Maven

Table Of Contents

[1. Installation 3](#_Toc517781818)

[2. Introduction 3](#_Toc517781819)

[2.1. Creating project Directory Structure 3](#_Toc517781820)

[2.2. Compiling the Project with MAVEN 4](#_Toc517781821)

[2.3. Packaging the Project with MAVEN 4](#_Toc517781822)

[3. Understanding archetype and pom.xml 4](#_Toc517781823)

[4. Build Lifecycle 5](#_Toc517781824)

[5. Adding Dependency 5](#_Toc517781825)

*Maven*

# Installation

* Download the latest Maven version.
* Set the M2\_HOME to “apache-maven-3.5.4”.
* Set the path: apache-maven-3.5.4\bin

Validate the successful installation through the command: ***mvn –version***

# Introduction

The following are the list of features provided by maven.

1. Managing Multiple JARS

Since every project incorporates many JARS to provide various functionality, MAVEN ensures that all the required JARS are properly in place at all times.

1. Dependencies between JARS and Compatible version
2. Project Directory Structure
3. Building, Publishing and Deploying.

## Creating project Directory Structure

One among the feature of MAVEN is that it supports the creation of a project directory structure with the help of maven repository which contains the same structure for various applications like WEB, spring boot, MVC.

The following command generates the folder structure: ***mvn archetype:generate***

Overview on the command: The execution of the above mentioned command first downloads all the sample project structure from the repository to the local machine upon completion of which it prompts the user to enter a number.

Every number corresponds to an archetype (which is the boiler plate template for a given project like WEB Project), based on the requirement the user may opt a suitable archetype number and the commands resumes the execution and generates all the required folder structure as requested. The number 106 is a default value for generating a standard project.

## Compiling the Project with MAVEN

Through maven it is possible to compile all the java classes that are contained in the project during the course of which MAVEN scans the pom.xml file validate if all the dependencies are satisfied and downloads all the dependent JARS.

**Command**: mvn complie

**Path**: {Root\_Folder}/

## Packaging the Project with MAVEN

When the packaging command is executed, MAVEN looks up the pom.xml file to determine the format to which the project has to be packaged. By default the option is “JAR” format.

**Path**: {Root\_Folder}/

**Command**: mvn package

The maven repository is composed of two main entities. Archetype Info and Dependency info.

1. *Archetype Info*: Defines the project structure.
2. *Dependency Info*: Speaks about the list of JARS and dependent JARS that required for any given project.

The term group id refers to the conventional package name and artefact id refers to the project name.

# Understanding archetype and pom.xml

The archetype defines the type of any given project. Hence while creating the project for the first time rather than programmer having to create the folder structure, we can make maven create the folder structure for us.

The command “**mvn archetype:generate**” initiates the creation process during which it would list all the available archetype and prompt user to select one archetype.

Whenever a project is created through maven the same is stored in the local repository implicitly and at the programmers will it is stored in maven repository.

In a scenario where user 1 creates a project A and does not wish to store the project in maven repository and days later he creates project B which requires some feature of project A even though the project is not available in the central repository the user will be able to add the JAR to his current project by adding the dependency tag with all the relevant detail which would extract the project from the local repository.

# Build Lifecycle

In general following are the stages that compose the build life cycle for given maven application.

1. validate – Ensure that all the required components are in place.
2. compile – Convert java to class files
3. test phase – Executes any embedded JUNIT test cases if available.
4. package phase - The outcome on the formatting can be a JAR or any format as mentioned in the pom.xml file.
5. verify – validate the result of the testing phase to make sure the quality criteria is met.
6. install phase – install the package in the local repository, for the use as a dependency in other projects.
7. deploy – done in the build environment, copies the final package to the central repository so as to make the packaging available for other developers.

If the maven command “***mvn package***” is executed by default all the predecessor commands get executed automatically.

Clean phase – this ensures that all the java files are compiled from scratch.

# Adding Dependency

In the event a new JAR needs to be added to the project, the changes has to go into the pom.xml file in the form of a dependency providing the co-ordinates (artefact id, group id, version) on the required jar following which when the command “***mvn compile***” is executed, maven downloads the required JAR from the maven repository and add it to the build path.

While we have provided all the tags for a new JAR, in the absence of scope tag maven considers the value to be “*compile*” by default.

The dependency information for any given JAR can be identified by using the “Maven repository search” option.

# Using Jetty

Maven has the ability to tag the container as a plugin to an application. Example: tomczat, jetty. And it is also possible to configure maven to constantly refresh deploy class changes through the configuration in the pom.xml file.

<configuration>

<scanIntervalSeconds>5<scanIntervalSeconds>

</configuration>

Note: while importing a project generated through maven command line execution into eclipse we may have to create a variable in the property of the project as “M2\_REPO” and initialize the “.m2/repository” path to it.