Introduction to the POM

What is a POM?

A Project Object Model or POM is the fundamental unit of work in Maven. It is an XML file that contains information about the project and configuration details used by Maven to build the project. It contains default values for most projects. Examples for this is the build directory, which is `target`; the source directory, which is `src/main/java`; the test source directory, which is `src/test/java`; and so on. When executing a task or goal, Maven looks for the POM in the current directory. It reads the POM, gets the needed configuration information, then executes the goal.

Some of the configuration that can be specified in the POM are the project dependencies, the plugins or goals that can be executed, the build profiles, and so on. Other information such as the project version, description, developers, mailing lists and such can also be specified.

[top] (./introduction-to-the-pom.html)

Super POM

The Super POM is Maven's default POM. All POMs extend the Super POM unless explicitly set, meaning the configuration specified in the Super POM is inherited by the POMs you created for your projects.

You can see the Super POM for Maven 3.6.3 (/ref/3.6.3/maven-model-builder/super-pom.html) in Maven Core reference documentation.

[top] (./introduction-to-the-pom.html)

Minimal POM

The minimum requirement for a POM are the following:

- `project` root
- `modelVersion` - should be set to 4.0.0
- `groupId` - the id of the project's group.
- `artifactId` - the id of the artifact (project)
- `version` - the version of the artifact under the specified group

Here's an example:

```
1. <project>
2.   <modelVersion>4.0.0</modelVersion>
3.
4.   <groupId>com.mycompany.app</groupId>
5.   <artifactId>my-app</artifactId>
6.   <version>1</version>
7. </project>
```

A POM requires that its groupId, artifactId, and version be configured. These three values form the project's fully qualified artifact name. This is in the form of <groupId>:<artifactId>:<version>. As for the example above, its fully qualified artifact name is "com.mycompany.app:my-app:1".

Also, as mentioned in the first section, if the configuration details are not specified, Maven will use their defaults. One of these default values is the packaging type. Every Maven project has a packaging type. If it is not specified in the POM, then the default value "jar" would be used.

Furthermore, you can see that in the minimal POM the *repositories* were not specified. If you build your project using the minimal POM, it would inherit the *repositories* configuration in the Super POM. Therefore when Maven sees the dependencies in the minimal POM, it would know that these dependencies will be downloaded from `https://repo.maven.apache.org/maven2` which was specified in the Super POM.

[top] (./introduction-to-the-pom.html)

Project Inheritance

Elements in the POM that are merged are the following:

- dependencies
- developers and contributors
- plugin lists (including reports)
- plugin executions with matching ids
- plugin configuration
- resources

The Super POM is one example of project inheritance, however you can also introduce your own parent POMs by specifying the parent element in the POM, as demonstrated in the following examples.

Example 1

The Scenario

As an example, let us reuse our previous artifact, com.mycompany.app:my-app:1. And let us introduce another artifact, com.mycompany.app:my-module:1.

```
1. <project>
2.   <modelVersion>4.0.0</modelVersion>
3.
4.   <groupId>com.mycompany.app</groupId>
5.   <artifactId>my-module</artifactId>
6.   <version>1</version>
7. </project>
```

And let us specify their directory structure as the following:

```
.
 |-- my-module
 |   `-- pom.xml
 `-- pom.xml
```

Note: `my-module/pom.xml` is the POM of com.mycompany.app:my-module:1 while `pom.xml` is the POM of com.mycompany.app:my-app:1

The Solution

Now, if we were to turn com.mycompany.app:my-app:1 into a parent artifact of com.mycompany.app:my-module:1,we will have to modify com.mycompany.app:my-module:1's POM to the following configuration:

com.mycompany.app:my-module:1's POM

```
 1. <project>
 2.   <modelVersion>4.0.0</modelVersion>
 3.
 4.   <parent>
 5.     <groupId>com.mycompany.app</groupId>
 6.     <artifactId>my-app</artifactId>
 7.     <version>1</version>
 8.   </parent>
 9.
10.   <groupId>com.mycompany.app</groupId>
11.   <artifactId>my-module</artifactId>
12.   <version>1</version>
13. </project>
```

Notice that we now have an added section, the parent section. This section allows us to specify which artifact is the parent of our POM. And we do so by specifying the fully qualified artifact name of the parent POM. With this setup, our module can now inherit some of the properties of our parent POM.

