
Apache
maven


Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

What is Maven

Maven is a project management tool which can manage the complete building life cycle.

Maven simplifies and standardizes the project build process. by handling compilation, testing, library dependency, distribution, documentation and team collaboration.

The Maven developers claim that Maven is more than just a build tool. We can think of Maven as a build tool with more features.

Maven provides developers ways to manage following:

- Builds
- Test
- Documentation
- Reporting
- Dependencies
- Releases
- Distribution
- Mailing List

When creating a project, Maven creates default project structure. Developer can just save files accordingly.

The following table shows the default values for project source code files, test case folders, resource files and other configurations.

${basedir} denotes the project root folder:

Item	Default
pom.xml	${basedir}/pom.xml
source code	${basedir}/src/main/java
resources	${basedir}/src/main/resources
test cases source files	${basedir}/src/test/java
test cases resource files	${basedir}/src/test/resources

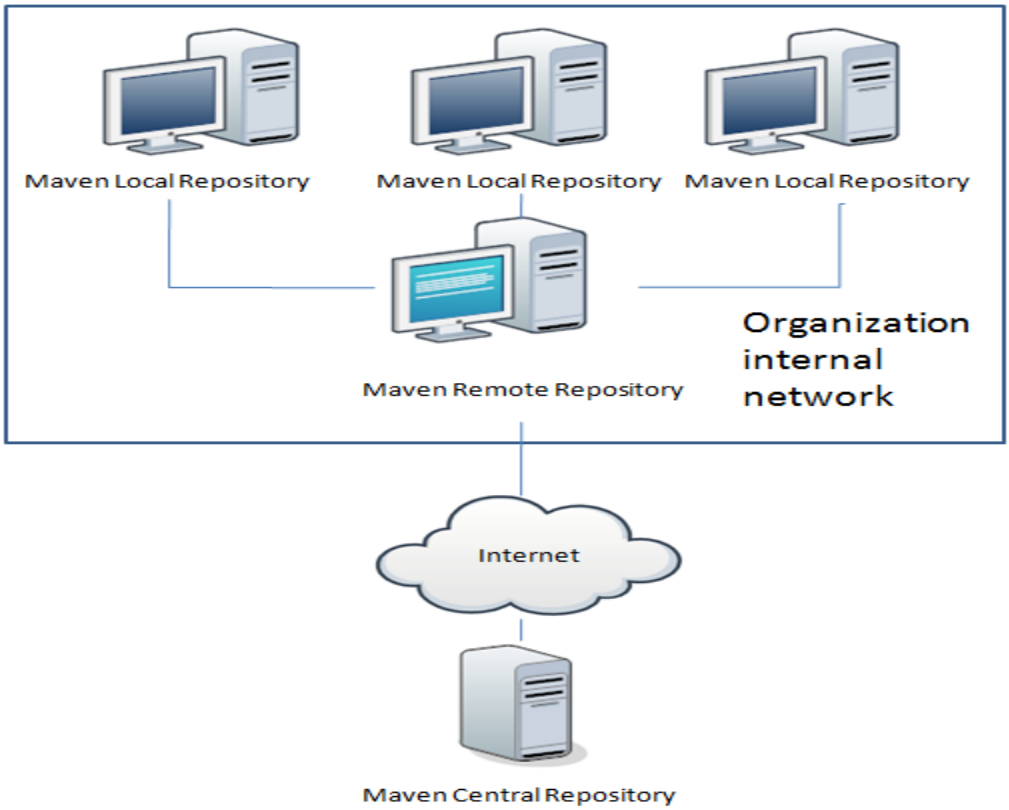
Complied source code	${basedir}/target
Generated JAR files	${basedir}/target/classes

How Maven uses the POM file

Maven uses the pom.xml file in the following steps.

- Read the pom.xml file, parse the content.
- Download dependencies to local dependency repository.
- Execute life cycle/build phase/goal.
  For example, mvn compile will do the compile
  mvn test will execute all unit test cases
  mvn package will do compile, then execute all unit test cases, finally zip the classes file to a jar/war/ear file.
- Execute plugins. Maven plugins are extensions for Maven core. Sometime we need to use plugins to do custom-project specific tasks.

