Maven Tutorial

Maven is a powerful project management tool that is based on POM (project object model). It is used for projects build, Dependancy and Dokumentation.

**Understanding the problem without Maven**

There are many problems that we face during the project development. They are discussed below:

**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

## What it does?

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

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

Apache Maven helps to manage

* Builds
* Documentation
* Reporing
* 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

# Difference between Ant and Maven

**Ant** and **Maven** both are build tools provided by Apache. The main purpose of these technologies is to ease the build process of a project.

There are many differences between ant and maven that are given below:

|  |  |
| --- | --- |
| **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. |

**What is a Maven Repository?**

In Maven terminology, a repository is a place i.e. directory where all the project jars, library jar, plugins or any other project specific artifacts are stored and can be used by Maven easily.

Maven repository are of three types

* local
* central
* remote

**Local Repository**

Maven local repository is a folder location on your machine. It gets created when you run any maven command for the first time.

Maven local repository keeps your project's all dependencies (library jars, plugin jars etc). When you run a Maven build, then Maven automatically downloads all the dependency jars into the local repository.It helps to avoid references to dependencies stored on remote machine every time a project is build.

Maven local repository by default get created by Maven in %USER\_HOME% directory. To override the default location, mention another path in Maven settings.xml file available at %M2\_HOME%\conf directory.

<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0

http://maven.apache.org/xsd/settings-1.0.0.xsd">

<localRepository>C:/MyLocalRepository</localRepository>

</settings>

When you run Maven command, Maven will download dependencies to your custom path.

**Central Repository**

Maven central repository is repository provided by Maven community. It contains a large number of commonly used libraries.

When Maven does not find any dependency in local repository, it starts searching in central repository using following URL: <http://repo1.maven.org/maven2/>

Key concepts of Central repository

* This repository is managed by Maven community.
* It is not required to be configured.
* It requires internet access to be searched.

To browse the content of central maven repository, maven community has provided a URL: <http://search.maven.org/#browse>. Using this library, a developer can search all the available libraries in central repository.

**Remote Repository**

Sometime, Maven does not find a mentioned dependency in central repository as well then it stopped build process and output error message to console. To prevent such situation, Maven provides concept of **Remote Repository** which is developer's own custom repository containing required libraries or other project jars.

For example, using below mentioned POM.xml,Maven will download dependency (not available in central repository) from Remote Repositories mentioned in the same 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>com.companyname.projectgroup</groupId>  <artifactId>project</artifactId>  <version>1.0</version>  <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>companyname.lib2</id>  <url>http://download.companyname.org/maven2/lib2</url>  </repository>  </repositories>  </project> |

**Maven Dependency Search Sequence**

When we execute Maven build commands, Maven starts looking for dependency libraries in the following sequence:

* **Step 1 -** Search dependency in local repository, if not found, move to step 2 else if found then do the further processing.
* **Step 2 -** Search dependency in central repository, if not found and remote repository/repositories is/are mentioned then move to step 4 else if found, then it is downloaded to local repository for future reference.
* **Step 3 -** If a remote repository has not been mentioned, Maven simply stops the processing and throws error (Unable to find dependency).
* **Step 4 -** Search dependency in remote repository or repositories, if found then it is downloaded to local repository for future reference otherwise Maven as expected stop processing and throws error (Unable to find dependency).

## What are 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 and which can be executed using 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 following command

mvn compiler:compile

## Plugin Types

Maven provided following two types of Plugins:

|  |  |
| --- | --- |
| **Type** | **Description** |
| Build plugins | They execute during the build and should be configured in the <build/> element of pom.xml |
| Reporting plugins | They execute during the site generation and they should be configured in the <reporting/> element of the pom.xml |

Following is the list of few common plugins:

|  |  |
| --- | --- |
| **Plugin** | **Description** |
| clean | Clean up target after the build. Deletes the target directory. |
| compiler | Compiles Java source files. |
| surefire | Run the JUnit unit tests. Creates test reports. |
| jar | Builds a JAR file from the current project. |
| war | Builds a WAR file from the current project. |
| javadoc | Generates Javadoc for the project. |
| antrun | Runs a set of ant tasks from any phase mentioned of the build. |

