Apache Maven

Build Tool

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

About the Tutorial

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

This tutorial will teach you how to use Maven in your day-to-day life of any project development using Java.

Audience

This tutorial has been prepared for the beginners to help them understand the basic functionality of Maven tool. After completing this tutorial you will find yourself at a moderate level of expertise in using Apache Maven from where you can take yourself to next levels.

Prerequisites

We assume you are going to use Maven to handle enterprise level Java projects development. So it is beneficial to have the knowledge of software development, Java SE, overview of Java EE development and deployment process.

Copyright and Disclaimer

Table of Contents

1. APACHE MAVEN — OVERVIEW

What is Maven?

Maven is a project management and comprehension tool that provides developers a complete build lifecycle framework. Development team can automate the project's build infrastructure in almost no time as Maven uses a standard directory layout and a default build lifecycle.

In case of multiple development teams environment, Maven can set-up the way to work as per standards in a very short time. As most of the project setups are simple and reusable, Maven makes life of developer easy while creating reports, checks, build and testing automation setups.

Maven provides developers ways to manage the following:

- Builds
- Documentation
- Reporting
- Dependencies
- SCMs
- Releases
- Distribution
- Mailing list

To summarize, Maven simplifies and standardizes the project build process. It handles compilation, distribution, documentation, team collaboration and other tasks seamlessly. Maven increases reusability and takes care of most of the build related tasks.

Maven Evolution

Maven was originally designed to simplify building processes in Jakarta Turbine project. There were several projects and each project contained slightly different ANT build files. JARs were checked into CVS.

Apache group then developed **Maven** which can build multiple projects together, publish projects information, deploy projects, share JARs across several projects and help in collaboration of teams.

Objective

The primary goal of Maven is to provide developer with the following:

- A comprehensive model for projects, which is reusable, maintainable, and easier to comprehend.
- Plugins or tools that interact with this declarative model.

Maven project structure and contents are declared in an xml file, pom.xml, referred as Project Object Model (POM), which is the fundamental unit of the entire Maven system. In later chapters, we will explain POM in detail.

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 detail. Maven provides sensible default behavior for projects. When a Maven project is created, Maven creates default project structure. Developer is only required to place files accordingly and he/she need not to define any configuration in pom.xml.

As an example, following table shows the default values for project source code files, resource files and other configurations. Assuming, **${basedir}** denotes the project location:

Item	Default
source code	${basedir}/src/main/java
Resources	${basedir}/src/main/resources
Tests	${basedir}/src/test
Complied byte code	${basedir}/target
distributable JAR	${basedir}/target/classes

In order to build the project, Maven provides developers with options to mention life-cycle goals and project dependencies (that rely on Maven plugin capabilities and on its default conventions). Much of the project management and build related tasks are maintained by Maven plugins.

Developers can build any given Maven project without the need to understand how the individual plugins work. We will discuss Maven Plugins in detail in the later chapters.

Features of Maven

- Simple project setup that follows best practices.
- Consistent usage across all projects.
- Dependency management including automatic updating.
- A large and growing repository of libraries.
- Extensible, with the ability to easily write plugins in java or scripting languages.
- Instant access to new features with little or no extra configuration.
- **Model-based builds**: Maven is able to build any number of projects into predefined output types such as jar, war, metadata.
- **Coherent site of project information**: Using the same metadata as per the build process, maven is able to generate a website and a PDF including complete documentation.
- **Release management and distribution publication**: Without additional configuration, maven will integrate with your source control system such as CVS and manages the release of a project.
- **Backward Compatibility**: You can easily port the multiple modules of a project into Maven 3 from older versions of Maven. It can support the older versions also.
- **Automatic parent versioning**: No need to specify the parent in the sub module for maintenance.
- **Parallel builds**: It analyzes the project dependency graph and enables you to build schedule modules in parallel. Using this, you can achieve the performance improvements of 20-50%.
- **Better Error and Integrity Reporting**: Maven improved error reporting, and it provides you with a link to the Maven wiki page where you will get full description of the error.

2. APACHE MAVEN — ENVIRONMENT SETUP

Maven is a Java based tool, so the very first requirement is to have JDK installed on your machine.

