**Apache maven**

## [What is Apache Maven?](https://www.vogella.com/tutorials/ApacheMaven/article.html#maven_overview)

[Apache Maven](http://maven.apache.org/) is an powerful build tool primary for Java software projects. It is implemented in Java which makes it platform-independent.

Build tools performs the compilation, testing, packaging and deployment of the final product. The advantages of performing these steps automatically instead of a human performing these steps is that the build is reproducible, faster and less error prone.

Maven can also perform related activities, e.g., create web sites, upload build results or generate reports.

Maven also supports the creation of the initial folder structure for applications to make it easy to get started.

**The key features of Maven are:**

* Convention over configuration: avoid as much configuration as possible, by choosing real world default values and supplying project templates (archtypes).
* Dependency management: support the definition and usage of dependencies to other libraries and projects. During the build, the Maven build system resolves the dependencies and it also builds the dependent projects if needed.
* Extensible via plug-ins: The Maven build system is extensible via plug-ins, which allows to keep the Maven core small. The Maven core does for example not know how to compile Java source code, this is handled by the compiler plug-in.

### [3.4. Proxy settings](https://www.vogella.com/tutorials/ApacheMaven/article.html#proxy-settings)

To configure you proxy, add a file named .m2/settings.xml to your user home with the proxy settings, similar to the following:

**<settings>**

**<proxies>**

**<proxy>**

**<id>**example-proxy**</id>**

**<active>**true**</active>**

**<protocol>**http**</protocol>**

**<host>**proxy.example.com**</host>**

**<port>**8080**</port>**

**<username>**proxyuser**</username>**

**<password>**somepassword**</password>**

**<nonProxyHosts>**www.google.com|\*.example.com**</nonProxyHosts>**

**</proxy>**

**</proxies>**

**</settings>**

User and password is only required if you proxy request base authentication.

### [Project identifiers via the group, artifact and version (GAV)](https://www.vogella.com/tutorials/ApacheMaven/article.html#maven_configuration_coordinates)

A Maven project uses the groupId, artifactId, version (also knows as GAV) and the packaging property to uniquely identify a Maven component. These attributes are explained in the following table.

| *Table 1. Coordinate attributes* | |
| --- | --- |
| **Name** | **Description** |
| groupId | Defines a unique base name of the organization or group that created the project. This is normally a reverse domain name or the name of an open source project. For the generation of new projects, the groupId also defines the package of the main class. |
| artifactId | Defines the unique name of the project within the groupId. If you generate a new project via Maven this is also used as root folder for the project. |
| version | This defines the version of the project. If a new version of the project is build, this version should change so that consumers can see that a different version is used. |
| packaging | Defines the packaging method. This could be e.g. a jar, war or ear file. If the packaging type is pom, Maven does not create anything for this project, it is just meta-data. |

For example, you can use the GAV to specific which exact version of a library you want ot use.

The full Maven coordinates are often written in the following format: groupId:artifactId:packaging:version

#### [Excluding transitive dependencies](https://www.vogella.com/tutorials/ApacheMaven/article.html#excluding-transitive-dependencies)

Sometimes your dependencies have conflicting transitive dependencies. For example library A requires library C in version 1.0.0 and library B requires it in version 1.1.0. To use library C in version 1.0.0 you can exclude C from the dependencies of A.

Lets for example B is the com.vogella.example artifactId.

**<dependency>**

**<groupId>**com.vogella.group**</groupId>**

**<artifactId>**com.vogella.example**</artifactId>**

**<version>**1.0**</version>**

**<scope>**compile , test**</scope>**

**<exclusions>**

**<exclusion>**

**<groupId>**c.groupId**</groupId>**

**<artifactId>**c.artifactId**</artifactId>**

**</exclusion>**

**</exclusions>**

**</dependency>**

## [Maven plug-ins, goals and the life cycle](https://www.vogella.com/tutorials/ApacheMaven/article.html#maven_coreconcepts)

### [5.1. Maven life cycle](https://www.vogella.com/tutorials/ApacheMaven/article.html#maven_lifecycle)

Maven builds follows a specified life cycle. The build life cycle is divided into build phases The build phases are divided into build goals.

