile the source code of the project

### Test: test the compile code, these test do not require to be packaged or deplyoed

### Package: used to convert your project into jar or war file

### Install: install package into local repository for use of another project.

Build Lifecycle has different build phases or stages, which are below:



**validate** − validate the project and check if everything is correct and all necessary information is available.

**compile** − this phase compiles the source code of your project.

**test** − tests the compiled source code by using a suitable unit testing framework. These tests should not require the code to be packaged or deployed

**package** − takes the compiled code and packages it in its distributable format.

**integration-test** − processes and deploys the package if possible into an environment where integration tests can be run.

**verify** − runs any checks to verify the package is valid and meets the required quality criteria.

**install** − installation of the package into the local repository. This is done to use it as a dependency in other projects locally.

**deploy** − done in an integration environment or release environment. Here the final package is

## **List the differences between ANT and Maven.**

|  |  |
| --- | --- |
| **Ant** | **Maven** |
| Ant doesn’t have formal conventions, so we need to provide information about the project structure in the build.xml file. | Maven has a convention to place source code, compiled code, etc. So we don’t need to provide information about the project structure in pom.xml file. |
| Ant is procedural, you need to provide information about what to do and when to do through code. You need to provide order. | Maven is declarative, everything you define in the pom.xml file. |
| There is no life cycle in Ant. | There is a life cycle in Maven. |
| Ant is a toolbox. | Maven is a framework. |
| It is mainly a build tool. | It is mainly a project management tool. |
| The ant scripts are not reusable. | The maven plugins are reusable. |
| It is less preferred than Maven. | It is more preferred than Ant. |

Difference Between Maven, Ant and Jenkins

| **Maven** | **Ant** | **Jenkins** |
| --- | --- | --- |
| It is a Build Automation Tool. | Java Library/Command Line Tool. | Continuous Integration Tool. |
| Defines how the software is built and describes the software dependencies. | Drives build process. | Automates the software development process with Continuous Integration and facilitates Continuous Deliver. |
| Supports projects written in C#, Ruby. | Supports projects written in C and C++. | Supports version control tools like Git, AccuRev. |
| Executes Unit Tests as a part of the normal build cycle. | Supports single file execution introduced with Java II. | Can execute Apache Ant and Apache Maven. |