



Maven for beginners - Handbook
(Build Automation tool)



Agenda

Topic	Subtopic
Introduction	What is a Build Tool?
	Build Tools available in the market
	History of Maven
Getting Started	Installation
	Setting up environment variables
	Verify Maven installation
Important terminologies	pom.xml
	local repository
	remote repository
	central repository
Configuring proxy in Maven	settings.xml
Maven Features	Archetype generation
	Maven build lifecycle
	Dependency management
	Eclipse settings for Dependency management
Maven with Eclipse	Maven with Eclipse

What is a Build Tool?

A build tool is a tool that automates everything related to building the software project. Building a software project typically includes one or more of these activities:

- Generating source code (if auto-generated code is used in the project).
- Generating documentation from the source code.
- Compiling source code.
- Packaging compiled code into JAR files or ZIP files.
- Installing the packaged code on a server, in a repository or somewhere else.

Any given software project may have more activities than these needed to build the finished software. Such activities can normally be plugged into a build tool, so these activities can be automated too.

The advantage of automating the build process is that you minimize the risk of humans making errors while building the software manually. Additionally, an automated build tool is typically faster than a human performing the same steps manually.

Build Tools available in market

Some of the build tools that are popular in the market are:

- Scala oriented Build Tool
- CMake
- Terraform
- Bower
- Gradle
- Apache Ant
- Apache Maven
- Apache Buildr
- NAnt

History of Maven

Maven was created by *Jason van Zyl* from Apache Software foundation in 2002. Maven v1.0 was released in 2004. Maven 2 was declared in October 2005 and Maven 3.0 was released in October 2010 being mostly backwards compatible with Maven 2.

Let's Get Started

Installation

You can download Maven from <https://maven.apache.org/download.cgi>

Binary zip archive	apache-maven-3.6.3-bin.zip
--------------------	--

Download the zip file and extract it in a folder

Setting environment variables

Prerequisites

Compatible JDK should be installed and Path should have been set

For Maven 3 we require JDK 7 and above

Set M2_HOME variable that points to Maven installation folder

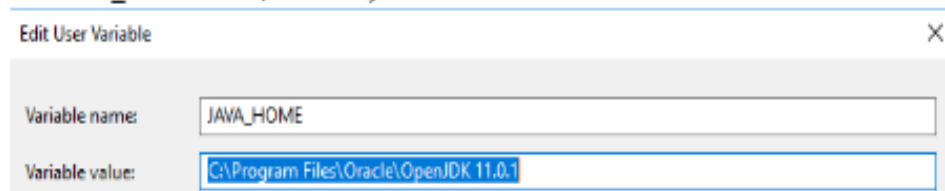


Edit User Variable

Variable name: M2_HOME

Variable value: D:\apache-maven-3.6.1

Set JAVA_HOME that points to java installation folder



Edit User Variable

Variable name: JAVA_HOME

Variable value: C:\Program Files\Oracle\OpenJDK 11.0.1

Path variable should point to bin folders of both JAVA_HOME and M2_HOME



C:\Program Files\Oracle\OpenJDK 11.0.1\bin
D:\apache-maven-3.6.1\bin

Verify Maven installation

Open a command prompt and enter mvn -version

```
C:\Users\anname>mvn -version
Apache Maven 3.6.1 (d66c9c8b3152b2e69ee9bac180bb8fcc8e6af555; 2019-04-05T00:30:29+05:30)
Maven home: D:\apache-maven-3.6.1\bin\..
Java version: 11.0.1, vendor: Oracle Corporation, runtime: C:\Program Files\Oracle\OpenJDK 11.0.1
Default locale: en_IN, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

Important terminologies

pom.xml

pom.xml file is the configuration file for Maven. Every Maven project will have this file. The mvn command does the following

- Reads pom.xml file
- Download dependencies defined in pom.xml file into local repository from central or remote repository

- Execute life-cycles, phases, goals, plugins etc that are defined in the build path

A sample pom.xml file

```
<?xml version="1.0" encoding="UTF-8"?>

<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>com.wipro</groupId>
  <artifactId>FirstMaven</artifactId>
  <version>1.0-SNAPSHOT</version>

  <name>FirstMaven</name>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Elements used for creating pom.xml file

project - It is the root element of the pom.xml file.

modelVersion - modelversion means what version of the POM model you are using. Use version 4.0.0 for maven 2 and maven 3.

groupId - groupId refers to the unique project category id in the organization

artifactId - artifactId refers to the unique project name under the organization project category

version - version element contains the version number of the project

name - this element is used to give name to our maven project.

