Contents

[MAVEN 2](#_Toc81216921)

[MAVEN PROJECT 2](#_Toc81216922)

[CREATING A SIMPLE MAVEN PROJECT 2](#_Toc81216923)

[MAVEN PHASES, PLUGINS AND GOALS 2](#_Toc81216924)

[MAVEN COORDINATES 4](#_Toc81216925)

[MAVEN SKIP TEST 4](#_Toc81216926)

[MAVEN SCOPES 4](#_Toc81216927)

[MAVEN MULTI MODULE PROJECT 5](#_Toc81216928)

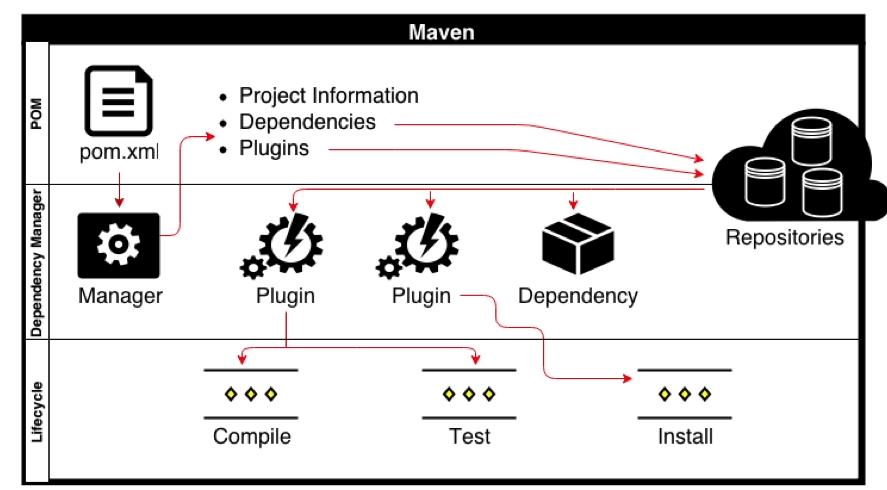
[POM.XML 6](#_Toc81216929)

[MANAGING VERSION OF DEPENDENCIES AND PLUGIN IN MULTI MODULE PROJECT 8](#_Toc81216930)

[MAVEN PROFILES 11](#_Toc81216931)

# MAVEN

* Maven is build and project management tool
* Build the project includes compiling, running the unit and integration tests, packaging the project and deploying. All these tasks can be automated using build tool like Maven
* Maven uses convention over configuration – means maven creates a basic project structure for us and we need to follow the structure rather than we creating the project structures.
* Maven provides templates for different project types known as archetype. This create the project structure based on these archetypes.
* Maven uses plugin Model.



## MAVEN PROJECT

### CREATING A SIMPLE MAVEN PROJECT

**mvn archetype:generate -DgroupId=com.demoapp -DartifactId=hellomaven -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false**

|  |  |
| --- | --- |
| **archetype:generate** | “archetype” is plugin and “generate” is one of the goal in “archetype” plugin . It takes parameters using “-D” option .They are called goal parameters |
| **groupId** | Package Name. With this name our project is stored in the maven repo |
| **artifactId** | Name of project |
| **archetypeArtifactId** | Project Type |

