

MAVEN

Vincent THOMASSIN

Projets exemples sur GitHub

<https://github.com/Vinthe/poei/tree/develop/maven>

C'est quoi **Maven**TM ?



C'est un **outil** de gestion et d'automatisation de **production** des **projets logiciels** Java

Maven gère :

- La **configuration du projet** : comment construire le livrable
Les IDE intègrent l'outil Maven pour configurer également l'environnement de développement,
- La **gestion des dépendances** : depuis le téléchargement sur internet jusqu'à son intégration dans le projet.
- Il suit la philosophie **Convention over Configuration**

https://fr.wikipedia.org/wiki/Convention_plut%C3%B4t_que_configuration

POM : Project Object Model

Chaque projet ou sous-projet est configuré par un POM qui contient les informations nécessaires à Maven pour traiter le projet

- nom du projet,
- numéro de version,
- dépendances vers d'autres projets,
- bibliothèques nécessaires à la compilation,
- noms des contributeurs,
- Etc ...

Ce **POM** se matérialise par un fichier **pom.xml** à la racine du projet.

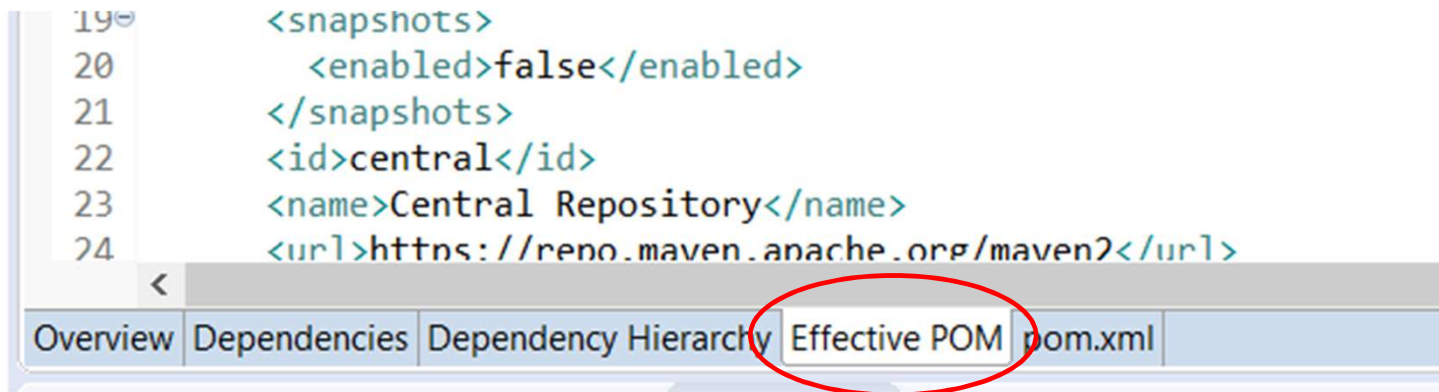
Le POM est identifié de manière unique par trois propriétés :

```
<artifactId>maven</artifactId>  
<groupId>fr.vinthech</groupId>  
<version>1-SNAPSHOT</version>
```

POM : L'approche en héritage

Tous les POM héritent du POM par défaut de Maven qui contient la configuration par défaut correspond aux **conventions** qui reflète l'état l'art.

Certains IDE permettent de voir le **POM effectif**, Vous y verrez Les variables Ils sont définies par défaut,



```
19 <snapshots>
20   <enabled>>false</enabled>
21 </snapshots>
22 <id>central</id>
23 <name>Central Repository</name>
24 <url>https://repo.maven.apache.org/maven2</url>
```

Overview Dependencies Dependency Hierarchy **Effective POM** pom.xml

Vous avez la possibilité si nécessaire de redéfinir chacune des variables pour votre projet.