Other Elements of pom.xml file

dependencies - dependencies element is used to defines a list of dependency of project.

dependency - dependency defines a dependency and used inside dependencies tag. Each dependency is described by its groupId, artifactId and version.

scope - this element used to define scope for this maven project that can be compile, runtime, test, provided system etc

Maven repositories

A Maven repository can be defined as a directory where all the projects APIs such as JARs, library JARs, plugins, etc. are placed that can be readily used by Maven when required.

There are three types of Maven repositories as specified below.

- Local Maven Repository
- Central Maven Repository
- Remote Maven Repository

Local Maven repository

A local Maven repository is a folder that is present in your local machine. The purpose of local repository is to keep all your project related dependencies such as library JARs, plugin JARs, etc. in one place. These dependencies are automatically downloaded into your local repository when you do a Maven build. By default the local repository is found in .m2 folder which is found in the user's home directory.

Eg) C:\Users\aname\.m2

Central Maven repository

Central Maven repository is referred to the repository which is provided by Maven community. It is a giant repository that contains a large number of library JARs which are used very commonly. It is maintained by the Maven community. No configuration is required for this repository in the pom.xml file.

Whenever a particular jar file specified in the pom.xml is not found in the local repository, it will be searched in the central repository.

Remote Maven repository

Sometime we need to set up a Maven repository inside a company or a project development team to host our own libraries. The company maintained repository is outside developer's machine and is called Maven remote repository.

The following pom.xml declares dependencies and also declared remote repository URL

```
<project ...>
  <dependencies>
    <dependency>
      <groupId>com.companyname.common-lib</groupId>
      <artifactId>common-lib</artifactId>
      <version>1.0.0</version>
    </dependency>
  </dependencies>
  <repositories>
    <repository>
      <id>companyname.lib1</id>
      <url>http://download.companyname.org/maven2/lib1</url>
    </repository>
    <repository>
      <id>org.springframework.spring-core</id>
      <url>https://mvnrepository.com/artifact/org.springframework/spring-
core</url>
    </repository>
  </repositories>
</project>
```

settings.xml file

The settings.xml file present in conf folder of the Maven installation directory.

Make sure the settings.xml file in the .m2 folder and the settings.xml file in the maven installation/conf folder are the same.

Maven Features

Archetype generation

In short, Archetype is a Maven project template toolkit. An archetype is defined as an original pattern or model from which all other things of the same kind are made. The name fits as we are trying to provide a system that provides a consistent means of generating Maven projects. Archetype will help authors create Maven project templates for users, and provides users with the means to generate parameterized versions of those project templates.

Example1:

Create a simple java project using mvn archetype plugin

Open command prompt and issue the below command in the directory in which you wanted to create the java project.

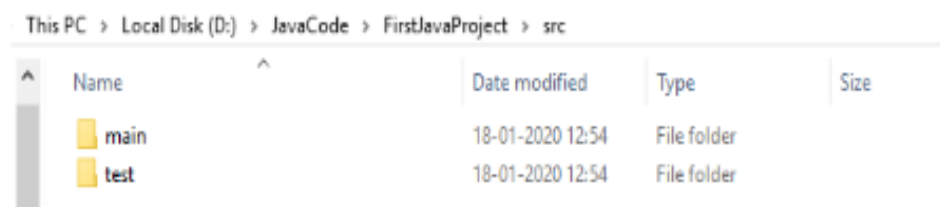
```
mvn archetype:generate -DgroupId=com.wipro -DartifactId=FirstJavaProject -DarchetypeVersion=1.0 -DarchetypeGroupId=org.apache.maven.archetypes -DarchetypeArtifactId=maven-archetype-quickstart
```

```
D:\JavaCode>mvn archetype:generate -DgroupId=com.wipro -DartifactId=FirstJavaProject -DarchetypeVersion=1.0 -DarchetypeGroupId=org.apache.maven.archetypes -DarchetypeArtifactId=maven-archetype-quickstart
[INFO] Scanning for projects...
[INFO] -----< org.apache.maven:standalone-pom >-----
[INFO] Building Maven Stub Project (No POM) 1
[INFO] -----[ pom ]-----
```


Once the build is success, you can open Windows explorer and check the FirstJavaProject folder.

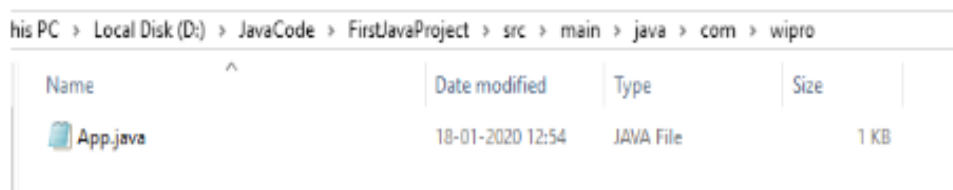


The src folder will contain main and test.