Maven has three default life cycle phases:

* default (cannot be executed)
* clean
* site

You can execute a full life cycle via command like clean or site. You can also execute build phases like verify or single goals like dependency:copy-dependencies.

The following lists gives an overview of the important Maven life cycle phases:

| **Build phase** | **Description** |
| --- | --- |
| validate | checks if the project is correct and all information is available |
| compile | compiles the source code into binary artifacts |
| test | executes the tests |
| package | takes the compiled code and package it, for example into a JAR file |
| integration-test | takes the packaged result and executes additional tests, which require the packaging |
| verify | performs checks if the package is valid |
| install | install the result of the package phase into the local Maven repository |
| deploy | deploys the package to a target, i.e. remote repository |
|  |  |

For a complete list of the Maven phases see <http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html>.

If you execute a build phase, all build phases before that build phase are executed in the above sequence. All relevant goals are executed during this process.

A goal is relevant for a phase, if the Maven plug-in or the pom binds this goal to the corresponding life cycle phase.

### [5.2. Plugins and goals](https://www.vogella.com/tutorials/ApacheMaven/article.html#plugins-and-goals)

Maven plugins provide the functionalities which are used in the Maven build.

Each Maven plugins provides one or more goals. A goal is a “unit of work” in Maven. It is possible to execute goals independently or as part of a larger chain of goals.

Goals can define parameters, which may have default values. The goals are executed based on the information found in the pom of the project, e.g., the compiler:compile goal checks the pom for relevant parameters. Goals can be attached to a life cycle phase.

### [5.3. Packages and goal bindings](https://www.vogella.com/tutorials/ApacheMaven/article.html#maven_packagesgoalbindings)

Each packaging contains a list of bindings for goals to a particular life cycle phase. For example, the jar packaging binds the following goals to the life cycle phases.

| *Table 2. Packaging goal* | |
| --- | --- |
| **Life cycle phase** | **Goal binding** |
| 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 |

### [5.4. Adding goals to life cycle phases](https://www.vogella.com/tutorials/ApacheMaven/article.html#maven_packagesgoalbindings2)

You can add goals to life cycle phases by configuring more Maven plug-ins and adding them to a life cycle in your pom file. You need to specify which goal should be executed. If the plug-in does not specify the default life cycle it should run, you must also specify the life cycle phase it should run.

**<plugin>**

**<groupId>**com.vogella.example**</groupId>**

**<artifactId>**vogella-some-maven-plugin**</artifactId>**

**<version>**1.0**</version>**

**<executions>**

**<execution>**

**<phase>**verify**</phase>**

**<goals>**

**<goal>**checklinks**</goal>**

**</goals>**

**</execution>**

**</executions>**

**</plugin>**

## [Maven and version control systems](https://www.vogella.com/tutorials/ApacheMaven/article.html#maven_vcs)

Maven generates its output into the target folder of each project. This build output should not get included into your version control system.

Add this directory to your ignore resources. For example, if you use Git as version control system, add the "target/" entry to your .gitignore file in the root of each project.

### [Review generated project](https://www.vogella.com/tutorials/ApacheMaven/article.html#exercise_maven_review)

Validate that Maven generated a project on your file system similar to the following structure.

*.*

└── com.vogella.maven.first

├── pom.xml

└── src

├── main

│   └── java

│   └── com

│   └── vogella

│   └── maven

│   └── first

│   └── App.java

└── test

└── java

└── com

└── vogella

└── maven

└── first

└── AppTest.java

Maven created a App.java class in the ./src/main/ folder, which is just a simple "Hello World" program. It also created an example test class in ./src/test/. In the root folder there is a pom.xml file.

### [Compile your sources](https://www.vogella.com/tutorials/ApacheMaven/article.html#exercise_maven_compile)

Now you want to compile your Java sources. For this switch on the command line into our projects root directory and trigger the following Maven command.

mvn compile

### [Create a JAR file](https://www.vogella.com/tutorials/ApacheMaven/article.html#exercise_maven_package)

Now you want to create an executable JAR file out of our project. The package goal creates a deployable JAR file.

