The Project Object Model (POM)

The Project Object Model (POM)

A Maven2-enabled project always comes with a *pom.xml* file which contains the Project Object Model (POM) for this project. The POM contains every important piece of information about your project, which can be your project description, versioning or distribution information, dependencies and much more. It is essentially a one-stop-shopping for finding anything related to your project.

In its most basic form, a POM has the following structure :

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns:xsi="http://www.w3.org/2001/XML-Schema-instance"
xmlns="http://maven.apache.org/POM/4.0"
xsi:schemaLocation="http://maven.apache.org/POM/4.0
http://maven.apache.org/maven-v4_0_0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>org.apache</groupId>
<artifactId>maven</artifactId>
<packaging>jar</packaging>
<version>2.0</version>
<name>Maven 2.0</name>
<url>http://maven.apache.org</url>
</project>
```

Encoding
Since every bit of information about your project comes from the POM, it is very important to take the time to set up the encoding correctly. You can't imagine how many problems it will save you from later on.

You can already spot a few key elements :

- **project** This is the top-level element in all Maven pom.xml files.
- **modelVersion**This element indicates what version of the object model this POM is using. The version of the model itself changes very infrequently but it is mandatory in order to ensure stability of use if and when the Maven developers deem it necessary to change the model.
- **groupId**This element indicates the unique identifier of the organization or group that created the project. The groupId is one of the key identifiers of a project and will be fully explored in the Naming Conventions lesson.
- **artifactId**This element indicates the unique base name of the primary artifact being generated by this project. The primary artifact for a project is typically a JAR file. Secondary artifacts like source bundles also use the artifactId as part of their final name. A typical artifact produced would have the form <artifactId>-<version>.<extension> (for example, `myapp-1.0.jar`). It is another key identifier of a project and is fully explored in the Naming Conventions lesson.
- **packaging**This element indicates the package type to be used by this artifact (e.g. JAR, WAR, EAR, etc.). This not only means if the artifact produced is JAR, WAR, or EAR but can also indicate a specific lifecycle to use as part of the build process. (The lifecycle is a topic we will deal with further on in the guide. For now, just keep in mind that the indicated packaging of a project can play a part in customizing the build lifecycle.) The default value for the `packaging` element is JAR so you do not have to specify this for most projects.
- **version**This element indicates the version of the artifact generated by the project. Maven goes a long way to help you with version management. It is another key identifier of a project which is fully explained in the Naming Conventions lesson.
- **name** This element indicates the display name used for the project. This is often used in Maven's generated documentation.
- **url** This element indicates where the project's site can be found. This is often used in Maven's generated documentation.
- **description** This elements provides a basic description of your project. This is often used in Maven's generated documentation.

One POM to rule them all

The POM is the basic unit of work in Maven. This is important to remember because Maven is inherently project-centric in that everything revolves around the notion of a project. Every task you ask Maven to perform will be conducted based upon the information found into your POM file. For instance, let's say you want to package your project. Before running the task, Maven will first read the packaging element of your POM to see what kind of archive is supposed to be generated for this project. Only then after knowing *what* you want, it will decide *how*to perform the operation based upon the retrieved value. As it was mentioned in the previous lesson, it is one of the main differences between Maven and the more traditional build tools.

Storing your POM
The POM is the central piece of information of your project. It has to be shared accross the team project members and a changes history should be kept as with any source code. Therefore, the project POM file should always be kept in your SCM repository at the very root of the project. From now on, this tutorial assumes you adhere to this rule.

