

**Note:**

The build tool is used to set up everything which is required to run code independently.

It generates source code, compiling code, and packaging code to a jar.

<Maven provides a common platform to perform these activities for Java>

***Apache Maven is a software project management and comprehension tool.***

Based on the concept of a **project object model (POM**), Maven can manage a project's build, reporting and documentation from a central piece of information.

Tool that can be used for building and managing any **Java-based project**.

Maven provides **pom.xml** which is the core of any project.

This is the **configuration file** where all required information’s are kept.

Many of the IDEs (Integrated Development Environments) are available which makes it easy to use.

Maven stores all **project jars**.

Library jar is in a place called **repository** which could be a **central, local or remote** repository.

Maven downloads the dependency jar from a **central repository**.

Downloaded libraries are stored in the **local repository** called .**m2**.

Maven uses the libraries available in an .m2 folder and if any new dependency added then maven downloads from the central repository to local repository.

If libraries are not available in the central repository then maven looks for the remote repository.

The user has to configure the remote repository in pom.xml to download from the remote repository.

**Maven’s Objectives**

Making the **build process** easy

Providing a **uniform build system**

Providing **quality project information**

Providing guidelines for **best practices development**

Allowing **transparent migration to new features**.

<**build** - The term build may refer to the **process** by which source code is **converted** into a stand-alone form that can be run on a computer or to the form itself.

One of the most important steps of a software build is the **compilation process**, where **source code** files are **converted** into **executable code.**

A “**build**” is a developed application for the customers that is given by development team to the **software** testing team. A “**release**” is an official launch of the application for the customers.

A **build** when tested and certified by the **software** testing team is provided to the customers as “**release**”.>

**Feature Summary of Maven:**

**Simple project setup** that follows best practices - get a new project or module started in seconds.

**Consistent usage** across all projects - means **no ramp up time** for new developers coming onto a project

**Superior dependency management** including automatic updating, dependency closures (also known as **transitive dependencies**)

Able to easily work with **multiple projects at the same time**

A large and growing repository of libraries and metadata to use out of the box, and arrangements in place with the largest Open Source projects for real-time availability of their **latest releases**

Extensible, with the ability to easily write plugins in Java or scripting languages

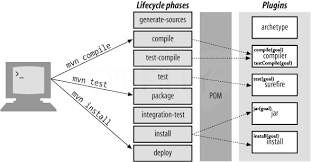
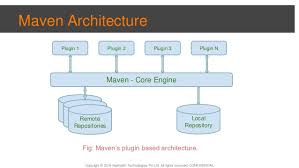
Instant access to new features with little or no extra configuration

**Dependency management:** Maven encourages the use of a central repository of JARs and other dependencies. Maven comes with a mechanism that your project's clients can use to **download any JARs required for building your project from a central JAR repository.**

This allows users of Maven to **reuse JARs** across projects and encourages communication between projects to ensure that **backward compatibility issues** are dealt with.

|  |  |  |
| --- | --- | --- |
| **Maven Life Cycle** | | |
| **3 Stages** |  | |
| ***Stages*** | ***Plug Ins*** | ***Purpose*** |
| **Compile** | Maven Compiler | Compile the code in the Java Project |
| **Test** | Maven Sure Fire | Run TestNG / Junit Test Suite |
| **Resources** | Maven Resources | Generate JAR/WAR/EAR |

**Maven Architecture**



***Working with Maven***

1. **Java Project 🡪 Maven Project**
2. **Maven Project (direct)**

**Java Project** 🡪 **Maven Project**

Maven Installation and Set-Up

==============================

On Windows:

1. Open the link "https://maven.apache.org/download.cgi"

2. Scroll down to 'Files' section

3. Click on the link corresponding to 'Binary zip archive' and download the zip file

4. Extract the file to a specific directory

5. Now open windows Start menu; type variables.

6. Click on "Edit system environment variables"

7. System Properties window will open up. Now click on 'Environment Variables' button on the window. And 'Environment Variables' window will open up.

8. Click on 'New' button under user variables. Provide "MAVEN\_HOME" under Variable name and directory location of maven (e.g. C:\apache-maven-3.3.9) needs to be provided under Variable value. Click OK

9. Check whether 'PATH' is present User variables.

a) If not create a new variable following the above step. Provide "PATH" under Variable name and "%MAVEN\_HOME%\bin" under Variable value.

b) If "PATH" variable exists then select the variable and click on Edit button. Under Variable value append "%MAVEN\_HOME%\bin" at the end. Click OK.

10. Open command prompt.

a) Type mvn -version and press enter. It should display the maven version.

