**Build Automation**

Build automation can be defined as the process of scripting and automating the compiling, running, deploying, and packaging of a source code.

For each kind of software language in the industry there are a variety of build tools that can be used for build automation. When it comes to Java or Java related languages, following are the major build tools that are being used by the software industry:

1. Ant
2. Maven
3. Gradle

**Ant (Another Neat Tool)**

Ant is one of the most commonly used build tools by the software industry for the build automation of Java-based products. It is configured using an XML file and by default the configuration file name for Ant is named as *build.xml*.

The main benefit of Ant is its flexibility. Ant doesn’t impose any coding conventions or project structures. Consequently, this means that Ant requires developers to write all the commands by themselves, which sometimes leads to huge XML build files which are hard to maintain.

**Since there are no conventions, just knowing Ant does not mean we’ll quickly understand any Ant build file. It’ll likely take some time to get accustomed with an unfamiliar Ant file, which is a disadvantage compared the other, newer tools.**

**Maven:**

Apache Maven is a popular tool for **build automation**, primarily Java projects. Maven

addresses two aspects of building software. First, it describes how software is built

and, second, it describes its dependencies. It uses conventions for the build procedure.

An XML file describes the software project being built, its dependencies on other external modules and components, the build order, directories, and required plugins. It comes with predefined targets to perform certain well-defined tasks, such as code compilation and its packaging. Maven dynamically downloads Java libraries and Maven plugins from one or more repositories, such as the Maven Central Repository, and stores them locally.

Ant modeled has targets and dependencies. Each target has a set of tasks. Ant doesn't have any conventions. It is procedural and does not have the concept of a build lifecycle. Maven has conventions, is declarative, and has a lifecycle.

Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (**POM**), Maven can manage a project's build, reporting and documentation from a central piece of information. Using maven we can build and manage any Java based project.

Understanding the problem without Maven

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.

Maven simplifies the above mentioned problems. It does mainly following tasks.

* It makes a project easy to build
* It provides uniform build process (maven project can be shared by all the maven projects)
* It provides project information (log document, cross referenced sources, mailing list, dependency list, unit test reports etc.)
* It is easy to migrate for new features of Maven

Apache Maven helps to manage

* **Builds**
* **Documentation**
* **Reporting**
* **SCMs**
* **Releases**
* **Distribution**

What is Build Tool?

A build tool takes care of everything for building a process. It does following:

* Generates source code (if auto-generated code is used)
* Generates documentation from source code
* **Compiles source code**
* **Packages compiled code into JAR of ZIP file**
* Installs the packaged code in local repository, server repository, or central repository

|  |  |
| --- | --- |
| **Ant** | **Maven** |
| Ant **doesn't has formal conventions**, so we need to provide information of the project structure in 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 **life cycle** in Maven. |
| It is **a tool** box. | It 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. |

**How to install Maven on windows?**

You can download and install maven on windows, linux and MAC OS platforms.

To install maven on windows, you need to perform following steps:

1. Download maven (e.g. apache-maven-3.6.0-bin.zip ) from <http://maven.apache.org/download.cgi> and extract it
2. Add JAVA\_HOME and MAVEN\_HOME (e.g. D:\apache-maven-3.6.0) in environment variable
3. Add %MAVEN\_HOME%\bin in ‘path’ environment variable
4. Verify Maven in command prompt by mvn –v. Below output is displayed

Maven always uses the JDK specified by JAVA\_HOME, no matter how many JDK installations are available on the system. This allows the user the flexibility to change JDKs as required or based on the project.

Hence, it is important to ensure JAVA\_HOME is defined. In the absence of this variable, Maven attempts to detect the presence of Java from PATH. This is typically JRE and not JDK.

C:\Users\jigar.mehta1>mvn -v

Apache Maven 3.6.0 (97c98ec64a1fdfee7767ce5ffb20918da4f719f3; 2018-10-25T00:11:4

7+05:30)

Maven home: D:\apache-maven-3.6.0\bin\..

Java version: 11.0.1, vendor: Oracle Corporation, runtime: C:\Program Files\Java

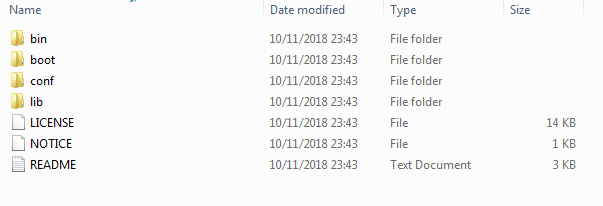
\jdk-11.0.1

Default locale: en\_GB, platform encoding: Cp1252

OS name: "windows 7", version: "6.1", arch: "amd64", family: "windows"

Open folder D:\apache-maven-3.6.0\ in windows explorer

Below contents will be visible



* The bin folder contains the batch files and shell scripts to run Maven on various platforms.
* The boot folder contains the jars required for Maven to start.
* The conf folder contains the default settings.xml file used by Maven.
* The lib folder contains the libraries used by Maven. It also contains an ext folder in which third-party extensions, which can extend or override the default Maven implementation, can be placed.

Creating a simple project with Maven

**Creating Maven project in Eclipse**

File-> New-> Other-> Maven-> Maven Project-> Next (select ‘Create a simple project) -> Next->

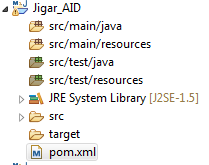
Enter below:

Group ID: Any Name

Artifact ID: Any Name

Select Finish

Below Folder structure is created in Eclipse



Open pom.xml

Below is the base structure of 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>Jigar\_GID</groupId>

<artifactId>Jigar\_AID</artifactId>

<version>0.0.1-SNAPSHOT</version>

</project>

IF we want to add ‘testng’ to our project folder using Maven POM then follow below steps

1. Navigate to **mvnrepository.com**
2. Search for testng
3. Select the required version e.g. 6.14.3
4. Below dependency is available

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>6.14.3</version>

<scope>test</scope>

</dependency>

Add it in POM and revised POM will be

<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>Jigar\_GID</groupId>

<artifactId>Jigar\_AID</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>6.14.3</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**Lets understand concept of Repositories in Maven**

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 🡪 Central repository 🡪 Remote repository

If dependency is not found in these repositories, maven stops processing and throws an error.

Maven Local Repository: By default, maven local repository is %USER\_HOME%/.m2 directory

Maven central repository is located on the web. It has been created by the apache maven community itself.

