**Apache Maven -3.5.4**

Maven is a build tool , help to test and run the java program independently

Maven runs purely based on POM xml [project object model] , it stores the configuration details

**Installation**

Download the Binary zip archive  apache-maven-3.6.0-bin.zip file maven apache website

Please note: it is open source , so no need to pay

Configure the variable and path details in your environmental variable gui , before doing all this install java on your machine , because maven is completely based on java

**Start with maven**

Cross check your maven by checking the version

#mvn –version

1. Apache Maven 3.6.0 (97c98ec64a1fdfee7767ce5ffb20918da4f719f3; 2018-10-24T20:41:47+02:00)
2. Maven home: D:\apache-maven-3.6.0\bin\..
3. Java version: 1.8.0\_161, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk1.8.0\_161\jre
4. Default locale: nl\_NL, platform encoding: Cp1252
5. OS name: "windows 7", version: "6.1", arch: "amd64", family: "windows"

Use below command to **create a project**

For example if you are running this in d:/firstproject path

mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=**my-app** -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4 -DinteractiveMode=false

explanation

archetype:generate  creating a new archetype project

groupId=com.mycompany.app   a unique naming schema for the Java's package

artifactId=my-app  name of the jar file without version which we are going to create while executing the maven packages

archetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4  informing maven what archetype to use to create the initial structure of the project. Maven looks it up from the archetypeCatalog.

interactiveMode=false  no interactive with the maven

You may test the newly compiled and packaged JAR with the following command:

java -cp target/my-app-1.0-SNAPSHOT.jar com.mycompany.app.App  gives the output

this is how the **directory structure** will be there in maven [if you create maven with this -DgroupId=com.mycompany.app -DartifactId=**my-app]**

1. my-app
2. |-- pom.xml
3. `-- src
4. |-- main
5. | `-- java
6. | `-- com
7. | `-- mycompany
8. | `-- app
9. | `-- App.java
10. `-- test
11. `-- java
12. `-- com
13. `-- mycompany
14. `-- app
15. `-- AppTest.java

As mentioned above POM xml is the heart of the maven , it guides the maven project with what and how to do..

url to know about pom  <https://maven.apache.org/pom.html#Build>

Basic POM XML files consist of

1. <project xmlns="<http://maven.apache.org/POM/4.0.0>"
2. xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"
3. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4. <http://maven.apache.org/xsd/maven-4.0.0.xsd>">
5. <modelVersion>4.0.0</modelVersion>
7. <groupId> com.mycompany.app</groupId>
8. <artifactId>my-app</artifactId>
9. <version>1.0</version>
10. </project>

If we use below command it will create a Default POM xml file like above ..

mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=**my-app** -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4 -DinteractiveMode=false

**Dependencies - POM**

By default almost all java programs needs some dependencies , so maven downloads and links the dependencies on compilation . As an added bonus, Maven brings in the dependencies of those dependencies too .

So go to maven repository using below url and search for your requirement and paste the same in your POM XML

<https://mvnrepository.com/>

such as here we need checkstyle plugin to check the complete code , checkstyle will give complete reports of your code warning and errors

so just search in maven repository and get the dependency codes and paste in your pom xml , it will help to fetch the same from internet and helps to run your java program

<dependency>

    <groupId>com.puppycrawl.tools</groupId>

    <artifactId>checkstyle</artifactId>

    <version>8.12</version>

</dependency>

One more example like combination of junit and checkstyle plugins

  <dependencies>

<dependency>

      <groupId>junit</groupId>

      <artifactId>junit</artifactId>

      <version>4.12</version>

      <type>jar</type>

      <scope>test</scope>

      <optional>true</optional>

    </dependency>

<dependency>

    <groupId>com.puppycrawl.tools</groupId>

    <artifactId>checkstyle</artifactId>

    <version>8.12</version>

</dependency>

  </dependencies>

**Build - POM**

Build has two kinds of functions , They are the build element, that handles things like declaring your project's directory structure and managing plugins; and the reporting element,

1 ) **base build**  declaring the folder and file paths like where to generate the jar or war files also we can instruct the maven build to look for specifically

Example of how to **mentioning the target files**..

<build>

  <defaultGoal>install</defaultGoal>

  <directory>${basedir}/target</directory>

  <finalName>${artifactId}-${version}</finalName>

  <filters>

    <filter>filters/filter1.properties</filter>

  </filters>

  ...