Alternatively, if you want the groupId or the version of your modules to be the same as their parents, you can remove the groupId or the version identity of your module in its POM.

```
 1. <project>
 2.   <modelVersion>4.0.0</modelVersion>
 3.
 4.   <parent>
 5.     <groupId>com.mycompany.app</groupId>
 6.     <artifactId>my-app</artifactId>
 7.     <version>1</version>
 8.   </parent>
 9.
10.   <artifactId>my-module</artifactId>
11. </project>
```

This allows the module to inherit the groupId or the version of its parent POM.

[top] (./introduction-to-the-pom.html)

Example 2

The Scenario

However, that would work if the parent project was already installed in our local repository or was in that specific directory structure (parent `pom.xml` is one directory higher than that of the module's `pom.xml`).

But what if the parent is not yet installed and if the directory structure is as in the following example?

```
.
 |-- my-module
 |   `-- pom.xml
 `-- parent
     `-- pom.xml
```

The Solution

To address this directory structure (or any other directory structure), we would have to add the `<relativePath>` element to our parent section.

```
 1. <project>
 2.   <modelVersion>4.0.0</modelVersion>
 3.
 4.   <parent>
 5.     <groupId>com.mycompany.app</groupId>
 6.     <artifactId>my-app</artifactId>
 7.     <version>1</version>
 8.     <relativePath>../parent/pom.xml</relativePath>
 9.   </parent>
10.
11.   <artifactId>my-module</artifactId>
12. </project>
```

As the name suggests, it's the relative path from the module's `pom.xml` to the parent's `pom.xml`.

Project Aggregation

Project Aggregation is similar to Project Inheritance. But instead of specifying the parent POM from the module, it specifies the modules from the parent POM. By doing so, the parent project now knows its modules, and if a Maven command is invoked against the parent project, that Maven command will then be executed to the parent's modules as well. To do Project Aggregation, you must do the following:

- Change the parent POMs packaging to the value "pom".
- Specify in the parent POM the directories of its modules (children POMs).

[top] (./introduction-to-the-pom.html)

Example 3

The Scenario

Given the previous original artifact POMs and directory structure:

com.mycompany.app:my-app:1's POM

```
<project>
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1</version>
</project>
```

com.mycompany.app:my-module:1's POM

```
<project>
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.app</groupId>
  <artifactId>my-module</artifactId>
  <version>1</version>
</project>
```

directory structure

```
.
 |-- my-module
 |    `-- pom.xml
 `-- pom.xml
```

The Solution

If we are to aggregate my-module into my-app, we would only have to modify my-app.

```
<project>
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1</version>
  <packaging>pom</packaging>

  <modules>
    <module>my-module</module>
  </modules>
</project>
```

In the revised com.mycompany.app:my-app:1, the packaging section and the modules sections were added. For the packaging, its value was set to "pom", and for the modules section, we have the element `<module>my-module</module>`. The value of `<module>` is the relative path from the com.mycompany.app:my-app:1 to com.mycompany.app:my-module:1's POM (*by practice, we use the module's artifactId as the module directory's name*).

Now, whenever a Maven command processes com.mycompany.app:my-app:1, that same Maven command would be ran against com.mycompany.app:my-module:1 as well. Furthermore, some commands (goals specifically) handle project aggregation differently.

[top] (./introduction-to-the-pom.html)

Example 4

The Scenario

But what if we change the directory structure to the following:

```
.
 |-- my-module
 |   `-- pom.xml
 `-- parent
     `-- pom.xml
```

How would the parent POM specify its modules?

The Solution

The answer? - the same way as Example 3, by specifying the path to the module.

```
 1. <project>
 2.   <modelVersion>4.0.0</modelVersion>
 3.
 4.   <groupId>com.mycompany.app</groupId>
 5.   <artifactId>my-app</artifactId>
 6.   <version>1</version>
 7.   <packaging>pom</packaging>
 8.
 9.   <modules>
10.     <module>../my-module</module>
11.   </modules>
12. </project>
```

Project Inheritance vs Project Aggregation

If you have several Maven projects, and they all have similar configurations, you can refactor your projects by pulling out those similar configurations and making a parent project. Thus, all you have to do is to let your Maven projects inherit that parent project, and those configurations would then be applied to all of them.