Repositories:



For Setting Custom path for local repository,below is the way.

```
<localRepository>C:/MyLocalRepository</localRepository>
```

Remote Repository

Sometime we need to set up a Maven repository inside a company or a project development team to host our own libraries.

The company mantained repository is outside developer's machine and is called Maven remove repository.

The following pom.xml declares dependencies and also declared remote repository URL.

```
<project ...>
  <dependencies>
    <dependency>
      <groupId>com.companyname.common-lib</groupId>
      <artifactId>common-lib</artifactId>
      <version>1.0.0</version>
    </dependency>
  <dependencies>
  <repositories>
    <repository>
      <id>companyname.lib1</id>
      <url>http://download.companyname.org/maven2/lib1</url>
    </repository>
    <repository>
      <id>companyname.lib2</id>
      <url>http://download.companyname.org/maven2/lib2</url>
    </repository>
  </repositories>
</project>
```

Maven Dependency Search Sequence

Maven searches for dependency libraries in the following sequence:

1. Search local dependency repository.
2. Search central dependency repository
3. Search the remote dependency repository

Maven stops the searching once it finds the jar file.

Maven build lifecycle:

A Build Lifecycle is a sequence of tasks we used to build a software. For example, compile, test, test more, package and publish or deploy are all tasks we need to do to build a software.

A Maven build lifecycle is a sequence of phases we need to go through in order to finishing building the software.

The following table lists some of the build lifecycle.

Lifecycle	Description
validate	validate the project is correct and all necessary information is available
compile	compile the source code
test	test the compiled source code using a unit testing
package	take the compiled code and package it in its distributable format, such as a JAR
integration-test	deploy the package into an environment where integration tests can be run
verify	verify the package is valid and meets quality criteria
install	install the package into the local repository
deploy	publish to integration or release environment

Maven has following three standard lifecycles:

- clean
- default (or build)
- site

These build phases are executed sequentially to complete the default lifecycle.

Given the build phases above, when the default lifecycle is used, Maven will

1. validate the project
2. compile the sources
3. run those against the tests
4. package the binaries (e.g. jar)
5. run integration tests against that package
6. verify the package
7. install the verifed package to the local repository
8. deploy the installed package in a specified environment

To do all those, you only need to call the last build phase to be executed, in this case, deploy:

```
mvn deploy
```

Calling a build phase will execute not only that build phase, but also every build phase prior to the called build phase.

Thus, doing

```
mvn integration-test
```

will do every build phase before it (validate, compile, package, etc.), before executing integration-test.

The same command can be used in a multi-module with one or more subprojects. For example:

```
mvn clean install
```

This command will traverse into all of the subprojects and run clean, then install including all of the prior steps.

Clean Lifecycle Reference

pre-clean	executes processes needed prior to the actual project cleaning
clean	remove all files generated by the previous build

post-clean	executes processes needed to finalize the project cleaning

Default Lifecycle Reference

validate	validate the project and ensure that all necessary information is available.
initialize	initialize build state, set properties or create directories.
generate-sources	generate any source code.
process-sources	process the source code.
generate-resources	generate resources.
process-resources	copy and process the resources into the destination directory for packaging.
compile	compile the source code.
process-classes	post-process the generated files from compilation.
generate-test-sources	generate any test source code.
process-test-sources	process the test source code.
generate-test-resources	create resources for testing.
process-test-resources	copy and process the resources into the test destination directory.

test-compile	compile the test source code
process-test-classes	post-process the generated files from test compilation.
test	run tests using a unit testing framework.
prepare-package	perform any operations necessary to prepare a package before the packaging.
package	package the compiled code into its distributable format, such as a JAR.
pre-integration-test	perform actions required before integration tests are executed.
integration-test	process and deploy the package into an environment where integration tests can be run.
post-integration-test	perform actions required after integration tests have been executed.
verify	run any checks to verify the package is valid.
install	install the package into the local repository.
deploy	publish the project.