## Example

We've used **maven-antrun-plugin** extensively in our examples to print data on console. See [Maven Build Profiles](https://www.tutorialspoint.com/maven/maven_build_profiles.htm) chapter. Let to understand it in a better way let's create a pom.xml in C:\MVN\project 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>com.companyname.projectgroup</groupId>

<artifactId>project</artifactId>

<version>1.0</version>

<build>

<plugins>

<plugin>

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

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

<version>1.1</version>

<executions>

<execution>

<id>id.clean</id>

<phase>clean</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>clean phase</echo>

</tasks>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

</project>

Next, open command console and go to the folder containing pom.xml and execute the following **mvn** command.

C:\MVN\project>mvn clean

Maven will start processing and display clean phase of clean life cycle

[INFO] Scanning for projects...

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

[INFO] Building Unnamed - com.companyname.projectgroup:project:jar:1.0

[INFO] task-segment: [post-clean]

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

[INFO] [clean:clean {execution: default-clean}]

[INFO] [antrun:run {execution: id.clean}]

[INFO] Executing tasks

[echo] clean phase

[INFO] Executed tasks

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

[INFO] BUILD SUCCESSFUL

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

[INFO] Total time: < 1 second

[INFO] Finished at: Sat Jul 07 13:38:59 IST 2012

[INFO] Final Memory: 4M/44M

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

The above example illustrates the following key concepts:

* Plugins are specified in pom.xml using plugins element.
* Each plugin can have multiple goals.
* You can define phase from where plugin should starts its processing using its phase element. We've used **clean** phase.
* You can configure tasks to be executed by binding them to goals of plugin. We've bound **echo** task with **run** goal of *maven-antrun-plugin*.
* That's it, Maven will handle the rest. It will download the plugin if not available in local repository and starts its processing.

Maven uses **archetype** plugins to create projects. To create a simple java application, we'll use maven-archetype-quickstart plugin. In example below, We'll create a maven based java application project in C:\MVN folder.

Let's open command console, go the C:\MVN directory and execute the following **mvn** command.

C:\MVN>mvn archetype:generate

-DgroupId=com.companyname.bank

-DartifactId=consumerBanking

-DarchetypeArtifactId=maven-archetype-quickstart

-DinteractiveMode=false

Maven will start processing and will create the complete java application project structure.

[INFO] Scanning for projects...

[INFO]

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

[INFO] Building Maven Stub Project (No POM) 1

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

[INFO]

[INFO] >>> maven-archetype-plugin:2.4:generate (default-cli) > generate-sources

@ standalone-pom >>>

[INFO]

[INFO] <<< maven-archetype-plugin:2.4:generate (default-cli) < generate-sources

@ standalone-pom <<<

[INFO]

[INFO] --- maven-archetype-plugin:2.4:generate (default-cli) @ standalone-pom --

-

[INFO] Generating project in Batch mode

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

---

[INFO] Using following parameters for creating project from Old (1.x) Archetype:

maven-archetype-quickstart:1.0

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

---

[INFO] Parameter: groupId, Value: com.companyname.bank

[INFO] Parameter: packageName, Value: com.companyname.bank

[INFO] Parameter: package, Value: com.companyname.bank

[INFO] Parameter: artifactId, Value: consumerBanking

[INFO] Parameter: basedir, Value: C:\MVN

[INFO] Parameter: version, Value: 1.0-SNAPSHOT

[INFO] project created from Old (1.x) Archetype in dir: C:\MVN\consumerBanking

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

[INFO] BUILD SUCCESS

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

[INFO] Total time: 03:19 min

[INFO] Finished at: 2015-09-26T12:18:26+05:30

[INFO] Final Memory: 15M/247M

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

Now go to C:/MVN directory. You'll see a java application project created named consumerBanking (as specified in artifactId). Maven uses a standard directory layout as shown below:



Using above example, we can understand following key concepts

|  |  |
| --- | --- |
| **Folder Structure** | **Description** |
| consumerBanking | contains src folder and pom.xml |
| src/main/java | contains java code files under the package structure (com/companyName/bank). |
| src/main/test | contains test java code files under the package structure (com/companyName/bank). |
| src/main/resources | it contains images/properties files (In above example, we need to create this structure manually). |

If you see, Maven also created a sample Java Source file and Java Test file. Open C:\MVN\consumerBanking\src\main\java\com\companyname\bank folder, you will see App.java.

package com.companyname.bank;

/\*\*

\* Hello world!