System Requirement

JDK	1.7 or above.
Memory	No minimum requirement.
Disk Space	No minimum requirement.
Operating System	No minimum requirement.

Step 1 - Verify Java Installation on your Machine

Open console and execute the following **java** command.

OS	Task	Command
Windows	Open Command Console	c:\> java -version
Linux	Open Command Terminal	$ java -version
Mac	Open Terminal	machine:~ joseph$ java -version

Let's verify the output for all the operating systems:

OS	Output
Windows	java version "1.7.0_60" Java(TM) SE Runtime Environment (build 1.7.0_60-b19) Java HotSpot(TM) 64-Bit Server VM (build 24.60-b09, mixed mode)
Linux	java version "1.7.0_60" Java(TM) SE Runtime Environment (build 1.7.0_60-b19) Java HotSpot(TM) 64-Bit Server VM (build 24.60-b09, mixed mode)

Mac	java version "1.7.0_60" Java(TM) SE Runtime Environment (build 1.7.0_60-b19) Java HotSpot(TM) 64-Bit Server VM (build 24.60-b09, mixed mode)

If you do not have Java installed, install the Java Software Development Kit (SDK) from http://www.oracle.com/technetwork/java/javase/downloads/index.html. We are assuming Java 1.7.0.60 as installed version for this tutorial.

Step 2: Set JAVA Environment

Set the **JAVA_HOME** environment variable to point to the base directory location where Java is installed on your machine. For example:

OS	Output
Windows	Set the environment variable JAVA_HOME to C:\Program Files\Java\jdk1.7.0_60
Linux	export JAVA_HOME=/usr/local/java-current
Mac	export JAVA_HOME=/Library/Java/Home

Append Java compiler location to System Path.

OS	Output
Windows	Append the string "**;**C:\Program Files\Java\jdk1.7.0.60\bin" to the end of the system variable, Path.
Linux	export PATH=$PATH:$JAVA_HOME/bin/
Mac	not required

Verify Java Installation using **java -version** command as explained above.

Step 3: Download Maven Archive

Download Maven 2.2.1 from http://maven.apache.org/download.html.

OS	Archive name
Windows	apache-maven-3.3.1-bin.zip

Linux	apache-maven-3.3.1-bin.tar.gz
Mac	apache-maven-3.3.1-bin.tar.gz

Step 4: Extract the Maven Archive

Extract the archive, to the directory you wish to install Maven 3.3.1. The subdirectory apache-maven-3.3.1 will be created from the archive.

OS	Location (can be different based on your installation)
Windows	C:\Program Files\Apache Software Foundation\apache-maven-3.3.1
Linux	/usr/local/apache-maven
Mac	/usr/local/apache-maven

Step 5: Set Maven Environment Variables

Add M2_HOME, M2, MAVEN_OPTS to environment variables.

OS	Output
Windows	Set the environment variables using system properties. M2_HOME=C:\Program Files\Apache Software Foundation\apache-maven-3.3.1 M2=%M2_HOME%\bin MAVEN_OPTS=-Xms256m -Xmx512m
Linux	Open command terminal and set environment variables. export M2_HOME=/usr/local/apache-maven/apache-maven-3.3.1 export M2=$M2_HOME/bin export MAVEN_OPTS=-Xms256m -Xmx512m
Mac	Open command terminal and set environment variables. export M2_HOME=/usr/local/apache-maven/apache-maven-3.3.1 export M2=$M2_HOME/bin export MAVEN_OPTS=-Xms256m -Xmx512m

Step 6: Add Maven bin Directory Location to System Path

Now append M2 variable to System Path.

OS	Output
Windows	Append the string ;%M2% to the end of the system variable, Path.
Linux	export PATH=$M2:$PATH
Mac	export PATH=$M2:$PATH

Step 7: Verify Maven Installation

Now open console and execute the following **mvn** command.