Site Lifecycle Reference

pre-site	executes processes prior to the project site generation
site	generates the project's site documentation

post-site	executes processes to finalize the site generation
site-deploy	deploys the generated site to the web server

Maven Build Profiles:

A *Build profile* is a set of configuration values which can be used to set or override default values of Maven build. Using a build profile, you can customize build for different environments such as *Production* v/s *Development* environments.

Profiles are specified in pom.xml file using its activeProfiles / profiles elements and are triggered in variety of ways. Profiles modify the POM at build time, and are used to give parameters different target environments (for example, the path of the database server in the development, testing, and production environments).

Profile Activation:

A Maven Build Profile can be activated in various ways.

- Explicitly using command console input.
- Through maven settings.
- Based on environment variables (User/System variables).
- OS Settings (for example, Windows family).
- Present/missing files.

Types of Build Profile:

Build profiles are majorly of three types

Type	Where it is defined
Per Project	Defined in the project POM file, pom.xml

Per User	Defined in Maven settings xml file (%USER_HOME%/.m2/settings.xml)
Global	Defined in Maven global settings xml file (%M2_HOME%/conf/settings.xml)

Example:

```
<profile>
    <id>db-localhost-oracle</id>
    <dependencies>
      <dependency>
        <groupId>ojdbc6</groupId>
        <artifactId>ojdbc6</artifactId>
      </dependency>
    </dependencies>
    <properties>
      <db.driver>oracle.jdbc.driver.OracleDriver</db.driver>
      <db.dialect>no.jbv.sergej.util.FixedOracle10gDialect</db.dialect>
      <db.url>jdbc:oracle:thin:@//localhost:1521/xe</db.url>
      <db.hbm2ddl>update</db.hbm2ddl>
    </properties>
  </profile>

  <profile>
    <id>db-localhost-mysql</id>
    <activation>
      <activeByDefault>true</activeByDefault>
    </activation>
    <dependencies>
      <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
      </dependency>
    </dependencies>
    <properties>
      <db.driver>com.mysql.jdbc.Driver</db.driver>
      <db.dialect>org.hibernate.dialect.MySQL5Dialect</db.dialect>
      <db.url>jdbc:mysql://localhost/${mysql.schema}</db.url>
      <db.hbm2ddl>update</db.hbm2ddl>
    </properties>
  </profile>
```

To see which profile will activate in a certain build, use the maven-help-plugin.

```
mvn help:active-profiles
```

What are Maven Plugins?

Maven is actually a plugin execution framework where every task is actually done by plugins. Maven Plugins are generally used to :

- create jar file
- create war file
- compile code files
- unit testing of code
- create project documentation
- create project reports

A plugin generally provides a set of goals and which can be executed using following syntax:

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

For example, a Java project can be compiled with the maven-compiler-plugin's compile-goal by running following command

```
mvn compiler:compile
```

Plugin Types

Maven provided following two types of Plugins:

Type	Description
Build plugins	They execute during the build and should be configured in the <build/> element of pom.xml
Reporting plugins	They execute during the site generation and they should be configured in the <reporting/> element of the pom.xml

Following is the list of few common plugins:

Plugin	Description
clean	Clean up target after the build. Deletes the target directory.
compiler	Compiles Java source files.
surefire	Run the JUnit unit tests. Creates test reports.
jar	Builds a JAR file from the current project.
war	Builds a WAR file from the current project.
javadoc	Generates Javadoc for the project.
antrun	Runs a set of ant tasks from any phase mentioned of the build.

Maven Package Project:

```
mvn clean package
```

Create a Project:

```
C:\mvn_test>mvn archetype:generate -DgroupId=com.caps.ide -DartifactId=xmlFileEditor -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

Run java Main:

```
mvn exec:java -Dexec.mainClass="com.caps.ide.App"
```

With arguments:

```
mvn exec:java -Dexec.mainClass="com.caps.ide.App" -Dexec.args="arg0 arg1 arg2"
```

With runtime dependencies in the CLASSPATH:

```
mvn exec:java -Dexec.mainClass="com.caps.ide.App" -Dexec.classpathScope=runtime
```

To create Documentation for the project:

```
mvn site
```

To run Unit test Via Maven:

```
mvn test
```

To run single test, issue this command :

```
mvn -Dtest=TestClass1 test
```

Skip test

We can skip test by using the following command.