\*

\*/

public class App

{

public static void main( String[] args )

{

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

}

}

Open C:\MVN\consumerBanking\src\test\java\com\companyname\bank folder, you will see AppTest.java.

package com.companyname.bank;

import junit.framework.Test;

import junit.framework.TestCase;

import junit.framework.TestSuite;

/\*\*

\* Unit test for simple App.

\*/

public class AppTest extends TestCase

{

/\*\*

\* Create the test case

\*

\* @param testName name of the test case

\*/

public AppTest( String testName )

{

super( testName );

}

/\*\*

\* @return the suite of tests being tested

\*/

public static Test suite()

{

return new TestSuite( AppTest.class );

}

/\*\*

\* Rigourous Test :-)

\*/

public void testApp()

{

assertTrue( true );

}

}

Developers are required to place their files as mentioned in table above and Maven handles the all the build related complexities.

# Maven - Build & Test Project

What we learnt in Project Creation chapter is how to create a Java application using Maven. Now we'll see how to build and test the application.

Go to C:/MVN directory where you've created your java application. Open *consumerBanking* folder.You will see the **POM.xml** file with following contents.

<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.companyname.projectgroup</groupId>

<artifactId>project</artifactId>

<version>1.0</version>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

</dependency>

</dependencies>

</project>

Here you can see, Maven already added Junit as test framework. By default Maven adds a source file **App.java** and a test file **AppTest.java** in its default directory structure discussed in previous chapter.

Let's open command console, go the C:\MVN\consumerBanking directory and execute the following **mvn** command.

C:\MVN\consumerBanking>mvn clean package

Maven will start building the project.

[INFO] Scanning for projects...

[INFO]

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

[INFO] Building consumerBanking 1.0-SNAPSHOT

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

[INFO]

[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ consumerBanking ---

[INFO] Deleting C:\MVN\consumerBanking\target

[INFO]

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

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

[INFO] skip non existing resourceDirectory C:\MVN\consumerBanking\src\main\resources

[INFO]

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

[INFO] Changes detected - recompiling the module!

[WARNING] File encoding has not been set, using platform encoding Cp1252, i.e. build is platform dependent!

[INFO] Compiling 1 source file to C:\MVN\consumerBanking\target\classes

[INFO]

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

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

i.e. build is platform dependent!

[INFO] skip non existing resourceDirectory C:\MVN\consumerBanking\src\test\resources

[INFO]

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

[INFO] Changes detected - recompiling the module!

[WARNING] File encoding has not been set, using platform encoding Cp1252, i.e. b

uild is platform dependent!

[INFO] Compiling 1 source file to C:\MVN\consumerBanking\target\test-classes

[INFO]

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

[INFO] Surefire report directory: C:\MVN\consumerBanking\target\surefire-reports

-------------------------------------------------------

T E S T S

-------------------------------------------------------

Running com.companyname.bank.AppTest

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.063 sec

Results :

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

[INFO]

[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ consumerBanking ---

[INFO] Building jar: C:\MVN\consumerBanking\target\consumerBanking-1.0-SNAPSHOT.jar

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

[INFO] BUILD SUCCESS

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

[INFO] Total time: 14.406 s

[INFO] Finished at: 2015-09-27T17:58:06+05:30

[INFO] Final Memory: 14M/247M

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

You've built your project and created final jar file, following are the key learning concepts

* We give maven two goals, first to clean the target directory (clean) and then package the project build output as jar(package).
* Packaged jar is available in consumerBanking\target folder as consumerBanking-1.0-SNAPSHOT.jar.
* Test reports are available in consumerBanking\target\surefire-reports folder.
* Maven compiled source code file(s) and then test source code file(s).
* Then Maven run the test cases.
* Finally Maven created the package.