OS	Task	Command
Windows	Open Command Console	c:\> mvn --version
Linux	Open Command Terminal	$ mvn --version
Mac	Open Terminal	machine:~ joseph$ mvn --version

Finally, verify the output of the above commands, which should be as follows:

OS	Output
Windows	Apache Maven 3.3.1 (r801777; 2009-08-07 00:46:01+0530) Java version: 1.7.0_60 Java home: C:\Program Files\Java\jdk1.7.0_60 \jre
Linux	Apache Maven 3.3.1 (r801777; 2009-08-07 00:46:01+0530) Java version: 1.7.0_60 Java home: C:\Program Files\Java\jdk1.7.0_60 \jre
Mac	Apache Maven 3.3.1 (r801777; 2009-08-07 00:46:01+0530) Java version: 1.7.0_60 Java home: C:\Program Files\Java\jdk1.7.0_60 \jre

3. APACHE MAVEN — POM

POM stands for Project Object Model. It is fundamental unit of work in Maven. It is an XML file that resides in the base directory of the project as pom.xml.

The POM contains information about the project and various configuration detail used by Maven to build the project(s).

POM also contains the goals and plugins. While executing a task or goal, Maven looks for the POM in the current directory. It reads the POM, gets the needed configuration information, and then executes the goal. Some of the configuration that can be specified in the POM are following:

- project dependencies
- plugins
- goals
- build profiles
- project version
- developers
- mailing list

Before creating a POM, we should first decide the project **group** (groupId), its **name** (artifactId) and its version as these attributes help in uniquely identifying the project in repository.

POM Example

```
<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.project-group</groupId>
   <artifactId>project</artifactId>
   <version>1.0</version>

</project>
```

It should be noted that there should be a single POM file for each project.

- All POM files require the **project** element and three mandatory fields: **groupId, artifactId, version.**
- Projects notation in repository is **groupId:artifactId:version.**
- Minimal requirements for a POM:

Node	Description
Project root	This is project root tag. You need to specify the basic schema settings such as apache schema and w3.org specification.
Model version	Model version should be 4.0.0.
groupId	This is an Id of project's group. This is generally unique amongst an organization or a project. For example, a banking group com.company.bank has all bank related projects.
artifactId	This is an Id of the project. This is generally name of the project. For example, consumer-banking. Along with the groupId, the artifactId defines the artifact's location within the repository.
version	This is the version of the project. Along with the groupId, It is used within an artifact's repository to separate versions from each other. For example: **com.company.bank:consumer-banking:1.0** **com.company.bank:consumer-banking:1.1.**

Super POM

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

Maven use the effective POM (configuration from super pom plus project configuration) to execute relevant goal. It helps developers to specify minimum configuration detail in his/her pom.xml. Although configurations can be overridden easily.

An easy way to look at the default configurations of the super POM is by running the following command: **mvn help:effective-pom**.

Create a pom.xml in any directory on your computer. Use the content of the above mentioned example pom.

In example below, we've created a pom.xml in C:\MVN\project folder.

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

```
C:\MVN\project>mvn help:effective-pom
```

Maven will start processing and display the effective-pom.

```
[INFO] Scanning for projects...
[INFO] Searching repository for plugin with prefix: 'help'.
[INFO] ------------------------------------------------------------------------
[INFO] Building Unnamed - com.companyname.project-group:project-name:jar:1.0
[INFO]    task-segment: [help:effective-pom] (aggregator-style)
[INFO] ------------------------------------------------------------------------
[INFO] [help:effective-pom {execution: default-cli}]
[INFO]

.....

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: < 1 second
[INFO] Finished at: Thu Jul 05 11:41:51 IST 2012
[INFO] Final Memory: 6M/15M
[INFO] ------------------------------------------------------------------------
```

Effective POM displayed as result in console, after inheritance, interpolation, and profiles are applied.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ============================================ -->
<!--                                              -->
<!-- Generated by Maven Help Plugin on 2015-04-09T11:41:51 -->
<!-- See: http://maven.apache.org/plugins/maven-help-plugin/ -->
<!--                                              -->
<!-- ============================================-->

<!-- ============================================-->
<!--                                              -->
```

```
<!-- Effective POM for project                                              -->
<!-- 'com.companyname.project-group:project-name:jar:1.0'                   -->
<!--                                                                        -->
<!-- ====================================================== -->