```
mvn package -Dmaven.test.skip=true
```

To run Specific profile: `mvn clean install -Pdb-localhost-oracle`

```
mvn -P debug
```

The Maven Surefire Plugin

The Maven unit test reports are generated by the Maven Surefire plugin. Therefore a unit test report is also some times referred to as surefire report.

Generating a Unit Test Report

You generate a Maven unit test report (Surefire unit test report) using the following Maven command:

mvn surefire-report:report

For more information about how the Maven command structure looks, see my Maven commands tutorial.

The generated unit test report can be found in the target/site directory. The unit test report is named surefire-report.html. The path to the unit test report is thus:

your-project/target/site/surefire-report.html

Skipping the Tests

Sometimes you might want Maven to generate a unit test report without running all the unit tests again. You might just want to use the results from the last run. You can get Maven to just generate the unit test report without rerunning the unit tests using this command:

mvn surefire-report:report-only

maven Deploy to tomcat:

We can use Maven-Tomcat plugin to package and deploy a WAR file to Tomcat for both Tomcat 6 and 7.

We used the following libraries.

- Maven 3
- Tomcat 6.0.37
- Tomcat 7.0.53

Command

For Tomcat 7, we have the following settings and command

Deploy URLhttp://localhost:8080/manager/text

Command mvn tomcat7:deploy

For Tomcat 6 we use the following url and command

Deploy URLhttp://localhost:8080/manager/

Command mvn tomcat6:deploy

Tomcat 7 Example

We can use the following steps to package and deploy a WAR file on Tomcat 7.

Add an user with roles manager-gui and manager-script in %TOMCAT7_PATH%/conf/tomcat-users.xml.

```
<?xml version='1.0' encoding='utf-8'?>

<tomcat-users>

...

  <role rolename="manager-gui"/>

  <role rolename="manager-script"/>

  <user username="admin" password="password" roles="manager-gui,manager-script" />

...
```

```
</tomcat-users>
```

The we have to add above Tomcat's user in the Maven setting file (%MAVEN_PATH%/conf/settings.xml), later Maven will use this user to login Tomcat server.

```
<?xml version="1.0" encoding="UTF-8"?>
<settings ...>
  <servers>

    <server>
      <id>TomcatServer</id>
      <username>admin</username>
      <password>password</password>
    </server>

  </servers>
</settings>
```

Then add the Tomcat7 Maven Plugin to pom.xml in the plugin section

```
  <plugin>
    <groupId>org.apache.tomcat.maven</groupId>
    <artifactId>tomcat7-maven-plugin</artifactId>
    <version>2.2</version>
    <configuration>
      <url>http://localhost:8080/manager/text</url>
      <server>TomcatServer</server>
      <path>/java2sWebApp</path>
    </configuration>
  </plugin>
```

We can issue the following code to deploy WAR file to Tomcat.

The deploy command deploys the WAR file to Tomcat server via "http://localhost:8080/manager/text" , on path "/java2sWebApp", using "TomcatServer" in settings.xml username and password for authentication.

mvn tomcat7:deploy

mvn tomcat7:undeploy

mvn tomcat7:redeploy

Tomcat 6 Example

We can use the following steps to deploy WAR file to Tomcat 6.

Add the following user name and role setting to %TOMCAT6_PATH%/conf/tomcat-users.xml.

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>

  <role rolename="manager-gui"/>
  <role rolename="manager-script"/>
  <user username="admin" password="password" roles="manager-gui,manager-script" />

</tomcat-users>
```

Add the following Maven Authentication settings to %MAVEN_PATH%/conf/settings.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<settings ...>
  <servers>

    <server>
      <id>TomcatServer</id>
      <username>admin</username>
      <password>password</password>
```