To ensure previous build artifacts are removed, you can use the clean goal.

mvn clean package

### [Validate](https://www.vogella.com/tutorials/ApacheMaven/article.html#validate)

Afterwards you can run the packed program.

java -cp target/com.vogella.maven.first-1.0-SNAPSHOT.jar com.vogella.maven.first.App

The output should be Hello World! on the command line.

### [9.6. Running the test](https://www.vogella.com/tutorials/ApacheMaven/article.html#running-the-test)

Instead of running a full build with packaging, it is also possible to just run the test phases of the Maven life cycle.

mvn test

### [Remove all build results / Clean the project](https://www.vogella.com/tutorials/ApacheMaven/article.html#remove-all-build-results-clean-the-project)

To remove all build artifacts (tge ./target/ folder), use the clean command.

mvn clean

**Q #1) What do you mean by Maven?**

**Answer**: Maven is a project management tool (introduced by Apache Software Foundation) that provides an entire framework for the build cycle. It is open-source and is mainly used for the project developed in Java.

Maven is driven by a project object model popularly known as POM. It is the central repository for all the dependencies. It maintains the same folder convention across organizations and can be easily integrated with continuous integration tools like Jenkins.

Besides, it takes care of the reporting and documentation of the project. It is developed in Java language.

**Q #2) What are the features or advantages of Maven?**

**Answer**:

**The features or advantages of Maven are as follows:**

* Not required to manually add jars for the project. The updates to the project dependencies and transitive dependencies are carried out automatically by Maven.
* Maven maintains a uniform directory structure across the organization.
* Both the deployment and build activities are taken care by Maven.
* Maven is simple, easy to grasp, to set up and utilize in the projects as it is driven by the POM file.
* Rapidly expanding repository of Maven contains a large number of libraries that can be used for multiple projects at a time.
* Maven encourages the use of extensible code design.

**Q #3) Name the aspects that are managed by Maven.**

**Answer**: The aspects that are managed by Maven are documentation, builds, SCMs, releases, distribution, mailing lists, dependencies, and reporting.

**Q #4) What is meant by Maven build lifecycle? Name these cycles.**

**Answer**: A Maven build life cycle comprises of certain phases. It determines the order in which the Maven goals need to be performed. If a particular phase is to be accomplished, then all the previous phases prior to that particular phase need to be run successfully in sequence.

**Maven build life cycles are listed below as the followings:**

1. **Clean**: Removes the artifact produced from the preceding build processes.
2. **Default**: Deployment and build process is taken care of.
3. **Site**: Project documentation is carried out in this cycle.

**Q #5) What is a build tool responsible for?**

**Answer**: Build tools are primarily responsible for producing the source code (provided the auto-generated code is utilized). It creates project documentation from the source code.

It compiles and packages the code in the form of JAR/ZIP file. Finally, places that code in the local, central, or remote repository.

**Q #6) What do you mean by POM and what does it contain?**

**Answer:**Project Object Model or pom forms the elementary part of the working of Maven. While performing a Maven goal or a task, the pom file residing in the present directory is searched and referred to collect information on the project.

It is basically an xml file. It contains configuration details like plugins, goals, developers, dependencies, profiles, versions, and mailing lists.

**For example**, to add excel dependencies in Selenium, we need to add the dependencies pasted below.

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/05/pom.png)

**Q #7)** **What do you mean by a Maven Repository?**

**Answer:**Maven repository is the location of a directory where all the related project artifacts, jars, libraries, plugins are kept and can be utilized by Maven easily.

**Q #8) Explain the various types of Maven Repositories.**

**Answer: There are three types of Maven repositories.**

**They are:**

* **Local Repository:**This is placed in our local machine generally in the **.m2** directory. This is generated once we are able to execute a Maven command successfully. All the project dependencies reside here. Once Maven scans the pom file, it first searches for its dependencies in the local repository.
* **Central Repository:**This is supplied by Apache Maven. It contains most of the routinely used libraries. Once any of the dependencies are not found in the local repository, then Maven connects to a central repository.
* **Remote Repository:**Sometimes companies develop their own custom repository comprising of their project artifacts, jars, and libraries. This is a type of repository that remains private for use inside that organization.