Note: Add in POM.xml for Java Version

**<properties>**

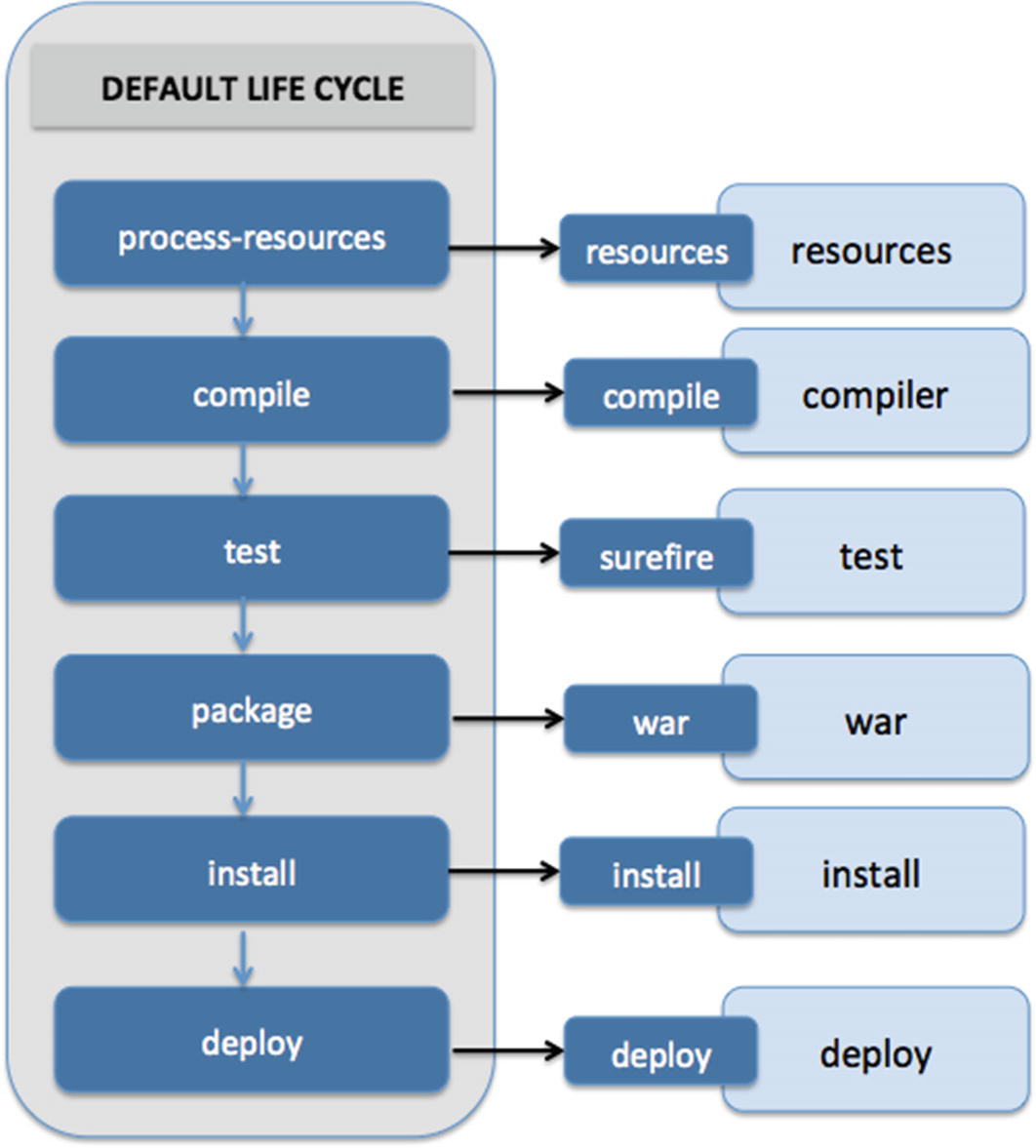
**<maven.compiler.source>11</maven.compiler.source>**

**<maven.compiler.target>11</maven.compiler.target>**

**</properties>**

### MAVEN PHASES, PLUGINS AND GOALS

* A maven plugin is collection one or more goals. For example **archetype:generate or install:install –** where **archetype and install**  are plugins and **generate and install** are goals.
* Goals can be a specific task of part of bigger task.



* Maven has life cycle phases. Each phase is associated with a goal. For example, “test” phase is associated with “test“ goal of “surefire” plugin
* When we execute a goal for example- ***mvn install:install.*** It will run all the previous life cycle phases and execute the goal associated with each phase.

#### MAVEN COMPILER PLUGIN

**The***compiler***plugin is used to compile the source code of a Maven project**. This plugin has two goals, which are already bound to specific phases of the default lifecycle:

* *compile***–** compile main source files
* *testCompile***–** compile test source files

##### **CONFIGURATION**

**By default, the***compiler***plugin compiles source code compatible with Java 5, and the generated classes also work with Java 5 regardless of the JDK in use.** We can modify these settings in the *configuration* element:

<**configuration**>

<**source**>1.8</**source**>

<**target**>1.8</**target**>

<-- other customizations -->

</**configuration**>

For convenience, we can set the Java version as properties of the POM:

<**properties**>

<**maven.compiler.source**>1.8</**maven.compiler.source**>

<**maven.compiler.target**>1.8</**maven.compiler.target**>

</**properties**>

###### EXAMPLE

|  |  |
| --- | --- |
| <build>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>3.7.0</version>  <configuration>  <maven.compiler.source>11</maven.compiler.source>  <maven.compiler.target>11</maven.compiler.target>  </configuration>  </plugin>  </plugins>  </build> | Enabling Java 11 in properties |

### MAVEN COORDINATES

|  |  |
| --- | --- |
| <groupId>com.demoapp</groupId>  <artifactId>hellomaven</artifactId>  <packaging>jar</packaging>  <version>1.0-SNAPSHOT</version> | * groupId,artifactId, packaging and version are called maven coordinates. * The uniquely define the location of the package of the project in Maven Repo * It defines the name of the package of the project |

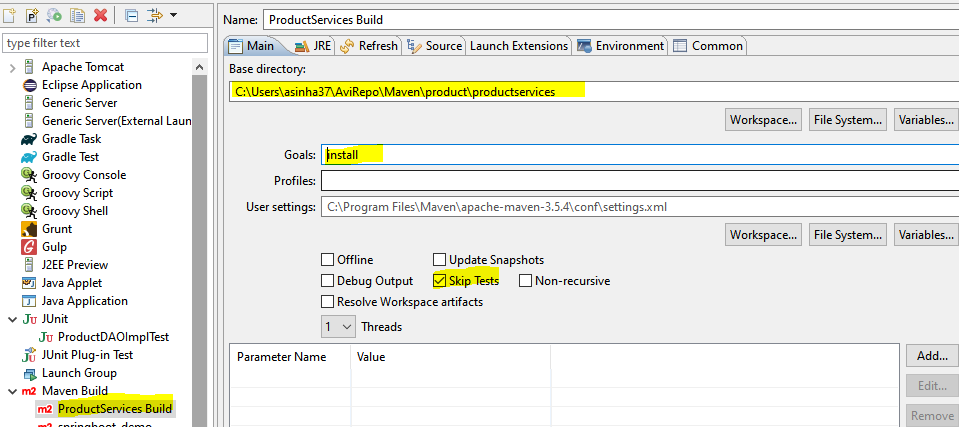
### MAVEN SKIP TEST

We can skip the unit/integration test during maven build. To skip the test pass “***skipTests***” as goal parameter

* From command Prompt: **mvn install -DskipTests**

#### SKIPPING TEST IN ECLIPSE

* Right Click on project 🡪 Run Configurations



### MAVEN SCOPES

**Maven has 6 default dependency scopes**.

#### COMPILE

|  |  |
| --- | --- |
| * **This is the default scope when no other scope is provided.** * Dependencies with this scope are available on the classpath of the project in all build tasks. | <dependency>  <groupId>commons-lang</groupId>  <artifactId>commons-lang</artifactId>  <version>2.6</version>  </dependency> |

#### PROVIDED

|  |  |
| --- | --- |
| * This scope is used to mark **dependencies that should be provided at runtime by JDK or a container**. * A good use case for this scope would be a web application deployed in some container, where the container already provides some libraries itself. * The *provided*dependencies are available only at compile-time and in the test classpath of the project. | For example, a web server that already provides the Servlet API at runtime, thus in our project, those dependencies can be defined with the ***provided*** scope:  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>servlet-api</artifactId>  <version>2.5</version>  <scope>provided</scope>  </dependency> |

#### RUNTIME

|  |  |
| --- | --- |
| * **The dependencies with this scope are required at runtime**, but they're not needed for compilation of the project code. * Dependencies marked with the *runtime* scope will be present in runtime and test classpath, but they will be missing from compile classpath. | A good example of dependencies that should use the runtime scope is a JDBC driver:  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>6.0.6</version>  <scope>runtime</scope>  </dependency> |

#### TEST

|  |  |
| --- | --- |
| * This scope is used to indicate that dependency isn't required at standard runtime of the application, but is used only for test purposes. * The standard use case for this scope is adding test library like JUnit to our application. | <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.12</version>  <scope>test</scope>  </dependency> |