**11) If you work in a proxy enabled environment you need to follow the below steps:**

**a) Go to the 'conf' folder of Maven package which you have downloaded from internet.**

**b) Open 'settings.xml'**

**c) Scroll down and go to 'proxies' section, uncomment it and modify the proxy settings as per internet proxy settings.**

**d) Save and close the file.**

**Maven Set-up in Eclipse:**

=========================

# In Eclipse go to Help menu -> Install New Software -> Enter " http://download.eclipse.org/technology/m2e/releases " 'Work With' field and press Enter

# Select check box corresponding to "Maven Integration for Eclipse" and click on Next

# accept the Agreement and click on Finish

# Restart Eclipse if required.

1. Create a normal Java project; right click on the project

2. Scroll down and hover the mouse on 'Configure' and click on 'Convert to Maven Project'

3. Click on Finish. You can see 'pom.xml' file gets created under project home directory.

4. Open the file. Click on 'pom.xml' from bottom panel.

5. Include the dependency as per the requirement. And save the file. Dependencies(i.e. jar files) will be downloaded from maven repository provided that

The internet connection exists.

- If you want to disable the maven nature of your project you may follow the below process:

# Right click on the project -> Scroll down and hover the mouse on Maven -> Click on 'Disable Maven Nature'

**Maven Project**

**Plugins**

1. **Compiler**
2. **Sure Fire**
3. **Resources**
4. **Compiler Plug In**

The Compiler Plugin has two goals. Both are already bound to their proper phases within the Maven Lifecycle and are therefore, automatically executed during their respective phases.

compiler: **compile** is bound to the compile phase and is used to compile the **main source files.**

compiler: **testCompile** is bound to the test-compile phase and is used to compile the **test source files.**

**Maven Dependency Search Sequence**

Maven searches for dependency libraries in the following sequence:

**1.Local dependency repository.**

**2.Central dependency repository**

**3. Remote dependency repository**

***Maven stops searching once it finds the jar file.***

**Maven Repositories: 3**

1. **Local Repository**

Maven local repository is a **local folder** on your machine.

Maven local repository is **created when you run any maven command for the first time**.

Maven local repository stores all dependency library jars, plugin jars, etc. on your development machine.

When Maven downloads the dependency jars it stores the jar files in the local Maven repository.

If a newer version is needed Maven would download the newer version.

If the version being declared in the dependency element in pom.xml file is already in the local Maven repository it just uses it without downloading.

Local Maven repository avoid referencing to dependencies stored on remote machine every time a project is build.

By **default** Maven creates the local repository under **%USER\_HOME% directory**.

We can set the folder for Maven local repository 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>

After setting the new local repository folder when running Maven command, Maven will download dependencies to the **custom path**.

1. **Central Repository**

Maven central repository is repository managed by **Maven community**.

Maven central repository contains a large number of commonly used libraries.

And we can publish our own libraries to Maven central repository as well.

When Maven cannot find any dependency jar file in your local repository, it starts searching in Maven central repository using following URL: **http://repo1.maven.org/maven2/**.

We don't need to configure the Maven central repository URL. But we do need internet access to download and search the Maven central repository.

To browse the central maven repository type in the following URL in your browser address bar.

http://search.maven.org/#browse

We can search a jar file in central repository.

1. **Remote 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>companyname.lib2</id>

<url>http://download.companyname.org/maven2/lib2</url>

</repository>

</repositories>

</project>

**Working with Maven:**

1. **Maven-compiler-plugin:**

Compiler-plugin is used to compile the sources of our project.

<**plugin**>

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

<**artifactId**>maven-compiler-plugin</**artifactId**>

<**version**>3.5.4</**version**>

<**configuration**>

<**source**>1.8</**source**>

<**target**>1.8</**target**>

</**configuration**>

</**plugin**>

1. **Maven-surefire-plugin**: Surefire-plugin is responsible for running tests that are placed in test source directory /src/test/java.

<**plugin**>

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

<**artifactId**>maven-surefire-plugin</**artifactId**>

<**version**>2.19.1</**version**>

<**configuration**>

<**suiteXmlFiles**>*<!-- TestNG suite XML files -->*<**suiteXmlFile**>testng.xml</**suiteXmlFile**></**suiteXmlFiles**>

</**configuration**>

</**plugin**>

1. **Maven Resources Plugin:**

The Resources Plugin handles the copying of project resources to the output directory. There are two different kinds of resources: main resources and test resources.

The difference is that the main resources are the resources associated to the main source code while the test resources are associated to the test source code.

Thus, this allows the separation of resources for the main source code and its unit tests.

Starting with version 2.3 this plugin uses the **Maven Filtering shared component** for filtering resources.