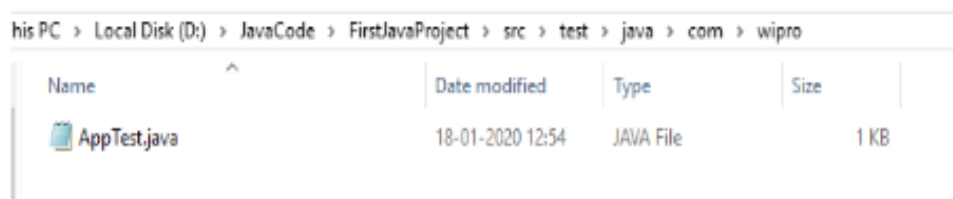


The java code will be kept in the main folder and the corresponding test programs will be kept in the test folder.

By default a Java file called App.java is created in main



The corresponding test program will be created in the test folder



Before proceeding further we need to have some understanding about Maven build life cycle.

Maven build lifecycle

A Build Lifecycle is a well-defined sequence of phases, which define the order in which the goals are to be executed. Here phase represents a stage in life cycle. As an example, a typical Maven Build Lifecycle consists of the following sequence of phases.

The important Maven build life cycle phases are:

validate	checks project correctness and availability of necessary information
compile	compiles the project's source code
test	using suitable test framework, it tests the project
package	takes care of packaging and distribution, such as a JAR
install	installs the package into the local repository
deploy	copies the package to the remote repository for sharing with other developers and projects, generally in integration or release environment

Maven Plugins:

Maven is actually a plugin execution framework where every task is actually done by plugins. Maven Plugins are generally used to –

- create jar file
- create war file
- compile code files
- unit testing of code
- create project documentation
- create project reports

A plugin generally provides a set of goals, which can be executed using the following syntax –

`mvn [plugin-name]:[goal-name]`

For example, a Java project can be compiled with the maven-compiler-plugin's compile-goal by running the following command.

```
mvn compiler:compile
```

Here compiler is the plugin and compile is the goal.

Continuation of Example1:

In the command prompt, move to FirstJavaProject folder and issue the command mvn compile

```
D:\JavaCode\FirstJavaProject>mvn compile
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.wipro:FirstJavaProject >-----
[INFO] Building FirstJavaProject 1
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ FirstJavaProject ---
```

At the end you would have received a Build failure message, indicating source option 5 is no longer supported.

```
[ERROR] Source option 5 is no longer supported. Use 6 or later.
[ERROR] Target option 1.5 is no longer supported. Use 1.6 or later.
[INFO] 2 errors
[INFO] -----
[INFO]
[INFO] BUILD FAILURE
[INFO] -----
[INFO] Total time: 8.091 s
[INFO] Finished at: 2020-01-27T11:25:57+05:30
[INFO]
```

To rectify this error, we'll edit our pom.xml and add the following lines (brown color) to change the compiler version to Java 8.

```
<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">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.wipro</groupId>
  <artifactId>FirstJavaProject</artifactId>
  <packaging>jar</packaging>
  <version>1</version>
  <name>FirstJavaProject</name>
  <url>http://maven.apache.org</url>

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

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Save the pom.xml file, go to the command prompt and again issue the mvn compile command and now you will notice that the build is success.

```
[INFO] Compiling 1 source file to D:\JavaCode\FirstJavaProject\target\classes
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 2.464 s
[INFO] Finished at: 2020-01-27T11:33:50+05:30
[INFO] -----
```

Now you can notice that a new folder called target is created in FirstJavaProject

his PC > Local Disk (D:) > JavaCode > FirstJavaProject

Name	Date modified	Type	Size
src	18-01-2020 12:54	File folder	
target	27-01-2020 11:37	File folder	
pom.xml	27-01-2020 11:31	XML Document	1 KB

If you go deep inside target folder, you'll notice that App.java file has got compiled and the corresponding .class file is found.

his PC > Local Disk (D:) > JavaCode > FirstJavaProject > target > classes > com > wipro

Name	Date modified	Type	Size
App.class	27-01-2020 11:37	CLASS File	1 KB

So the **mvn compile** command has helped to compile the source files present in the src folder.

Task1: You can add another Calc.java in the src folder. Then issue the mvn compile command and check if the Calc.class file is created.

Calc.java

```
package com.wipro;
```

```
public class Calc
{
    public int add(int a, int b)
    {
        return a+b;
    }
}
```

You can also edit the App.java to create an object of this Calc and invoke the add method.

App.java

```
package com.wipro;

public class App
{
    public static void main( String[] args )
    {
        Calc c=new Calc();
        System.out.println("The sum of 2 and 3 is "+c.add(2,3));
    }
}
```

Now go to the command prompt and issue **mvn clean**.

