**What Is Maven**

Maven is a build and project management tool that is generally used in frameworks built in Java. It is developed by Apache Software Foundation. Maven, a word from the Yiddish language, means ‘gatherer of knowledge’. It was introduced to make the process of triggering build in Jakarta Turbine Project.

Maven is controlled by the Project Object Model (pom) file. While working with frameworks built-in Java, we often have to deal with a number of dependencies.

Before Maven came into the picture, all dependencies, that are nothing but JAR files had to be added to our framework manually. Also, we needed to take care of the software upgrade in our project.

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/05/c-users-ghs6kor-pictures-saved-pictures-maven3-pn.png)

Maven ensures that project JARs and libraries are downloaded automatically. Only the information relating to the versions of the software and type of dependencies need to be described in the pom .xml file.

Maven can take care of projects in Ruby, C#, and other languages. It takes up the task of building projects, their dependencies, and documentation.

ANT, another tool developed by Apache Software Foundation, is also used for building and deployment of projects. But Maven is more advanced than ANT. Like ANT, Maven has made the process of build simple. Thus, in short, Maven has made the life of developers easy.

**Why Use Maven**

**Maven performs the below activities:**

* Repository to get the dependencies.
* Having a similar folder structure across the organization.
* Integration with Continuous Integration tools like Jenkins.
* Plugins for test execution.
* It provides information on how the software/ project is being developed.
* The build process is made simpler and consistent.
* Provides guidelines for the best practices to be followed in the project.
* Enhances project performance.
* Easy to move to new attributes of Maven.
* Integration with version control tools like Git.

Maven takes care of processes like releases, distribution, reporting, builds, documentation, and SCMs. Maven connects to the Maven Central repository and loads them in local. Some of the IDEs that support project development with Maven are NetBeans, Eclipse, IntelliJ, and so on.

**Maven should be used in our projects in the following scenarios:**

* If the project requires a large number of dependencies.
* If the version of the dependencies needs frequent up-gradation.
* If the project needs to have quick documentation, compiling and packaging of the source code into JAR or ZIP files.

**Operational Steps Of Maven**

**#1)** First Maven goes through the POM .xml file.

**#2)** The dependencies are loaded into the local repository.

**#3)** Goes through the built-in life cycles of Maven as shown below:

* **Default:** Takes care of the deployment of the project.
* **Clean:** Removes any errors, thereby cleans the project, and removes the artifact produced from the previous process of the build.
* **Site:**Takes care of the documentation of the project.

**#4)** Each built-in cycles have several phases. **For example,** default has twenty-three phases while clean and site has three and four phases respectively.

**#5)** Each Maven cycle goes through several stages where a particular stage has a specific objective.

**Some of the important phases are listed below:**

* **Validate:**Verifies if all the prerequisites data for the build to complete are available.
* **Compile:**Source code is compiled.
* **Test- compile:**Test source code is compiled.
* **Test:**Unit test cases are executed.
* **Package:** Source code is compiled and packaged into JAR or ZIP files.
* **Integration- test:**Package is deployed and if there are any issues, integration test cases are executed.
* **Install-Package:** It is installed in the local repository.
* **Deploy:**A copy of the package is made available from the remote repository.

These phases have to be executed in order. Also, if the **deploy** phase i.e. the end phase of the Maven cycle has to be executed then all the prior phases to that cycle have to be completed successfully.

**From the command prompt, the phases are run in the following way:**

**mvn <name of phase>**, **for example,** mvn validate

**#6)** A group of Maven goals makes up a phase. Just like phases of Maven, each goal has to be run in a specific order.

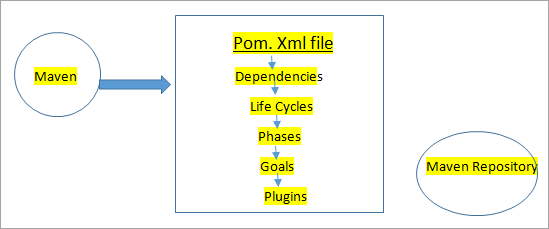
A goal has the following syntax:

**Plugin: goal**

**Here, we have discussed some of the phases along with the goals tied to them:**

* compiler: compile ( used in the phase of compilation)
* compiler: test ( used in the phase of test compilation )
* surefire: test ( used in the phase of testing )
* install: install ( used in the phase of installation )
* jar: war ( used in the phase of packaging )
* war: war ( used in the phase of packaging )

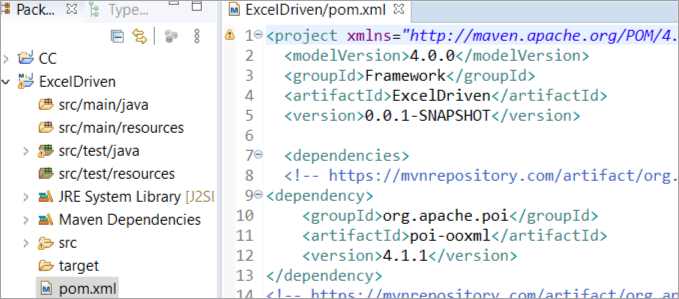
**The operational steps of how Maven works is diagrammatically shown below:**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/05/operational-steps-of-how-Maven-works1.png)

**Terminologies In Maven**

* **POM**: It stands for Project Object Model. It is an XML file that has the information about the project, the dependencies present in the project, directory of the source file, plugin information, and so on. These are the necessary data for Maven to completely build the project. Maven reads the pom file to get all this information.