If you want to change the default maven local repository then go to %MAVEN\_HOME%\conf (e.g. D:\apache-maven-3.6.0\conf) and edit “settings.xml”

Below code will be commented

<!-- localRepository

| The path to the local repository maven will use to store artifacts.

|

| Default: ${user.home}/.m2/repository

<localRepository></localRepository>

-->

In <localRepository> tag provide the new path and unhide this tag

Whenever the Maven project is saved the local repository will be saved in the new path.

To find out the USER\_HOME type following in command prompt

echo %HOMEDRIVE%%HOMEPATH%

Maven Remote Repository: 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. **(Ignore for now)**

Maven Central Repository:

The path of central repository is: <https://repo.maven.apache.org/maven2/> (This will contain all files related to required APIs)

The central repository contains a lot of common libraries that can be viewed by this url http://search.maven.org/#browse.

You can search any repository from Maven official website **mvnrepository.com**

**How to import existing Maven Project in Eclipse?**

Create a folder “D:\Maven Practise\Maven\_Test”. Inside this create below folder structure

D:\Maven Practise\Maven\_Test -> src 🡪 main 🡪 java

resources

test 🡪 java

resources

Copy “pom.xml” having below contents in “Maven\_Test” folder

<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>Jigar\_GID1</groupId>

<artifactId>Jigar\_AID1</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>6.14.3</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

Add below java file inside src/main/java

**public** **class** HelloWorld {

**public** **static** **void** main(String[] args) {

System.***out***.println("Hello World");

}

}

Add below test file in src/test/java

**import** org.testng.Assert;

**import** org.testng.annotations.Test;

**public** **class** AppTest {

@Test

**public** **void** testApp() {

System.***out***.println("Inside AppTest");

Assert.*assertEquals*(0, 0);

}

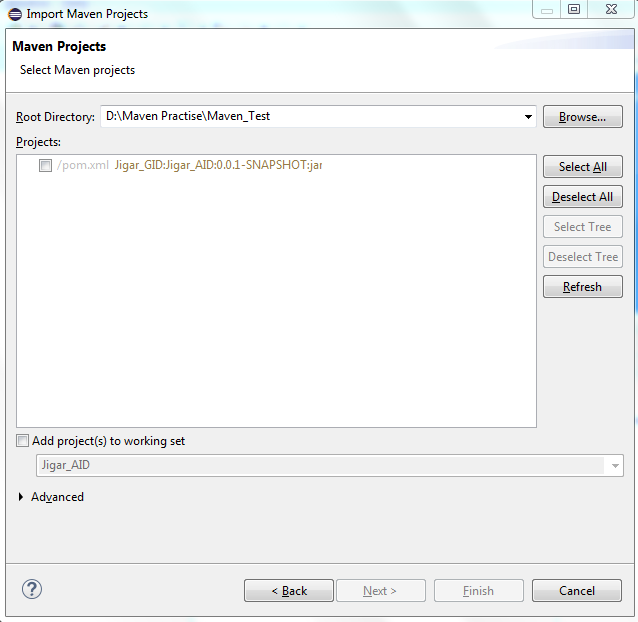
}

In Eclipse navigate to

File -> Import -> Maven -> Existing Maven Projects -> Next

Select “D:\Maven Practise\Maven\_Test” in Root Directory

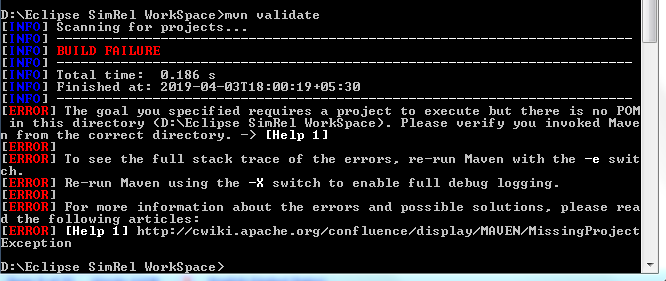
Select pom.xml and click Finish



**How to compile all java files using Maven in command prompt?**

In the command prompt navigate to folder where the pom.xml does not reside e.g. “D:\Eclipse SimRel WorkSpace”

Run command “mvn validate” and see the result. It basically checks whether a valid pom.xml is present or not.

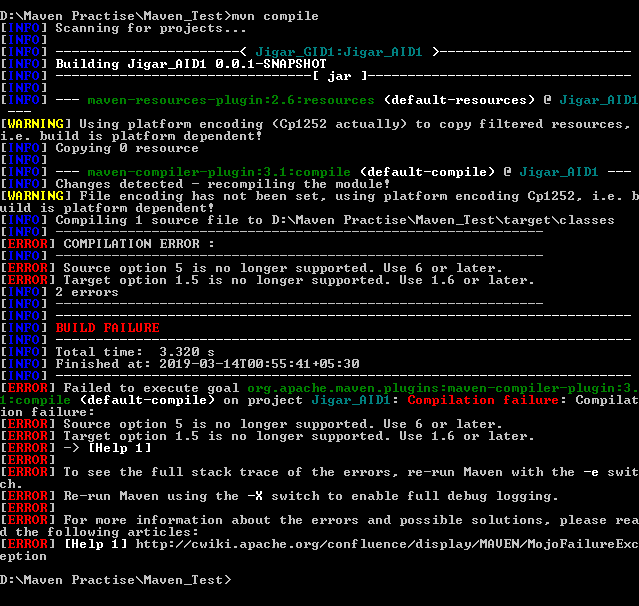


Goto folder where valid pom.xml is present and run the same command and see the success result



mvn compile (Compiles all java files in src/main/java and creates class files in target/classes/)