```
D:\JavaCode\FirstJavaProject>mvn clean
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.wipro:FirstJavaProject >-----
[INFO] Building FirstJavaProject 1
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ FirstJavaProject ---
[INFO] Deleting D:\JavaCode\FirstJavaProject\target
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] -----
[INFO] Total time:  0.374 s
[INFO] Finished at: 2020-01-27T12:14:59+05:30
[INFO] -----
```

You can notice that the target folder is deleted. mvn clean will help to clean all the earlier builds and will delete the target folder.

Now again issue **mvn compile** and check if .class files are created for both App.java and Calc.java

his PC > Local Disk (D:) > JavaCode > FirstJavaProject > target > classes > com > wipro

Name	Date modified	Type	Size
App.class	27-01-2020 12:20	CLASS File	1 KB
Calc.class	27-01-2020 12:20	CLASS File	1 KB

Now we can create a jar file out of this with the help of mvn package.

Note: JAR stands for Java ARchive. It's a file format based on the popular ZIP file format and is used for aggregating many files into one.

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

```
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ FirstJavaProject ---
[INFO] Building jar: D:\JavaCode\FirstJavaProject\target\FirstJavaProject-1.jar
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 10.716 s
[INFO] Finished at: 2020-01-27T14:40:15+05:30
[INFO]
```

You can notice that in the target folder, FirstJavaProject_1.jar file is created.

We can now execute this jar file with the following command:

```
C:\Windows\system32\cmd.exe

D:\JavaCode\FirstJavaProject>java -cp target\FirstJavaProject-1.jar com.wipro.App
The sum of 2 and 3 is 5
```

Note: -cp is the <class search path of directories and zip/jar files>

In case if some other applications are dependent on this jar file, we need to make this jar file available in our local repository. We can do so with the mvn install command.


```

[INFO] --- maven-install-plugin:2.4:install (default-install) @ FirstJavaProject ---
[INFO] Installing D:\JavaCode\FirstJavaProject\target\FirstJavaProject-1.jar to C:\Users\anrame\.m2\repository\com\wipro\FirstJavaProject\1\FirstJavaProject-1.jar
[INFO] Installing D:\JavaCode\FirstJavaProject\pom.xml to C:\Users\anrame\.m2\repository\com\wipro\FirstJavaProject\1\FirstJavaProject-1.pom
[INFO] BUILD SUCCESS
[INFO] Total time: 8.766 s
[INFO] Finished at: 2020-01-27T14:11:39+05:39
D:\JavaCode\FirstJavaProject>

```

You can see that the FirstJavaProject-1.jar is now copied to your local repository.

is PC > Local Disk (C:) > Users > anrame > .m2 > repository > com > wipro > FirstJavaProject > 1

Name	Date modified	Type	Size
_remote.repositories	27-01-2020 14:51	REPOSITORIES File	1 KB
FirstJavaProject-1.jar	27-01-2020 14:40	JAR File	3 KB
FirstJavaProject-1.pom	27-01-2020 11:31	POM File	1 KB

Note: An important point in maven lifecycle is when a phase is called via Maven command, only phases up to and including that phase will execute.

So if I run mvn clean and do mvn install, then automatically compile, test, package and install will get executed.

Task: Try out the other life cycle phases and note the results.

Example 2: Create a Web Application and check the result.

The Web application can be created with the following command in the command prompt.