Now open command console, go the C:\MVN\consumerBanking\target\classes directory and execute the following java command.

C:\MVN\consumerBanking\target\classes>java com.companyname.bank.App

You will see the result

Hello World!

## Adding Java Source Files

Let's see how we can add additional Java files in our project. Open C:\MVN\consumerBanking\src\main\java\com\companyname\bank folder, create Util class in it as Util.java.

package com.companyname.bank;

public class Util

{

public static void printMessage(String message){

System.out.println(message);

}

}

Update App class to use Util class.

package com.companyname.bank;

/\*\*

\* Hello world!

\*

\*/

public class App

{

public static void main( String[] args )

{

Util.printMessage("Hello World!");

}

}

Now open command console, go the C:\MVN\consumerBanking directory and execute the following **mvn** command.

C:\MVN\consumerBanking>mvn clean compile

After Maven build is successful, go the C:\MVN\consumerBanking\target\classes directory and execute the following java command.

C:\MVN\consumerBanking\target\classes>java -cp com.companyname.bank.App

You will see the result

Hello World!

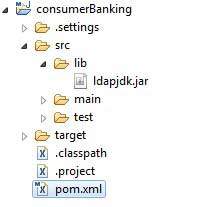
# Maven - External Dependencies

Now as you know Maven does the dependency management using concept of [Maven Repositories](https://www.tutorialspoint.com/maven/maven_repositories.htm). But what happens if dependency is not available in any of remote repositories and central repository? Maven provides answer for such scenario using concept of **External Dependency**.

For an example, let us do the following changes to project created in [Maven Creating Project](https://www.tutorialspoint.com/maven/maven_creating_project.htm) section.

* Add **lib** folder to src folder
* Copy any jar into the lib folder. We've used **ldapjdk.jar**, which is a helper library for LDAP operations.

Now our project structure should look like following:



Here you are having your own library specific to project, which is very usual case and it can contain jars which may not be available in any repository for maven to download from. If your code is using this library with Maven then Maven build will fail because it cannot download or refer to this library during compilation phase.

To handle the situation, let's add this external dependency to maven **pom.xml** using following way.

<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.companyname.bank</groupId>

<artifactId>consumerBanking</artifactId>

<packaging>jar</packaging>

<version>1.0-SNAPSHOT</version>

<name>consumerBanking</name>

<url>http://maven.apache.org</url>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>ldapjdk</groupId>

<artifactId>ldapjdk</artifactId>

<scope>system</scope>

<version>1.0</version>

<systemPath>${basedir}\src\lib\ldapjdk.jar</systemPath>

</dependency>

</dependencies>

</project>

Look at the second dependency element under dependencies in above example which clears following key concepts about **External Dependency**.

* External dependencies (library jar location) can be configured in pom.xml in same way as other dependencies.
* Specify groupId same as name of the library.
* Specify artifactId same as name of the library.
* Specify scope as system.
* Specify system path relative to project location.

Hope now you are clear about external dependencies and you will be able to specify external dependencies in your Maven project.

# Maven - Project Documents

This tutorial will teach you how to create documentation of the application in one go. So let's start, go to C:/MVN directory where you had created your java **consumerBanking** application. Open *consumerBanking* folder and execute the following **mvn** command.

C:\MVN>mvn site

Maven will start building the project.

[INFO] Scanning for projects...

[INFO]

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

[INFO] Building consumerBanking 1.0-SNAPSHOT

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

[INFO]

[INFO] --- maven-site-plugin:3.3:site (default-site) @ consumerBanking ---

[WARNING] Report plugin org.apache.maven.plugins:maven-project-info-reports-plugin has an empty version.

[WARNING]

[WARNING] It is highly recommended to fix these problems because they threaten the stability of your build.

[WARNING]

[WARNING] For this reason, future Maven versions might no longer support building such malformed projects.