This will give below error



Add below lines in pom.xml and run the command “mvn compile” again

<properties>

<maven.compiler.source>1.6</maven.compiler.source>

<maven.compiler.target>1.6</maven.compiler.target>

</properties>

New pom.xml would be

<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>Jigar\_GID1</groupId>

<artifactId>Jigar\_AID1</artifactId>

<version>0.0.1-SNAPSHOT</version>

<properties>

<maven.compiler.source>1.6</maven.compiler.source>

<maven.compiler.target>1.6</maven.compiler.target>

</properties>

<dependencies>

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>6.14.3</version>

<scope>test</scope>

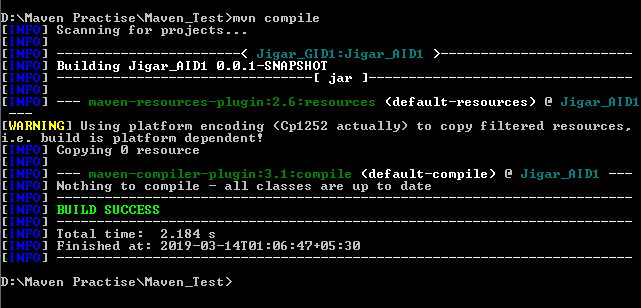
</dependency>

</dependencies>

</project>

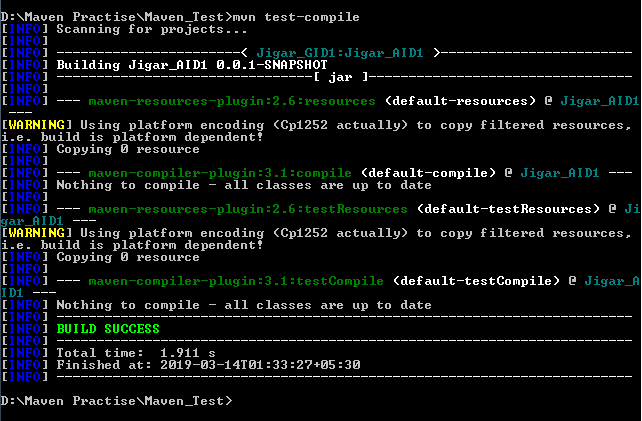
mvn complile (Compiles all java files in src/main/java and creates class files in target/classes/ )

Below is the result



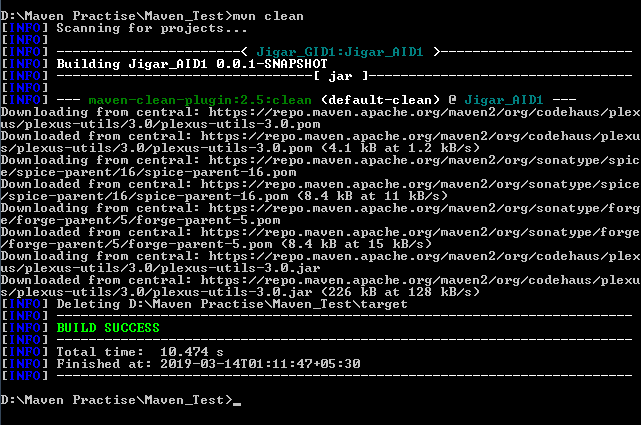
HelloWorld.class is created in [\\target\classes](file:///\\target\classes) folder

mvn test-compile (Compiles all java files in src/main/java and src/test/java and creates class files in target/classes/)

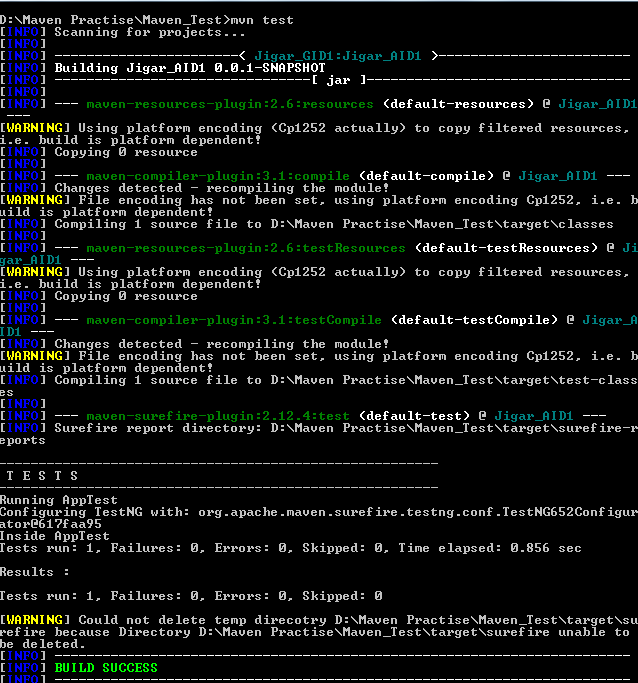


AppTest.class file is created in [\\target\test-classes](file:///\\target\test-classes) folder

mvn clean (deletes all files in target folder and target folder itself)



mvn test (does all activities of compile, test-compile and then run test class files)



**Maven Lifecyle**

Maven is implemented based around the concept of a build lifecycle. The stages of a lifecycle are called phases. In each phase, one or more goals can be executed.

Maven Lifecycle -> Phases -> Goals (LPG…)

Maven has three built-in build lifecycles:

* **default**: The default lifecycle handles project build and deployment
* **clean**: The clean lifecycle cleans up the files and folders produced by Maven. We can also customize this to delete user created folder/files during compilation.
* **site**: The site lifecycle handles the creation of project documentation

Maven infers the lifecycle based on the phase specified.

Below are the important phases for each lifecycle:

* The clean lifecycle: The clean phase removes all the files and folders created by Maven as part of its build.

Attempt to clean a project's working directory of the files that we're generated at build-time. By default, it discovers and deletes the directories configured in **project.build.directory**, **project.build.outputDirectory**, **project.build.testOutputDirectory**, and **project.reporting.outputDirectory**.

* The site lifecycle: The site phase generates the project's documentation, which can be published, as well as a template that can be customized further
* The default lifecycle: The following in **bold** are some of the important phases of the default lifecycle:
  + **validate: This phase validates that all project information is available and correct ‰**
  + process-resources: This phase copies project resources to the destination to package ‰
  + **compile: This phase compiles the source code**
  + **test: This phase compiles unit tests within a suitable framework**
  + **package: This phase packages the compiled code in its distribution format**
  + integration-test: This phase processes the package in the integration test environment ‰
  + **verify: This phase runs checks to verify that the package is valid**
  + **install: This phase installs the package in the local repository deploy**
  + **deploy: This phase installs the final package in the configured repository**

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 **test**, **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.

The phases named with hyphenated-words (pre-\*, post-\*, or process-\*) are not usually directly called from the command line. These phases sequence the build, producing intermediate results that are not useful outside the build. In the case of invoking integration-test, the environment may be left in a hanging state

mvn install

This command executes each default life cycle phase in order (validate, compile, package, etc.), before executing install. You only need to call the last build phase to be executed, in this case, install:

Each phase is made up of plugin goals. A plugin goal is a specific task that builds the project.

