**Maven -**

## Introduction

Build tools are used basically for **compiling** in order to create an image for all the code of the project. By the time the build tools added beside compiling other useful features like:

     - management of the project dependencies

     - handling management tasks

     - adapting to the changing of the product requirements

     - automatization .

With **Maven,**programmers do not have to create a build process themselves, they only have to declare **life-cycle goals** and **dependencies** for the project.

Maven helps the companies to **standardize**:

      - the programming rules

      - locations for source files

      - documentation

      - dependencies

      - output .

Maven is not only a build tool but is also a **project management tool**. With Maven can be used to generate reports, web sites and improve team collaboration.

## Installing Maven

### Installing Maven on Mac OSX or Linux

environment variables have to be exported in the .bash\_login file

|  |  |
| --- | --- |
| 1  2 | export M2\_HOME=/usr/local/maven  export PATH=${PATH}:${M2\_HOME}/bin |

### Installing Maven on Microsoft Windows

C:\Users\Some user >  set M2\_HOME=c: \ Program Files \ apache - maven - 3.2.3

C:\Users\Some user >  set PATH=%PATH% ; %M2\_HOME% \bin

Do this in environment variable

## Create new project with Maven

## Mvn archetype:generate

Choose webapp or quick start

Then

#### GroupId

This identifier will identify the project uniquely across all the projects. In case of multiple module project a new name should be added to the parent’s groupId as root package. In Maven repository this name represents the hierarchical location.

#### ArtifactId

The identifier represents the name of the jar without version. This identifier can have whatever name but with lowercase letters and no strange symbols.

#### Version

If the project is distributed the version should be one typical with numbers and dots, e.g.

1.0

C:\mvn archetype:generate

-DgroupId=com.company.insurance

-DartifactId=carAccident

-DarchetypeArtifactId=maven-archetype-quickstart

-DinteractiveMode=false

## Testing a Maven Installation

**mvn -v or mvn—version**

from command-line. If Maven has been installed, the result will be like:

## Maven properties

Maven properties can be defined in pom.xml file or in any resource that will be processed by the Maven Resource plugin’s filtering features.

**Properties have the syntax: ${ and}.**

**The implicit properties for a Maven project are:**

**project.\***

This prefix can be used to reference values in a Maven POM.

**settings.\***

This prefix can be used to reference values from Maven Settings in ~/.m2/ settings.xml

**env.\***

This prefix can be used to reference environment variables ( PATH and M2\_HOME)

### Maven settings

Maven settings.xml file is used to define some values for configuration of the local repository location, alternate remote repository servers, user configuration as authentication information for private users, active build profiles.

There are two settings files located at:

[?](http://www.wideskills.com/maven/maven-installation)

|  |  |
| --- | --- |
| 1  2 | $M2\_HOME/conf/settings.xml  ${user.home}/.m2/settings.xml |

The files are optional. If both files are present, the values in the Maven installation settings file are overriden by the user home settings.

When the file settings.xml is located in root directory of an application it will be automatically used to configure Maven at compile time.

If the settings.xml is not located in the root directory of the application or if the settings configurations are changing often this file should be put in a custom location.

Top level elements of the settings.xml:

**localRepository**

The location of the local repository can be changed by the user configuration. The default value is

|  |  |
| --- | --- |
| 1 | ${user.home}/.m2/repository/ |

In settings.xml is configured :

|  |  |
| --- | --- |
| 1  2  3  4  5 | <settings>  …..      <localRepository>/path/to/local/repo/</localRepository>  …..  </settings> |

The local repository must have an absolute path.

### Project properties

The Maven Project Property is referenced through the property name from Maven Project Object Model (POM).

Anything in Maven POM can be referenced with a property. For example:

**project.name and project.description**

The project name and project description are used to reference from documentation.

**project.groupId and project.version**

If the projects are build in a multi - module build type they share the same groupId and version identifiers.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <dependencies>  <dependency>  <groupId>${project.groupId}</groupId>  <artifactId>main-project</artifactId>  <version>${project.version}</version>  </dependency>  </dependencies> |

### Resource filtering : FIND ?

## Maven configuration

After the Maven is installed it could be noticed that some local user-specific files have been created and a local repository in the home directory.

