MAVEN 3.0

Maven is more than a build tool and described as Project Management Framework.

KANDEPU RAMESH

ASPIRE Technologies

Phone:7799108899/ 7799208899

E-Mail:Ramesh@aspirecareers.in

1. INTRODUCTION

The Maven is a **Project Management Framework** and it is indeed much more than just a simple build scripting tool. Maven's declarative, standard approach to project build management simplifies many aspects of the project lifecycle.

Maven does following aspects of the project lifecycle automatically:

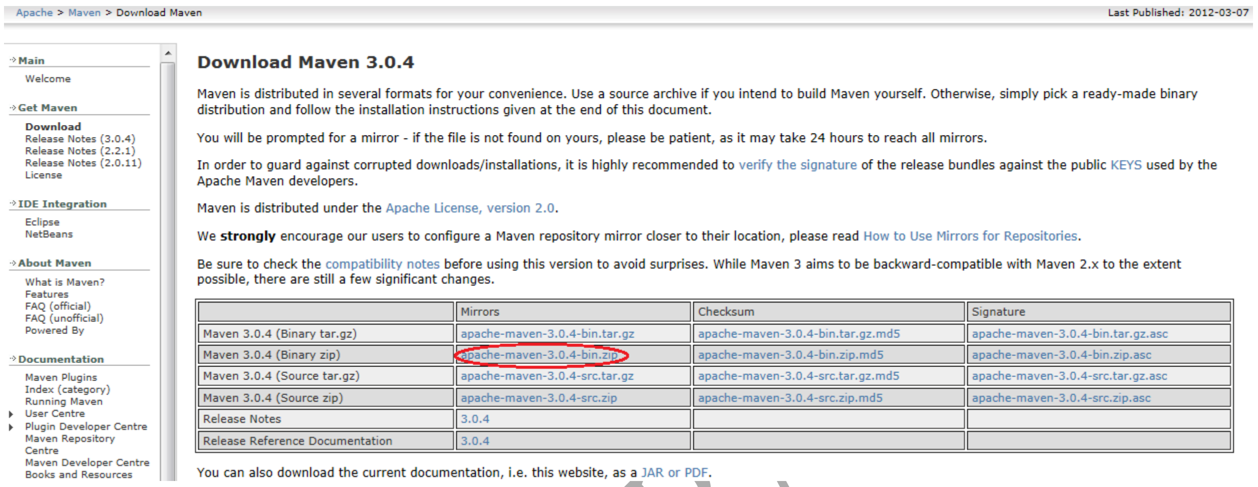
1) Downloads required jars
2) Compiles source code
3) Packaging project as .jar, .war, .ear, .pom, etc
4) Deploy project into Web server or Application server
5) Starts Server
6) Performs Unit testing
7) Prepares Test Report including Code coverage report
8) Builds Technical Site
9) Un-deploy application from server
10) Stop Server

Maven is an invaluable tool for Java development. Indeed, **maven touches so many phases of the SDLC**.

2. INSTALLATION

Maven is a pure java framework, so first of all we need to ensure JDK1.5 or above is installed and JAVA_HOME is set.

Download latest maven version (Maven 3) software from http://maven.apache.org/download.html website.



Apache > Maven > Download Maven — Last Published: 2012-03-07

Main: Welcome
Get Maven: Download, Release Notes (3.0.4), Release Notes (2.2.1), Release Notes (2.0.11), License
IDE Integration: Eclipse, NetBeans
About Maven: What is Maven?, Features, FAQ (official), FAQ (unofficial), Powered By
Documentation: Maven Plugins, Index (category), Running Maven, User Centre, Plugin Developer Centre, Maven Repository Centre, Maven Developer Centre, Books and Resources

Download Maven 3.0.4

Maven is distributed in several formats for your convenience. Use a source archive if you intend to build Maven yourself. Otherwise, simply pick a ready-made binary distribution and follow the installation instructions given at the end of this document.

You will be prompted for a mirror - if the file is not found on yours, please be patient, as it may take 24 hours to reach all mirrors.

In order to guard against corrupted downloads/installations, it is highly recommended to verify the signature of the release bundles against the public KEYS used by the Apache Maven developers.

Maven is distributed under the Apache License, version 2.0.

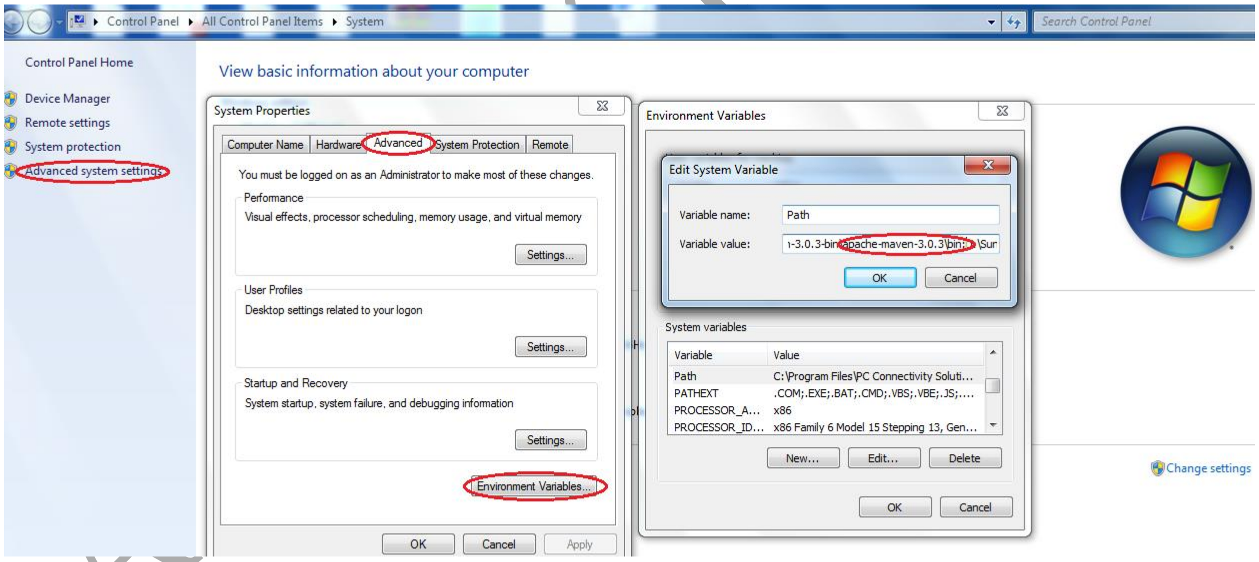
We **strongly** encourage our users to configure a Maven repository mirror closer to their location, please read How to Use Mirrors for Repositories.

