









POM stands for project object model and this is the file to provide details for a maven project to execute goals.

What are goals?

Goals are basically the plugins available in maven. The complete list of available plugins can be found at <https://maven.apache.org/plugins/index.html>.

What is below in pom file:

<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">

xmlns defines default namespace, which states, that all nodes within project node and without a namespace-alias will be in http://maven.apache.org/POM/4.0.0 namespace by default.

[**xmlns**](http://www.w3.org/TR/xml-names/#ns-decl) is part of the [W3C Namespaces in XML Recommendation](http://www.w3.org/TR/xml-names/):

The prefix **xmlns** is used only to declare namespace bindings and is by definition bound to the namespace name <http://www.w3.org/2000/xmlns/>.

[**xmlns:xsi**](http://www.w3.org/TR/xmlschema-1/#Instance_Document_Constructions) declares a standard namespace prefix (xsi) for a core namespace used in XSD: http://www.w3.org/2001/XMLSchema-instance

Archetype:

An archetype is a template of a project which is combined with some user input to produce a working Maven project that has been tailored to the user's requirements.

POM example:

1. <project xmlns="http://maven.apache.org/POM/4.0.0"
2. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4. http://maven.apache.org/xsd/maven-4.0.0.xsd">
5. <modelVersion>4.0.0</modelVersion>
6. <groupId>com.mycompany.app</groupId>
7. <artifactId>my-app</artifactId>
8. <packaging>jar</packaging>
9. <version>1.0-SNAPSHOT</version>
10. <name>Maven Quick Start Archetype</name>
11. <url>http://maven.apache.org</url>
12. <dependencies>
13. <dependency>
14. <groupId>junit</groupId>
15. <artifactId>junit</artifactId>
16. <version>4.11</version>
17. <scope>test</scope>
18. </dependency>
19. </dependencies>
20. </project>

pom.xml contains the Project Object Model (POM) for this project. The POM is the basic unit of work in Maven. This is important to remember because Maven is inherently project-centric in that everything revolves around the notion of a project. In short, the POM contains every important piece of information about your project and is essentially one-stop-shopping for finding anything related to your project. Understanding the POM is important and new users are encouraged to refer to the [Introduction to the POM](https://maven.apache.org/guides/introduction/introduction-to-the-pom.html).

This is a very simple POM but still displays the key elements every POM contains, so let's walk through each of them to familiarize you with the POM essentials:

* **project** This is the top-level element in all Maven pom.xml files.
* **modelVersion** This element indicates what version of the object model this POM is using. The version of the model itself changes very infrequently but it is mandatory in order to ensure stability of use if and when the Maven developers deem it necessary to change the model.
* **groupId** This element indicates the **unique identifier** of the organization or group that created the project. The groupId is one of the key identifiers of a project and is typically based on the fully qualified domain name of your organization. For example org.apache.maven.plugins is the designated groupId for all Maven plugins.
* **artifactId** This element indicates the unique base name of the primary artifact being generated by this project. The primary artifact for a project is typically a JAR file. Secondary artifacts like source bundles also use the artifactId as part of their final name. A typical artifact produced by Maven would have the form <artifactId>-<version>.<extension> (for example, myapp-1.0.jar).
* **packaging** This element indicates the package type to be used by this artifact (e.g. JAR, WAR, EAR, etc.). This not only means if the artifact produced is JAR, WAR, or EAR but can also indicate a specific lifecycle to use as part of the build process. (The lifecycle is a topic we will deal with further on in the guide. For now, just keep in mind that the indicated packaging of a project can play a part in customizing the build lifecycle.) The default value for the packaging element is JAR so you do not have to specify this for most projects.
* **version** This element indicates the version of the artifact generated by the project. Maven goes a long way to help you with version management and you will often see the SNAPSHOT designator in a version, which indicates that a project is in a state of development. We will discuss the use of [snapshots](https://maven.apache.org/guides/getting-started/index.html#What_is_a_SNAPSHOT_version) and how they work further on in this guide.
* **name** This element indicates the display name used for the project. This is often used in Maven's generated documentation.
* **url** This element indicates where the project's site can be found. This is often used in Maven's generated documentation.
* **description** This element provides a basic description of your project. This is often used in Maven's generated documentation.

The minimum requirements for a POM are the following:

* project root
* modelVersion - should be set to 4.0.0
* groupId - the id of the project's group.
* artifactId - the id of the artifact (project)
* version - the version of the artifact under the specified group

Here's an example:

1. <project>
2. <modelVersion>4.0.0</modelVersion>
3. <groupId>com.mycompany.app</groupId>
4. <artifactId>my-app</artifactId>
5. <version>1</version>
6. </project>

A POM requires that its groupId, artifactId, and version be configured. These three values form the project's fully qualified artifact name. This is in the form of <groupId>:<artifactId>:<version>. As for the example above, its fully qualified artifact name is "com.mycompany.app:my-app:1".

Default values for some elements:

the build directory ->target;

the source directory-> src/main/java;

the test source directory-> src/test/java;

the packaging -> jar

*repositories ->* https://repo.maven.apache.org/maven2

SUPER POM is a parent file which all POM files inherits unless specified otherwise. To specify the other parent file we need to add the <parent/> section as below:

1. <project>
2. <parent>
3. <groupId>com.mycompany.app</groupId>
4. <artifactId>my-app</artifactId>
5. <version>1</version>
6. </parent>
7. <modelVersion>4.0.0</modelVersion>
8. <groupId>com.mycompany.app</groupId>
9. <artifactId>my-module</artifactId>
10. <version>1</version>
11. </project>

**Project Model Variables**

Any field of the model that is a single value element can be referenced as a variable. For example, ${project.groupId}, ${project.version}, ${project.build.sourceDirectory} and so on. Refer to the POM reference to see a full list of properties.