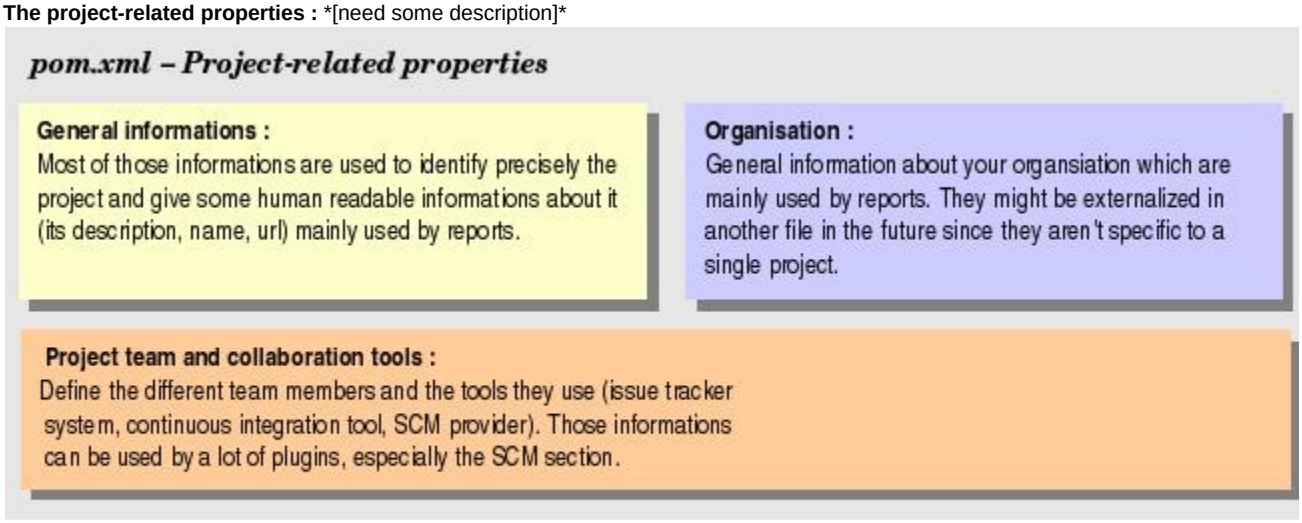
It may seem very obvious but it's worth repeating that you should make sure your POM is always up-to-date as the project evolves. The faster your team finds integration issues, the easier they will be to fix. You should tackle them as early as possible!

What is a POM made of ?

If you look at all the elements that can go in a POM file, you'll be lost. The Maven team has defined a LOT of standard information (coming mostly from Maven 1 experience) that can be put in your POM file. Honestly, I don't suggest this exciting reading to anyone unless you are specifically looking for some properties.

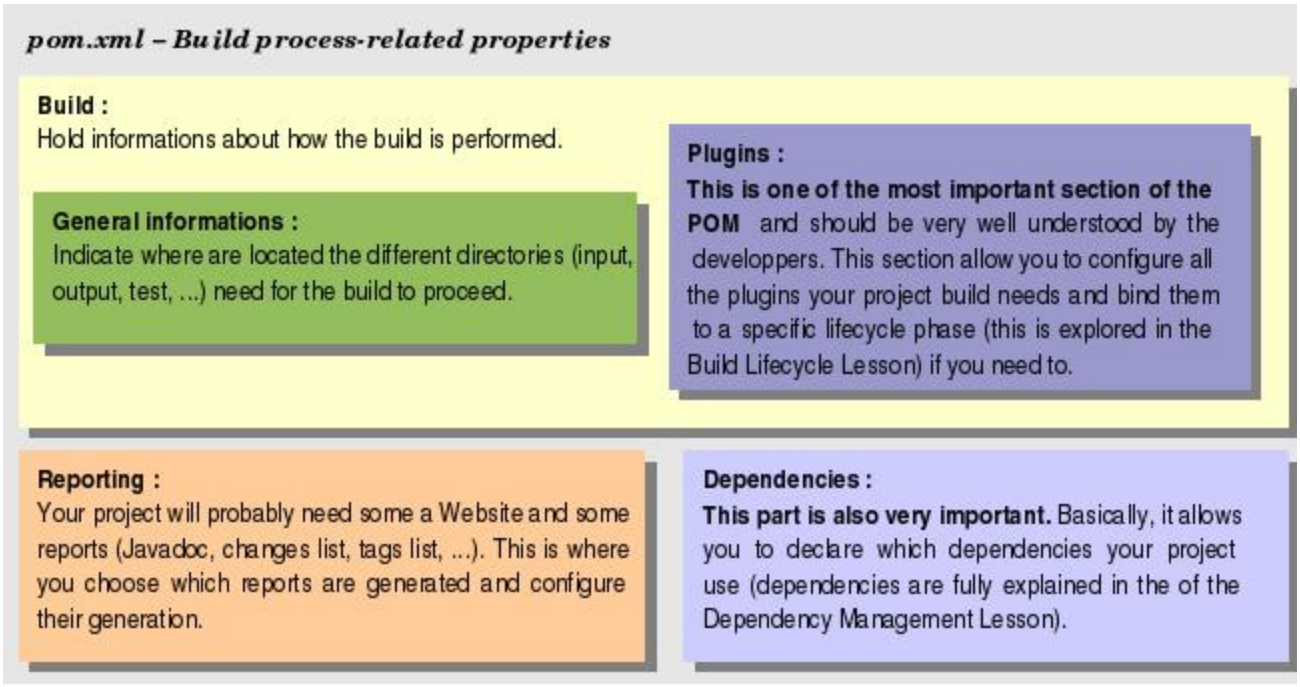
