What Maven Provides

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

What POM Contains

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

What is Build Lifecycle?

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. As an example, a typical **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

A **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.

The order of execution depends on the order in which the goal(s) and the build phase(s) are invoked. For example, consider the command below. The **clean**and **package** arguments are build phases while the **dependency:copy-dependencies** is a goal.

## What is SNAPSHOT?

SNAPSHOT is a special version that indicates a current development copy. Unlike regular versions, Maven checks for a new SNAPSHOT version in a remote repository for every build.

Now data-service team will release SNAPSHOT of its updated code every time to repository, say data-service: 1.0-SNAPSHOT, replacing an older SNAPSHOT jar.

## Snapshot vs Version

In case of Version, if Maven once downloaded the mentioned version, say data-service:1.0, it will never try to download a newer 1.0 available in repository. To download the updated code, data-service version is be upgraded to 1.1.

In case of SNAPSHOT, Maven will automatically fetch the latest SNAPSHOT (data-service:1.0-SNAPSHOT) every time app-ui team build their project.

## app-ui pom.xml

**app-ui** project is using 1.0-SNAPSHOT of data-service.

<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>app-ui</groupId>

<artifactId>app-ui</artifactId>

<version>1.0</version>

<packaging>jar</packaging>

<name>health</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>data-service</groupId>

<artifactId>data-service</artifactId>

<version>1.0-SNAPSHOT</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

## data-service pom.xml

**data-service** project is releasing 1.0-SNAPSHOT for every minor change.

<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>data-service</groupId>

<artifactId>data-service</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging>jar</packaging>

<name>health</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

</project>

Dependency Order

<groupId>com.companyname.project-group</groupId>

<artifactId>project</artifactId>

<version>1.0</version>

<scope>test</scope>

<dependencies>

<dependency>

<groupId>org.glassfish.jersey.containers</groupId>

<artifactId>jersey-container-servlet-core</artifactId>

</dependency>

<dependency>

<groupId>org.glassfish.jersey.media</groupId>

<artifactId>jersey-media-json-jackson</artifactId>

</dependency>

<dependency>

<groupId>org.jmockit</groupId>

<artifactId>jmockit</artifactId>

<version>1.33</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.jdbi</groupId>

<artifactId>jdbi</artifactId>

<version>2.73</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.39</version>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<version>1.16.12</version>

</dependency>

</dependencies>

What is a Maven Repository?

In Maven terminology, a repository is a directory where all the project jars, library jar, plugins or any other project specific artifacts are stored and can be used by Maven easily.

Maven repository are of three types. The following illustration will give an idea regarding these three types.

* local
* central
* remote

## Local Repository

Maven local repository is a folder location on your machine. It gets created when you run any maven command for the first time.

Maven local repository keeps your project's all dependencies (library jars, plugin jars etc.). When you run a Maven build, then Maven automatically downloads all the dependency jars into the local repository. It helps to avoid references to dependencies stored on remote machine every time a project is build.

Maven local repository by default get created by Maven in %USER\_HOME% directory. To override the default location, mention another path in Maven settings.xml file available at %M2\_HOME%\conf directory.

Central Repository

Maven central repository is repository provided by Maven community. It contains a large number of commonly used libraries.

When Maven does not find any dependency in local repository, it starts searching in central repository using following URL − <https://repo1.maven.org/maven2/>

Key concepts of Central repository are as follows −

* This repository is managed by Maven community.
* It is not required to be configured.
* It requires internet access to be searched.

## Remote Repository

Sometimes, Maven does not find a mentioned dependency in central repository as well. It then stops the build process and output error message to console. To prevent such situation, Maven provides concept of **Remote Repository**, which is developer's own custom repository containing required libraries or other project jars.

For example, using below mentioned POM.xml, Maven will download dependency (not available in central repository) from Remote Repositories mentioned in the same pom.xml.

## Maven Dependency Search Sequence

When we execute Maven build commands, Maven starts looking for dependency libraries in the following sequence −

* **Step 1** − Search dependency in local repository, if not found, move to step 2 else perform the further processing.
* **Step 2** − Search dependency in central repository, if not found and remote repository/repositories is/are mentioned then move to step 4. Else it is downloaded to local repository for future reference.
* **Step 3** − If a remote repository has not been mentioned, Maven simply stops the processing and throws error (Unable to find dependency).
* **Step 4** − Search dependency in remote repository or repositories, if found then it is downloaded to local repository for future reference. Otherwise, Maven stops processing and throws error (Unable to find dependency).

**MVN CLEAN : Cleans all the artifacts from the repository.**