**Maven:**

* Is a **build** and **dependency management tool** for Java based application development.

**Maven installation:**

* It is not installed as Windows service, rather it is configured using environment variables.

To run maven from command prompt, it is necessary to update PATH variable to **bin** directory.

[ note: JDK must be already installed and environment variables updated.]

To verify maven installation: **>** mvn –version

**Location of the Maven HOME directory: >**${user.home}/.m2/repository

When we compile a Maven project, Maven will download all the project’s dependency and plugins into Maven local repository, saves time for next complication.

To check local repository: **>** mvn help:evaluate -Dexpression=settings.localRepository

**Maven – Settings File:**

* Maven setting.xml file contains configuration that are not specific to a project but are global in nature.
* It also contains information that I not meant to be distributed (e.g. passwords).

**Location of Maven Setting File:**

The Maven installation directory: $M2\_HOME/conf/settings.xml [global settings]

The user’s home directory: ${user.home}/.m2/settings.xml [user settings]

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

<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/> -- copies plugins and dependencies (artifacts) locally.

<interactiveMode/> --will determine whether maven prompts you when it needs input. (true).

<usePluginRegistry/> -- decides ${user.home}/.m2/plugin-repository.xml file plug-in versions.

<offline/> -- instruct Maven to operate in an offline mode. Default is false.

<pluginGroups/> -- contains a list of pluginGroup elements

<servers/> -- Maven can interact with variety of Server (allows to specify security credentials).

<mirrors/> -- allows you to specify alternative location for repositories.

<proxies/> -- contains the HTTP proxy info need to connect to the internet.

<profiles/> -- allows to group certain configuration elements.

<activeProfiles/> -- allows to specify a default profile to be active for Maven to use.

</settings>

**Maven Dependency Management:**

* In Maven, dependency is another archive-JAR, ZIP, etc. on which the **current project needs** in order to **compile, build, test and/or to run.**
* The dependencies are gathered in **pom.xml** file, inside of a <dependencies> tag.

When we run a build or execute a maven goal, these dependencies are resolved, and are then loaded from the local repository. If they are not present there, then Maven will download them from remote repository and store them in local repository (we are allowed to manually install as well).

**Maven Dependency Example:**

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>io.cucumber</groupId>

<artifactId>cucumber-junit</artifactId>

<version>4.7.4</version>

</dependency>

</dependencies>

If pom.xml points to many artifacts of same groupId, then we use **properties** in order to factorize the code for easy maintenance.

<properties>

<junit.version>4.12</junit.version>

<cucumber.version>4.3.5.RELEASE</cucumber.version>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>${junit.version}</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>io.cucumber</groupId>

<artifactId>cucumber-junit</artifactId>

<version>${cucumber.version}</version>

</dependency>

</dependencies>

**External Dependency:** Sometimes we have to refer jar files which are not Maven repository. We use these jars by placing them in project’s lib folder and configure the external dependency.

<dependency>

<groupId>extDependency</groupId> --set to name extDependency

<artifactId>extDependency</artifactId> -- set to name extDependency

<scope>system</scope> -- scope element value is set to system.

<version>1.0</version>

<systemPath>${basedir}\war\WEB-INF\lib\extDependency.jar</systemPath> --refers to JAR file.

</dependency>

**Maven Dependency Tree:**  To view list of all dependences into the project.

**>**mvn dependency:tree

**Maven Dependency Exclusion:**

Apart from version mismatches issue caused with transitive dependency there can be version mismatch between project artifacts and artifacts from the platform of deployment (Servers).

To resolve such version mismatches issues Maven provides <exclusion> tag in order to break the transitive dependency.

For example we have junit 4.12 in classpath and including DBUnit dependency, then to remove Junit dependency can be done by <exclusion> tag.

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>${junit.version}</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.dbunit</groupId>

<artifactId>dbunit</artifactId>

<version>${dbunit.version}</version>

<scope>test</scope>

<exclusions>

<!--Exclude transitive dependency to JUnit-3.8.2 -->

<exclusion>

<artifactId>junit</artifactId>

<groupId>junit</groupId>

</exclusion>

</exclusions>

</dependency>

**Maven Dependency Scopes:**

* Scope attribute is used to specify the visibility of a dependency, relative to different lifecycle phases (**build, test, runtime,** etc.).

Maven provides six scopes

Compile scope:

Provided scope:

Runtime scope:

Test scope:

System scope:

Import scope:

Transitivity Resolution:

**Difference between Dependency and plugins:**

A normal dependency are artifacts we use in your application like libraries.a packaged piece of classes that the project depends on. It can be jar, war etc. For example, if we want to be able to write JUnit test, we have to use JUnit annotations and classes thus we declare that the project depends on JUnit.

A plugin is a tool that is used at the execution of your maven build (used to accomplish different tasks). For example, to compile the project maven uses maven-compiler-plugin, to run tests - maven-surefire-plugin and so on.

(for example, the compiler plugin compiles your Java code into Bytecode).

Both plugins and artifacts can have transitive dependencies. That means they have dependencies themselves.