And if you have a group of projects that are built or processed together, you can create a parent project and have that parent project declare those projects as its modules. By doing so, you'd only have to build the parent and the rest will follow.

But of course, you can have both Project Inheritance and Project Aggregation. Meaning, you can have your modules specify a parent project, and at the same time, have that parent project specify those Maven projects as its modules. You'd just have to apply all three rules:

- Specify in every child POM who their parent POM is.
- Change the parent POMs packaging to the value "pom" .
- Specify in the parent POM the directories of its modules (children POMs)

[top] (./introduction-to-the-pom.html)

Example 5

The Scenario

Given the previous original artifact POMs again,

com.mycompany.app:my-app:1's POM

```
<project>
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1</version>
</project>
```

com.mycompany.app:my-module:1's POM

```
<project>
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.app</groupId>
  <artifactId>my-module</artifactId>
  <version>1</version>
</project>
```

and this **directory structure**

```
.
 |-- my-module
 |    `-- pom.xml
 `-- parent
      `-- pom.xml
```

The Solution

To do both project inheritance and aggregation, you only have to apply all three rules.

com.mycompany.app:my-app:1's POM

```
<project>
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1</version>
  <packaging>pom</packaging>

  <modules>
    <module>../my-module</module>
  </modules>
</project>
```

com.mycompany.app:my-module:1's POM

```
<project>
  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>com.mycompany.app</groupId>
    <artifactId>my-app</artifactId>
    <version>1</version>
    <relativePath>../parent/pom.xml</relativePath>
  </parent>

  <artifactId>my-module</artifactId>
</project>
```

NOTE: Profile inheritance the same inheritance strategy as used for the POM itself.

[top] (./introduction-to-the-pom.html)

Project Interpolation and Variables

One of the practices that Maven encourages is *don't repeat yourself*. However, there are circumstances where you will need to use the same value in several different locations. To assist in ensuring the value is only specified once, Maven allows you to use both your own and pre-defined variables in the POM.

For example, to access the `project.version` variable, you would reference it like so:

```
  <version>${project.version}</version>
```

One factor to note is that these variables are processed *after* inheritance as outlined above. This means that if a parent project uses a variable, then its definition in the child, not the parent, will be the one eventually used.

Available Variables

Project Model Variables

Any field of the model that is a single value element can be referenced as a variable. For example, `${project.groupId}`, `${project.version}`, `${project.build.sourceDirectory}` and so on. Refer to the POM reference to see a full list of properties.

These variables are all referenced by the prefix "`project.`". You may also see references with `pom.` as the prefix, or the prefix omitted entirely - these forms are now deprecated and should not be used.

Special Variables

`project.basedir`	The directory that the current project resides in.
`project.baseUri`	The directory that the current project resides in, represented as an URI. *Since Maven 2.1.0*
`maven.build.timestamp`	The timestamp that denotes the start of the build (UTC). *Since Maven 2.1.0-M1*

The format of the build timestamp can be customized by declaring the property `maven.build.timestamp.format` as shown in the example below:

```
<project>
  ...
  <properties>
    <maven.build.timestamp.format>yyyy-MM-dd'T'HH:mm:ss'Z'</maven.build.timestamp.format>
  </properties>
  ...
</project>
```

The format pattern has to comply with the rules given in the API documentation for SimpleDateFormat (https://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html). If the property is not present, the format defaults to the value already given in the example.

Properties

You are also able to reference any properties defined in the project as a variable. Consider the following example:

```
<project>
  ...
  <properties>
    <mavenVersion>3.0</mavenVersion>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.apache.maven</groupId>
      <artifactId>maven-artifact</artifactId>
      <version>${mavenVersion}</version>
    </dependency>
    <dependency>
      <groupId>org.apache.maven</groupId>
      <artifactId>maven-core</artifactId>
      <version>${mavenVersion}</version>
    </dependency>
  </dependencies>
  ...
</project>
```

[top] (./introduction-to-the-pom.html)