<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
h
ttp://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.companyname.project-group</groupId>
  <artifactId>project</artifactId>
  <version>1.0</version>
  <build>
    <sourceDirectory>C:\MVN\project\src\main\java</sourceDirectory>
    <scriptSourceDirectory>src/main/scripts</scriptSourceDirectory>
    <testSourceDirectory>C:\MVN\project\src\test\java</testSourceDirectory>
    <outputDirectory>C:\MVN\project\target\classes</outputDirectory>
    <testOutputDirectory>C:\MVN\project\target\test-
classes</testOutputDirectory>
    <resources>
      <resource>
        <mergeId>resource-0</mergeId>
        <directory>C:\MVN\project\src\main\resources</directory>
      </resource>
    </resources>
    <testResources>
      <testResource>
        <mergeId>resource-1</mergeId>
        <directory>C:\MVN\project\src\test\resources</directory>
      </testResource>
    </testResources>
    <directory>C:\MVN\project\target</directory>
    <finalName>project-1.0</finalName>
    <pluginManagement>
      <plugins>
```

```
        <plugin>
          <artifactId>maven-antrun-plugin</artifactId>
          <version>1.3</version>
        </plugin>
        <plugin>
          <artifactId>maven-assembly-plugin</artifactId>
          <version>2.2-beta-2</version>
        </plugin>
        <plugin>
          <artifactId>maven-clean-plugin</artifactId>
          <version>2.2</version>
        </plugin>
        <plugin>
          <artifactId>maven-compiler-plugin</artifactId>
          <version>2.0.2</version>
        </plugin>
        <plugin>
          <artifactId>maven-dependency-plugin</artifactId>
          <version>2.0</version>
        </plugin>
        <plugin>
          <artifactId>maven-deploy-plugin</artifactId>
          <version>2.4</version>
        </plugin>
        <plugin>
          <artifactId>maven-ear-plugin</artifactId>
          <version>2.3.1</version>
        </plugin>
        <plugin>
          <artifactId>maven-ejb-plugin</artifactId>
          <version>2.1</version>
        </plugin>
        <plugin>
          <artifactId>maven-install-plugin</artifactId>
          <version>2.2</version>
        </plugin>
```

```
        <plugin>
          <artifactId>maven-jar-plugin</artifactId>
          <version>2.2</version>
        </plugin>
        <plugin>
          <artifactId>maven-javadoc-plugin</artifactId>
          <version>2.5</version>
        </plugin>
        <plugin>
          <artifactId>maven-plugin-plugin</artifactId>
          <version>2.4.3</version>
        </plugin>
        <plugin>
          <artifactId>maven-rar-plugin</artifactId>
          <version>2.2</version>
        </plugin>
        <plugin>
          <artifactId>maven-release-plugin</artifactId>
          <version>2.0-beta-8</version>
        </plugin>
        <plugin>
          <artifactId>maven-resources-plugin</artifactId>
          <version>2.3</version>
        </plugin>
        <plugin>
          <artifactId>maven-site-plugin</artifactId>
          <version>2.0-beta-7</version>
        </plugin>
        <plugin>
          <artifactId>maven-source-plugin</artifactId>
          <version>2.0.4</version>
        </plugin>
        <plugin>
          <artifactId>maven-surefire-plugin</artifactId>
          <version>2.4.3</version>
        </plugin>
```

```
        <plugin>
          <artifactId>maven-war-plugin</artifactId>
          <version>2.1-alpha-2</version>
        </plugin>
      </plugins>
    </pluginManagement>
    <plugins>
      <plugin>
        <artifactId>maven-help-plugin</artifactId>
        <version>2.1.1</version>
      </plugin>
    </plugins>
  </build>
  <repositories>
    <repository>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
      <id>central</id>
      <name>Maven Repository Switchboard</name>
      <url>http://repo1.maven.org/maven2</url>
    </repository>
  </repositories>
  <pluginRepositories>
    <pluginRepository>
      <releases>
        <updatePolicy>never</updatePolicy>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
      <id>central</id>
      <name>Maven Plugin Repository</name>
      <url>http://repo1.maven.org/maven2</url>
    </pluginRepository>
  </pluginRepositories>
```