```
D:\>mvn archetype:generate -DgroupId=com.mywipro -DartifactId=TestWeb -DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false
```

Run mvn install and deploy this in Tomcat webapp folder and test your application



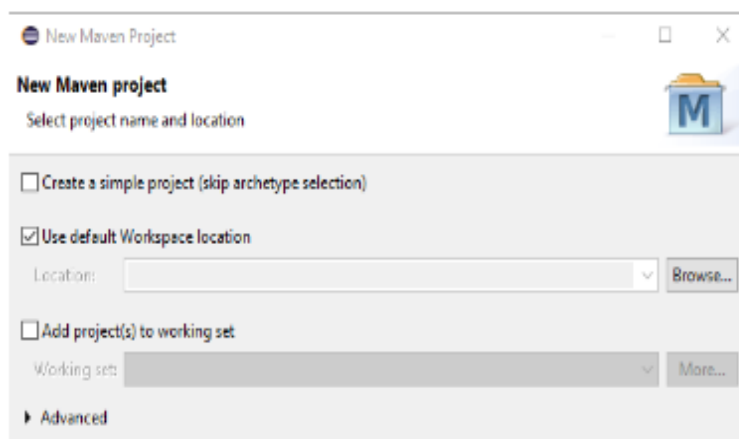
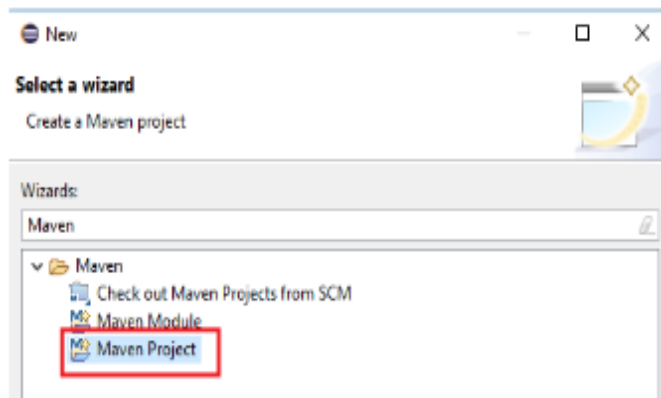
Hello World!

Maven with Eclipse

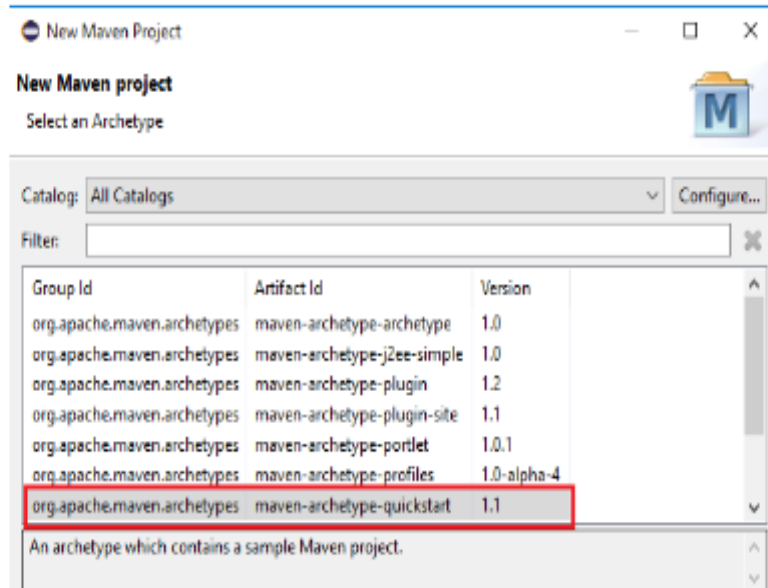
A simple “Hello World” application

To do list:

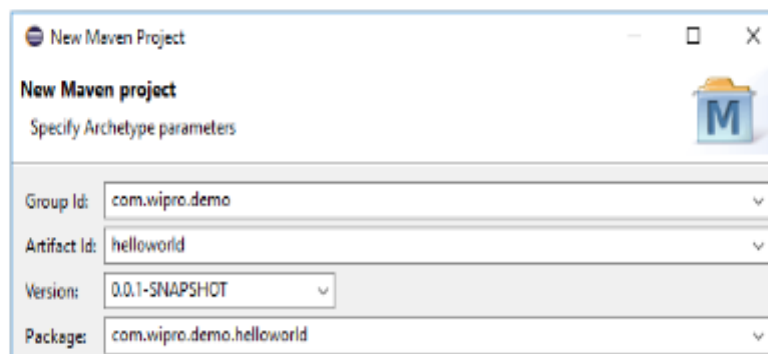
- Create a Maven Quick Start Application
 - Change compiler version
 - Package the application as JAR file
 - Execute and check the output
- In Eclipse, go to File → New → Other and search for Maven
Choose Maven Project.



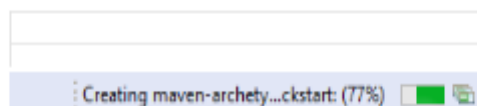
2. Choose this archetype to create a simple maven project



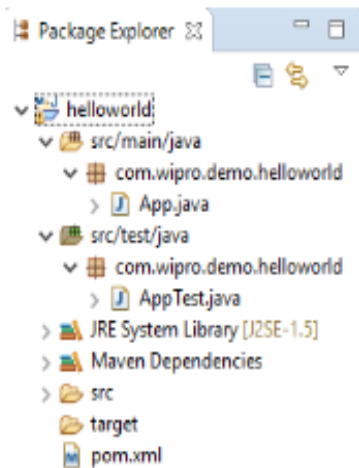
3. Provide Group Id and Artifact Id. Package name will be auto filled.



4. Click Finish and you can find the progress of your project structure getting created.



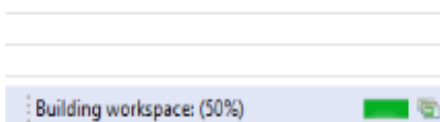
5. Project structure



6. To get JRE System Library 1.8, add the compiler properties in pom.xml file

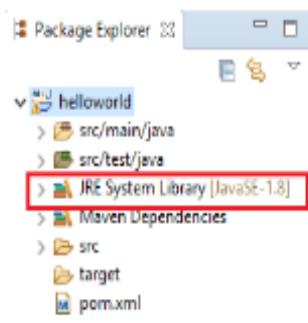
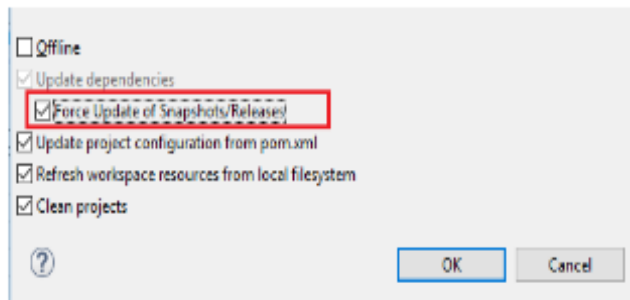
```
<properties>  
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
  <maven.compiler.source>1.8</maven.compiler.source>  
  <maven.compiler.target>1.8</maven.compiler.target>  
</properties>
```

7. Save the file and wait till the workspace gets updated.



-
8. Update the project: right click on the project name → Maven → Update Project

Remember to check this option and click ok



9. Sample “Hello World” program is available in App.java file

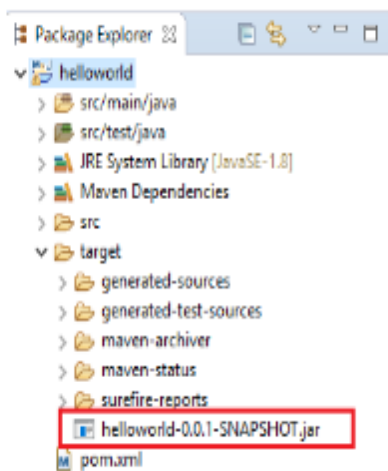
```
App.java
1 package com.wipro.demo.helloworld;
2
3 /**
4  * Hello world!
5  *
6  */
7 public class App
8 {
9     public static void main( String[] args )
10    {
11        System.out.println( "Hello World!" );
12    }
13 }
```

-
10. Executing **mvn install** command from Eclipse: right click on the project → Run as → Maven install

Wait till you see this output in the console