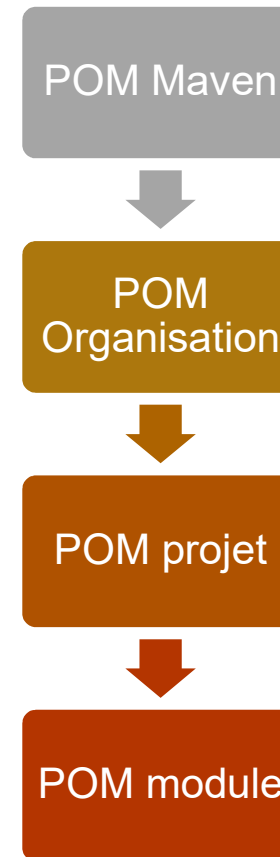
POM : Le POM parent

Il est possible de définir un **POM parent** qui reflètera les spécificités votre projets et/ou de votre organisation. Le POM parent est designé par ses identifiant.

```
10
11
12  <parent>
13    <groupId>fr.vinthech</groupId>
14    <artifactId>maven</artifactId>
15    <version>1-SNAPSHOT</version>
16  </parent>
```

À chaque étape de l'héritage, il est possible de définir ou redéfinir des propriétés/configuration qui deviendront le **défaut pour les POM héritants**.

Note : lorsqu'un parent est désigné le groupId est optionnel. Par défaut, il s'agit du même que le parent.



Dépendances : identification

Il est possible d'ajouter des dépendances à votre projet en indiquant les identifiants de la dépendance dans une balise « dependency »

```
<dependencies>
  <dependency>
    <groupId>com.google.guava</groupId>
    <artifactId>guava</artifactId>
    <version>28.2-jre</version>
  </dependency>
</dependencies>
```

Dépendances : localisation

Les dépendances sont mises à disposition dans des « repositories ».

Les dépendances doivent être mises à disposition sur des repository Public ou privé.

```
<repositories>
  <repository>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
    <id>central</id>
    <name>Central Repository</name>
    <url>https://repo.maven.apache.org/maven2</url>
  </repository>
</repositories>
```

Par défaut, Le central repository de Maven Apache est configuré. Mais il est possible de configurer des repository supplémentaires. Cette configuration peut se faire via le POM, le POM parent, ou de la configuration sur le poste de travail de l'outils Maven : settings.xml

Repositories privés et Proxis

L'accès à repository privée peut-être conditionné Par des identifiants de connexion.

- Il sera alors possible pour les développeurs de l'entreprise d'y mettre à disposition les librairie qu'ils développent

Une organisation peut contraindre d'utiliser sont « repository » privé pour l'accès aux ressources publiques. Repository privée joue alors également le rôle de proxy.

Cela permet 2 choses :

- De contrôler quelles librairies sont mises à disposition des projets.
- De sécuriser la disponibilité des librairies utilisées sur les projets de l'organisation. Le proxy maintenant un cache des librairie.

Dépendances : mise en local

Lorsque les librairies sont identifiées elles sont mis à disposition par Maven sur le poste du développeur.



	Nom	Modifié le	Type	Taille
	_remote.repositories	03/02/2020 09:21	Fichier REPOSITOR...	1 Ko
	guava-28.2-jre.jar	01/02/2020 20:08	Fichier JAR	2 723 Ko
	guava-28.2-jre.sha1	01/02/2020 20:08	Fichier SHA1	1 Ko
	guava-28.2-jre.pom	01/02/2020 20:08	Fichier POM	11 Ko
	guava-28.2-jre.pom.sha1	01/02/2020 20:08	Fichier SHA1	1 Ko
	guava-28.2-jre-sources.jar	03/02/2020 09:21	Fichier JAR	1 628 Ko
	guava-28.2-jre-sources.jar.sha1	03/02/2020 09:21	Fichier SHA1	1 Ko
	m2e-lastUpdated.properties	03/02/2020 09:21	Fichier PROPERTIES	1 Ko

En général, vos IDE intègre directement ses fichiers au classpath de vos programmes.

Le dossier des librairies peut devenir rapidement conséquent il faut prévoir plusieurs Go d'espace disque.

Dépendances : portées

Par défaut ce scope est *compile*. Ce scope indique que la dépendance est utilisée lors de la compilation et sera accessible dans tous les contextes (test, execution).

Le scope *test* indique que la dépendance n'est accessible que lors de la compilation des tests et leur exécution.

La dépendance en scope test ne fera donc pas partie du livrable.

Attention : Dans l'éclipse, l'exécution du programme se faire avec les dépendances test.

Le scope provided indique que la dépendance est disponible à la compilation, mais elle devra être fournie par le contexte d'exécution (par exemple par le serveur d'application)

Le scope runtime indique que la dépendance n'est pas accessible lors de la compilation, mais elle est disponible à l'exécution.

Dépendances : transitivités

Maven gère la **transitivité des dépendances**.

Maven ajoute automatiquement les dépendances requises par les dépendances que vous avez défini.