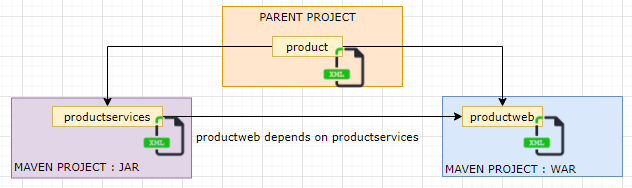
#### SYSTEM

|  |  |
| --- | --- |
| * [***system*** scope is deprecated](https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html#system-dependencies) * ***system* scope is much like the***provided***scope.** The main difference between those two scopes is that *system* requires us to directly point to a specific jar on the local machine. * The important thing to remember is that building the project with *system* scope dependencies may fail on different machines if dependencies aren't present in the same path | <dependency>  <groupId>com.baeldung</groupId>  <artifactId>custom-dependency</artifactId>  <version>1.3.2</version>  <scope>system</scope>  <systemPath>${project.basedir}/libs/custom-dependency-1.3.2.jar</systemPath>  </dependency> |

#### IMPORT

|  |  |
| --- | --- |
| * This scope was added in Maven 2.0.9 and **it's only available for the dependency type***pom***.** | <dependency>  <groupId>com.baeldung</groupId>  <artifactId>custom-project</artifactId>  <version>1.3.2</version>  <type>pom</type>  <scope>import</scope>  </dependency> |

## MAVEN MULTI MODULE PROJECT



* In the multimodule maven project - “product” is the parent project
* The productweb project is dependent upon the productservices project.

### POM.XML

#### PARENT POM

* We need to create a pom.xml at the parent folder and include the child projects as module.
* The packaging of the parent project is “**pom**”
* Assign suitable maven coordinates to parent project.

|  |
| --- |
| <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.products.crud</groupId>  <artifactId>product</artifactId>  <version>1.0</version>  <packaging>pom</packaging>  <name>product</name>  <url>http://maven.apache.org</url>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <maven.compiler.source>11</maven.compiler.source>  <maven.compiler.target>11</maven.compiler.target>  </properties>  <dependencies>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.4</version>  <scope>test</scope>  </dependency>  </dependencies>  <modules>  <module>productservices</module>  <module>productweb</module>  </modules>  </project> |

#### CHILD POM

##### productservices- pom

* Refer the parent project using the “parent”

|  |
| --- |
| <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>  <parent>  <groupId>com.products.crud</groupId>  <artifactId>product</artifactId>  <version>1.0</version>  </parent>  <artifactId>productservices</artifactId>  <packaging>jar</packaging>  <name>productservices</name>  <url>http://maven.apache.org</url>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <maven.compiler.source>11</maven.compiler.source>  <maven.compiler.target>11</maven.compiler.target>  </properties>  <dependencies>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.4</version>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>5.3.9</version>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>3.7.0</version>  <configuration>  <maven.compiler.source>11</maven.compiler.source>  <maven.compiler.target>11</maven.compiler.target>  </configuration>  </plugin>  </plugins>  </build>  </project> |

##### productweb- pom

|  |
| --- |
| <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/maven-v4\_0\_0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>com.products.crud</groupId>  <artifactId>product</artifactId>  <version>1.0</version>  </parent>  <groupId>com.products.crud.web</groupId>  <artifactId>productweb</artifactId>  <packaging>war</packaging>  <name>productweb Maven Webapp</name>  <url>http://maven.apache.org</url>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <maven.compiler.source>11</maven.compiler.source>  <maven.compiler.target>11</maven.compiler.target>  </properties>  <dependencies>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.4</version>  <scope>test</scope>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>javax.servlet-api</artifactId>  <version>4.0.1</version>  <scope>provided</scope>  </dependency>  <dependency>  <groupId>com.products.crud</groupId>  <artifactId>productservices</artifactId>  <version>1.0</version>  </dependency>  </dependencies>  <build>  <finalName>productweb</finalName>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>3.7.0</version>  <configuration>  <maven.compiler.source>11</maven.compiler.source>  <maven.compiler.target>11</maven.compiler.target>  </configuration>  </plugin>  </plugins>  </build>  </project> |

### MANAGING VERSION OF DEPENDENCIES AND PLUGIN IN MULTI MODULE PROJECT

* For the consistency of version of dependencies and plugins we declare the versions of dependencies and plugins in parent pom – which the inherited by all child poms/ projects.