```
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ helloworld ---  
[INFO] Building jar: E:\MavenSpace\helloworld\target\helloworld-0.0.1-SNAPSHOT.jar  
[INFO] --- maven-install-plugin:2.4:install (default-install) @ helloworld ---  
[INFO] Installing E:\MavenSpace\helloworld\target\helloworld-0.0.1-SNAPSHOT.jar to C:\i  
[INFO] Installing E:\MavenSpace\helloworld\pom.xml to C:\Users\YU377726\.m2\repository\  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 13.920 s  
[INFO] Finished at: 2020-06-17T15:40:31+05:30  
[INFO] -----
```

11. Expand target folder



12. Run the App.java file

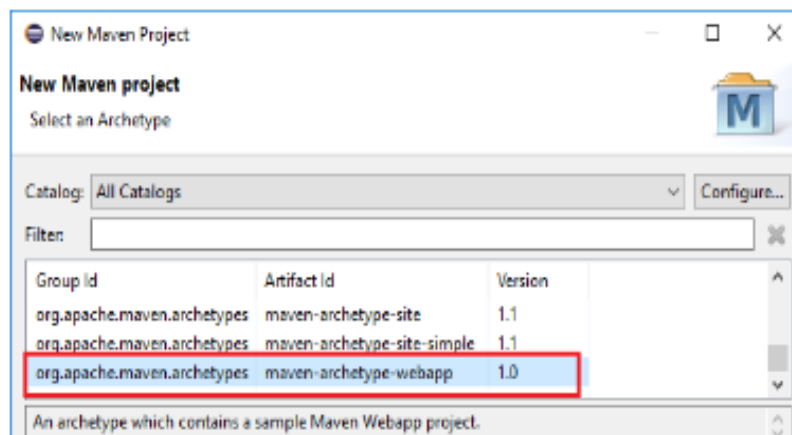


A simple "Hello World" Web Application

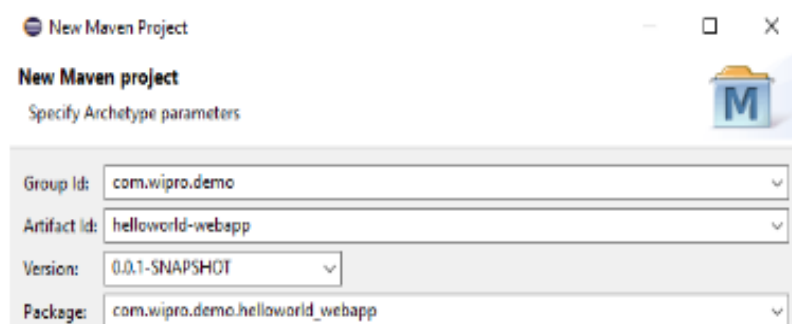
To do list:

- a) Create a Maven Web Application
- b) Change compiler version
- c) Add servlet-api JAR file
- d) Package the web application as WAR file
- e) Add jetty maven plugin
- f) Start the jetty server and run the application

1. In Eclipse, go to File → New → Other → Maven Project
2. Choose this archetype to create a simple maven webapp project

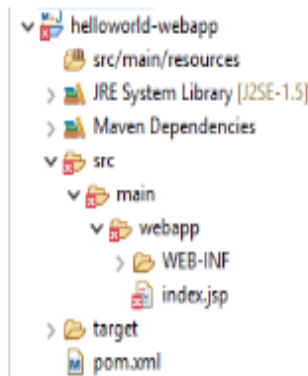


3. Provide Group Id and Artifact Id. Package name will be auto filled.



4. Click Finish and you can find the progress of your project structure getting created.

5. Project structure

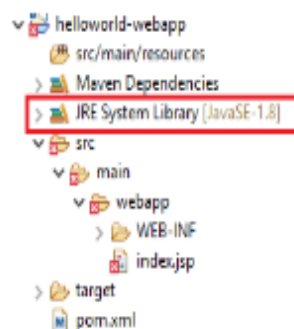


6. To get JRE System Library 1.8, add the compiler properties in pom.xml file