Here is a table of phases, plugins, and goals:

|  |  |  |
| --- | --- | --- |
| **Phase (Command Prompt arguments)** | **Plugin that are runs in this Phase**  <plugin-group-id>:<plugin-artifact-id>[:<plugin-version>]:<goal> | **Goal** |
| Clean (mvn clean) | Maven Clean  (maven-clean-plugin:2.5:clean) | plugin clean |
| **Site** | **Maven Site plugin**  **(Ignore for now)** | **Site** |
| process-resources (mvn process-resources) | maven-resources-plugin  (maven-resources-plugin:2.6:resources) | resource |
| Compile (mvn compile) | maven-compiler-plugin  (maven-resources-plugin:2.6:resources)  (maven-compiler-plugin:3.1:compile) | Compile |
| Test (mvn test) | maven-surefire-plugin  (maven-resources-plugin:2.6:resources)  (maven-resources-plugin:2.6:testResources)  (maven-compiler-plugin:3.1:compile)  (maven-compiler-plugin:3.1:testCompile)  maven-surefire-plugin:2.12.4:test | Test |
| Package (mvn package) | Varies based on the packaging; for instance, the Maven JAR plugin  (maven-resources-plugin:2.6:resources)  (maven-resources-plugin:2.6:testResources)  (maven-compiler-plugin:3.1:compile)  (maven-compiler-plugin:3.1:testCompile)  (maven-surefire-plugin:2.12.4:test)  (maven-jar-plugin:2.4:jar) | jar (in the case of a Maven JAR plugin) |
| Install (mvn install) | Maven Install plugin  (maven-resources-plugin:2.6:resources)  (maven-resources-plugin:2.6:testResources)  (maven-compiler-plugin:3.1:compile)  (maven-compiler-plugin:3.1:testCompile)  (maven-surefire-plugin:2.12.4:test)  (maven-jar-plugin:2.4:jar)  (maven-install-plugin) | Install |
| Deploy (mvn deploy) | Maven Deploy plugin  (maven-resources-plugin:2.6:resources)  (maven-resources-plugin:2.6:testResources)  (maven-compiler-plugin:3.1:compile)  (maven-compiler-plugin:3.1:testCompile)  (maven-surefire-plugin:2.12.4:test)  (maven-jar-plugin:2.4:jar)  (maven-install-plugin)  (maven-deploy-plugin:2.7:deploy) | Deploy |

To run an executable jar file, in the command prompt navigate to the folder where the jar file is created and run below command

**java -cp jarname.jar packagename1.packagename2.Classname** having main method

**Understanding pom.xml**

Below is listing of the elements directly under the POM's project element. Notice that modelVersion contains 4.0.0. That is currently the only supported POM version for both Maven 2 & 3, and is always required.

<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>

<!-- The Basics -->

<groupId>...</groupId>

<artifactId>...</artifactId>

<version>...</version>

<packaging>...</packaging>

<dependencies>...</dependencies>

<parent>...</parent>

<dependencyManagement>...</dependencyManagement>

<modules>...</modules>

<properties>...</properties>

<!-- Build Settings -->

<build>...</build>

<reporting>...</reporting>

<!-- More Project Information -->

<name>...</name>

<description>...</description>

<url>...</url>

<inceptionYear>...</inceptionYear>

<licenses>...</licenses>

<organization>...</organization>

<developers>...</developers>

<contributors>...</contributors>

<!-- Environment Settings -->

<issueManagement>...</issueManagement>

<ciManagement>...</ciManagement>

<mailingLists>...</mailingLists>

<scm>...</scm>

<prerequisites>...</prerequisites>

<repositories>...</repositories>

<pluginRepositories>...</pluginRepositories>

<distributionManagement>...</distributionManagement>

<profiles>...</profiles>

</project>

A pom file is an XML file that is based on a specific schema, as specified at the top of the file:

<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/maven-v4\_0\_0.xsd"*>

There is also a modelVersion element that defines the version of this schema:

<modelVersion>4.0.0</modelVersion>

The groupId element is a unique identifier of the organization to which the project belongs.

<groupId>com.in28minutes.maven</groupId>

The artifactId element is the name of the project. For our sample project, it is simple-project:

<artifactId>basic-maven-example</artifactId>

The version element is the specific instance of the project, corresponding to the source code at a particular instance of time. In our case, it is 1.0-SNAPSHOT, which is a default version during development: <version>1.0.1-SNAPSHOT</version>

The combination of **groupId**, **artifactId**, and **version** uniquely identifies the project. In this sense, they are the coordinates of the project.

The packaging element indicates the artifact type of the project produces. This is typically a jar, war, zip, or in some cases, a pom: <packaging>jar</packaging>

The dependencies element section of the pom file defines all the dependent projects of this project. This would typically be third-party libraries required to build, test, and run the project:

<dependencies> </dependencies>

The parent section is used to indicate a relationship, specifically a parent-child relationship. If the project is part of a multi-module project or inherits project information from another project, then the details are specified in this section: <parent> </parent>

Maven properties are placeholders. Their values are accessible anywhere in the pom file by using ${key}, where key is the property name: <properties> </properties>

A project with modules is known as a multi-module or aggregator project. Modules are projects that this pom file lists and are executed as a group: <modules> </modules>

**Specifying source encoding for platform independent builds i.e. How to remove build warning shown after compilation using maven**

‘[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources’

Add the following code and compile using ‘mvn package’ in cmd

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

**Various setting properties of Maven**

1. Open the settings.xml file in the conf folder of your Maven installation

Maven has a global settings file called settings.xml in the conf folder of the Maven installation. The settings file contains configurations that are not specific to a project, but are global in nature.

Like the pom file, the settings file is also an XML file based on an XML schema. It starts as follows:

<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"*>

**Typical setting configurations:**

1. **localRepository** element:

<!-- localRepository

| The path to the local repository maven will use to store artifacts.

|

| Default: ${user.home}/.m2/repository

<localRepository>C:/software/maven</localRepository>

1. **offline** element

The following code represents the offline element in the settings file: false. This setting indicates whether Maven should operate in offline mode; that is, it should not download updates or dependencies if they are not available.

<!-- offline

| Determines whether maven should attempt to connect to the network when executing a build.

| This will have an effect on artifact downloads, artifact deployment, and others.

|

| Default: false

<offline>false</offline>

-->

1. The proxies element

This allows us to specify a proxy server to connect to the Internet. This is relevant in enterprises where direct access to the Internet might be blocked due to security or other reasons.