#### PARENT POM

|  |
| --- |
| <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.products.crud</groupId>  <artifactId>product</artifactId>  <version>1.0</version>  <packaging>pom</packaging>  <name>product</name>  <url>http://maven.apache.org</url>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <maven.compiler.source>11</maven.compiler.source>  <maven.compiler.target>11</maven.compiler.target>  </properties>  <dependencies>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <scope>test</scope>  </dependency>  </dependencies>  <dependencyManagement>  <dependencies>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.4</version>  </dependency>  </dependencies>  </dependencyManagement>  <build>  <pluginManagement>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>3.7.0</version>  <configuration>  <maven.compiler.source>11</maven.compiler.source>  <maven.compiler.target>11</maven.compiler.target>  </configuration>  </plugin>  </plugins>  </pluginManagement>  </build>  <modules>  <module>productservices</module>  <module>productweb</module>  </modules>  </project> |

#### CHILD POM

|  |
| --- |
| <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>  <parent>  <groupId>com.products.crud</groupId>  <artifactId>product</artifactId>  <version>1.0</version>  </parent>  <artifactId>productservices</artifactId>  <packaging>jar</packaging>  <name>productservices</name>  <url>http://maven.apache.org</url>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <maven.compiler.source>11</maven.compiler.source>  <maven.compiler.target>11</maven.compiler.target>  </properties>  <dependencies>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>5.3.9</version>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |
| <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/maven-v4\_0\_0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>com.products.crud</groupId>  <artifactId>product</artifactId>  <version>1.0</version>  </parent>  <groupId>com.products.crud.web</groupId>  <artifactId>productweb</artifactId>  <packaging>war</packaging>  <name>productweb Maven Webapp</name>  <url>http://maven.apache.org</url>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <maven.compiler.source>11</maven.compiler.source>  <maven.compiler.target>11</maven.compiler.target>  </properties>  <dependencies>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <scope>test</scope>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>javax.servlet-api</artifactId>  <version>4.0.1</version>  <scope>provided</scope>  </dependency>  <dependency>  <groupId>com.products.crud</groupId>  <artifactId>productservices</artifactId>  <version>1.0</version>  </dependency>  </dependencies>  <build>  <finalName>productweb</finalName>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

## MAVEN PROFILES

* We use maven profiles for build portability. For example – if we have configuration which is different in different environment(dev/test/prod)- we create profiles pertaining to each environment.
* The specific configuration is picked up when a particular profile is activated

|  |  |
| --- | --- |
|  | * Create a “Source Folder” – **src/main/profiles** * Create a folder (e.g. **dev, prod, test**) where we can keep configuration files specific to a particular profile. * Create configuration file(application.properties) and configure some properties     **TO ACTIVATE A PARTICULAR PROFILE**   * **TO ACTIVATE DEV PROFILE: *mvn clean install -pdev*** * **TO ACTIVATE TEST PROFILE: *mvn clean install -ptest*** * **TO ACTIVATE PROD PROFILE: *mvn clean install -pprod*** |
| <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.profiles.demo</groupId>  <artifactId>profilesdemo</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>jar</packaging>  <name>profilesdemo</name>  <url>http://maven.apache.org</url>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <maven.compiler.source>11</maven.compiler.source>  <maven.compiler.target>11</maven.compiler.target>  </properties>  <dependencies>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>3.8.1</version>  <scope>test</scope>  </dependency>  </dependencies>  <profiles>  <profile>  <id>dev</id>  <properties>  <build.profile.id>dev</build.profile.id>  </properties>  <build>  <resources>  <resource>  <directory>src/main/profiles/dev</directory>  </resource>  </resources>  </build>  </profile>  <profile>  <id>test</id>  <properties>  <build.profile.id>test</build.profile.id>  </properties>  <build>  <resources>  <resource>  <directory>src/main/profiles/test</directory>  </resource>  </resources>  </build>  </profile>  <profile>  <id>prod</id>  <properties>  <build.profile.id>prod</build.profile.id>  </properties>  <build>  <resources>  <resource>  <directory>src/main/profiles/prod</directory>  </resource>  </resources>  </build>  </profile>  </profiles>  </project> | |