```
  <reporting>
    <outputDirectory>C:\MVN\project\target/site</outputDirectory>
  </reporting>
</project>
```

In above pom.xml, you can see the default project source folders structure, output directory, plug-ins required, repositories, reporting directory, which Maven will be using while executing the desired goals.

Maven pom.xml is also not required to be written manually. Maven provides numerous archetype plugins to create projects, which in order, create the project structure and pom.xml

4. APACHE MAVEN — BUILD LIFE CYCLE

What is Build Lifecycle?

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

Phase	Handles	Description
prepare-resources	resource copying	Resource copying can be customized in this phase.
validate	Validating the information	Validates if the project is correct and if all necessary information is available.
compile	compilation	Source code compilation is done in this phase.
Test	Testing	Tests the compiled source code suitable for testing framework.
package	packaging	This phase creates the JAR/WAR package as mentioned in the packaging in POM.xml.
install	installation	This phase installs the package in local/remote maven repository.
Deploy	Deploying	Copies the final package to the remote repository.

There are always **pre** and **post** phases to register **goals,** which must run prior to, or after a particular phase.

When Maven starts building a project, it steps through a defined sequence of phases and executes goals, which are registered with each phase.

Maven has the following three standard lifecycles:

- clean
- default(or build)

- site

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.

The order of execution depends on the order in which the goal(s) and the build phase(s) are invoked. For example, consider the command below. The **clean** and **package** arguments are build phases while the **dependency:copy-dependencies** is a goal.

```
mvn clean dependency:copy-dependencies package
```

Here the *clean* phase will be executed first, followed by the **dependency:copy-dependencies goal**, and finally *package* phase will be executed.

Clean Lifecycle

When we execute *mvn post-clean* command, Maven invokes the clean lifecycle consisting of the following phases.

- pre-clean
- clean
- post-clean

Maven clean goal (clean:clean) is bound to the *clean* phase in the clean lifecycle. Its **clean:cleangoal** deletes the output of a build by deleting the build directory. Thus, when *mvn clean* command executes, Maven deletes the build directory.

We can customize this behavior by mentioning goals in any of the above phases of clean life cycle.

In the following example, We'll attach maven-antrun-plugin:run goal to the pre-clean, clean, and post-clean phases. This will allow us to echo text messages displaying the phases of the clean lifecycle.

We've created 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.pre-clean</id>
         <phase>pre-clean</phase>
         <goals>
            <goal>run</goal>
         </goals>
         <configuration>
            <tasks>
               <echo>pre-clean phase</echo>
            </tasks>
         </configuration>
      </execution>
      <execution>
         <id>id.clean</id>
         <phase>clean</phase>
         <goals>
          <goal>run</goal>
         </goals>
         <configuration>
            <tasks>
               <echo>clean phase</echo>
            </tasks>
         </configuration>
      </execution>
      <execution>
         <id>id.post-clean</id>
         <phase>post-clean</phase>
```

```
         <goals>
            <goal>run</goal>
         </goals>
         <configuration>
            <tasks>
               <echo>post-clean phase</echo>
            </tasks>
         </configuration>
      </execution>
   </executions>
   </plugin>
</plugins>
</build>
</project>
```

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

```
C:\MVN\project>mvn post-clean
```

Maven will start processing and displaying all the phases 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] [antrun:run {execution: id.pre-clean}]
[INFO] Executing tasks
     [echo] pre-clean phase
[INFO] Executed tasks
[INFO] [clean:clean {execution: default-clean}]
[INFO] [antrun:run {execution: id.clean}]
[INFO] Executing tasks
     [echo] clean phase
[INFO] Executed tasks
[INFO] [antrun:run {execution: id.post-clean}]
[INFO] Executing tasks
```

```
     [echo] post-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] ------------------------------------------------------------------------
```

You can try tuning **mvn clean** command, which will display **pre-clean** and **clean.** Nothing will be executed for **post-clean** phase.

End of ebook preview

If you liked what you saw…

Buy it from our store @ **https://store.tutorialspoint.com**