1. mirrors elements

Instead of downloading dependencies from Maven Central, you can configure Maven to download them from a mirror of the central repository. This is extremely useful in an organization where the repository can be mirrored in a repository manager within an organization and all users can download dependencies from this mirror

1. Repositories are remote collections of projects that Maven uses to populate the required dependencies to a local repository. There are two types of repositories—releases and snapshots—and Maven allows specific configurations for each, as illustrated in the following code

**Command line options**

mvn –h (A number of options that Maven supports are displayed)

-e (When there is an error while running Maven, this flag will result in Maven displaying a detailed stack trace of the error)

-q or –quiet (When the quiet option is enabled, only errors are displayed. The other outputs are not printed. This permits speeding up builds where verbose outputs are usually displayed)

-v or –version (Check if Maven is installed and working)

-o or –offline (Maven does not attempt to download any dependency or plugin from the Internet. This option will work correctly, provided Maven has all the information required for the project to be built and run.)

**Maven Profiles**

Maven provides three type of profiles:

1. Per Project profile as defined in the pom file of the project
2. Per User profile as defined in the user settings file (in the .m2 subfolder of the user's HOME folder)
3. A Global profile as defined in the global settings file (in the conf folder of MAVEN\_HOME)

By creating different profiles for different variations of the project build, you can use the same pom file to create differing builds.

There are two ways to create a profile: in the project's pom file or in the settings file. It is important to note that, if a profile is active from the settings file, its values will override any profiles with equivalent IDs in the pom file.

The profile in pom.xml can have the following elements:

<profile>

<id>test</id>

<activation>...</activation>

<build>...</build>

<modules>...</modules>

<repositories>...</repositories>

<pluginRepositories>...</pluginRepositories>

<dependencies>...</dependencies>

<reporting>...</reporting>

<dependencyManagement>...</dependencyManagement>

<distributionManagement>...</distributionManagement>

</profile>

The profile in settings.xml can only have the following elements:

<profile>

<id>test</id>

<activation>...</activation>

<repositories>...</repositories>

<pluginRepositories>...</pluginRepositories>

<properties>…</properties>

</profile>

To deactivate a profile, set the following value in the activeByDefault element to ‘false’

<activeByDefault>false</activeByDefault>

Run below command to list the active profiles

mvn help:active-profiles

Create a maven project and add below 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>EG1</groupId>

<artifactId>EG12</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>6.14.3</version>

<scope>test</scope>

</dependency>

</dependencies>

<profiles>

<profile>

<id>test1</id>

<activation>

<activeByDefault>true</activeByDefault>

</activation>

</profile>

<profile>

<id>test2</id>

<activation>

<activeByDefault>true</activeByDefault>

</activation>

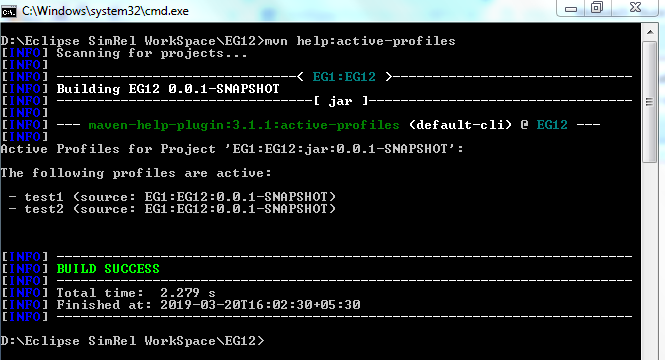
</profile>

</profiles>

</project>

mvn help:active-profiles

Output:



**Properties in Maven**

Maven allows us to define as well as use properties. Properties allow us to avoid hardcoding values in multiple places such as versions of dependencies.

Syntax

<name>value</name>

Define a property testng.version and assign a value 3.8.1

<properties>

<testng.version>3.8</testng.version>

</properties>

Usage of Property:

testng.version is defined in <properties> tag and used in <dependency> tag

<dependencies>

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

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

<scope>test</scope>

</dependency>

</dependencies>

<properties>

<maven.compiler.target>1.8</maven.compiler.target>

<maven.compiler.source>1.8</maven.compiler.source>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<testng.version>6.14.3</testng.version>

<name>value</name>

</properties>

There are different types of properties. They are as follows:

* Environment variables: Prefixing a variable with env. will return the value of the Windows environment variable. For example, ${env.PATH} will return the value of the PATH variable.

Run below pom.xml in eclipse and it will print value of ‘MAVEN\_HOME’ in console which is an environment variable.

<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>EG1</groupId>

<artifactId>EG12</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

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

<scope>test</scope>

</dependency>

</dependencies>

<properties>

<maven.compiler.target>1.8</maven.compiler.target>

<maven.compiler.source>1.8</maven.compiler.source>

<project.build.sourceEncoding></project.build.sourceEncoding>

<testng.version>6.14.3</testng.version>

<name>value</name> //Normal properties

</properties>

<build>

<plugins>

<plugin>

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

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

<version>1.8</version>

<executions>

<execution>

<phase>validate</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<target>

<echo>Displaying value of pom.xml element</echo>

<echo>[project.artifactId] ${project.artifactId}</echo>

<echo>Maven Home = ${env.MAVEN\_HOME}</echo>

<echo>${settings.offline}</echo>

<echo>Java property java\_home = ${java.home}</echo>

</target>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

<profiles>

<profile>

<id>student1</id>

<activation>

<activeByDefault>true</activeByDefault>

</activation>

</profile>

<profile>

<id>student2</id>

<activation>

<activeByDefault>true</activeByDefault>

</activation>

</profile>

</profiles>

</project>

Output:

[WARNING]

[WARNING] Some problems were encountered while building the effective settings

[WARNING] Unrecognised tag: 'snapshotPolicy' (position: START\_TAG seen ...</layout>\n <snapshotPolicy>... @209:27) @ D:\apache-maven-3.6.0\conf\settings.xml, line 209, column 27

[WARNING]

[INFO] Scanning for projects...

[INFO]

[INFO] ------------------------------< EG1:EG12 >------------------------------

[INFO] Building EG12 0.0.1-SNAPSHOT

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

[INFO]

[INFO] --- maven-antrun-plugin:1.8:run (default) @ EG12 ---

[INFO] Executing tasks

main:

[echo] Displaying value of pom.xml element

[echo] [project.artifactId] EG12

[echo] Maven Home = D:\apache-maven-3.6.0

[echo] ${settings.offline}

[echo] Java property java\_home = C:\Program Files\Java\jdk-11.0.1

[INFO] Executed tasks

[INFO]

[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ EG12 ---

[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is platform dependent!

[INFO] Copying 1 resource

[INFO]

[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ EG12 ---

[INFO] Nothing to compile - all classes are up to date

[INFO]

[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ EG12 ---

[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is platform dependent!

