**Maven**

**What is Maven?**

Maven is a powerful project management tool that is based on POM (project object model). It is used for projects build, dependency and documentation. It simplifies the build process like ANT. But it is too much advanced than ANT.

**What does maven do?**

It does mainly following tasks.

1. **Adding set of Jars in each project:** In case of struts, spring, hibernate frameworks, we need to add set of jar files in each project. It must include all the dependencies of jars also.
2. **Creating the right project structure:** We must create the right project structure in servlet, struts etc, otherwise it will not be executed.
3. **Building and deploying the project:** We must have to build and deploy the project so that it may work.
4. It makes a project easy to build
5. It provides uniform build process (maven project can be shared by all the maven projects)
6. It provides project information (log document, cross referenced sources, mailing list, dependency list, unit test reports etc.)
7. It is easy to migrate for new features of Maven

**Maven** **Repository:**

A maven repository is a directory of packaged JAR file with pom.xml file. Maven searches for dependencies in the repositories. There are 3 types of maven repository:

1. Local Repository
2. Central Repository
3. Remote Repository

Maven searches for the dependencies in the following order:

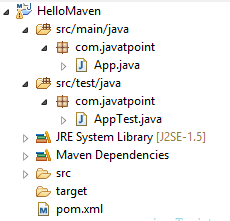
Local repository then Central repository then Remote repository.

1. By default, maven local repository is %USER\_HOME%/.m2 directory.
2. The path of central repository is: <http://repo1.maven.org/maven2/>.
3. Maven **remote repository** is located on the web. Most of libraries can be missing from the central repository such as JBoss library etc, so we need to define remote repository in pom.xml file.

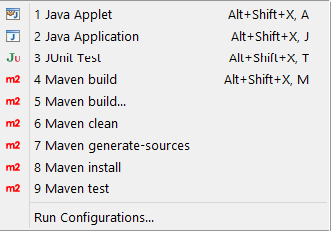
**Maven Eclipse Example**

In eclipse, click on File menu → New → Project → Maven → Maven Project. → Next → Next → Next. Now write the group Id, artifact Id, Package as shown in below figure → finish.

Now you will see a maven project with complete directory structure. All the files will be created automatically such as Hello Java file, pom.xml file, test case file etc. The directory structure of the maven project is shown in the below figure.



If you right click on the project → Run As, you will see the maven options to build the project.



**Maven Build Life cycle:**

Maven is based around the central concept of a build lifecycle. What this means is that the process for building and distributing a particular artifact (project) is clearly defined.

For the person building a project, this means that it is only necessary to learn a small set of commands to build any Maven project, and the POM will ensure they get the results they desired.

There are three built-in build lifecycles: default, clean and site. The default lifecycle handles your project deployment, the clean lifecycle handles project cleaning, while the site lifecycle handles the creation of your project's site documentation.

**A Build Lifecycle is Made Up of Phases:**

Each of these build lifecycles is defined by a different list of build phases, wherein a build phase represents a stage in the lifecycle.

For example, the default lifecycle comprises of the following phases (for a complete list of the lifecycle phases, refer to the Lifecycle Reference):

**Validate:** - validate the project is correct and all necessary information is available

**Compile:** - compile the source code of the project

**Test:** - test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed

**Package:** - take the compiled code and package it in its distributable format, such as a JAR.

**Verify:** - run any checks on results of integration tests to ensure quality criteria are met

**Install:** - install the package into the local repository, for use as a dependency in other projects locally

**Deploy:** - done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.

These lifecycle phases (plus the other lifecycle phases not shown here) are executed sequentially to complete the default lifecycle. Given the lifecycle phases above, this means that when the default lifecycle is used, Maven will first validate the project, then will try to compile the sources, run those against the tests, package the binaries (e.g. jar), run integration tests against that package, verify the integration tests, install the verified package to the local repository, then deploy the installed package to a remote repository.

**Maven pom.xml file**

**POM** is an acronym for **Project Object Model**. Maven reads the pom.xml file, then executes the goal.

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 maven 2, it was named as project.xml file. But, since maven 2 (also in maven 3), it is renamed as pom.xml.

**Automatically Generated pom.xml file**



Group Id and artifact Id will come as package name in the project.

**Maven Commands:**

Command to execute processes needed prior to the actual project cleaning: mvn pre-clean

Command to remove all files generated by the previous build: mvn clean

Command to execute processes needed to finalize the project cleaning: mvn post-clean

Command to compile the source code: mvn compile

Command to run the tests in project: mvn test

Command to package the code as motioned in pom file: mvn package

Command to install package in the local repository: mvn install

Command to copy the jar file to the remote repository: mvn deploy

Command to add the dependency: mvn dependency:sources

Command to add the java doc to the project is : mvn dependency:resolve –Dclassifier=javadoc

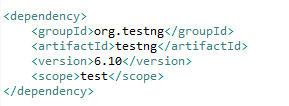
Command to run a profile in the in the project is: mvn test -psanityTest

**How to add dependencies to the project:**

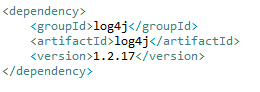
Navigate to the pom.xml and add the required decadency. So when we build our project it looks for dependencies in the local Repository and if it don’t find it in the local repository it will download from the central repository and store it in the local repository. It is not required for us to download the dependencies and add it to the project, maven will take care of it.

**Examples:**

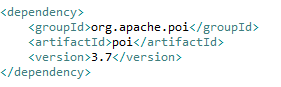
**TestNG Dependency:**

****

**Log4J** **Dependency:**

****

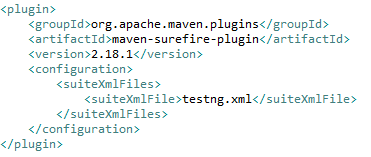
**Apache POI Dependency:**

****

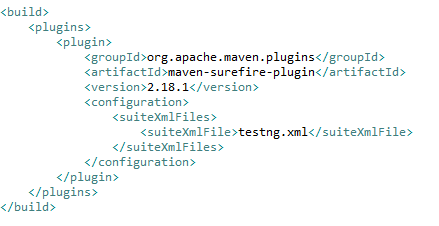
**Site:** https://mvnrepository.com/

**Maven Plugin’s:**

plugins are used to create jar files, create war files, compile code, unit test code, create project documentation, and on and on. Almost any action that you can think of performing on a project is implemented as a **Maven plugin**.



On the top of plugins we have build tag which process for building and distributing a particular artifact (project) is clearly defined.



Using a build **profile**, you can customize build for different environments such as Production v/s Development environments. In the below example we have integrated TestNG with Maven.

Command to run tests bases in the profile is : mvn test -PSanityTest