The POM can basically be decomposed into three different sections :

- **The project-related properties :** *[need some description]*



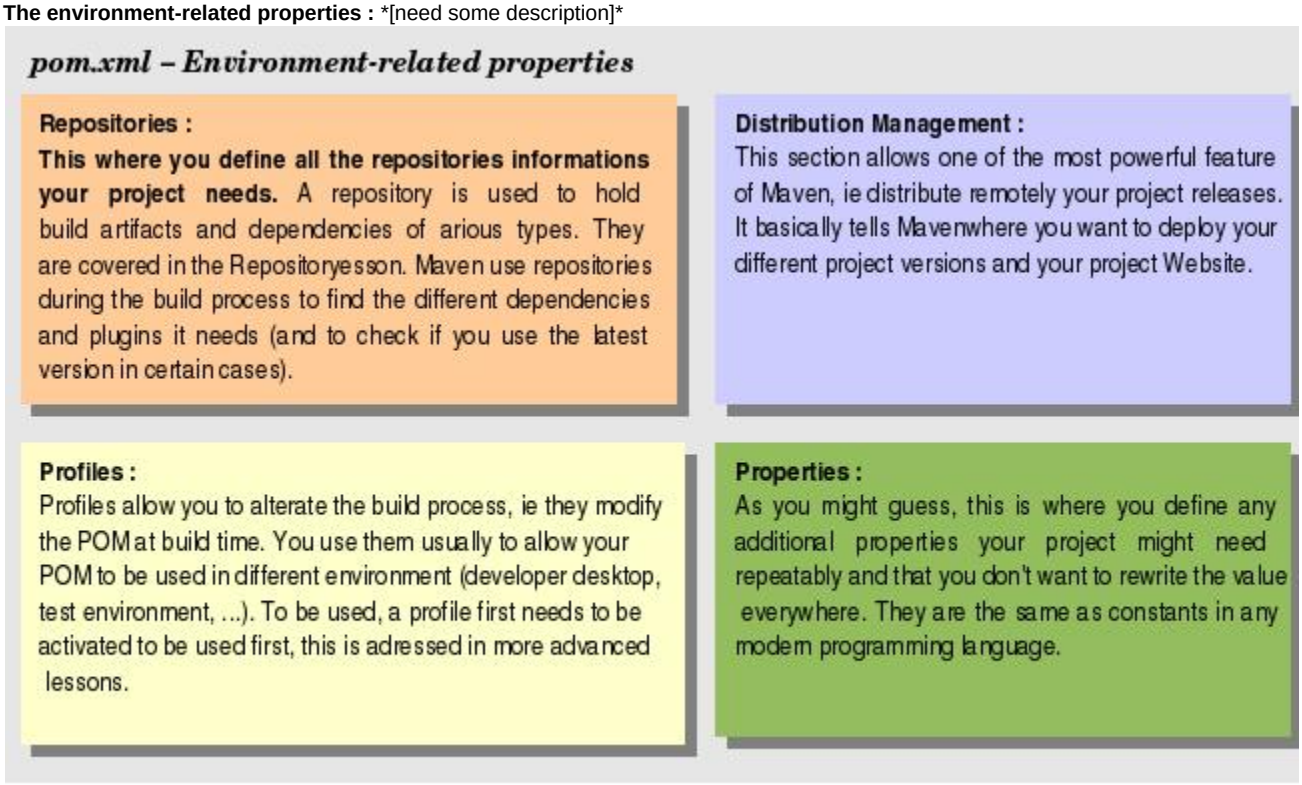
The project-related properties

- **The build process-related properties :** *[need some description]*



Build process-related properties

- **The environment-related properties :** *[need some description]*



Environment-related properties

- **The overall POM structure :** *[need some description]*

Overall pom.xml structure

POM elements order
The POM has restrictions on the different elements order (since its structure is validated by an XML schema). I have chosen to regroup the properties to make them easier to read.