Be sure to check the compatibility notes before using this version to avoid surprises. While Maven 3 aims to be backward-compatible with Maven 2.x to the extent possible, there are still a few significant changes.

	Mirrors	Checksum	Signature
Maven 3.0.4 (Binary tar.gz)	apache-maven-3.0.4-bin.tar.gz	apache-maven-3.0.4-bin.tar.gz.md5	apache-maven-3.0.4-bin.tar.gz.asc
Maven 3.0.4 (Binary zip)	apache-maven-3.0.4-bin.zip	apache-maven-3.0.4-bin.zip.md5	apache-maven-3.0.4-bin.zip.asc
Maven 3.0.4 (Source tar.gz)	apache-maven-3.0.4-src.tar.gz	apache-maven-3.0.4-src.tar.gz.md5	apache-maven-3.0.4-src.tar.gz.asc
Maven 3.0.4 (Source zip)	apache-maven-3.0.4-src.zip	apache-maven-3.0.4-src.zip.md5	apache-maven-3.0.4-src.zip.asc
Release Notes	3.0.4		
Release Reference Documentation	3.0.4		

You can also download the current documentation, i.e. this website, as a JAR or PDF.

Extract it into an appropriate directory and add the bin subdirectory to the system path.



Control Panel ▸ All Control Panel Items ▸ System

Control Panel Home: Device Manager, Remote settings, System protection, Advanced system settings

View basic information about your computer

System Properties — Computer Name | Hardware | Advanced | System Protection | Remote

You must be logged on as an Administrator to make most of these changes.

Performance: Visual effects, processor scheduling, memory usage, and virtual memory — Settings...

User Profiles: Desktop settings related to your logon — Settings...

Startup and Recovery: System startup, system failure, and debugging information — Settings...

Environment Variables... — OK, Cancel, Apply

Environment Variables — Edit System Variable

Variable name: Path

Variable value: ı-3.0.3-bin\apache-maven-3.0.3\bin;\Sur

OK, Cancel

System variables

Variable	Value
Path	C:\Program Files\PC Connectivity Soluti...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;....
PROCESSOR_A...	x86
PROCESSOR_ID...	x86 Family 6 Model 15 Stepping 13, Gen...

New..., Edit..., Delete — OK, Cancel

Change settings

Now, check that Maven is correctly installed by running mvn -version:

C:\>mvn –version

Apache Maven 3.0.3

3. PROJECT OBJECT MODEL (POM)

The heart of a Maven project, the POM, describes our project, its structure, and its dependencies. It contains a detailed description of project, including information about versioning and configuration management, dependencies, application and testing resources, team members and structure, and much more.
The POM takes the form of an XML file (called **pom.xml** by default), which is placed in project home directory. Maven **standardizes directory structure**. Hence, we would not need to declare source and target directories.

PROJECT DESCRIPTION

At the start of each Maven POM file, we will find a list of descriptive elements describing things like the project name, version number, how it is to be packaged, and so on as shown below:

#pom.xml

```
<project xmlns=”..” xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
        <!-- PROJECT DESCRIPTION -->
        <modelVersion>4.0.0</modelVersion>
        <groupId>com.mycompany.accounting</groupId>
        <artifactId>accounting-core</artifactId>
        <version>1.0</version>
        <packaging>jar</packaging>
        <name>Accounting application</name>
        <description>My First Maven Application</description>
        ...
</project>
```

The **<modelVersion>** element declares to which version of project descriptor this POM conforms.
The **<groupId>** element denotes a universally unique identifier for a project. By convention, it often corresponds to the initial part of the Java package used for the application classes (eg: edu.mycompany.accounting). When the artifact is deployed to a Maven repository, the groupId is split out into a matching **directory structure** on the repository. (eg. edu\mycompany\accounting).
The **<artifactId>** represents the actual name of the project. This, combined with the groupId, should uniquely identify the project.
Every project also has a **<version>** element, which indicates the current version number, which refers to major releases. Each version has its own directory on the Maven repository, which is a subdirectory of the project directory.
The **<packaging>** element specifies different deliverable file formats such as .jar, .war, .ear, .pom, etc. For example, in this listing, Maven will generate a file called **accounting-core-1.1.jar**.
The Maven 2 artifact is:

$$\overbrace{\underbrace{\text{com/mycompany/accounting}}_{\text{groupId}}\text{/}\underbrace{\text{accounting-core}}_{\text{artifactId}}\text{/}\underbrace{\text{1.0}}_{\text{version}}}^{\text{Artifact path}}\text{/}\overbrace{\underbrace{\text{accounting-core}}_{\text{artifactId}}\text{-}\underbrace{\text{1.1}}_{\text{version}}\text{.}\underbrace{\text{jar}}_{\text{packaging}}}^{\text{Artifact}}$$

The information in this section allows Maven to derive a unique path to the artifact generated by this project.
The generated artifact would be stored on the Maven repository in a directory called com/mycompany/accounting/accounting-core/1.0.