**<plugin>**

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

**<artifactId>maven-source-plugin</artifactId>**

**<executions>**

**<execution>**

**<id>attach-sources</id>**

**<goals>**

**<goal>jar</goal>**

**</goals>**

**</execution>**

**</executions>**

**</plugin>**

## To skip test cases in Maven

***mvn package -DskipTests***

add the below tag

**<properties>**

**<maven.test.skip>true</maven.test.skip>**

**</properties>**

**Command:** ***mvn package –Dmaven.test.skip=true***

**Note:** ***Maven will download all the 3 plugins automatically for basic executions.***

**POM:**

A **Project Object Model** or POM is the fundamental unit of work in Maven.

It is an **XML** file that contains information about the **project** and **configuration** details used by Maven to build the project.

It contains **default values** for most projects.

Examples:

**build directory**- **target**;

the **source directory**- **src/main/java**;

the **test source directory**- **src/test/java**;

The POM was renamed from project.xml in Maven 1 to pom.xml in Maven 2.

Instead of having a maven.xml file that contains the goals that can be executed, the goals or plugins are now configured in the pom.xml.

When executing a task or goal, Maven looks for the POM in the current directory. It reads the POM, gets the needed configuration information, then executes the goal.

Some of the **configuration** that can be specified in the **POM** are the **project dependencies,** the **plugins** or **goals** that can be executed, the **build profiles**, and so on. Other information such as the project version, description, developers, mailing lists and such can also be specified.

**Super POM**

The Super POM is **Maven's default POM**. All POMs extend the Super POM unless explicitly set, meaning the configuration specified in the Super POM is inherited by the POMs you created for your projects.

**Minimal POM**

**project root**

**modelVersion** - should be set to 4.0.0

**groupId** - the id of the project's group.

**artifactId** - the id of the artifact (project)

**version** - the version of the artifact under the specified group

**FAQs:**

***What are the steps involved in project deployment?***

1. Check-in the code from all projects in progress into the SVN or source code repository and tag it.
2. Download the complete source code from SVN.
3. Build the application.
4. Store the build output either WAR or EAR file to a common network location.
5. Get the file from network and deploy the file to the production site.
6. Updated the documentation with date and updated version number of the application.

***What does it mean when you say Maven uses Convention over Configuration?***

Maven uses Convention over Configuration which means developers are not required to create build process themselves. Developers do not have to mention each and every configuration details.

***What are the aspects Maven manages?***

Maven provides developers ways to manage following −

Builds

Documentation

Reporting

Dependencies

SCMs

Releases

Distribution

mailing list

***What is Maven artifact?***

***An artifact is a file, usually a JAR that gets deployed to a Maven repository.***

A Maven build produces one or more artifacts, such as a compiled JAR and a "sources" JAR.

Each artifact has a group ID (usually a reversed domain name, like com.example.foo), an artifact ID (just a name), and a version string. The three together uniquely identify the artifact.

***A project's dependencies are specified as artifacts.***

***What is 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.

**3 build lifecycle of Maven.**

The three build lifecycles are −

clean:cleans up artifacts created by prior builds.

default (or build):This is used to build the application.

site: generates site documentation for the project.

**What is the command to quickly build your Maven site?**

Type the command −

mvn site

**What would the command *mvn clean* do?**

This command removes the target directory with all the build data before starting the build process.

What are the phases of a Maven Build Lifecycle?

Following are the phases −

**validate −** validate the project is correct and all necessary information is available.

**compile −** compile the source code of the project.

**test −** test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed

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

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

**verify −** run any checks to verify the package is valid and meets quality criteria.

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

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

What is a goal in Maven terminology?

A goal represents a specific task which contributes to the building and managing of a project. It may be bound to zero or more build phases. A goal not bound to any build phase could be executed outside of the build lifecycle by direct invocation.

*mvn clean dependency:copy-dependencies package*?

This command will clean the project, copy the dependencies and package the project (executing all phases up to *package*).

What phases does a Clean Lifecycle consist?

The clean lifecycle consists of the following phases −

pre-clean

clean

post-clean

What phases does a Site Lifecycle consist?

The phases in Site Lifecycle are −

pre-site

site

post-site

site-deploy

What is Build Profile?

A Build profile is a set of configuration values which can be used to set or override default values of Maven build. Using a build profile, you can customize build for different environments such as Production v/s Development environments.

What are different types of Build Profiles?

Build profiles are of three types −

**Per Project −** Defined in the project POM file, pom.xml.

**Per User −** Defined in Maven settings xml file (%USER\_HOME%/.m2/settings.xml).

**Global −** Defined in Maven global settings xml file (%M2\_HOME%/conf/settings.xml)

How can you activate profiles?