[INFO] Copying 0 resource

[INFO]

[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ EG12 ---

[INFO] Nothing to compile - all classes are up to date

[INFO]

[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ EG12 ---

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

[INFO] BUILD SUCCESS

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

[INFO] Total time: 3.159 s

[INFO] Finished at: 2019-03-21T02:50:51+05:30

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

* pom variables: Prefixing a variable with **project**. will return the value of that element in the pom file. For example, ${project.version} will return the value in the tag of the pom file.

In the above pom.xml example ${project. artifactId} prints the value of <artifactId> in which the pom.xml is stored i.e. EG12 in this case

* The settings variable: Prefixing a variable with **settings**. will return the value of that element in the settings file. For example, ${settings.offline} will return the value in the settings file.
* Java properties: Any property available through the System.getProperties() method in Java is available. For example, ${**java**.home}.

Below is the program to fetch all the System Properties name and their values:

**package** pack1;

**import** java.util.Iterator;

**import** java.util.Properties;

**import** java.util.Set;

**public** **class** LearningAnt {

**public** **static** **void** main(String[] args) {

Properties prop = System.*getProperties*();

Set<Object> keys = prop.keySet();

Iterator<Object> iter = keys.iterator();

**while** (iter.hasNext()) {

String key = iter.next().toString();

System.***out***.println(key + " : " + System.*getProperty*(key));

}

}

}

Output:

sun.desktop : windows

awt.toolkit : sun.awt.windows.WToolkit

java.specification.version : 11

sun.cpu.isalist : amd64

sun.jnu.encoding : Cp1252

java.class.path : D:\Eclipse SimRel WorkSpace\HW\bin

java.vm.vendor : Oracle Corporation

sun.arch.data.model : 64

user.variant :

java.vendor.url : http://java.oracle.com/

user.timezone :

os.name : Windows 7

java.vm.specification.version : 11

sun.java.launcher : SUN\_STANDARD

user.country : GB

sun.boot.library.path : C:\Program Files\Java\jdk-11.0.1\bin

sun.java.command : pack1.LearningAnt

jdk.debug : release

sun.cpu.endian : little

user.home : C:\Users\jigar.mehta1

user.language : en

java.specification.vendor : Oracle Corporation

java.version.date : 2018-10-16

java.home : C:\Program Files\Java\jdk-11.0.1

file.separator : \

java.vm.compressedOopsMode : 32-bit

line.separator :

java.specification.name : Java Platform API Specification

java.vm.specification.vendor : Oracle Corporation

java.awt.graphicsenv : sun.awt.Win32GraphicsEnvironment

user.script :

sun.management.compiler : HotSpot 64-Bit Tiered Compilers

java.runtime.version : 11.0.1+13-LTS

user.name : Jigar.Mehta1

path.separator : ;

os.version : 6.1

java.runtime.name : Java(TM) SE Runtime Environment

file.encoding : Cp1252

java.vm.name : Java HotSpot(TM) 64-Bit Server VM

java.vendor.version : 18.9

java.vendor.url.bug : http://bugreport.java.com/bugreport/

java.io.tmpdir : C:\Users\JIGAR~1.MEH\AppData\Local\Temp\

java.version : 11.0.1

user.dir : D:\Eclipse SimRel WorkSpace\HW

os.arch : amd64

java.vm.specification.name : Java Virtual Machine Specification

java.awt.printerjob : sun.awt.windows.WPrinterJob

sun.os.patch.level : Service Pack 1

java.library.path : C:\Program Files\Java\jdk-11.0.1\bin;C:\Windows\Sun\Java\bin;C:\Windows\system32;C:\Windows;C:/Program Files/Java/jdk-11.0.1/bin/server;C:/Program Files/Java/jdk-11.0.1/bin;C:\Program Files (x86)\Intel\iCLS Client\;C:\Program Files\Intel\iCLS Client\;C:\Program Files (x86)\RSA SecurID Token Common;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Program Files (x86)\CrSSL\bin;C:\Program Files\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files\Intel\Intel(R) Management Engine Components\IPT;C:\Program Files (x86)\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files (x86)\Intel\Intel(R) Management Engine Components\IPT;C:\Program Files (x86)\Intel\OpenCL SDK\2.0\bin\x86;C:\Program Files (x86)\Intel\OpenCL SDK\2.0\bin\x64;C:\Program Files\Java\jdk-11.0.1\bin;D:\apache-maven-3.6.0\bin;D:\SeleniumDrivers;D:\apache-ant-1.10.5\bin;D:\Eclipse SimRel Latest\eclipse;;.

java.vendor : Oracle Corporation

java.vm.info : mixed mode

java.vm.version : 11.0.1+13-LTS

sun.io.unicode.encoding : UnicodeLittle

java.class.version : 55.0

* Normal properties: Values that are specified in the <properties> tag

**Maven Plugins**

Maven is - at its heart - a plugin execution framework; all work is done by plugins.

There are the build and the reporting plugins:

* Build plugins will be executed during the build and they should be configured in the <build/> element from the POM.
  + clean
  + compiler
  + deploy
  + failsafe
  + install
  + resources
  + site
  + surefire
  + verifier
* Reporting plugins will be executed during the site generation and they should be configured in the <reporting/> element from the POM.
  + changelog
  + changes
  + checkstyle
  + doap
  + dock
  + javadoc
  + jdeps
  + jxr
  + linkcheck
  + pmd
  + project-info-reports
  + surefire-report

To see the most up-to-date list browse the Maven repository, specifically the [org/apache/maven/plugins](https://repo.maven.apache.org/maven2/org/apache/maven/plugins/) subfolder.