REPOSITORY

Exactly where Maven looks for dependencies will depend on how our repositories are set up. The default Maven2 repository is located at **http://repo1.maven.org/maven2** (it is defined in the **Super POM file** located at %MAVEN_HOME%\lib\maven-model-builder-3.0.3.jar).

Other popular public repositories are:

1) http://repository.jboss.org/nexus/content/groups/public/.
2) http://mvnrepository.com/

Note: Every public repository may not provide all jars with all versions. Hence some time we may need to configure some other public repository explicitly as show below:

Example:

```
<repositories>
    <repository>
      <id>jboss</id>
      <name>jboss repo</name>
      <url>https://repository.jboss.org/nexus/content/groups/public/</url>
    </repository>
  </repositories>
```

DEPENDECY MANAGEMENT

Dependency management is a major feature of Maven 2. **In Maven, JAR files are rarely, if ever, stored in the project directory structure.** Instead, dependencies are declared within the pom.xml file. Dependencies are the libraries we need to compile, test, and run our application. In a Maven project, we list the libraries our application needs, including the exact version number of each library. Using this information, Maven will do its best to find, retrieve, and assemble the libraries it needs during the different stages in the build lifecycle. In addition, using a powerful feature called **Transitive Dependencies**, maven will include not only the libraries that we declare but also all the extra libraries that our declared libraries need to work correctly.

DECLARING DEPENDENCIES

In the <dependencies> section of the POM file, we declare the libraries that we need to compile, test, and run our application. Dependencies are retrieved from local or remote repositories, and cached locally on our development machine, in the **C:\Users\Adminstrator\.m2\repository** directory structure (This location is specified in %MAVEN_HOME%\conf\settings.xml file). If we use the same jar in two projects, it will only be downloaded (and stored) once, which saves time and disk space.

In Maven, dependencies are handled declaratively.

Example: Suppose that our project needs to use Hibernate.

```
<dependencies>
        <dependency>
              <groupId>org.hibernate</groupId>
              <artifactId>hibernate-core</artifactId>
              <version>3.5.6-Final</version>
    </dependency>
</ dependencies>
```

In this case, Maven will look for the Hibernate JAR file in the following directory:

http://repo1.maven.org/maven2/org/hibernate/hibernate-core/3.5.6-Final/

If we tell Maven 2 that our project needs a particular library, it will try to work out what other libraries this library needs, and retrieve them as well is called as **Transitive Dependencies**.

DEPENDENCY SCOPE

We may not need to include all the dependencies in the deployed application. Some JARs are needed only for unit testing, while others will be provided at runtime by the application server. Using a technique called dependency scope, maven lets us use certain JARs only when we really need them and **excludes** them from the classpath when we don't. Maven provides several dependency scopes.

The default scope is the **compile** scope. The compile-scope dependencies are available in all phases.

```
<dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate</artifactId>
        <version>3.1</version>
</dependency>
```

A **provided dependency scope** is used to compile the application but will not be deployed. We would use this scope when we expect the JDK or server to provide the JAR. The servlet APIs is a good example:

```
<dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>servlet-api</artifactId>
        <version>2.4</version>
        <scope>provided</scope>
</dependency>
```

The **runtime dependency scope** is used for dependencies that are not needed for compilation, only for execution, such as Java Database Connectivity (JDBC) drivers:

```
<dependency>
       <groupId>oracle</groupId>
       <artifactId>oracle-jdbc</artifactId>
       <version>10.2.0.1.0XE</version>
       <scope>runtime</scope>
</dependency>
```

We use the **test dependency scope** for dependencies that are only needed to compile and run tests, and that don't need to be distributed (JUnit or TestNG, for example):

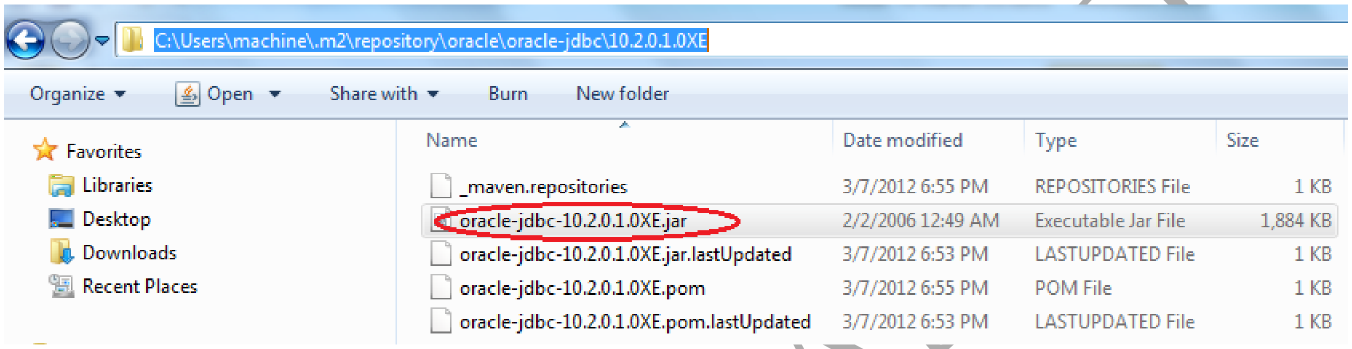
```
<dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>3.8.1</version>
        <scope>test</scope>
</dependency>
```

PROPRIETARY DEPENDENCIES

For commercial and copyright reasons, not all of the commonly used libraries are available on the public Maven repositories. A common example is the Oracle JDBC Driver, which is available free-of-charge on the Oracle web site, but it cannot be redistributed via a public Maven repository. Before downloading (manually) these libraries we required to accept license agreement.