The **~/.m2 home**directory there will be:

**~/m2/settings.xml**

See the 2.6.1 section where this file is described

**~/m2/repository**

This directory contains local Maven repository. When a dependency is downloaded from the Maven repository a copy of this is stored in the local repository.

### Plugins Configuration

Plugins can be customized by configuring them in the project’s POM by the properties defined by goals in a plugin.

For example for the **compile**goal there are more configuration parameters like

**Maven Help Plugin**can be used to describe a particular plugin goal if a plugin goal

configuration parameter is searched.

With the command **mvnhelp:describe**a particular plugin can be described from the command line.

|  |  |
| --- | --- |
|  | $ mvn help: describe -Dcmd=compiler:compile -Ddetail |

**-Ddetail** argument will determine Maven to print all of the goals parameters for the

entire plugin.

## ****Description****

**POM** is an xml file where a Maven project is described, configured and customized like the web.xml file for the Java web applications, Makefile or an Ant build.xml.

**POM** contains all the parameters and settings shared across the company - multiple projects, developers, paths, maven plugins, many dependencies with different versions.

In **POM** unique coordinates can be assigned to the project (groupId, artifactId, version), the attributes of the projects, developers or persons that contribute to the project, dependencies are described.

In conclusion by **POM** file in Maven defines the **model** for every project.

By using this approach with POM, Maven can be considered a **declarative** build system declaring what **we would like to have as result** not **what to do(imperative build system**).

Like in Java where all the objects have parent java.lang.Object, in Maven all **POM**s have parent in Maven **Super POM**.This **POM** contains all the default information and it is not necessary to be repeated with every new **POM**.

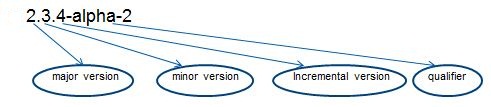
<https://maven.apache.org/ref/3.6.3/maven-model-builder/super-pom.html>

## ****POM Syntax****

### **Project versions**

Maven project version represents a number used to order the releases for a project.  This number is composed of other three numbers separated by dots every one of the number having an important role.

|  |  |
| --- | --- |
| 1 | <major version>.<minor version>.<incremental version>-<qualifier> |



### **NAPSHOT versions**

If the version contains the string “**SNAPSHOT**” Maven will add to this token date and time converted to UTC.

For example:

0.1-SNAPSHOT will be extended by Maven to **0.1-20141003-120305-1** this means the time for release 12:02 AM on October 3rd, 2014 UTC.

A project that depends on **SNAPSHOT** is not stable. All the **SNAPSHOT** versions of the dependencies should be resolved on release versions when a project is released**. SNAPSHOT** versions are only for development.

We can visualize the pom.xml with the command line

|  |  |
| --- | --- |
| 1 | mvnhelp:effective-pom |

or declaring **help:effective-pom**  as goal in Eclipse. This will help to replace references (type of ${..} ) to properties with its values

## ****Parent POM ? check****

## ****Dependencies****

### **External dependencies**

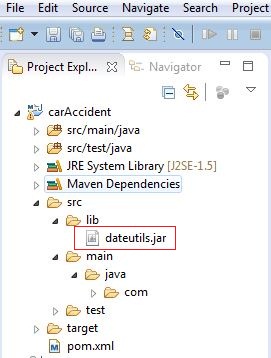
If necessary dependencies for a project are not available neither in central repository nor in remote repository or local repository then they can be called **External Dependencies**.

The dependency can be located in the local machine and moved then to the lib directory. This type of dependencies are named external because are external to Maven repository system.