Clean plugin:

mvn clean

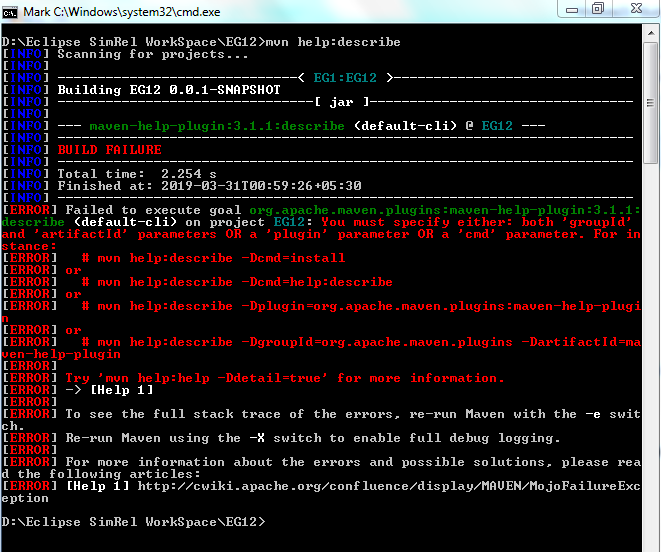
A valid lifecycle phase or a goal must be specified in the format

**<plugin-prefix>:<goal> or**

**<plugin-group-id>:<plugin-artifact-id>[:<plugin-version>]:<goal>**

Run below command

**mvn help:describe**

****

You must specify either: both 'groupId' and 'artifactId' parameters OR a 'plugin' parameter OR a 'cmd' parameter. For instance:

mvn help:describe -Dcmd=install

or

mvn help:describe -Dcmd=help:describe

or

mvn help:describe -Dplugin=org.apache.maven.plugins:maven-help-plugin

or

mvn help:describe -DgroupId=org.apache.maven.plugins -DartifactId=maven-help-plugin

or

mvn help:help -Ddetail=true

e.g.

Maven **help** plugin

Run below in command prompt

D:\Eclipse SimRel WorkSpace\EG12>**mvn help:help**

[INFO] Scanning for projects...

[INFO]

[INFO] ------------------------------< EG1:EG12 >------------------------------

[INFO] Building EG12 0.0.1-SNAPSHOT

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

[INFO]

[INFO] --- maven-help-plugin:3.1.1:help (default-cli) @ EG12 ---

[INFO] Apache Maven Help Plugin 3.1.1

The Maven Help plugin provides goals aimed at helping to make sense out of the

build environment. It includes the ability to view the effective POM and

settings files, after inheritance and active profiles have been applied, as

well as a describe a particular plugin goal to give usage information.

This plugin has 8 goals:

**help:active-profiles**

Displays a list of the profiles which are currently active for this build.

**help:all-profiles**

Displays a list of available profiles under the current project. Note: it will list all profiles for a project. If a profile comes up with a status inactive then there might be a need to set profile activation switches/property.

**help:describe**

Displays a list of the attributes for a Maven Plugin and/or goals (aka Mojo - Maven plain Old Java Object).

**help:effective-pom**

Displays the effective POM as an XML for this build, with the active profiles factored in, or a specified artifact.

**help:effective-settings**

Displays the calculated settings as XML for this project, given any profile enhancement and the inheritance of the global settings into the user-level settings.

**help:evaluate**

Evaluates Maven expressions given by the user in an interactive mode.

**help:help**

Display help information on maven-help-plugin. Call mvn help:help -Ddetail=true -Dgoal=<goal-name> to display parameter details.

**help:system**

Displays a list of the platform details like system properties and environment variables.

Available lifecycle phases are:

* validate
* initialize
* generate-sources
* process-sources
* generate-resources
* process-resources
* compile
* process-classes
* generate-test-sources
* process-test-sources
* generate-test-resources
* process-test-resources
* test-compile
* process-test-classes
* test
* prepare-package
* package
* pre-integration-test
* integration-test
* post-integration-test
* verify
* install
* deploy
* pre-clean
* clean
* post-clean
* pre-site
* site
* post-site
* site-deploy

**Inheritance in POM.xml**

If the <packaging></packaging> element in pom.xml is set to ‘pom’ then it is a parent pom

EG13\_AID is parent 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>EG13</groupId>

<artifactId>EG13\_AID</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>pom</packaging>

<properties>

<selenium.version>3.13.0</selenium.version>

<jigar.version>6.14.3</jigar.version>

<jigar.jar>jar</jigar.jar>

</properties>

<dependencies>

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>

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

</dependency>

<dependency>

<groupId>org.apache.poi</groupId>

<artifactId>poi</artifactId>

<version>4.0.1</version>

</dependency>

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

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

<type>${jigar.jar}</type>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

<version>1.8</version>

<executions>

<execution>

<phase>validate</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<target>

<echo>Displaying value of EG13\_AID pom.xml element</echo>

<echo>[project.artifactId] ${project.artifactId}</echo>

<echo>[project.version] ${project.version}</echo>

<echo> Selenium version = ${selenium.version}</echo>

<echo>Java Home = ${env.JAVA\_HOME}</echo>

<echo>Maven Home = ${env.MAVEN\_HOME}</echo>

<echo>Ant Home = ${env.ANT\_HOME}</echo>

<echo>Base directory = ${project.basedir}</echo>

</target>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

</project>

Run as Maven test in Eclipse. Below is the result:

[INFO] Scanning for projects...

[INFO]

[INFO] ---------------------------< EG13:EG13\_AID >----------------------------

[INFO] Building EG13\_AID 0.0.1-SNAPSHOT

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

[INFO]

[INFO] --- maven-antrun-plugin:1.8:run (default) @ EG13\_AID ---

[INFO] Executing tasks

main:

[echo] Displaying value of EG13\_AID pom.xml element

[echo] [project.artifactId] EG13\_AID

[echo] [project.version] 0.0.1-SNAPSHOT

[echo] Selenium version = 3.13.0

[echo] Java Home = C:\Program Files\Java\jdk-11.0.1

[echo] Maven Home = D:\apache-maven-3.6.0

[echo] Ant Home = D:\apache-ant-1.10.5

[echo] Base directory = D:\Eclipse SimRel WorkSpace\EG13\_AID

[INFO] Executed tasks

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

[INFO] BUILD SUCCESS

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

[INFO] Total time: 2.689 s

[INFO] Finished at: 2019-04-14T03:17:01+05:30

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

EG14\_AID is child pom.xml

1. <relativePath> tag in child.xml should point to parent pom.xml path
2. <packaging> should be jar
3. All elements under <build> are inherited by EG14\_AID from EG13\_AID

<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>EG14</groupId>

<artifactId>EG14\_AID</artifactId>

<packaging>jar</packaging>

<parent>

<groupId>EG13</groupId>

<artifactId>EG13\_AID</artifactId>

<version>0.0.1-SNAPSHOT</version>

<relativePath>../EG13\_AID/pom.xml</relativePath>

</parent>

<dependencies>

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>6.14.3</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

Output of Maven test in Eclipse

[INFO] Scanning for projects...