Maven n'ajoute que les dépendances nécessaires

Scope de votre dépendance dans votre		compile	provided	runtime	test
Scope de la dépendance de votre dépendance	compile	compile	-	runtime	-
	provided	provided	-	provided	-
	runtime	runtime	-	runtime	-
	test	test	-	test	-
		Scope de la sous-dépendance pour votre projet			

Gestion des Versions

Une version publiée doit être stable.

Tout changement que ce soit du code, des ressources, des modifications au niveau des dépendances (choix ou version) doit se traduire par une modification de la version de votre projet maven

Une exception : Il est entendu que les versions suffixées de « -SNAPSHOT » n'ont pas un contenu figé.

Ressource utile :

Gestion sémantique de version <https://semver.org/lang/fr/spec/v2.0.0.html>

Projet Modulaire

Un POM peut être chargé de de gestion plusieurs modules,

```
<modules>
  <module>../maven-project</module>
  <module>../maven-projet2</module>
</modules>
```

À l'exécution d'une tâche, le POM se charge d'ordonner les modules en fonction de leur dépendance respective.

Scanning for projects...

Reactor Build Order:

maven-parent	[pom]
maven-projet java library	[jar]
maven-projet2	[jar]

Cycle de vie

3 « lifecycles » de base dans Maven :

default : qui permet de **construire** et **déployer** le projet

clean : qui permet de **nettoyer** le projet en supprimant les éléments issus de la construction de celui-ci

site : qui permet de créer un site web pour le projet



Phases du lifecycle Default

validate	valider le projet est correct et toutes les informations nécessaires sont disponibles.
generate-sources	générer n'importe quel code source à inclure dans la compilation.
generate-resources	générer des ressources à inclure dans le package.
compile	compiler le code source du projet.
process-classes	post-traiter les fichiers générés à partir de la compilation, par exemple pour améliorer le bytecode sur les classes Java.
test-compile	compiler le code source du test dans le répertoire de destination du test
test	exécuter des tests en utilisant un cadre de test unitaire approprié. Ces tests ne doivent pas exiger que le code soit conditionné ou déployé.
package	prenez le code compilé et empaquetez-le dans son format distribuable, tel qu'un JAR.
integration-test	traiter et déployer le package si nécessaire dans un environnement où les tests d'intégration peuvent être exécutés.
verify	effectuer des vérifications pour vérifier que le package est valide et répond aux critères de qualité.
install	installez le package dans le référentiel local, pour l'utiliser en tant que dépendance dans d'autres projets localement.
deploy	fait dans un environnement d'intégration ou de publication, copie le package final dans le référentiel distant pour le partager avec d'autres développeurs et projets.

<http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html>

Goals

Les *phases* sont découpées en *tâches*. Chaque tâche est assurée par un **plugin Maven**. Dans le jargon de *Maven*, ces tâches s'appellent des **goals**.

Par défaut, les goals suivants des plugins core de Maven sont rattachées au phase :

Phase	plugin:goal
process-resources	resources:resources
compile	compiler:compile
process-test-resources	resources:testResources
test-compile	compiler:testCompile
test	surefire:test
package	jar:jar or war:war or
install	install:install
deploy	deploy:deploy

Voir : <http://maven.apache.org/ref/3.6.3/maven-core/default-bindings.html>

Plugin

Pour ajouter d'autres **tâches** aka **goals** aux **phases** est de configurer des plugins dans votre projet.

Les plugins sont des artefacts qui fournissent des « goals » à Maven. En outre, un plugin peut avoir un ou plusieurs « goals », chaque « goals » représentant une capacité de ce plugin.

Par exemple, le plugin « **compile** » a deux **goals** : « **compile** » et « **testCompile** ». Le premier compile le code source de votre code principal, tandis que le second compile le code source de votre code de test.



Command mvn

Syntaxe :

```
mvn [options] [<goal(s)>] [<phase(s)>]
```

Exemples

```
mvn package
```

```
mvn clean install
```

```
mvn install -DskipTests=true
```

Build du livrable

Bonnes pratiques

Pour une livraison,

Afin que garantir la reproductibilité du livrable :

- Toujours partir d'une version propre (prestine) du repository
- Toujours construire le livrable en dehors de le l'IDE.
- Il est donc conseiller d'avoir un dossier projets versionné dédié à la production des livrable.



Fin