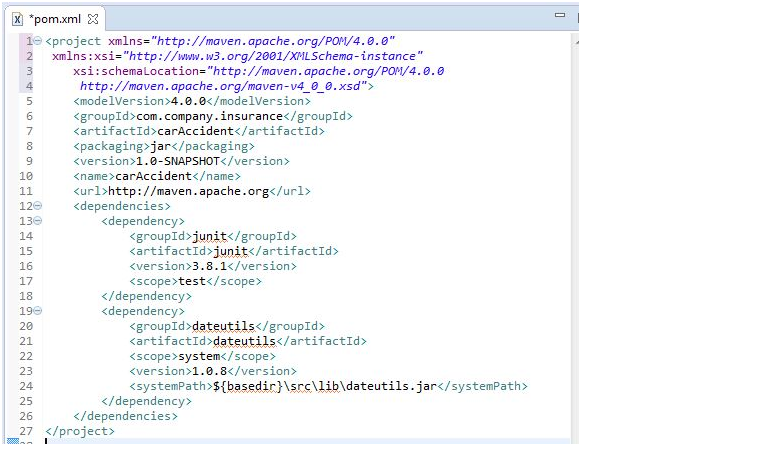
**External Dependencies** can be added to the **POM** as other dependencies. For example if a library will be added to a project in pom.xml it has to be specified:

* groupId ( name of the library)
* artifactId (name of the library)
* scope = system
* version ( respect the pom rules for version syntax )
* system relative path to the project location

For example if a local jar of typedateUtils ( necessary classes to format date and time ) and this jar does not exist in the any repository this jar can be added to lib folder ( should be created in src folder). The project will have now the following structure:



In POM this jar will be added as external dependency:



We can try this

mvn install:install-file -Dfile=C:\Users\DELL\Desktop\ojdbc6.jar -DgroupId=com.oracle -DartifactId=ojdbc6 -Dversion=11.2.0.4 -Dpackaging=jar

or

We can also use this

<dependency>

<groupId>com.oracle</groupId>

<artifactId>ojdbc6</artifactId>

<version>11.2.0.3</version>

</dependency>

<repositories> <repository>

<id>codelds</id>

<url>https://code.lds.org/nexus/content/groups/main-repo</url>

</repository> </repositories>

<https://mkyong.com/maven/how-to-add-oracle-jdbc-driver-in-your-maven-local-repository/>

### **Snapshot dependencies**

Snapshot dependencies are dependencies (jar files) which are currently under deployment. The snapshot dependencies are always downloaded in the local repository in order to have the latest version for the build.

<https://stackoverflow.com/questions/22015685/maven-3-does-not-update-snapshot-dependency-from-local-repository>

## FIND?

## ****Exclusions :****

Exclusions are necessary when different versions of the same JAR are fetched in the project. This can lead to classpath issues.

This problem appears when transitive dependencies (projects that depend on other projects) are fetched in the project.

To avoid this, the <exclusions> tag has to be added.

<dependency>

<groupId>mydependency</groupId>

<artifactId>project-x</artifactId>

<version>1.0-SNAPSHOT</version>

<exclusions>

<exclusion>

<groupId>mydependency</groupId>

<artifactId>project-y</artifactId>

</exclusion>

</exclusions>

</dependency>

#### Why exclusions are made on a per-dependency basis, rather than at the POM level

This is mainly to be sure the dependency graph is predictable, and to keep inheritance effects from excluding a dependency that should not be excluded. If you get to the method of last resort and have to put in an exclusion, you should be absolutely certain which of your dependencies is bringing in that unwanted transitive dependency.

If you truly want to ensure that a particular dependency appears nowhere in your classpath, regardless of path, the [banned dependencies rule](https://maven.apache.org/enforcer/enforcer-rules/bannedDependencies.html) can be configured to fail the build if a problematic dependency is found. When the build fails, you'll need to add specific exclusions on each path the enforcer finds.

## <https://maven.apache.org/guides/introduction/introduction-to-optional-and-excludes-dependencies.html>

## find aggregation in maven?

## ****Types of repositories****

## **Local repository**

## **Central repository**

## **Remote repository**

## Maven repositories are folders used to store artifacts resulted from development work.

## Storing Maven artifacts (jars and poms) in specific repositories is preferable to storing them in CVS or SVN (control versioning systems) because jars are binary files and these types of environments handle text files better and updates are faster.

* are open source and free
* provide repository browser
* backup support
* are deployable on web server

### **Local repository**

**Local repository**is the folder from the developer machine or build system where Maven stores project’s dependencies:

         - plugins

         - jar

         - other files downloaded by Maven.