</build>

One more example of how to **mention specifically** to search for files

<build>

    ...

    <resources>

      <resource>

        <targetPath>META-INF/plexus</targetPath>

        <filtering>false</filtering>

        <directory>${basedir}/src/main/plexus</directory>

        <includes>

          <include>configuration.xml</include>

        </includes>

        <excludes>

          <exclude>\*\*/\*.properties</exclude>

        </excludes>

      </resource>

    </resources>

    <testResources>

      ...

    </testResources>

    ...

  </build>

2) build plugin  looping the plugin under build

Simple example

<build>

    <pluginManagement><!-- lock down plugins versions to avoid using Maven defaults (may be moved to parent pom) -->

      <plugins>

        <plugin>

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

          <version>3.0.0</version>

        </plugin>

                   </plugins>

</build>

By mentioning with the **few specifications**

  <build>

    ...

    <plugins>

      <plugin>

        <groupId>org.apache.maven.plugins</groupId>

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

        <version>2.6</version>

        <extensions>false</extensions>

**<inherited>true</inherited>**

**<configuration>**

          <classifier>test</classifier>

        </configuration>

        <dependencies>...</dependencies>

        <executions>...</executions>

      </plugin>

    </plugins>

  </build>

Still more there in POM files such as Developers , contributors, profile , Continuous integration management , SCM , Mailing , repositories we can refer that in the below url

<https://maven.apache.org/pom.html#Properties>

**Maven Build cycle**

Maven can execute in multiple ways

1. Eclipse it is java tool
2. Console mode

In order to use the console mode ,  we need to install a plugin – Surefire plugin under build or dependencies[google it and paste it]

Build lifecycles is a process of defining the building and distributing a particular artifact project

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,

validate - validate the project is correct and all necessary information is available

compile - compile the source code of the project

clean – handles the project cleaning , such as removing unwanted dependencies

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.

Default Lifecycle

validate --> validate the project is correct and all necessary information is available.

initialize --> initialize build state, e.g. set properties or create directories.

generate-sources --> generate any source code for inclusion in compilation.

process-sources --> process the source code, for example to filter any values.

generate-resources --> generate resources for inclusion in the package.

process-resources --> copy and process the resources into the destination directory, ready for packaging.

compile --> compile the source code of the project.

process-classes --> post-process the generated files from compilation, for example to do bytecode enhancement on Java classes.

generate-test-sources --> generate any test source code for inclusion in compilation.

process-test-sources --> process the test source code, for example to filter any values.

generate-test-resources --> create resources for testing.

process-test-resources --> copy and process the resources into the test destination directory.

test-compile --> compile the test source code into the test destination directory

process-test-classes --> post-process the generated files from test compilation, for example to do bytecode enhancement on Java classes. For Maven 2.0.5 and above.

test --> run tests using a suitable unit testing framework. These tests should not require the code be packaged or deployed.

prepare-package --> perform any operations necessary to prepare a package before the actual packaging. This often results in an unpacked, processed version of the package. (Maven 2.1 and above)

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

pre-integration-test --> perform actions required before integration tests are executed. This may involve things such as setting up the required environment.

integration-test --> process and deploy the package if necessary into an environment where integration tests can be run.

post-integration-test --> perform actions required after integration tests have been executed. This may including cleaning up the environment.

verify --> run any checks to verify the package is valid and meets quality criteria.

install --> install the package into the local repository, for use as a dependency in other projects locally.

deploy --> done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.

pre-site --> execute processes needed prior to the actual project site generation

site --> generate the project's site documentation

post-site --> execute processes needed to finalize the site generation, and to prepare for site deployment

site-deploy --> deploy the generated site documentation to the specified web server

pre-clean --> execute processes needed prior to the actual project cleaning

clean --> remove all files generated by the previous build

post-clean --> execute processes needed to finalize the project cleaning

**Compiling a maven project**

Go to the concetn path , and execute the above commands like

#mvn clean

#mvn test

#mvn install

Or even

#mvn clean deploy  The same command can be used in a multi-module scenario (i.e. a project with one or more subprojects). Maven traverses into every subproject and executes clean, then executes deploy (including all of the prior build phase steps).

Thanks,

Naveen S M