```
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.source>1.8</maven.compiler.source>
  <maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

7. Save the file and wait till the workspace gets updated.

8. Update the project: right click on the project name → Maven → Update Project



9. To get servlet-api JAR file, add the dependency details in pom.xml, inside the `<dependencies>` tag

```
<!-- https://mvnrepository.com/artifact/javax.servlet/servlet-api -->
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>4.0.1</version>
  <scope>provided</scope>
</dependency>
```

How to get these dependency details?

- i) Go to the mvnrepository.com and search for servlet-api

The screenshot shows the mvnrepository.com search results for 'servlet-api'. The search bar at the top contains 'servlet-api'. The left sidebar lists various repositories with their artifact counts. The main content area shows 'Found 32712 results' and a list of search results. The first result, '1. Java Servlet API', is highlighted with a red box. Below this, the details for 'Java Servlet API' are shown, including its license (CDL, GPL 2.0), categories (Java Specifications), tags (standard, servlet, javax, api, specs), and that it is used by 12,892 artifacts. At the bottom, a table shows the available versions and their repositories. The version '4.0.1' is highlighted with a red box, and it is listed as being available from the 'Central' repository.

Repository	Count
Central	23.9k
Sonatype	7.8k
Spring Plugins	4.7k
Spring Lib M	4.4k
JCenter	1.4k
Clojars	1.0k
Spring Lib Release	931

Found 32712 results

Sort: **relevance** | popular | newest

1. Java Servlet API
javax.servlet » javax.servlet-api
Java Servlet API
Last Release on Apr 26, 2018

Java Servlet API
Java Servlet API

License: **CDL** **GPL 2.0**

Categories: **Java Specifications**

Tags: **standard** **servlet** **javax** **api** **specs**

Used By: 12,892 artifacts

Central (20) | Redhat GA (1) | ICM (5)

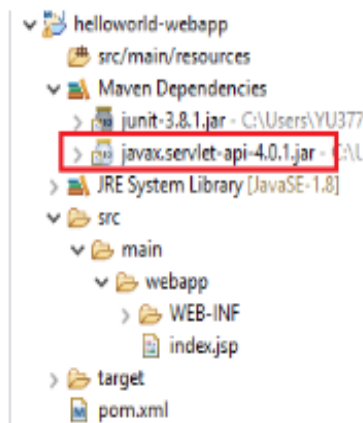
Version	Repository
4.0.1	Central

- ii) Copy the dependency details and place it in pom.xml file



```
<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>4.0.1</version>
  <scope>provided</scope>
</dependency>
```

10. Save the file and wait till the workspace gets updated.



11. Sample "Hello World" HTML code is available in index.jsp file



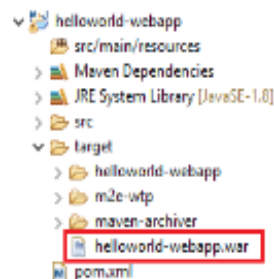
```
1 <html>
2 <body>
3   <h2>Hello World!</h2>
4 </body>
5 </html>
```