 During the build of a project all the dependency files will be stored in the local repository.

The default Maven repository is **.m2** folder in %USER HOME% directory.

**Unix/Mac OS X - ~/.m2**

Windows - C:\Documents and Settings\{your username}\.m2

In order to install a project artifact into the local repository the following command line has to be executed:

**mvn install**

This command builds the project and copies the packaged jars into the Maven local repository. The command triggers all the build phases before install but we will discuss at the next chapters about the build phases.

<settings>

<!--localRepository

   | The path to the local repository maven will use to store artifacts.

   |

   | Default: ~/.m2/repository

<localRepository>/path/to/local/repo</localRepository>

## ****Central repository****

When the project is built, Maven checks the pom.xml file and determines which dependency to download.

## http://www.wideskills.com/sites/default/files/subjects/Maven%20Tutorial/04/m01.png

First of all Maven gets the dependency from the local repository and in case that is not found Maven try to get it from the default repository - central repository.

[**http://repo1.maven.org/maven2/**](http://repo1.maven.org/maven2/)

[**http://repo2.maven.org/maven2/**](http://repo2.maven.org/maven2/)

[**http://repo3.maven.org/maven2/**](http://repo3.maven.org/maven2/)

The site is not available and finally it can be redirected to [**http://search.maven.org/**](http://search.maven.org/).

The Central Repository is managed by the Maven community.

if a project is published to the **Central Repository** the individuals have to respect some basic requirements regarding the quality of the projects metadata.

#### **Mirrors**

Because repositories are declared inside a project (custom repositories) the persons who   sharethe project can get the right settings from the Maven files. The mirror is an alternative which can be used for a particular repository without changing the project settings files.

Mirrors are used because:

    ▪       Exist a synchronized mirror on the internet that is geographically closer and faster

    ▪       A particular repository should be replaced with an internal repository with a greater control over

    ▪       A repository manager should provide a local cache to a mirror and it need to use its URL instead

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

|  |  |
| --- | --- |
|  | <settings>    ...  <mirrors>  <mirror>  <id>UK</id>  <name>UK Central</name>  <url>http://uk.maven.org/maven2</url>  <mirrorOf>central</mirrorOf>  </mirror>  </mirrors>    ...  </settings> |

## Publishing your artifacts to the Central Repository

### <https://dzone.com/articles/publish-your-artifacts-to-maven-central>

### **Remote repository**

Projects have unique dependencies that are not available in the Maven Apache Central Repository. Local repositories cannot be used every time from the team members to download manually new libraries.

To prevent such situations Maven provides the solution of centralized **Remote Repository**where all the dependencies need to be made maintained and available. The missing dependencies are automatically installed on the local repositories from the remote repository.

**The Remote Repository**is a repository located on a web server similar to Central Repository from which Maven can download dependencies. It can be located on the local network of a company or anywhere on the internet.

This repository is used usually for hosting the internal projects of a company (organization) which then are shared among other internal projects. For example projects containing common utilities tools for programming (date, string parsing), security access, printing modules, object model etc. These projects are not public and not accessible outside the company and cannot be hosted on the public central repository.



## http://www.wideskills.com/sites/default/files/subjects/Maven%20Tutorial/04/m02.png

Maven can change its properties on the fly (at runtime).This means that Maven can be customized in different ways in order to support any software environment:



Running Maven requires write access to the ~/.m2 home directory and network access to download dependencies.

If the environment is restricted or a firewall existsthen in the file settings.xml inside the tag <proxies> the protocol, host, proxy password and user have to be added inside in order to have full network access to download the dependencies:

|  |  |
| --- | --- |
|  | <settings>  …  <proxies>  <proxy>  <active>true</active>  <protocol>http</protocol>  <host>proxy.somewhere.com</host>  <port>8080</port>  <username>proxyuser</username>  <password>proxypassssword</password>  <nonProxyHosts></nonProxyHosts>  </proxy>  </proxies>  …  </settings> |

**Running in Offline Mode**

If the dependencies are checked only locally then Maven can be run in offline mode called **Maven offline mode**. In the command line it has to be added the argument **–o**

