What is Maven?

Maven is a project management and comprehension tool that provides developers a complete build lifecycle framework. Development team can automate the project's build infrastructure in almost no time as Maven uses a standard directory layout and a default build lifecycle.

In case of multiple development teams environment, Maven can set-up the way to work as per standards in a very short time. As most of the project setups are simple and reusable, Maven makes life of developer easy while creating reports, checks, build and testing automation setups.

Maven provides developers ways to manage the following −

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

To summarize, Maven simplifies and standardizes the project build process. It handles compilation, distribution, documentation, team collaboration and other tasks seamlessly. Maven increases reusability and takes care of most of the build related tasks.

Maven Evolution

Maven was originally designed to simplify building processes in Jakarta Turbine project. There were several projects and each project contained slightly different ANT build files. JARs were checked into CVS.

Apache group then developed **Maven** which can build multiple projects together, publish projects information, deploy projects, share JARs across several projects and help in collaboration of teams.

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. In later chapters, we will explain POM in detail.

Convention over Configuration

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

Developers do not have to mention each and every configuration detail. Maven provides sensible default behavior for projects. When a Maven project is created, Maven creates default project structure. Developer is only required to place files accordingly and he/she need not to define any configuration in pom.xml.

As an example, 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 |

In order to build the project, Maven provides developers with options to mention life-cycle goals and project dependencies (that rely on Maven plugin capabilities and on its default conventions). Much of the project management and build related tasks are maintained by Maven plugins.

Developers can build any given Maven project without the need to understand how the individual plugins work. We will discuss Maven Plugins in detail in the later chapters.

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.

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

Before creating a POM, we should first decide the project **group** (groupId), its **name** (artifactId) and its version as these attributes help in uniquely identifying the project in repository.

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.

Maven use the effective POM (configuration from super pom plus project configuration) to execute relevant goal. It helps developers to specify minimum configuration detail in his/her pom.xml. Although configurations can be overridden easily.

An easy way to look at the default configurations of the super POM is by running the following command: **mvn help:effective-pom**

Create a pom.xml in any directory on your computer.Use the content of above mentioned example pom.

In example below, We've created a pom.xml in C:\MVN\project folder.

Now open command console, go the folder containing pom.xml and execute the following **mvn** command.

C:\MVN\project>mvn help:effective-pom

Maven will start processing and display the effective-pom.

[INFO] Scanning for projects...

[INFO] Searching repository for plugin with prefix: 'help'.

[INFO] ------------------------------------------------------------------------

[INFO] Building Unnamed - com.companyname.project-group:project-name:jar:1.0

[INFO] task-segment: [help:effective-pom] (aggregator-style)

[INFO] ------------------------------------------------------------------------

[INFO] [help:effective-pom {execution: default-cli}]

[INFO]

.....

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESSFUL

[INFO] ------------------------------------------------------------------------

[INFO] Total time: < 1 second

[INFO] Finished at: Thu Jul 05 11:41:51 IST 2012

[INFO] Final Memory: 6M/15M

[INFO] -------------------------------------------

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.

mvn clean dependency:copy-dependencies package

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

Clean Lifecycle

When we execute *mvn post-clean* command, Maven invokes the clean lifecycle consisting of the following phases.

* pre-clean
* clean
* post-clean

Maven clean goal (clean:clean) is bound to the *clean* phase in the clean lifecycle. Its **clean:cleangoal** deletes the output of a build by deleting the build directory. Thus, when *mvn clean* command executes, Maven deletes the build directory.

We can customize this behavior by mentioning goals in any of the above phases of clean life cycle.

In the following example, We'll attach maven-antrun-plugin:run goal to the pre-clean, clean, and post-clean phases. This will allow us to echo text messages displaying the phases of the clean lifecycle.

We've created a pom.xml in C:\MVN\project folder.

<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.projectgroup</groupId>

<artifactId>project</artifactId>

<version>1.0</version>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-antrun-plugin</artifactId>

<version>1.1</version>

<executions>

<execution>

<id>id.pre-clean</id>

<phase>pre-clean</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>pre-clean phase</echo>

</tasks>

</configuration>

</execution>

<execution>

<id>id.clean</id>

<phase>clean</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>clean phase</echo>

</tasks>

</configuration>

</execution>

<execution>

<id>id.post-clean</id>

<phase>post-clean</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>post-clean phase</echo>

</tasks>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

</project>

Now open command console, go to the folder containing pom.xml and execute the following **mvn** command.

C:\MVN\project>mvn post-clean

Maven will start processing and displaying all the phases of clean life cycle.

[INFO] Scanning for projects...

[INFO] -----------------------------------------------------------------

-

[INFO] Building Unnamed - com.companyname.projectgroup:project:jar:1.0

[INFO] task-segment: [post-clean]

[INFO] ------------------------------------------------------------------

[INFO] [antrun:run {execution: id.pre-clean}]

[INFO] Executing tasks

[echo] pre-clean phase

[INFO] Executed tasks

[INFO] [clean:clean {execution: default-clean}]

[INFO] [antrun:run {execution: id.clean}]

[INFO] Executing tasks

[echo] clean phase

[INFO] Executed tasks

[INFO] [antrun:run {execution: id.post-clean}]

[INFO] Executing tasks

[echo] post-clean phase

[INFO] Executed tasks

[INFO] ------------------------------------------------------------------

[INFO] BUILD SUCCESSFUL

[INFO] ------------------------------------------------------------------

[INFO] Total time: > 1 second

[INFO] Finished at: Sat Jul 07 13:38:59 IST 2012

[INFO] Final Memory: 4M/44M

[INFO] ------------------------------------------------------------------

You can try tuning **mvn clean** command, which will display **pre-clean** and clean. Nothing will be executed for **post-clean** phase.