12. Executing `mvn install` command from Eclipse: right click on the project → Run as → Maven install

Wait till you see this output in the console

```
[INFO] Packaging webapp
[INFO] Assembling webapp [helloworld-webapp] in [E:\MavenSpace\helloworld-webapp\tar
[INFO] Processing war project
[INFO] Copying webapp resources [E:\MavenSpace\helloworld-webapp\src\main\webapp]
[INFO] Webapp assembled in [124 msecs]
[INFO] Building war: E:\MavenSpace\helloworld-webapp\target\helloworld-webapp.war
[INFO] WEB-INF\web.xml already added, skipping
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ helloworld-webapp ---
[INFO] Installing E:\MavenSpace\helloworld-webapp\target\helloworld-webapp.war to C:
[INFO] Installing E:\MavenSpace\helloworld-webapp\pom.xml to C:\Users\YU377726\.m2\r
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 6.994 s
[INFO] Finished at: 2020-06-17T16:58:55+05:30
[INFO]
```

13. Expand target folder

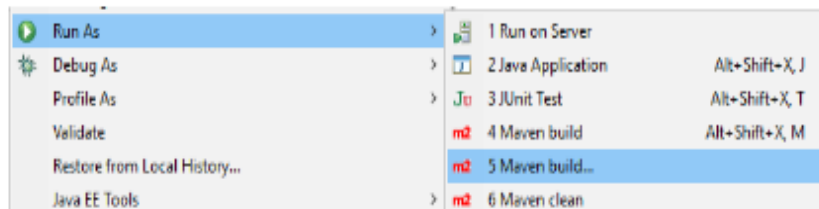


```
helloworld-webapp
├── src/main/resources
├── Maven Dependencies
├── JRE System Library [JavaSE-1.8]
├── src
├── target
│   ├── helloworld-webapp
│   ├── m2e-wtp
│   ├── maven-archiver
│   └── helloworld-webapp.war
└── pom.xml
```

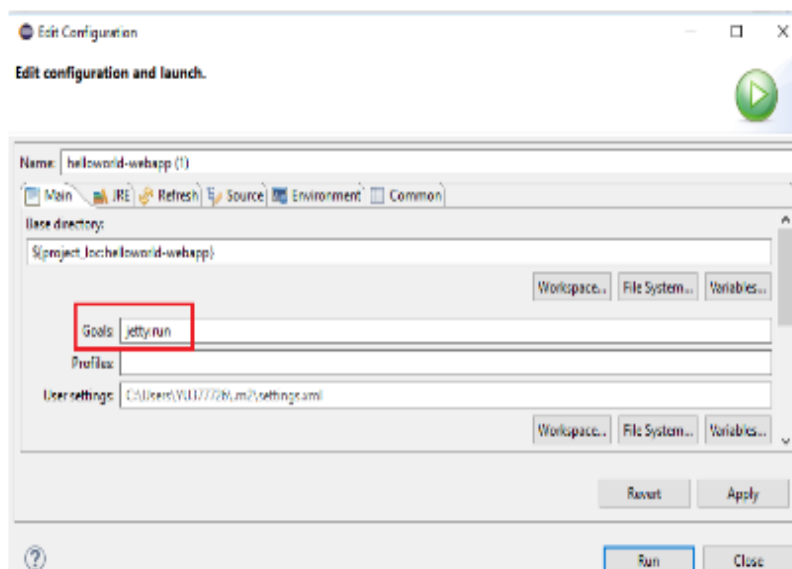
14. To add jetty maven plugin, add the plugin details in pom.xml, inside `<build>` tag

```
<build>
  <finalName>helloworld-webapp</finalName>
  <plugins>
    <plugin>
      <groupId>org.eclipse.jetty</groupId>
      <artifactId>jetty-maven-plugin</artifactId>
      <version>9.2.11.v20150529</version>
    </plugin>
  </plugins>
</build>
```

15. Executing `mvn jetty:run` command from Eclipse: right click on the project → Run As → Maven build...



16. Provide maven goals and click Run



To start the jetty server in a different port number (in case if 8080 is not available): `mvn jetty:run -Djetty.port=<port number>`

Goals: `jetty:run -Djetty.port=9001`

17. Wait till you see this output in the console

```
[WARNING] !RequestLog  
[INFO] Started ServerConnector@55f4887d{HTTP/1.1}{0.0.0.0:9001}  
[INFO] Started @11636ms  
[INFO] Started Jetty Server
```

18. From browser



Hello World!