**mvn –o clean install**

**mv –o test**

Instead of adding  –o argument to CLI, this could be mentioned in **settings.xml**where the offline tag has to be set to **true**:

|  |  |
| --- | --- |
|  | <settings>  …  <localrepository> c:\temp</localrepository>  <offline>true</offline>  …  </settings> |

If a dependency is not available the build will fail.  This can be avoided by installing manually the dependencies from a packaged file (jar, war, ear):

|  |  |
| --- | --- |
|  | mvninstall:install –file –DgroupId=%GROUP\_ID%  -DartifactId=%ARTIFACT\_ID%  -Dversion=%VERSION%  -Dfile=%COMPONENT%.jar –Dpackaging=jar |

**-DgeneratePOM=true**

## ****Building with Maven****

Maven is an open source tool for Java enterprise projects designed to make easier the build process. Once understood Maven helps to save time with all types of projects – small or big.

Maven carries out of repetitive tasks increasing the productivity.

## ****6.1 Build****

Build is a process by which source code is converted in a stand-alone form that can be run on a computer. The most important part of a build is the compilation process where source code files are converted in executable code.

Building process is managed by a build tool. Builds are created when the code is ready for implementation, for testing or for release.

Building includes one of more of the following activities:

* Source code generating (auto-generating code)
* Documentation and release notes generating from source code
* Compiling source code to binary code
* Packaging of binary code
* Running automated tests
* Installing the packaged code on a server or in a repository

### **Build lifecycles**

In Maven the process of building and distributing an artifact (project) is precisely defined. The process represents a central concept of Maven: **build lifecycle**.

For building there are necessary some commands to be learnt and with the help of the POM the build will drive to the desired results.

 Built-in Maven Lifecycles are: **default**, **clean** and **site**.

The build lifecycles are composed by build phases and a build phase represents a stage in a lifecycle. Phases give order to a set of goals. Goals are bound to phases. Goals that are not bound to the phases can be executed outside the build lifecycle by directly calling them.

Build phases are executed sequentially to complete a build lifecycle.

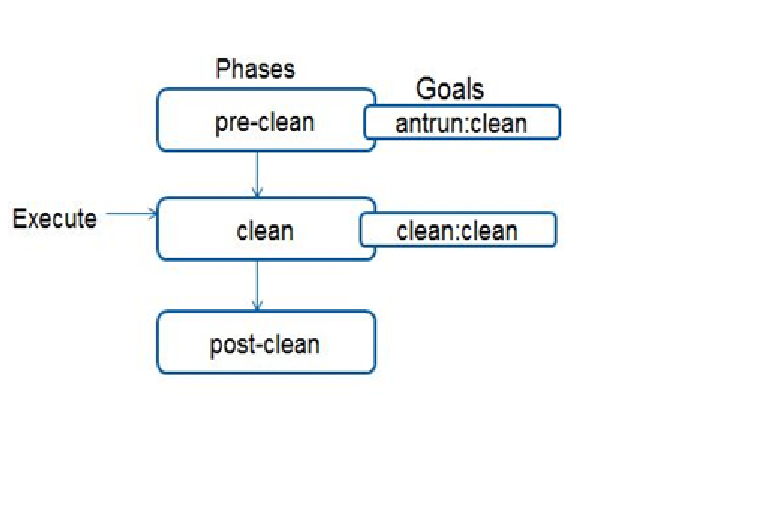
#### **Clean**

Clean is the simplest lifecycle in Maven. It can be run with the command

**mvn clean**

This lifecycle is composed of three lifecycle phases:

* **Pre-clean**
* **Clean**
* **Post-clean**

****

After running clean Maven will delete the build directory. If the location of the build directory is not customized then this is ${basedir}/target directory defined by the Super POM.

#### **Default**

If the default lifecycle is applied for a project then the builds becomes automated.