**Q #9) Explain Maven Artifact.**

**Answer:**Maven artifact is known as a file (generally a jar) that is extended to the Maven repository. Maven build generates multiple artifacts comprising of source jar and compiled jar. **GroupId, artifactId, and version** together form an artifact and all of the three combined solely identifies it.

**The below code snippet shows an example of the Maven artifact.**

<dependency>

<groupId> org.selenium </groupId>

<artifactId> seleniumJava </artifactId>

<version> 4.0.2 </version>

</dependency>

**Q #10) List down the various scopes of Maven Dependency.**

**Answer:**

Dependency scope is used to limit the transitivity of a dependency and to determine when a dependency is included in a classpath.

**There are 6 scopes:**

**compile:**

This is the default scope, used if none is specified. Compile dependencies are available in all classpaths of a project. Furthermore, those dependencies are propagated to dependent projects.

**Provided:**

This is much like compile, but indicates you expect the JDK or a container to provide the dependency at runtime. For example, when building a web application for the Java Enterprise Edition, you would set the dependency on the Servlet API and related Java EE APIs to scope provided because the web container provides those classes. A dependency with this scope is added to the classpath used for compilation and test, but not the runtime classpath. It is not transitive.

**Runtime:**

This scope indicates that the dependency is not required for compilation, but is for execution. Maven includes a dependency with this scope in the runtime and test classpaths, but not the compile classpath.

**test:**

This scope indicates that the dependency is not required for normal use of the application, and is only available for the test compilation and execution phases. This scope is not transitive. Typically this scope is used for test libraries such as JUnit and Mockito. It is also used for non-test libraries such as Apache Commons IO if those libraries are used in unit tests (src/test/java) but not in the model code (src/main/java).

**system**

This scope is similar to provided except that you have to provide the JAR which contains it explicitly. The artifact is always available and is not looked up in a repository.

**import**

This scope is only supported on a dependency of type pom in the <dependencyManagement> section. It indicates the dependency is to be replaced with the effective list of dependencies in the specified POM's <dependencyManagement> section. Since they are replaced, dependencies with a scope of import do not actually participate in limiting the transitivity of a dependency.

**Q #11) What are the differences between Maven and Ant?**

**Answer: The differences between Maven and Ant are listed below:**

| **Maven** | **Ant** |
| --- | --- |
| Maven is descriptive and most of the project information are defined in the pom file. | Ant is in form of procedure or method where all the instructions to be performed are given in order. |
| Maven follows a life cycle. | Ant does not follow any life cycle. |
| Maven is considered as a framework. | Ant is considered as a tool box. |
| Maven is a project management and build tool. | Ant is a tool used only for build process. |
| Maven plugins can be reused. | Ant scripts cannot be reused. |
| Maven follows a protocol for example, Maven projects follow a uniform directory structure across organization. | Ant does not follow any specific protocol. |

**Q #12) What do you mean by Maven Archetype?**

**Answer:**Maven archetype is basically a plugin assigned with the job of generating the project structure according to a particular skeleton or template.

**The following command is used to create a project template:**

**mvn archetype: generate**

**Q #13) How to determine the version of Maven in our system?**

**Answer:**To determine the version of Maven we are using in our system, we need to enter the below command in the console.

**mvn –version**

**Q #14) What is known as SNAPSHOT in Maven?**

**Answer:**Maven snapshot is that version which is still unreleased. Prior to the first release is completed, there is a 1.0-SNAPSHOT version. In the future, this version will emerge to be 1.0 version. It is called as the present development copy. Maven looks for the new SNAPSHOT version in the remote repository.

**Q #15) How to mention profiles in Maven?**

**Answer:**The profiles are mentioned in Maven with the help of a subset of elements present in the pom file.

**Q #16)** **What are Maven Plugins?**

**Answer:**Maven plugins are the basic component of a Maven framework. Each of the plugins has a specific task to be performed.

**Maven generally performs the following functionalities:**

* Generates jar files.
* Generates war files.
* Compiles the code.
* Executes unit testing of code.
* Generates documentation of the project.
* Generates customized reports.