These variables are all referenced by the prefix "project.". You may also see references with pom. as the prefix, or the prefix omitted entirely - these forms are now deprecated and should not be used.

Special Variables:

|  |  |
| --- | --- |
| project.basedir | The directory that the current project resides in. |
| project.baseUri | The directory that the current project resides in, represented as an URI. *Since Maven 2.1.0* |
| maven.build.timestamp | The timestamp that denotes the start of the build (UTC). *Since Maven 2.1.0-M1* |

#### A Build Lifecycle is Made Up of Phases

Each of these build lifecycles is defined by a different list of build phases, wherein a build phase represents a stage in the lifecycle.

For example, the default lifecycle comprises of the following phases (for a complete list of the lifecycle phases, refer to the [Lifecycle Reference](https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html#Lifecycle_Reference)):

* validate - validate the project is correct and all necessary information is available
* compile - compile the source code of the project
* test - test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
* package - take the compiled code and package it in its distributable format, such as a JAR.
* verify - run any checks on results of integration tests to ensure quality criteria are met
* install - install the package into the local repository, for use as a dependency in other projects locally
* deploy - done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.

These lifecycle phases (plus the other lifecycle phases not shown here) are executed sequentially to complete the default lifecycle. Given the lifecycle phases above, this means that when the default lifecycle is used, Maven will first validate the project, then will try to compile the sources, run those against the tests, package the binaries (e.g. jar), run integration tests against that package, verify the integration tests, install the verified package to the local repository, then deploy the installed package to a remote repository.

**Mention the difference between Apache Ant and Maven?**

Apache Ant Maven  
• Ant is a toolbox – Maven is a framework  
• Ant does not have formal conventions like project directory structure – Maven has conventions  
• Ant is procedural; you have to tell to compile, copy and compress – Maven is declarative ( information on what to make & how to build)  
• Ant does not have lifecycle; you have to add sequence of tasks manually – Maven has a lifecycle  
• Ant scripts are not reusable – Maven plugins are reusable

In Maven, the setting files are called settings.xml, and the two setting files are located at

• Maven installation directory: $M2\_Home/conf/settings.xml -> Global settings  
• User’s home directory: ${ user.home }/ .m2 / settings.xml -> User settings

Some of the main features of Maven are:

* + **Simple to use**: Maven provides easy project settings that are based on genuine practices.
  + **Fast**: You can receive a fresh project or module that began in fewer seconds in Maven.
  + **Easy to learn:** Maven usage and commands are pretty easy to learn across all projects. Therefore ramp-up time for new developers coming onto a project is very less.
  + **Dependency management:** Maven provides superior dependency management including automatic updates and transitive dependencies.
  + **Multiple Projects**: You can easily work on multiple projects at the same time by using Maven.
  + **Huge Library:** Maven has a large and growing repository of libraries and metadata to use out of the box.

## **What is Maven artifact?**

* An artifact is a file, normally a JAR that gets deployed to a Maven repository. A Maven build creates one or more artifacts, such as a compiled JAR and a source JAR.
* Each artifact consists of a group ID, an artifact ID, and a version string. The three together uniquely identify the artifact. A project’s dependencies are specified as artifacts.

## **Name the 3 build lifecycle of Maven.**

The three build lifecycles are −

**clean**: cleans up artifacts created by prior builds.

**default**: used to build the application.

**site**: generates site documentation for the project.

Good websites for interview questions:

<https://career.guru99.com/top-20-maven-interview-questions/>

<https://www.edureka.co/blog/interview-questions/maven-interview-questions/>

## Using Mirrors for Repositories

With [Repositories](https://maven.apache.org/guides/introduction/introduction-to-repositories.html) you specify from which locations you want to *download* certain artifacts, such as dependencies and maven-plugins. Repositories can be declared inside a project, which means that if you have your own custom repositories, those sharing your project easily get the right settings out of the box. However, you may want to use an alternative mirror for a particular repository without changing the project files.

Some reasons to use a mirror are:

* There is a synchronized mirror on the internet that is geographically closer and faster
* You want to replace a particular repository with your own internal repository which you have greater control over
* You want to run a [repository manager](https://maven.apache.org/repository-management.html) to provide a local cache to a mirror and need to use its URL instead

To configure a mirror of a given repository, you provide it in your settings file (${user.home}/.m2/settings.xml), giving the new repository its own id and url, and specify the mirrorOf setting that is the ID of the repository you are using a mirror of. For example, the ID of the main Maven Central repository included by default is central, so to use the different mirror instance, you would configure the following:

1. <settings>
2. ...
3. <mirrors>
4. <mirror>
5. <id>other-mirror</id>
6. <name>Other Mirror Repository</name>
7. <url>https://other-mirror.repo.other-company.com/maven2</url>
8. <mirrorOf>central</mirrorOf>
9. </mirror>
10. </mirrors>
11. ...
12. </settings>

Note that there can be at most one mirror for a given repository. In other words, you cannot map a single repository to a group of mirrors that all define the same <mirrorOf> value. Maven will not aggregate the mirrors but simply picks the first match. If you want to provide a combined view of several repositories, use a [repository manager](https://maven.apache.org/repository-management.html) instead.