The main phases of the default lifecycle are:

|  |  |
| --- | --- |
| **validate** | validates the project information |
| **compile** | compiles the source code |
| **Test** | test unit tests with a test framework |
| **package** | package code into distributable format (JAR) |
| **integration-test** | processes the package in the integration test environment where integration tests can be run |
| **verify** | runs checks to verify if package is valid |
| **install** | installs the package in the local repository |
| **deploy** | installs the final package in the remote repository |

#### **Skip tests**

Tests can be skipped by adding the argument **“-Dmaven.test.skip=true”**in command line

**Mvn install –Dmaven.test.skip=true**

**Mvn package –Dmaven.test.skip=true**

**In maven-surefire-plugin  defineskipTests tag:**

|  |  |
| --- | --- |
|  | <plugin>      <groupId>org.apache.maven.plugins</groupId>      <artifactId>maven-surefire-plugin</artifactId>      <version>2.12.4</version>      <configuration>          <skipTests>true</skipTests>      </configuration>  </plugin> |

In Eclipse it can be checked the Skip Tests option from the window Run Configurations:

#### **Site**

Maven site generator can create a project with professional quality and low maintenance. This kind of site is useful in a company for technical project information and team communication.

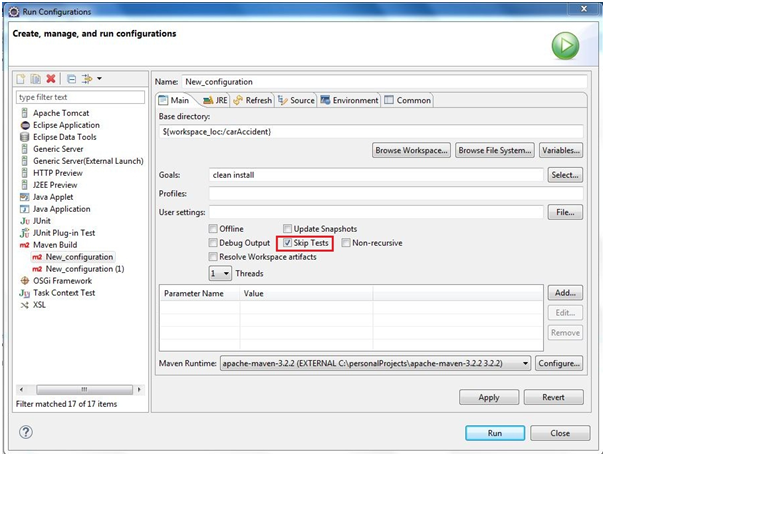
Maven sites are used on open source projects. A project site contains information about the project, some reports generated and project specific content.

This lifecycle contains the following phases

* **Pre-site**
* **Site**
* **Post-site**
* **Site-deploy**

It can be generated from command-line with the statement:

**mvn site**



**Build profiles**

A project can be configured with Maven differently for different environments. Different instances will be generated depending on the target environment. In this scope a set of Maven profiles can be used: one profile for each environment.

For the test environment for example the test profile can be activated in order to get test configuration.

Different servers need to have a different configuration of the application. That could be different database connection configurations or logging configuration that should be visible only during development.

Using a profile for build does not means that the resulting artifact after build will be identical with the artifact that should result without profile.

The profiles can be created in POM.xml , for example if we want to create two profiles, one for development and one for production we have to add in POM.xml the following:

[?](http://www.wideskills.com/maven/building-maven)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | <project>  ...  <profiles>  <profile>  <id>profile-dev</id>  <activation>  <activeByDefault>true</activeByDefault>  <property>  <name>environment</name>  <value>dev</value>  </property>  </activation>  </profile>  <profile>  <id>profile-prod</id>  <activation>  <property>  <name>environment</name>  <value>prod</value>  </property>  </activation>  </profile>  </profiles>  ...  </project> |
|  |  |

The id of the created profiles is represented by the name of the profiles. Inside of the tag activation there is the tag name that specify that the profile should be activated by the variable environment.  The variable can be passed to Maven while doing a new build.