Such proprietary libraries like these will need to add it manually to our local repository using **mvn install** command.

```
C:\>mvn install:install-file -DgroupId=oracle  -DartifactId=oracle-jdbc -Dpackaging=jar -Dversion=10.2.0.1.0XE
                         -DgeneratePom=true -Dfile=ojdbc14_g.jar
```



C:\Users\machine\.m2\repository\oracle\oracle-jdbc\10.2.0.1.0XE

Name	Date modified	Type	Size
_maven.repositories	3/7/2012 6:55 PM	REPOSITORIES File	1 KB
oracle-jdbc-10.2.0.1.0XE.jar	2/2/2006 12:49 AM	Executable Jar File	1,884 KB
oracle-jdbc-10.2.0.1.0XE.jar.lastUpdated	3/7/2012 6:53 PM	LASTUPDATED File	1 KB
oracle-jdbc-10.2.0.1.0XE.pom	3/7/2012 6:55 PM	POM File	1 KB
oracle-jdbc-10.2.0.1.0XE.pom.lastUpdated	3/7/2012 6:53 PM	LASTUPDATED File	1 KB

PROJECT INHERITANCE

All modules would have a common parent POM file which is very much like any other POM file.

```
        <modelVersion>4.0.0</modelVersion>
        <groupId>edu.aspire.webservices.rest</groupId>
        <artifactId>rest-pom</artifactId>
        <packaging>pom</packaging>
        <version>1.0</version>
        <name>RESTful Examples</name>
```

The main distinguishing factor is the <packaging> element, which is declared as a POM, rather than the WAR or JAR values. Indeed, all **parent POM files** must use the **pom** packaging type.

Then, within each child project, we need to declare a **<parent>** element that refers to the parent POM file:

```
        <parent>
            <groupId>edu.aspire.webservices.rest</groupId>
            <artifactId>rest-pom</artifactId>
            <version>1.0</version>
        </parent>
```

Note that we don't need to define the version or groupId of the child project—these values are inherited from the parent pom file.

BUILD CONFIGURATION

Maven uses components called **plug-ins** to do most of the work. The plug-in that handles java compilation is called **maven-compiler-plugin**. Many plug-ins are available from the Maven web site (http://mvnrepository.com/plugins.html).

Example:

#pom.xml

```
        <!—Build configurataion -- >
```

```
          <build>
                    <plugins>
                            <plugin>
                               <groupId>org.apache.maven.plugins</groupId>
                               <artifactId>maven-compiler-plugin</artifactId>
                               <configuration>
                                 <!-- using java 6 -->
                                  <source>1.6</source>
                                  <target>1.6</target>
                               </configuration>
                            </plugin>
                    </plugins>
          </build>
```

Refer **APPENDIX A** for related example.

The <build> section also where resource directories are defined. By default, any files placed in the src\main\config will be packaged into the generated project artifact. Any files in src\test\resources will be made available on the project classpath during unit tests.

We also can add additional resource directories. In the following example, we set up an additional resource directories for Hibernate configuration and mapping files. At build-time, these files automatically will be bundled into the resulting project artifact, along with the compiled classes.

Example:

```
<build>
          …
          <resources>
                    <resource>
                              <directory>src/main/config</directory>
                    </resource>
                    <resource>
                              <directory>src/main/resources</directory>
                    </resource>
          </resources>
</build>
```

BUILD PROFILES

Profiles are a useful way to customize the build lifecycle for different environments. They let us define properties that change depending on target environment, such as database connections, etc. At compile time, these properties can be inserted into project configuration files. For example, we may need to configure different database connections for different platforms. Suppose JDBC configuration details are stored in file called jdbc.properties, stored in the src/main/resources directory. In this file, we would use a variable expression in the place of the property value, as shown here:

```
jdbc.connection.url=${jdbc.connection.url}
```

In this case, we will define two profiles: one for a development database, and one for a test database. The <profiles> section of the POM file would look like this:

```
<profiles>
	<!-- Development environment -->
	<profile>
		<id>development</id>
		<activation>
			<activeByDefault>true</activeByDefault>
		</activation>
		<properties>
			<!-- The development database -->
			<jdbc.connection.url>jdbc:mysql://localhost/devdb</jdbc.connection.url>
		</properties>
	</profile>
	<!-- Test environment -->
	<profile>
		<id>test</id>
		<properties>
			<!-- The test database -->
			<jdbc.connection.url>jdbc:mysql://localhost/testdb</jdbc.connection.url>
		</properties>
	</profile>
</profiles>
```

Each profile has an identifier (<id>) that lets us invoke the profile by name, and a list of property values to be used for variable substitution.

Profiles can be activated in several ways. In this case, we use the activeByDefault property to define the development profile as the default profile. Therefore, running a standard Maven compile with no profiling options will use this profile:

```
C:\> mvn compile
```

In this case, the generated jdbc.properties file in the target/classes directory will look like this:

```
jdbc.connection.url=jdbc:mysql://localhost/devdb
```

To activate the test profile, we need to name it explicitly, using the -P command line option as shown here:

```
C:\> mvn compile -Ptest
```

Now, the generated jdbc.properties file, in the target/classes directory, will be configured for the test database:

```
jdbc.connection.url=jdbc:mysql://localhost/testdb
```

4. MAVEN 2 LIFE CYCLE

Ant build scripts typically have targets with names like compile, test, and deploy. In Maven 2, this notion is standardized into a set of well-known and well-defined **lifecycle phases**. Instead of invoking tasks or targets, the Maven 2 developer invokes a lifecycle phase. For example, to compile the application source code, we invoke the “compile” lifecycle phase:

C:\>mvn package

Below table summaries life cycle phases:

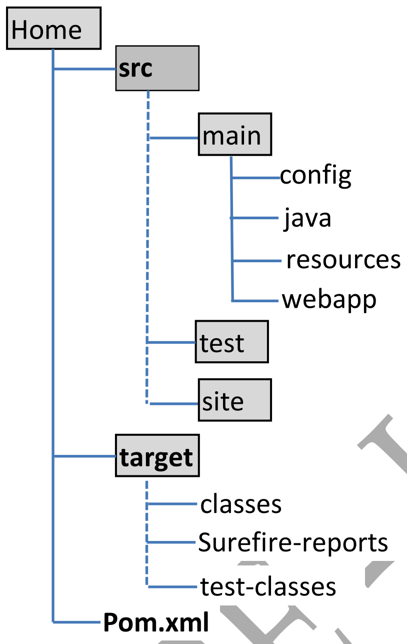
Life cycle phases	**Description**
generate-sources	Generates any extra source code needed for the application, which is generally accomplished using the appropriate plug-ins.
Compile	compiles the project source code.
test-compile	Compiles the project unit tests.
Test	Runs the unit tests (typically using JUnit) in the src/test/java directory. If any tests fail, the build will stop. In all cases, Maven generates a set of test reports in text and XML test reports in the target/surefire-reports directory.
Package	Packages the compiled code in its distributable format (JAR, WAR, etc.).
integration-test	Processes and deploys the package if necessary into an environment in which integration tests can be run.
Install	Installs the package into the local repository
Deploy	In an integration or release environment, this copies the final package to the remote repository for sharing with other developers and projects.

Note: For more details refer http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html

5. MAVEN DIRECTORY STRUCTURE

Much of Maven's power comes from the standard practices that it encourages. A developer who had previously worked on a Maven project immediately will feel familiar with the structure and organization of a new one. Time need not be wasted for reinventing directory structures, conventions, etc for each project. Although we can override any particular directory location for our own specific ends, but we really should respect the standard Maven 2 directory structure as much as possible, for several reasons:

1. It makes POM file smaller and simpler.
2. It makes the project easier to understand and makes life easier.
3. It makes it easier to integrate plug-ins.



Where 'src' is for all source code and target for generated artifacts.

The src directory has a number of subdirectories, each of which has a clearly defined purpose:

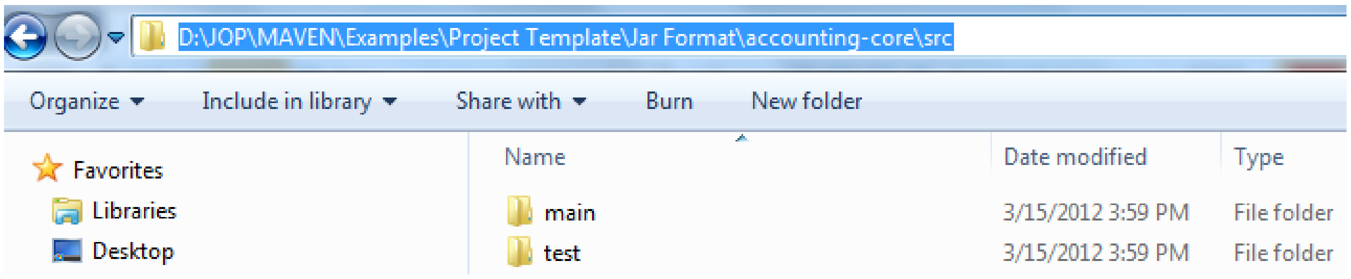
Src\main\java	Source code goes here
Src\main\resources	Other resources our application needs
Src\main\config	Configuration files
Src\main\webapp	The web application directory for a WAR project.
Src\test\java	Source code for unit tests, by convention in a directory structure mirroring the one in our main source code directory.
Src\test\resources	Resources to be used for unit tests, but that will not be deployed.
Src\site	Files used to generate the Maven project web site.

PROJECT TEMPLATE WITH ARCHETYPES

By using **archetype** plug-in, we can create standard directory structure. This is an excellent way to get a basic project environment up and running quickly. The default archetype model will produce a JAR library project.

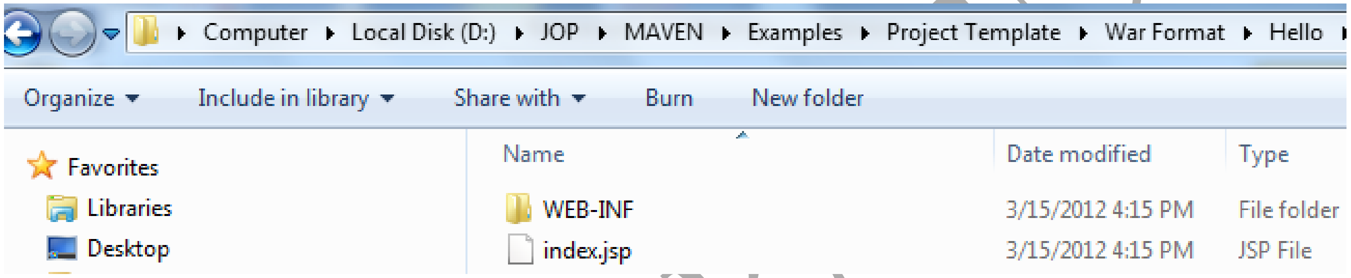
```
C:\>mvn archetype:create -DgroupId=edu.aspire.accounting -DartifactId=accounting-core -
Dpackagename=edu.aspire.core
```

This will create a complete, correctly structured, working Maven project, including a sample POM file, a sample class, and a unit test.


D:\JOP\MAVEN\Examples\Project Template\Jar Format\accounting-core\src

Name	Date modified	Type
main	3/15/2012 3:59 PM	File folder
test	3/15/2012 3:59 PM	File folder

We can use a different archetype by using the **archetypeArtifactId** command-line option. For more details refer, http://myjeeva.com/exclusive-maven-archetype-list.html web site.
C:\>mvn archetype:create -DgroupId=edu.aspire -DartifactId=Hello -DarchetypeArtifactId=**maven-archetype-webapp**
This example uses the maven-archetype-webapp archetype, which creates an empty WAR project.


