**Maven**

* 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.

Maven provides developers ways to manage the following −

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

## **===========================================================**

## **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.

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

## **Convention over Configuration**

Maven uses **convention over configuration**, which means developers are not required to create build process themselves.

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 |

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

## **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.

# ==============================================**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

## 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**

|  |  |
| --- | --- |
| Sr.No. | Node & Description |
| 1 | **Project root**  This is project root tag. You need to specify the basic schema settings such as apache schema and w3.org specification. |
| 2 | **Model version**  Model version should be 4.0.0. |
| 3 | **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. |
| 4 | **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. |
| 5 | **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.

**mvn help:effective-pom**

===============================================================**Maven - Build Life Cycle**

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.

**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

**Goal**

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

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

## **Default (or Build) Lifecycle**

This is the primary life cycle of Maven and is used to build the application. It has the following 23 phases

|  |  |
| --- | --- |
| Sr.No. | Lifecycle Phase & Description |
| 1 | **validate**  Validates whether project is correct and all necessary information is available to complete the build process. |
| 2 | **initialize**  Initializes build state, for example set properties. |
| 3 | **generate-sources**  Generate any source code to be included in compilation phase. |
| 4 | **process-sources**  Process the source code, for example, filter any value. |
| 5 | **generate-resources**  Generate resources to be included in the package. |
| 6 | **process-resources**  Copy and process the resources into the destination directory, ready for packaging phase. |
| 7 | **compile**  Compile the source code of the project. |
| 8 | **process-classes**  Post-process the generated files from compilation, for example to do bytecode enhancement/optimization on Java classes. |
| 9 | **generate-test-sources**  Generate any test source code to be included in compilation phase. |
| 10 | **process-test-sources**  Process the test source code, for example, filter any values. |
| 11 | **test-compile**  Compile the test source code into the test destination directory. |
| 12 | **process-test-classes**  Process the generated files from test code file compilation. |
| 13 | **test**  Run tests using a suitable unit testing framework (Junit is one). |
| 14 | **prepare-package**  Perform any operations necessary to prepare a package before the actual packaging. |
| 15 | **package**  Take the compiled code and package it in its distributable format, such as a JAR, WAR, or EAR file. |
| 16 | **pre-integration-test**  Perform actions required before integration tests are executed. For example, setting up the required environment. |
| 17 | **integration-test**  Process and deploy the package if necessary into an environment where integration tests can be run. |
| 18 | **post-integration-test**  Perform actions required after integration tests have been executed. For example, cleaning up the environment. |
| 19 | **verify**  Run any check-ups to verify the package is valid and meets quality criteria. |
| 20 | **install**  Install the package into the local repository, which can be used as a dependency in other projects locally. |
| 21 | **deploy**  Copies the final package to the remote repository for sharing with other developers and projects. |

There are few important concepts related to Maven Lifecycles, which are worth to mention −

* When a phase is called via Maven command, for example **mvn compile**, only phases up to and including that phase will execute.
* Different maven goals will be bound to different phases of Maven lifecycle depending upon the type of packaging (JAR / WAR / EAR).