[INFO] configuring report plugin org.apache.maven.plugins:maven-project-info-reports-plugin:2.8.1

[INFO] Relativizing decoration links with respect to project URL: http://maven.apache.org

[INFO] Rendering site with org.apache.maven.skins:maven-default-skin:jar:1.0 skin.

[INFO] Generating "Dependencies" report --- maven-project-info-reports-plugin:2.8.1

[INFO] Generating "Dependency Convergence" report --- maven-project-info-reports-plugin:2.8.1

[INFO] Generating "Dependency Information" report --- maven-project-info-reports-plugin:2.8.1

[INFO] Generating "About" report --- maven-project-info-reports-plugin:2.8.1

[INFO] Generating "Plugin Management" report --- maven-project-info-reports-plugin:2.8.1

[INFO] Generating "Project Plugins" report --- maven-project-info-reports-plugin:2.8.1

[INFO] Generating "Project Summary" report --- maven-project-info-reports-plugin:2.8.1

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

[INFO] BUILD SUCCESS

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

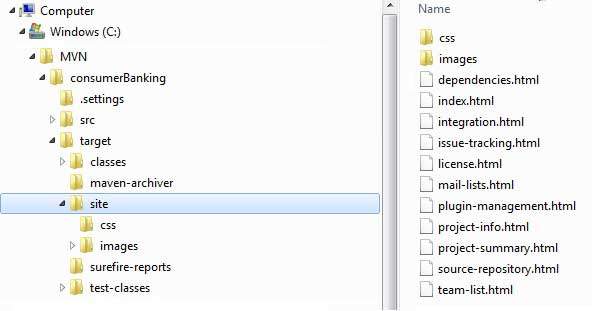
[INFO] Total time: 37.828 s

[INFO] Finished at: 2015-09-27T12:11:27+05:30

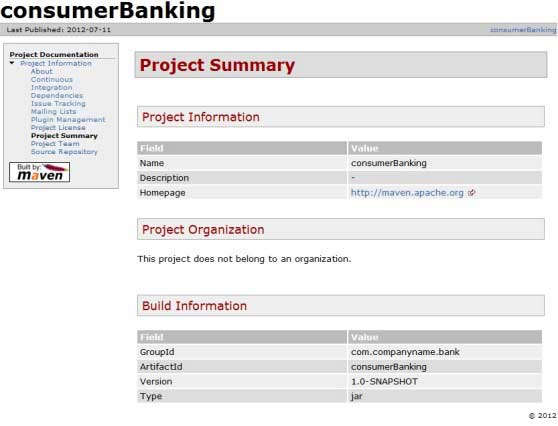
[INFO] Final Memory: 23M/247M

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

That's it. Your project documentation is ready. Maven has created a site within target directory.



Open C:\MVN\consumerBanking\target\site folder. Click on index.html to see the documentation.



Maven creates the documentation using a documentation-processing engine called [Doxia](http://maven.apache.org/doxia/index.html) which reads multiple source formats into a common document model.To write documentation for your project, you can write your content in a following few commonly used formats which are parsed by Doxia.

|  |  |  |
| --- | --- | --- |
| **Format Name** | **Description** | **Reference** |
| APT | A Plain Text document format | [doxia format](http://maven.apache.org/doxia/format.html) |
| XDoc | A Maven 1.x documentation format | [jakarta format](http://jakarta.apache.org/site/jakarta-site2.html) |
| FML | Used for FAQ documents | [fml format](http://maven.apache.org/doxia/references/fml-format.html) |
| XHTML | Extensible HTML | [XHTML wiki](http://en.wikipedia.org/wiki/XHTML) |

## Dependency Scope

Transitive Dependencies Discovery can be restricted using various Dependency Scope as mentioned below

|  |  |
| --- | --- |
| **Scope** | **Description** |
| compile | This scope indicates that dependency is available in classpath of project. It is default scope. |
| provided | This scope indicates that dependency is to be provided by JDK or web-Server/Container at runtime |
| runtime | This scope indicates that dependency is not required for compilation, but is required during execution. |
| test | This scope indicates that the dependency is only available for the test compilation and execution phases. |
| system | This scope indicates that you have to provide the system path. |
| import | This scope is only used when dependency is of type pom. This scopes indicates that the specified POM should be replaced with the dependencies in that POM's <dependencyManagement> section. |

# Maven - Creating Project

Maven uses **archetype** plugins to create projects. To create a simple java application, we'll use maven-archetype-quickstart plugin. In example below, We'll create a maven based java application project in C:\MVN folder.