Computer ▸ Local Disk (D:) ▸ JOP ▸ MAVEN ▸ Examples ▸ Project Template ▸ War Format ▸ Hello ▸

Name	Date modified	Type
WEB-INF	3/15/2012 4:15 PM	File folder
index.jsp	3/15/2012 4:15 PM	JSP File

6. DEPLOYING AN APPLICATION USING CARGO

There are many third-party tools and libraries that can help us to deploy web application. One of the most versatile is Cargo (http://cargo.codehaus.org/). Cargo is a powerful tool that allows us to deploy web application to a number of different servers, including Tomcat, Jetty, JBoss, and Weblogic. In this chapter, we will just look at how to configure Cargo to deploy a WAR application to a **running remote Tomcat server**.
Go to Appendix C for Example.

7. APPENDIX A

The maven with standalone application.

The plug-in that handles Java compilation is called **maven-compiler-plugin**. So, to set up Java compilation in Maven script, all we need to do is to configure this plug-in, which we do as follows:

Directory Structure

```
Sample
  |
  |-- src/main/java
  |       |
  |       |-- edu.aspire.core
  |               |
  |               |-- Welcome.java
  |
  |-- src/test/java
  |       |
  |       |-- edu.aspire.test
  |               |
  |               |-- WelcomeTest.java
  |
  |-- pom.xml
```

#pom.xml

```
<?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/maven-
v4_0_0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>edu.aspire</groupId>
        <artifactId>sample</artifactId>
        <version>1.0</version>
        <name>Sample Project</name>
        <description>sample project using maven framework</description>

        <!-- dependencies -->
        <dependencies>
                <dependency>
                        <groupId>junit</groupId>
                        <artifactId>junit</artifactId>
                        <version>4.9</version>
                        <scope>test</scope>
                </dependency>
          </dependencies>
        <build>
                <plugins>
                        <!-- using java 6 -->
                      <plugin>
                            <groupId>org.apache.maven.plugins</groupId>
```

```
                    <artifactId>maven-compiler-plugin</artifactId>
                    <configuration>
                        <source>1.6</source>
                        <target>1.6</target>
                    </configuration>
                </plugin>
            </plugins>
    </build>
</project>
```

//Welcome.java

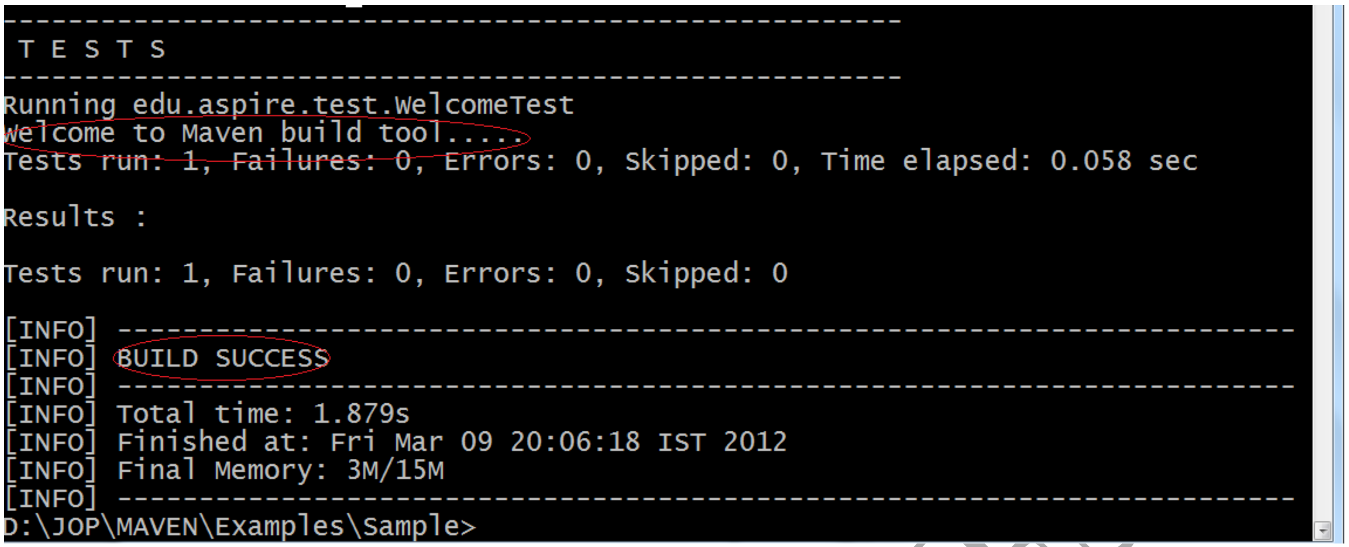
```
package edu.aspire.core;
import java.sql.*;
public class Welcome{
    public void disp(){
        System.out.println("Welcome to Maven build tool.....");
    }
}
```

#WelcomeTest.java

```
package edu.aspire.test;
import edu.aspire.core.Welcome;
import org.junit.Assert;
import org.junit.Test;
public class WelcomeTest{
    @Test
    public void testDisp(){
        Welcome obj = new Welcome();
        obj.disp();
        Assert.assertEquals(true, true);
    }
}
```

Given following command to execute.

D:\JOP\MAVEN\Examples\Sample>**mvn test**



```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running edu.aspire.test.WelcomeTest
Welcome to Maven build tool.....
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.058 sec

Results :

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

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.879s
[INFO] Finished at: Fri Mar 09 20:06:18 IST 2012
[INFO] Final Memory: 3M/15M
[INFO] ------------------------------------------------------------------------
D:\JOP\MAVEN\Examples\Sample>
```

8. APPENDIX B

This application is used to execute JDBC Application.