```
    </server>

  </servers>
</settings>
```

Add the Tomcat6 Maven Plugin to POM.xml file

```
  <plugin>
    <groupId>org.apache.tomcat.maven</groupId>
    <artifactId>tomcat6-maven-plugin</artifactId>
    <version>2.2</version>
    <configuration>
      <url>http://localhost:8080/manager</url>
      <server>TomcatServer</server>
      <path>/java2sWebApp</path>
    </configuration>
  </plugin>
```

Use the following command to deploy to Tomcat

mvn tomcat6:deploy

mvn tomcat6:undeploy

mvn tomcat6:redeploy

SNAPSHOT vs VERSION:

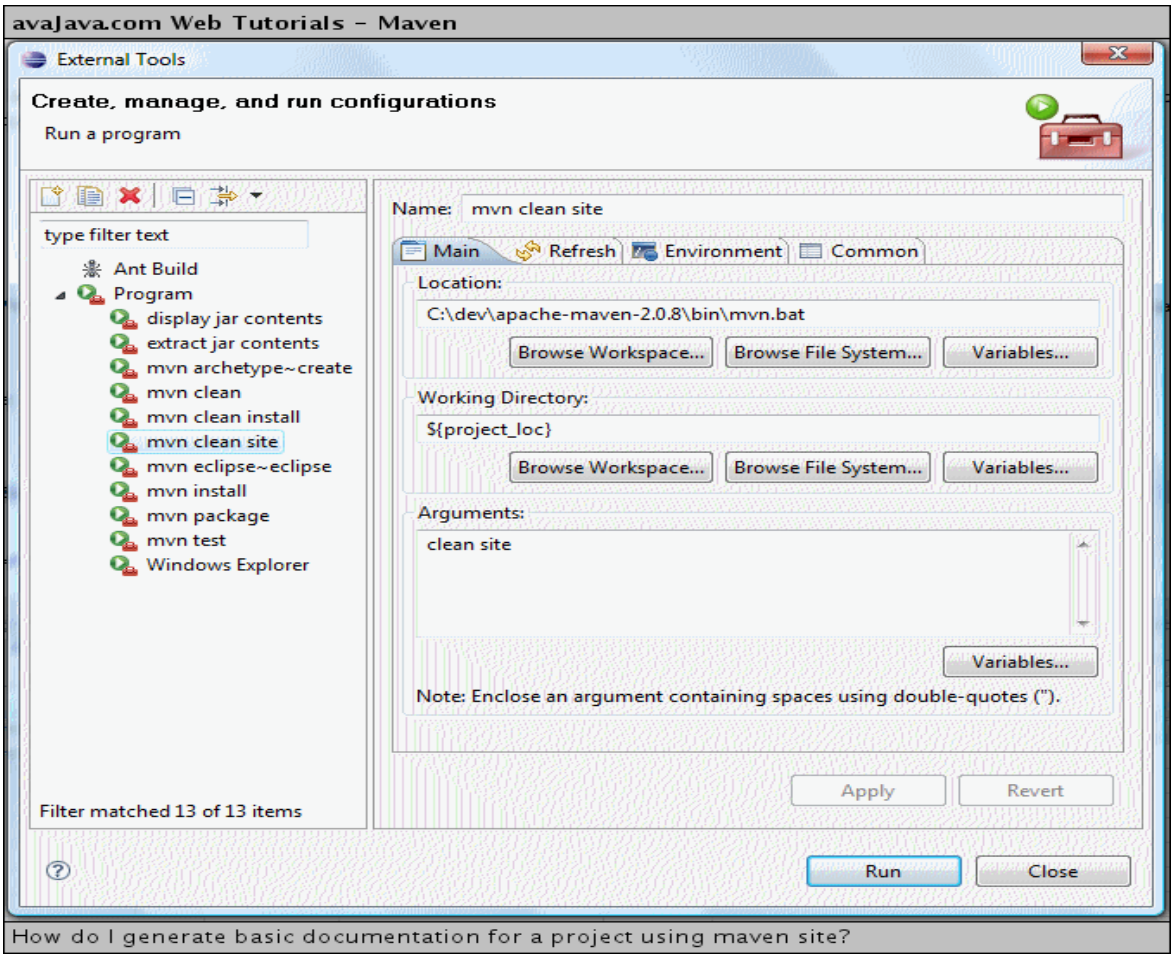
A "release" is the final build for a version which does not change.

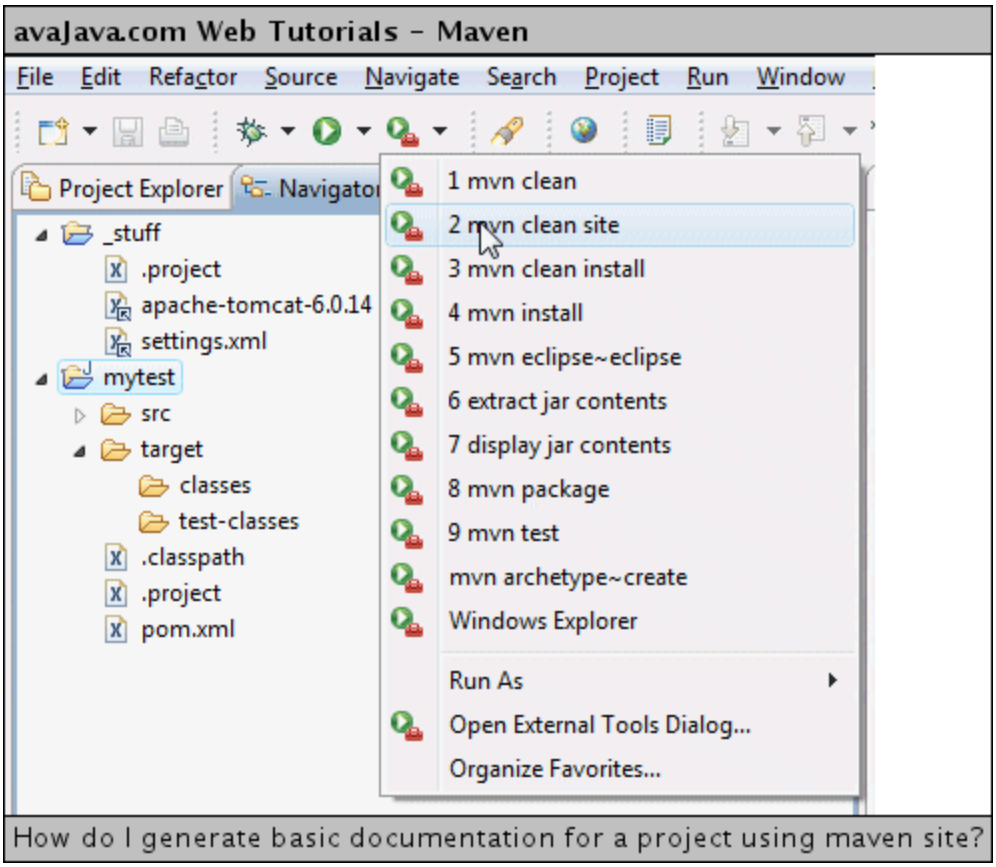
A "snapshot" is a build which can be replaced by another build which has the same name. It is implies the build could change at any time and is still under active development.

You have different artifacts for different builds based on the same code.

E.g. you might have one with debugging and one without. One for Java 5.0 and one for Java 6. Generally its simpler to have one build which does everything you need.

Adding Goals in Eclipse:




avaJava.com Web Tutorials - Maven
File Edit Refactor Source Navigate Search Project Run Window
Project Explorer
Navigator
_stuff
.project
apache-tomcat-6.0.14
settings.xml
mytest
src
target
classes
test-classes
.classpath
.project
pom.xml
1 mvn clean
2 mvn clean site
3 mvn clean install
4 mvn install
5 mvn eclipse~eclipse
6 extract jar contents
7 display jar contents
8 mvn package
9 mvn test
mvn archetype~create
Windows Explorer
Run As
Open External Tools Dialog...
Organize Favorites...
How do I generate basic documentation for a project using maven site?