[INFO]

[INFO] ---------------------------< EG14:EG14\_AID >----------------------------

[INFO] Building EG14\_AID 0.0.1-SNAPSHOT

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

[INFO]

[INFO] --- maven-antrun-plugin:1.8:run (default) @ EG14\_AID ---

[INFO] Executing tasks

main:

[echo] Displaying value of EG13\_AID pom.xml element

[echo] [project.artifactId] EG14\_AID

[echo] [project.version] 0.0.1-SNAPSHOT

[echo] Selenium version = 3.13.0

[echo] Java Home = C:\Program Files\Java\jdk-11.0.1

[echo] Maven Home = D:\apache-maven-3.6.0

[echo] Ant Home = D:\apache-ant-1.10.5

[echo] Base directory = D:\Eclipse SimRel WorkSpace\EG14\_AID

[INFO] Executed tasks

[INFO]

[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ EG14\_AID ---

[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is platform dependent!

[INFO] Copying 0 resource

[INFO]

[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ EG14\_AID ---

[INFO] Nothing to compile - all classes are up to date

[INFO]

[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ EG14\_AID ---

[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is platform dependent!

[INFO] Copying 0 resource

[INFO]

[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ EG14\_AID ---

[INFO] Nothing to compile - all classes are up to date

[INFO]

[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ EG14\_AID ---

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

[INFO] BUILD SUCCESS

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

[INFO] Total time: 3.429 s

[INFO] Finished at: 2019-04-14T03:20:39+05:30

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

Example of <element> overridden in Child pom.xml

<build> tag is overridden in below pm.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>EG14</groupId>

<artifactId>EG14\_AID</artifactId>

<packaging>jar</packaging>

<parent>

<groupId>EG13</groupId>

<artifactId>EG13\_AID</artifactId>

<version>0.0.1-SNAPSHOT</version>

<relativePath>../EG13\_AID/pom.xml</relativePath>

</parent>

<dependencies>

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>6.14.3</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<version>1.8</version>

<executions>

<execution>

<phase>validate</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<target>

<echo>Displaying value of EG14\_AID pom.xml element</echo>

<echo>[project.artifactId] EG13\_AID</echo>

<echo>[project.version] 0.0.1-SNAPSHOT</echo>

<echo>Selenium version = 3.13.0</echo>

<echo>Java Home = C:\Program Files\Java\jdk-11.0.1</echo>

<echo>Maven Home = D:\apache-maven-3.6.0</echo>

<echo>Ant Home = D:\apache-ant-1.10.5</echo>

<echo>Base directory = D:\Eclipse SimRel WorkSpace\EG13\_AID</echo>

</target>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

</project>

Output of Maven test in Eclipse

[INFO] Scanning for projects...

[INFO]

[INFO] ---------------------------< EG14:EG14\_AID >----------------------------

[INFO] Building EG14\_AID 0.0.1-SNAPSHOT

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

[INFO]

[INFO] --- maven-antrun-plugin:1.8:run (default) @ EG14\_AID ---

[INFO] Executing tasks

main:

[echo] Displaying value of EG14\_AID pom.xml element

[echo] [project.artifactId] EG13\_AID

[echo] [project.version] 0.0.1-SNAPSHOT

[echo] Selenium version = 3.13.0

[echo] Java Home = C:\Program Files\Java\jdk-11.0.1

[echo] Maven Home = D:\apache-maven-3.6.0

[echo] Ant Home = D:\apache-ant-1.10.5

[echo] Base directory = D:\Eclipse SimRel WorkSpace\EG13\_AID

[INFO] Executed tasks

[INFO]

[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ EG14\_AID ---

[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is platform dependent!

[INFO] Copying 0 resource

[INFO]

[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ EG14\_AID ---

[INFO] Nothing to compile - all classes are up to date

[INFO]

[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ EG14\_AID ---

[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is platform dependent!

[INFO] Copying 0 resource

[INFO]

[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ EG14\_AID ---

[INFO] Nothing to compile - all classes are up to date

[INFO]

[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ EG14\_AID ---

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

[INFO] BUILD SUCCESS

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

[INFO] Total time: 3.514 s

[INFO] Finished at: 2019-04-14T03:22:45+05:30

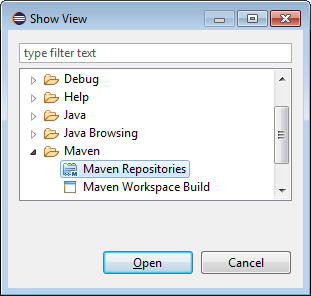
[INFO] ------------------------------------------------------------------------

Note: below elements are not inherited by child pom.xml

* artifactid
* name
* prerequisites

**How do I add a Maven dependency in Eclipse?**

1. On the top menu bar, open Window -> Show View -> Other
2. In the Show View window, open Maven -> Maven Repositories

[](https://i.stack.imgur.com/9CMXO.png)

1. In the window that appears, right-click on Global Repositories and select Go Into.
2. Right-click on "central (<http://repo.maven.apache.org/maven2>)" and select "Rebuild Index"
   1. Note that it will take a while to complete the download
3. Once indexing is complete, Right-click on the project -> Maven -> Add Dependency and start typing the name of the project you want to import (such as "hibernate").
   1. The search results will auto-fill in the "Search Results" box.

**Aggregation (or Multi-Module)**

A project with modules is known as a multimodule or aggregator project. Modules are projects that this POM lists, and are executed as a group. A pom packaged project may aggregate the build of a set of projects by listing them as modules, which are relative paths to the directories or the POM files of those projects.

Note: Ensure that below artificats are created in same workspace

Project Aggregation is similar to Project Inheritance. But instead of specifying the parent POM from the module, it specifies the modules from the parent POM. By doing so, the parent project now knows its modules, and if a Maven command is invoked against the parent project, that Maven command will then be executed to the parent's modules as well. To do Project Aggregation, you must do the following:

* Change the parent POMs packaging to the value "pom".
* Specify in the parent POM the directories of its modules (children POMs).

Below is pom.xml for Group\_Modules1 maven project

<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>Group\_Modules</groupId>

<artifactId>Group\_Modules1</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>pom</packaging>

<modules>

<module>../Group\_Modules2</module>

<module>../Group\_Modules3</module>

</modules>

</project>

Below is pom.xml for Group\_Modules2 maven project

<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>Group\_Modules</groupId>

<artifactId>Group\_Modules2</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>org.apache.poi</groupId>

<artifactId>poi</artifactId>

<version>4.0.1</version>

</dependency>

</dependencies>

</project>

Below is pom.xml for Group\_Modules3 maven project

<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>Group\_Modules</groupId>

<artifactId>Group\_Modules3</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>6.14.2</version>

</dependency>

</dependencies>

</project>