Default (or Build) Lifecycle

This is the primary life cycle of Maven and is used to build the application. It has the following 21 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).

In the following example, we will attach maven-antrun-plugin:run goal to few of the phases of Build lifecycle. This will allow us to echo text messages displaying the phases of the lifecycle.

We've updated pom.xml in C:\MVN\project folder.

<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.projectgroup</groupId>

<artifactId>project</artifactId>

<version>1.0</version>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-antrun-plugin</artifactId>

<version>1.1</version>

<executions>

<execution>

<id>id.validate</id>

<phase>validate</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>validate phase</echo>

</tasks>

</configuration>

</execution>

<execution>

<id>id.compile</id>

<phase>compile</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>compile phase</echo>

</tasks>

</configuration>

</execution>

<execution>

<id>id.test</id>

<phase>test</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>test phase</echo>

</tasks>

</configuration>

</execution>

<execution>

<id>id.package</id>

<phase>package</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>package phase</echo>

</tasks>

</configuration>

</execution>

<execution>

<id>id.deploy</id>

<phase>deploy</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>deploy phase</echo>

</tasks>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

</project>

Now open command console, go the folder containing pom.xml and execute the following **mvn** command.

C:\MVN\project>mvn compile

Maven will start processing and display phases of build life cycle up to the compile phase.

[INFO] Scanning for projects...

[INFO] -----------------------------------------------------------------

-

[INFO] Building Unnamed - com.companyname.projectgroup:project:jar:1.0

[INFO] task-segment: [compile]

[INFO] -----------------------------------------------------------------

-

[INFO] [antrun:run {execution: id.validate}]

[INFO] Executing tasks

[echo] validate phase

[INFO] Executed tasks

[INFO] [resources:resources {execution: default-resources}]

[WARNING] Using platform encoding (Cp1252 actually) to copy filtered

resources,

i.e. build is platform dependent!

[INFO] skip non existing resourceDirectory

C:\MVN\project\src\main\resources

[INFO] [compiler:compile {execution: default-compile}]

[INFO] Nothing to compile - all classes are up to date

[INFO] [antrun:run {execution: id.compile}]

[INFO] Executing tasks

[echo] compile phase

[INFO] Executed tasks

[INFO] -----------------------------------------------------------------

-

[INFO] BUILD SUCCESSFUL

[INFO] -----------------------------------------------------------------

-

[INFO] Total time: 2 seconds

[INFO] Finished at: Sat Jul 07 20:18:25 IST 2012

[INFO] Final Memory: 7M/64M

[INFO] -----------------------------------------------------------------

-

Site Lifecycle

Maven Site plugin is generally used to create fresh documentation to create reports, deploy site, etc. It has the following phases −

* pre-site
* site
* post-site
* site-deploy

In the following example, we will attach **maven-antrun-plugin:run** goal to all the phases of Site lifecycle. This will allow us to echo text messages displaying the phases of the lifecycle.

We've updated pom.xml in C:\MVN\project folder.

<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.projectgroup</groupId>

<artifactId>project</artifactId>

<version>1.0</version>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-antrun-plugin</artifactId>

<version>1.1</version>

<executions>

<execution>

<id>id.pre-site</id>

<phase>pre-site</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>pre-site phase</echo>

</tasks>

</configuration>

</execution>

<execution>

<id>id.site</id>

<phase>site</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>site phase</echo>

</tasks>

</configuration>

</execution>

<execution>

<id>id.post-site</id>

<phase>post-site</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>post-site phase</echo>

</tasks>

</configuration>

</execution>

<execution>

<id>id.site-deploy</id>

<phase>site-deploy</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<tasks>

<echo>site-deploy phase</echo>

</tasks>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

</project>

Now open the command console, go the folder containing pom.xml and execute the following **mvn** command.

C:\MVN\project>mvn site

Maven will start processing and displaying the phases of site life cycle up to site phase.

[INFO] Scanning for projects...

[INFO] ------------------------------------------------------------------

[INFO] Building Unnamed - com.companyname.projectgroup:project:jar:1.0

[INFO] task-segment: [site]

[INFO] ------------------------------------------------------------------

[INFO] [antrun:run {execution: id.pre-site}]

[INFO] Executing tasks

[echo] pre-site phase

[INFO] Executed tasks

[INFO] [site:site {execution: default-site}]

[INFO] Generating "About" report.

[INFO] Generating "Issue Tracking" report.

[INFO] Generating "Project Team" report.

[INFO] Generating "Dependencies" report.

[INFO] Generating "Project Plugins" report.

[INFO] Generating "Continuous Integration" report.

[INFO] Generating "Source Repository" report.

[INFO] Generating "Project License" report.

[INFO] Generating "Mailing Lists" report.

[INFO] Generating "Plugin Management" report.

[INFO] Generating "Project Summary" report.

[INFO] [antrun:run {execution: id.site}]

[INFO] Executing tasks

[echo] site phase

[INFO] Executed tasks

[INFO] ------------------------------------------------------------------

[INFO] BUILD SUCCESSFUL

[INFO] ------------------------------------------------------------------

[INFO] Total time: 3 seconds

[INFO] Finished at: Sat Jul 07 15:25:10 IST 2012

[INFO] Final Memory: 24M/149M

[INFO] --------------------------------------------------------------