Let's open command console, go the C:\MVN directory and execute the following **mvn** command.

C:\MVN>mvn archetype:generate

-DgroupId=com.companyname.bank

-DartifactId=consumerBanking

-DarchetypeArtifactId=maven-archetype-quickstart

-DinteractiveMode=false

Maven will start processing and will create the complete java application project structure.

[INFO] Scanning for projects...

[INFO]

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

[INFO] Building Maven Stub Project (No POM) 1

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

[INFO]

[INFO] >>> maven-archetype-plugin:2.4:generate (default-cli) > generate-sources

@ standalone-pom >>>

[INFO]

[INFO] <<< maven-archetype-plugin:2.4:generate (default-cli) < generate-sources

@ standalone-pom <<<

[INFO]

[INFO] --- maven-archetype-plugin:2.4:generate (default-cli) @ standalone-pom --

-

[INFO] Generating project in Batch mode

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

---

[INFO] Using following parameters for creating project from Old (1.x) Archetype:

maven-archetype-quickstart:1.0

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

---

[INFO] Parameter: groupId, Value: com.companyname.bank

[INFO] Parameter: packageName, Value: com.companyname.bank

[INFO] Parameter: package, Value: com.companyname.bank

[INFO] Parameter: artifactId, Value: consumerBanking

[INFO] Parameter: basedir, Value: C:\MVN

[INFO] Parameter: version, Value: 1.0-SNAPSHOT

[INFO] project created from Old (1.x) Archetype in dir: C:\MVN\consumerBanking

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

[INFO] BUILD SUCCESS

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

[INFO] Total time: 03:19 min

[INFO] Finished at: 2015-09-26T12:18:26+05:30

[INFO] Final Memory: 15M/247M

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

Now go to C:/MVN directory. You'll see a java application project created named consumerBanking (as specified in artifactId). Maven uses a standard directory layout as shown below:



Using above example, we can understand following key concepts

|  |  |
| --- | --- |
| **Folder Structure** | **Description** |
| consumerBanking | contains src folder and pom.xml |
| src/main/java | contains java code files under the package structure (com/companyName/bank). |
| src/main/test | contains test java code files under the package structure (com/companyName/bank). |
| src/main/resources | it contains images/properties files (In above example, we need to create this structure manually). |

If you see, Maven also created a sample Java Source file and Java Test file. Open C:\MVN\consumerBanking\src\main\java\com\companyname\bank folder, you will see App.java.

package com.companyname.bank;

/\*\*

\* Hello world!

\*

\*/

public class App

{

public static void main( String[] args )

{

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

}

}

Open C:\MVN\consumerBanking\src\test\java\com\companyname\bank folder, you will see AppTest.java.

package com.companyname.bank;

import junit.framework.Test;

import junit.framework.TestCase;

import junit.framework.TestSuite;

/\*\*

\* Unit test for simple App.

\*/

public class AppTest extends TestCase

{

/\*\*

\* Create the test case

\*

\* @param testName name of the test case

\*/

public AppTest( String testName )

{

super( testName );

}

/\*\*

\* @return the suite of tests being tested

\*/

public static Test suite()

{

return new TestSuite( AppTest.class );

}

/\*\*

\* Rigourous Test :-)

\*/

public void testApp()

{

assertTrue( true );

}

}

Developers are required to place their files as mentioned in table above and Maven handles the all the build related complexities.

In next section, we'll disc

Now, Maven’s dependency library look-up sequences is changed to :

1. Search in Maven local repository, if not found, go step 2, else exit.
2. Search in Maven central repository, if not found, go step 3, else exit.
3. Search in java.net Maven remote repository, if not found, prompt error message, else exit.