Download **ojdbc14.jar** file manually from oracle web site. Such proprietary libraries like these will need to add it manually to our local repository using **mvn install** command.

```
C:\>mvn install:install-file -DgroupId=oracle  -DartifactId=oracle-jdbc -Dpackaging=jar -Dversion=10.2.0.1.0XE
                                -DgeneratePom=true -Dfile=ojdbc14.jar
```

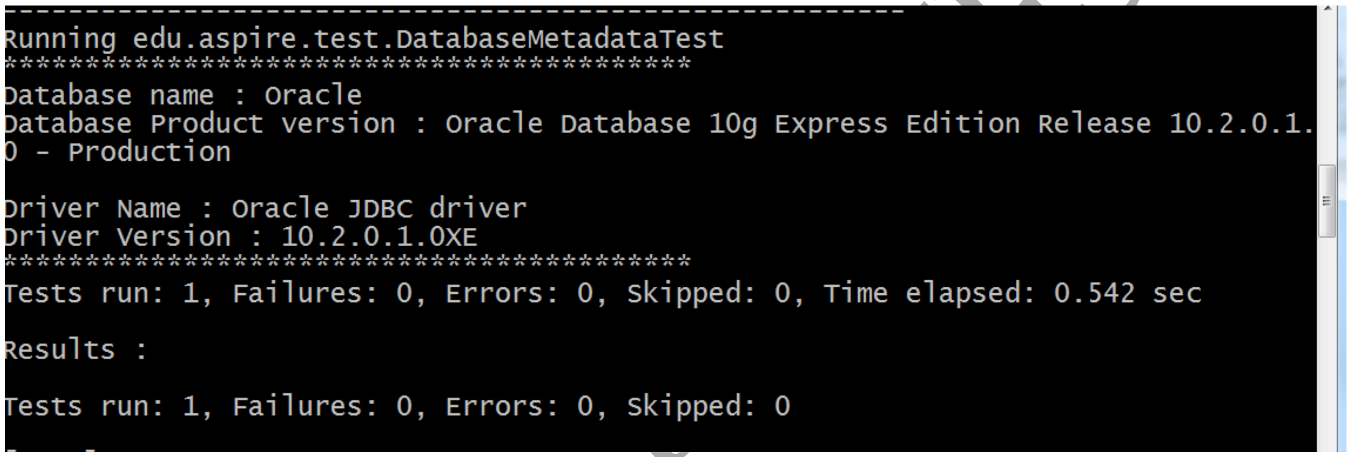
Add following <dependency> element in the pom.xml file.

```
        <dependency>
                <groupId>oracle</groupId>
                <artifactId>oracle-jdbc</artifactId>
                <version>10.2.0.1.0XE</version>
                <scope>runtime</scope>
        </dependency>
```

The dependency scope "**runtime**" indicates the jar file is required only for execution but not for compilation.

#pom.xml

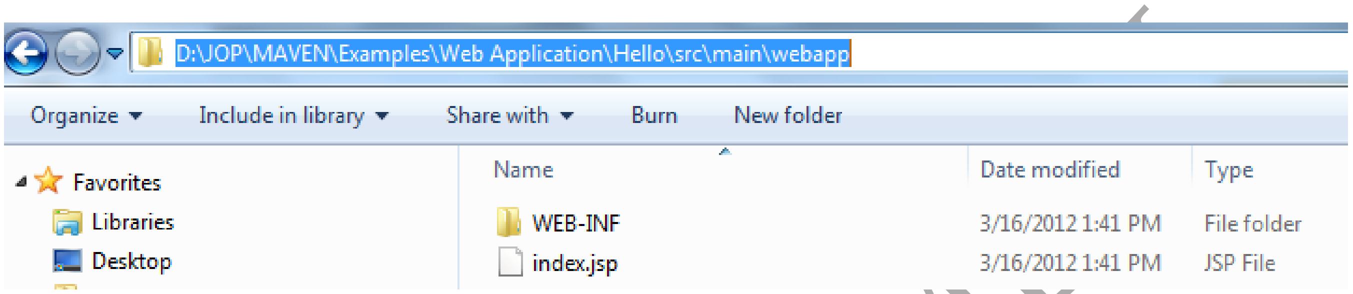
```
<?xml version="1.0" encoding="UTF-8"?>
<project …>
        <modelVersion>4.0.0</modelVersion>
        <groupId>edu.aspire</groupId>
        <artifactId>sample</artifactId>
        <version>1.0</version>
        <name>Sample Project</name>
        <description>sample project using maven framework</description>

        <!-- Dependencies -->
        <dependencies>
                <dependency>
                        <groupId>oracle</groupId>
                        <artifactId>oracle-jdbc</artifactId>
                        <version>10.2.0.1.0XE</version>
                        <scope>runtime</scope>
                </dependency>
                <dependency>
                        <groupId>junit</groupId>
                        <artifactId>junit</artifactId>
                        <version>4.9</version>
                        <scope>test</scope>
                </dependency>
  </dependencies>
  <!—Build configuration -- >
  <build>
        <plugins>
        <!-- using java 6 -->
              <plugin>
                      <groupId>org.apache.maven.plugins</groupId>
                      <artifactId>maven-compiler-plugin</artifactId>
                      <configuration>
                              <source>1.6</source>
                              <target>1.6</target>
                      </configuration>
              </plugin>
           </plugins>
        </build>
</project>
D:\JOP\MAVEN\Examples\JDBC Application>mvn clean install
```



```
-------------------------------------------------------
Running edu.aspire.test.DatabaseMetadataTest
******************************************
Database name : Oracle
Database Product version : Oracle Database 10g Express Edition Release 10.2.0.1.
0 - Production

Driver Name : Oracle JDBC driver
Driver Version : 10.2.0.1.0XE
******************************************
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.542 sec

Results :

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

9. APPENDIX C

This application is used to deploy Web Application in Tomcat Web Server.
Create Web Application template using archetype:
D:\WEBSERVICES\MAVEN\Examples\Web Application>mvn archetype:create -DgroupId=edu.aspire -DartifactId=Hello -DarchetypeArtifactId=**maven-archetype-webapp**



D:\JOP\MAVEN\Examples\Web Application\Hello\src\main\webapp

Name	Date modified	Type
WEB-INF	3/16/2012 1:41 PM	File folder
index.jsp	3/16/2012 1:41 PM	JSP File

#pom.xml