A Maven Build Profile can be activated in various ways −

Explicitly using command console input.

Through maven settings.

Based on environment variables (User/System variables).

OS Settings (for example, Windows family).

Present/missing files.

What is the default location for your local repository?

~/m2./repository.

What is the command to install JAR file in local repository?

mvn install

What is Central Repository?

It 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/.

What is Remote Repository?

Sometimes, Maven does not find a mentioned dependency in central repository as well then it stops the 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.

What is the sequence in which Maven searches for dependency libraries?

Following is the search pattern −

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).

Why are Maven Plugins used?

Maven Plugins are used to −

create jar file.

create war file.

compile code files.

unit testing of code.

create project documentation.

create project reports.

What are the types of Maven Plugins?

Maven provides following two types of Plugins −

**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

When does Maven use External Dependency concept?

Maven dependency management using concept of Maven Repositories (Local, Central, Remote). Suppose dependency is not available in any of remote repositories and central repository; in such scenarios Maven uses concept of External Dependency.

What are the things you need to define for each 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.

What is Archetype?

Archetype is a Maven plugin whose task is to create a project structure as per its template.

What is the command to create a new project based on an archtype?

Type the following command −

mvnarchetype:generate

What is SNAPSHOT in Maven?

SNAPSHOT is a special version that indicates a current development copy. Unlike regular versions, Maven checks for a new SNAPSHOT version in a remote repository for every build.

What is difference between Snapshot and Version?

In case of Version, if Maven once downloaded the mentioned version say data-service:1.0, it will never try to download a newer 1.0 available in repository. To download the updated code, data-service version is be upgraded to 1.1.

In case of SNAPSHOT, Maven will automatically fetch the latest SNAPSHOT (data-service:1.0-SNAPSHOT) everytime app-ui team build their project.

What is transitive dependency in Maven?

Transitive dependency means to avoid needing to discover and specify the libraries that your own dependencies require, and including them automatically.

What does dependency management mean in the context of transitive dependency?

It means to directly specify the versions of artifacts to be used when they are encountered in transitive dependencies. For an example project C can include B as a dependency in its dependencyManagement section and directly control which version of B is to be used when it is ever referenced.

Maven determines what version of a dependency is to be used when multiple versions of an artifact are encountered. If two dependency versions are at the same depth in the dependency tree, the first declared dependency will be used. This is called dependency mediation.

What is dependency scope? Name all the dependency scope.

Dependency scope includes dependencies as per the current stage of the build. Various Dependency Scopes are −

**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 scope indicates that the specified POM should be replaced with the dependencies in that POM's <dependencyManagement> section.

What is the minimal set of information for matching a dependency references against a dependencyManagement section ?

{groupId,artifactId,type,classifier}.

How do you reference a property defined in your pom.xml file?

To reference a property defined in your pom.xml, the property name uses the names of the XML elements that define the value, with "pom" being allowed as an alias for the project (root) element.

So ${pom.name} refers to the name of the project, ${pom.version} refers to the version of the project, ${pom.build.finalName} refers to the final name of the file created when the built project is packaged, etc.

What are the default values for packaging element? If there is no packaging element defined? What is the default value for that?

Some of the valid packaging values are jar, war, ear and pom. If no packaging value has been specified, it will default to jar.

What is the value for packaging element in pom for a project that is purely meta-data?

pom

What is the use of execution element in pom file?

The <execution> element contains information's required for the execution of a plugin.

What is a project's fully qualified artifact name?

<groupId>:<artifactId>:<version>

If you do not define any information, where does your pom inherits that information from?

All POMs inherit from a parent (despite explicitly defined or not). This base POM is known as the Super POM, and contains values inherited by default.

How profiles are specified in Maven?

Profiles are specified using a subset of the elements available in the POM itself.

What are the elements in POM that a profile can modify when specified in the POM?

<repositories>, <pluginRepositories>,<dependencies>, <plugins> ,<properties>, <modules><reporting>,<dependencyManagement>,<distributionManagement>

Why profile is used in Maven?

To give portability to projects ( e.g. windows, linux etc).

What are the benefit of storing JARS/external dependencies in local repository instead of remote one?

It uses less storage, it makes checking out project quicker, non need for versioning JAR files.

How can you build your project offline?

Use the command −

mvn o package.

How do you exclude dependency?

Using the *exclusion* element.

What is a system dependency?

Dependency with scope system are always available and are not looked up in repository, they are usually used to tell Maven about dependencies which are provided by the JDK or the VM. Thus, system dependencies are especially useful for resolving dependencies on artifacts which are now provided by the JDK.