For example:

**mvn clean compile package –Denvironment=prod**

In order to check which profile is currently active it has to be used the following command:

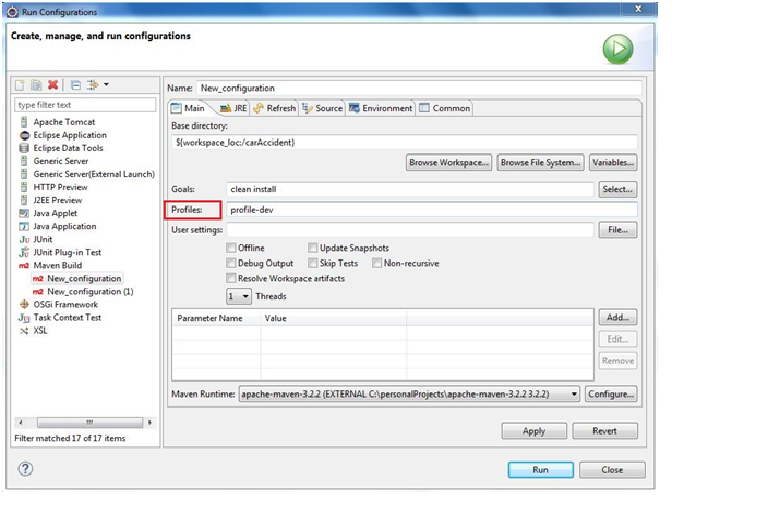
**mvn clean compile package –Denvironment=prodhelp:active-profiles**

In the console will be displayed the result:

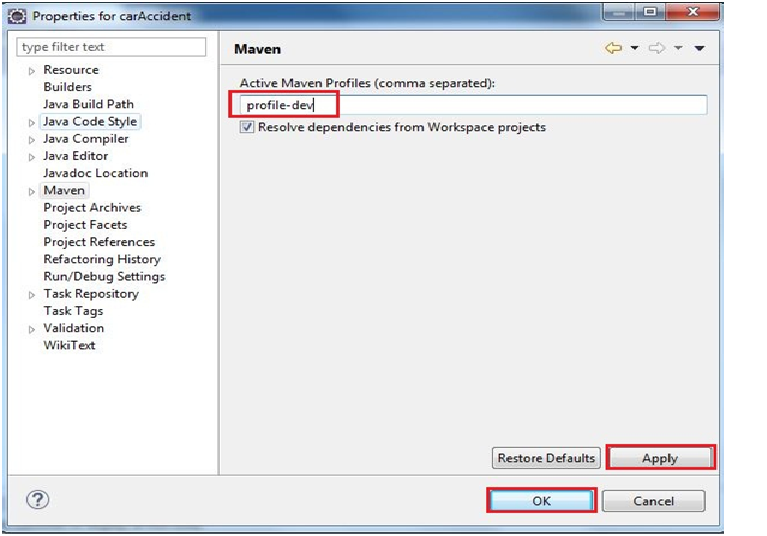
[?](http://www.wideskills.com/maven/building-maven)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | [INFO]  Active Profiles for Project 'com.myarchetype:archetype-example:jar:1.0-SNAPSHOT':  The following profiles are active:   - profile-prod (source: com.myarchetype:archetype-example:1.0-SNAPSHOT)  [INFO] ------------------------------------------------------------------------  [INFO] BUILD SUCCESS  [INFO] ------------------------------------------------------------------------  [INFO] Total time: 23.653 s  [INFO] Finished at: 2014-10-17T00:29:47+03:00  [INFO] Final Memory: 12M/30M  [INFO] ------------------------------------------------------------------------ |

In eclipse the profile can be declared in the window **Run Configuration:**



If the profile applies only for a module then in the properties of the module the profile has to be specified too (right click in Eclipse on the Module then select Properties):



If a profile is used to exclude some folders at deployment in production environment then this has to be configured with the tag **<packagingExcludes>**in POM file.

|  |  |
| --- | --- |
|  | <project>  ….  <profiles>      <profile>          <id>profile-prod</id>          <activation>          <activeByDefault>true</activeByDefault>          </activation>          <build>              <plugins>                  <plugin>                  <artifactId>archetype-example</artifactId>                      <configuration>   <packagingExcludes>WEB-INF/cms</packagingExcludes>                      </configuration>                      </plugin>                  </plugins>              </build>          </profile>  <profiles>  …  </project> |

## <https://www.baeldung.com/maven-profiles>