```
 <!-- Build Configuration -->
 <build>
      <plugins>
           <!--deploy in tomcat6.x -->
           <plugin>
                      <groupId>org.codehaus.cargo</groupId>
                      <artifactId>cargo-maven2-plugin</artifactId>
                      <executions>
                                 <execution>
                                            <id>verify-deploy</id>
                                            <phase>pre-integration-test</phase>
                                            <goals>
                                                       <goal>deployer-redeploy</goal>
                                            </goals>
                                 </execution>
                      </executions>
                      <configuration>
                             <wait>false</wait>
                             <container>
                                      <containerId>tomcat6x</containerId>
                                      <type>remote</type>
                              </container>
                              <configuration>
                                     <type>runtime</type>
                                          <properties>
                                                    <cargo.hostname>localhost</cargo.hostname>
                                                    <cargo.servlet.port>9090</cargo.servlet.port>
```

```
                            <cargo.tomcat.manager.url>http://localhost:9090/manager</cargo.tomcat.manager.url>
                            <cargo.remote.username>admin</cargo.remote.username>
                            <cargo.remote.password>admin</cargo.remote.password>
                            </properties>
                        </configuration>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

In this example, maven automatically deploys the packaged WAR file just before the integration tests phase. This section is optional and is designed to make it easier to run automatic integration or functional tests against the latest version of the application. The **deployer-redeploy** goal will redeploy the application on the targeted Tomcat server.

The **<configuration>** element is used to define the type of application server, and provide some server-specific configuration details indicating how to deploy to this server. For Tomcat, this consists of the URL for the Tomcat Manager application, as well as a valid Tomcat username and password that will give us access to this server.

For this to work correctly, we need to have defined a Tomcat user with the "manager" role. This is not the case by default, so we may have to modify our Tomcat configuration manually.

In case of tomocat6, %TOMCAT_HOME%\conf\tomcat-users.xml

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
        <role rolename="tomcat"/>
        <role rolename="manager"/>
        <user name="admin" password="admin" roles="tomcat,manager"/>
</tomcat-users>
```

In case of tomcat7, the %TOMCAT_HOME%\conf\tomcat-users.xml

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
        <role rolename="tomcat"/>
        <role rolename="manager-gui"/>
        <role rolename="manager-script"/>
        <user name="admin" password="admin" roles="tomcat,manager-gui,manager-script" />
</tomcat-users>
```

The pom file for Tomcat 7 is:

```
                <!--Deploy in tomcat7.x -->
                <plugin>
                        <groupId>org.apache.tomcat.maven</groupId>
                        <artifactId>tomcat7-maven-plugin</artifactId>
                        <version>2.1</version>
                        <executions>
                                <execution>
                                        <id>verify-deploy</id>
```

```
                                                <!--Process and deploy the package if necessary into an
environment where integration tests can be run-->
                                                <phase>pre-integration-test</phase>
                                                <goals>
                                                        <goal>redeploy</goal>
                                                </goals>
                                        </execution>
                                </executions>
                                <configuration>
                                  <url>http://localhost:9090/manager/text</url>
                                  <username>admin</username>
                                  <password>admin</password>
                                </configuration>
                        </plugin>
```

10. APPENDIX D

This application is used to execute Hibernate Example.

Download third part proprietary jars named ojdbc6.jar.

```
D:\WEBSERVICES\MAVEN\Examples\jars>mvn install:install-file -DgroupId=oracle  -DartifactId=oracle-jdbc -Dpackaging=jar -Dversion=11 -DgeneratePom=true -Dfile=ojdbc6.jar
```

#pom.xml

```
<project …>
        <!—Project Description -- >
        <modelVersion>4.0.0</modelVersion>
        <groupId>edu.aspire</groupId>
        <artifactId>StudentDao</artifactId>
        <version>1.0</version>
        <name>Student Dao Example</name>
        <description>Dao Impl using Hiberante Framework</description>

        <!--Repository URL -->
          <repositories>
            <repository>
              <id>jboss</id>
              <name>jboss repo</name>
              <url>https://repository.jboss.org/nexus/content/groups/public/</url>
            </repository>
          </repositories>

        <!—Dependency Management -->
        <dependencies>
                <dependency>
                      <groupId>org.hibernate</groupId>
                      <artifactId>hibernate-core</artifactId>
                      <version>3.5.6-Final</version>
                 </dependency>
                 <dependency>
                      <groupId>javassist</groupId>
                      <artifactId>javassist</artifactId>
                        <version>3.9.0.GA</version>
                </dependency>
                <dependency>
                        <groupId>org.hibernate</groupId>
                        <artifactId>hibernate-c3p0</artifactId>
                      <version>4.1.1.Final</version>
                </dependency>
                <dependency>
                <groupId>org.slf4j</groupId>
                <artifactId>slf4j-jdk14</artifactId>
                 <version>1.5.8</version>
            </dependency>
          <dependency>
                <groupId>oracle</groupId>
                <artifactId>oracle-jdbc</artifactId>
                <version>11</version>
                <scope>runtime</scope>
          </dependency>
          <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.9</version>
            <scope>test</scope>
          </dependency>
 </dependencies>
```

<!—Build Configuration -- >

```
 <build>
      <plugins>
            <!-- using java 6 -->
           <plugin>
             <groupId>org.apache.maven.plugins</groupId>
             <artifactId>maven-compiler-plugin</artifactId>
             <configuration>
               <source>1.6</source>
               <target>1.6</target>
             </configuration>
             </plugin>
        </plugins>
        <resources>
            <resource>
                 <directory>src/main/config</directory>
            </resource>
            <resource>
                 <directory>src/main/resources</directory>
            </resource>
        </resources>
</build>
</project>
```