**Maven project in Eclipse IDE with a code snippet from the POM XML file is shown below.**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/05/Maven-Tutorial-Series-Part-1-2.png)

* **GroupId**: Recognizes our project uniquely from all the projects. GroupId is a part of the pom file. It is often said as an identity for the group of projects.
* **ArtifactId**: A jar file that is deployed to the Maven repository. ArtifactId is a part of the pom file. It is often said as an identity and name of our project.
* **Version**: Specifies the version of the jar of the project. Version is also a part of the pom file.

As depicted in the image above, we can see that <groupId> , <artifactId> and <version> tags form the part of the dependencies defined for the project.

* **Maven Central Repository**: This the placeholder where jars, libraries, plugins, and configuration data required by Maven for building the project are present.

If we want to add Selenium dependencies in our project, then we need to have the below code snippet to the pom file.

<!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java -->

<dependency>

<groupId> org.seleniumhq.selenium </groupId>

<artifactId> selenium-java </artifactId>

<version> 4.0.0-alpha-5 </version>

</dependency>

Out of the multiple projects hoisted in the Maven central repository, it identifies Selenium with the help of the groupId. Similarly, ArtifactId is like a subproject under the groupId.

**For example,** Selenium code is supported by languages like Java, C#, and so on. Here, from the artifactId, it is evident that we are only connecting to the JARs associated with Java. Also, the version of the JAR to be associated to the project is also a part of the dependencies.

We can also publish our own project in the Maven repository by providing these three pieces of information.

**Maven Repository**

**Maven Repository can be of three types:**

1. Local Repository
2. Remote Repository
3. Central Repository

Once Maven reads the dependencies from the POM file, it first searches them in the local repository, then to the central, and finally to the remote repository. If the dependencies are not found in any of the three repositories, then the user is notified with an error and the process is stopped.

**#1) Maven Local Repository**

The local repository is located in our local system – mostly in **.m2** (C:/Users /superdev /.m2) directory which shows its presence once Maven is installed in our system and we have been able to successfully execute a Maven command.

It is also possible to modify this location in settings.xml (MAVEN\_HOME /conf /settings.xml) with the help of the **localRepository** tag.

**Below xml code snippet, shows how to change the locations of the local repository:**

<settings>

<localRepository>

C: \Maven \m2

</localRepository>

</settings

**#2) Maven Central Repository**

[Central repository](https://repo.maven.apache.org/maven2/) is developed by the Apache Maven group and is hosted on the web. This is considered as the central repository and it has all the common libraries. Like a local repository, we can also modify the location where they are to be downloaded by default by changing the setting.xml.

**#3) Maven Remote Repository**

Remote repository is also hosted on the web. In some scenarios, a company can develop its own remote repository and perform deployments on their private projects. These will be owned by that specific company and can be operated only inside it.

The remote repository has similar working patterns like a central repository. Whenever any dependencies or configurations are required from these repositories, they shall first be downloaded into our local and then used.

**A sample xml code for remote repository with the id and url is shown below.**

  <repositories>

<repository>

<id>com.src.repository</id>

<url>http://maven.comp.com/maven2/</url>

</repository>

</repositories>

search

# Maven Build Lifecycle, Phases, and Goals

[1 Comments](https://www.journaldev.com/33659/maven-build-lifecycle-phases-goals#comments)

Filed Under: [Maven](https://www.journaldev.com/maven)

* [Home](https://www.journaldev.com/) » [Maven](https://www.journaldev.com/maven) » Maven Build Lifecycle, Phases, and Goals

## What is Maven Build Lifecycle?

When we build a Maven project, it executes a set of clearly defined tasks based on the project pom.xml configuration and the command-line options. This standard set of tasks creates the maven build lifecycle.

The benefit of a clearly defined lifestyle is that we have to remember only a few sets of commands to compile, build, install, and deploy our projects.

**Recommended Reading**: [20+ Maven Commands and Options (Cheat Sheet)](https://www.journaldev.com/33645/maven-commands-options-cheat-sheet)

## Built-in Build Lifecycles

There are three built-in build lifecycles.

1. **default**: handles project build and deployment
2. **clean**: handles project cleaning
3. **site**: handles the creation of project site documentation

## Maven Build Phases

Maven build lifecycle goes through a set of stages, they are called build phases. For example, the default lifecycle is made up of the following phases.

* validate
* compile
* test
* package
* verify
* install
* deploy

The build phases are executed sequentially. When we run a maven build command, we specify the phase to be executed. Any maven build phases that come before the specified phase is also executed. For example, if we run mvn package then it will execute validate, compile, test, and package phases of the project.

## Maven Build Goals

A build phase is made up of a set of goals. Maven goals represent a specific task that contributes to the building and managing of a project.

Sometimes, a maven goal is not bound to a build phase. We can execute these goals through the command line.

The syntax to execute a goal is:

$ mvn plugin-prefix:goal

$ mvn plugin-group-id:plugin-artifact-id[:plugin-version]:goal

Here is an example to execute the dependency tree goal from the command line. It’s not part of any build phases.

mvn dependency:tree

## Role of POM in Build Lifecycle

The maven builds executed through the command line runs a set of phases and goals. The pom.xml configuration plays a major role in setting up the project build lifecycle.

The packaging value of pom.xml file defines the set of goals to be executed by the maven build. For example, if it’s jar then the following phases and goals will be executed.

|  |  |
| --- | --- |
| Phase | Goal |
| process-resources | resources:resources |
| compile | compiler:compile |
| process-test-resources | resources:testResources |
| test-compile | compiler:testCompile |
| test | surefire:test |
| package | jar:jar |
| install | install:install |
| deploy | deploy:deploy |

We can also configure goals in the pom.xml file using the plugins element. This is mostly required when you have created a custom plugin and want to execute any specific goal for a