**Maven plugin gives a group of goals that can be run with the following command syntax:**

**mvn [plugin-name]:[goal-name]**

**Q #17) What are the different types of Maven Plugins?**

**Answer:**

**The different types of Maven plugins are listed below:**

* **Building Plugins:**These plugins are used at the time of build and are defined in the building element of the pom file.
* **Reporting Plugins:**These plugins are used at the time of site generation and are defined in the reporting element of the pom file.

**Q #18) What does goal in Maven mean?**

**Answer:**A collection of Maven goals constitute a phase. Each goal is defined for a particular job involved in project management and execution of the build.

**Q #19) Name the build phases in Maven Build Lifecycle.**

**Answer:**The build phases in Maven Build Lifecycle are listed down:

* **Validate:**Checks if all the preconditions information to trigger the build is obtained.
* **Compile:**Project source code is compiled.
* **Test:**The Source code that is compiled is tested with the unit test framework. In this phase, the code is not deployed or packaged.
* **Package:**Source code after compilation is packaged in the form of ZIP or JAR files.
* **Integration- test:**After the package is deployed in an environment, the integration test cases are executed.
* **Verify:**Examines to ensure that the package is correct and it meets all the required quality specifications.
* **Install:**Installation of packages into the local repository.
* **Deploy:**A specimen of the final package is made accessible to the remote repository for distribution among the other developers across projects.

**Q #20) What is the location where Maven dependencies are downloaded?**

**Answer:**The project artifacts, dependencies, and jars downloaded by Maven are placed in the local repository of Maven. The folder **.m2**is by default the location for the local repository. This default location can be changed from the settings.xml file.

**Q #21) Name the phases of the Maven Clean Lifecycle.**

**Answer:**

**The phases of Maven Clean Lifecycle are listed below:**

* pre-clean
* clean
* post-clean

**Q #22) What is the purpose of command mvn clean in Maven?**

**Answer:**mvn clean aims to clean the project artifacts created by the previous Maven builds from the target directories. This is generally executed before initiating a new build process.

**Q #23) What do you mean by Build profiles in Maven?**

**Answer:**Build profile is a collection of configurations that helps to place a value or overrule the default values required for the build process in Maven.

We can design the build process for various environments like development, testing, and production. Profiles are described in the pom files with the help of <profiles> and is usually used to point to different environments.

**Q #24) List down the types of Maven Build profiles.**

**Answer:**

**The types of Maven Build profiles are listed below:**

* **Per-User:** This is described in Maven settings.xml file.
* **Per Project:** This is described in pom.xml of the project.
* **Global**: This is described in the global Maven settings.xml file.

**Q #25) What do you mean by Maven’s External Dependencies?**

**Answer:**Once Maven reads through the pom file, it gets to know the list of dependencies for the project. It searches for these dependencies in Local, Central and Remote repositories. In case any of the dependency not found in any of the repositories, then Maven utilizes the external dependency.

**Q #26) What are the parameters required to define External Dependency in Maven?**

**Answer:**Maven external dependencies are described in the pom xml file just like other dependencies. The parameters required to define the external dependencies are **groupid, artifactId, scope set to system and system path as per the project location** (relative path).

**Q #27) What is meant by Mojo in the terminology of Maven?**

**Answer:**Mojo is referred to as **Maven plain Old Java Object**. A mojo is basically an executable goal to increase the features of Maven. A plugin is a distribution of one or multiple connected MOJOs.

**Q #28) What is the job of the exclusion element in Maven?**

**Answer:**In order to keep out any of the dependencies, the exclusion element is used in Maven.

**Q #29) Why is it recommended to keep the external dependencies in the local repository rather than a remote repository in Maven?**

**Answer:**It is recommended to keep the external dependencies in the local repository instead of remote in Maven because local repository consumes less space, are easily accessible and there is no need to take care of versioning for jars.

**Q #30) How to perform a force update in Maven?**

**Answer:**A forced update in Maven can be done with the following commands:

**mvn clean install –U**

**-U** is used for forcing a Maven update. However, the dependencies that are release based cannot be performed in this manner.

**Q #31) Explain the error “You cannot have two plugin executions with the same ( or missing) elements” in Maven.**

**Answer:**This error message comes in the situations where we have run a single plugin more than one time with the same id. We need to give a unique id for each execution.