

Apache Maven 2

Didier DONSEZ

Université Joseph Fourier – Grenoble 1

PolyTech'Grenoble – LIG/ADELE

`Didier.Donsez@imag.fr`

`Didier.Donsez@ieee.org`

`donsez@apache.org`



Licence

- Cette présentation est couverte par le contrat Creative Commons By NC ND
 - <http://creativecommons.org/licenses/by-nc-nd/2.0/fr/>

Au sommaire

- Motivation
- Installation
- Cycle de vie
- Structure d'un projet Maven
- Plugins Maven
- Développement d'un plugin
- Bibliographie et Webographie

Motivations

- Rappel sur ANT
 - Remplaçant de make (pour les développements Java)
 - NAnt pour .NET
 - Séquenceur de tâches (regroupées en cible)
 - Graphe de dépendance de « cibles »
 - Très grand nombre de tâches développées
- Défaut
 - Pas de structure « standard » de projet
 - Pas de cycle de vie « standard » d'un projet
 - Pas de métadonnées « standard » sur les projets
 - scm, website, ML, ...
 - Pas de séparation de préoccupation
 - Libs externes à récupérer pour le projet et pour les tâches
 - problème de la MAJ des versions

Motivations de Maven

- Modèle abstrait de projet (POM)
 - Orienté objet, héritage
 - Séparation de préoccupations
- Cycle de vie standard
 - Séquencement d'états (goal) standards
 - Action des plugins en fonction des états
- Structure « standard » de projet
 - Nommage standard des variables (src.dir, ...)
- Gestion automatique des dépendances avec d'autres projets
 - Chargement des MAJ
- Dépôts des projets
 - publiques ou privés, local ou distants
 - caching et proxy
- Extensible via l'ajout des plugins
 - Eux même des projets Maven

Installation

- Télécharger la distribution binaire
 - <http://maven.apache.org>
- Dézipper dans un répertoire
- Positionner les variables d'environnement
 - `set JAVA_HOME=c:\j2sdk1.x.y`
 - `set MAVEN_HOME=c:\maven-2.z.w`
 - `set PATH=%JAVA_HOME%\bin;%MAVEN_HOME%\bin`
 - `mvn -version`
 - `mvn --help`
- (éventuellement) Configurer `~/.m2/settings.xml`
 - repositories, plugins repositories, proxies, ...
- Intégration à votre IDE (Eclipse, NetBeans, IDEA, ...)
 - <http://m2eclipse.codehaus.org/>, <http://mevenide.codehaus.org> ...

Évitez d'être ennuyé par des VM « parasites »

mvn --help

usage: mvn [options] [<goal(s)>] [<phase(s)>]

Options:

- q,--quiet Quiet output - only show errors
- C,--strict-checksums Fail the build if checksums don't match
- c,--lax-checksums Warn if checksums don't match
- P,--activate-profiles Comma-delimited list of profiles to activate**
- ff,--fail-fast Stop at first failure in reactorized builds
- fae,--fail-at-end Only fail the build afterwards; allow all non-impacted builds to continue
- B,--batch-mode Run in non-interactive (batch) mode
- fn,--fail-never NEVER fail the build, regardless of project result
- up,--update-plugins Synonym for cpu**
- N,--non-recursive Do not recurse into sub-projects
- npr,--no-plugin-registry Don't use ~/.m2/plugin-registry.xml for plugin versions
- U,--update-snapshots Forces a check for updated releases and snapshots on remote repositories**
- cpu,--check-plugin-updates Force upToDate check for any relevant registered plugins
- npu,--no-plugin-updates Suppress upToDate check for any relevant registered plugins
- D,--define Define a system property**
- X,--debug Produce execution debug output
- e,--errors Produce execution error messages
- f,--file Force the use of an alternate POM file.**
- h,--help Display help information
- o,--offline Work offline**
- r,--reactor Execute goals for project found in the reactor
- s,--settings Alternate path for the user settings file**
- v,--version Display version information

Le modèle de projet (POM pour Project Object Model)

- Description d'un projet indépendante des actions à accomplir
 - Orienté objet → héritage du modèle
- Exemple

```
<project>
```

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

```
  <groupId>com.mycompany.app</groupId>
```

```
  <artifactId>my-app</artifactId>
```

```
  <version>1.0.0-SNAPSHOT</version>
```

```
  <packaging>jar</packaging>
```

```
  <dependencies>
```

```
    <dependency>
```

```
      <groupId>junit</groupId>
```

```
      <artifactId>junit</artifactId>
```

```
      <version>3.8.1</version>
```

```
      <scope>test</scope>
```

```
    </dependency>
```

```
  </dependencies>
```

```
  ...
</project>
```

Identifiant (unique) du projet :
Identifiant de l'artifact produit

type du projet:
pom, jar, war, ear, bundle, ...

dépendances du projet envers
d'autres projets (artifact)

constitue le \$CLASSPATH

id d'une dépendance

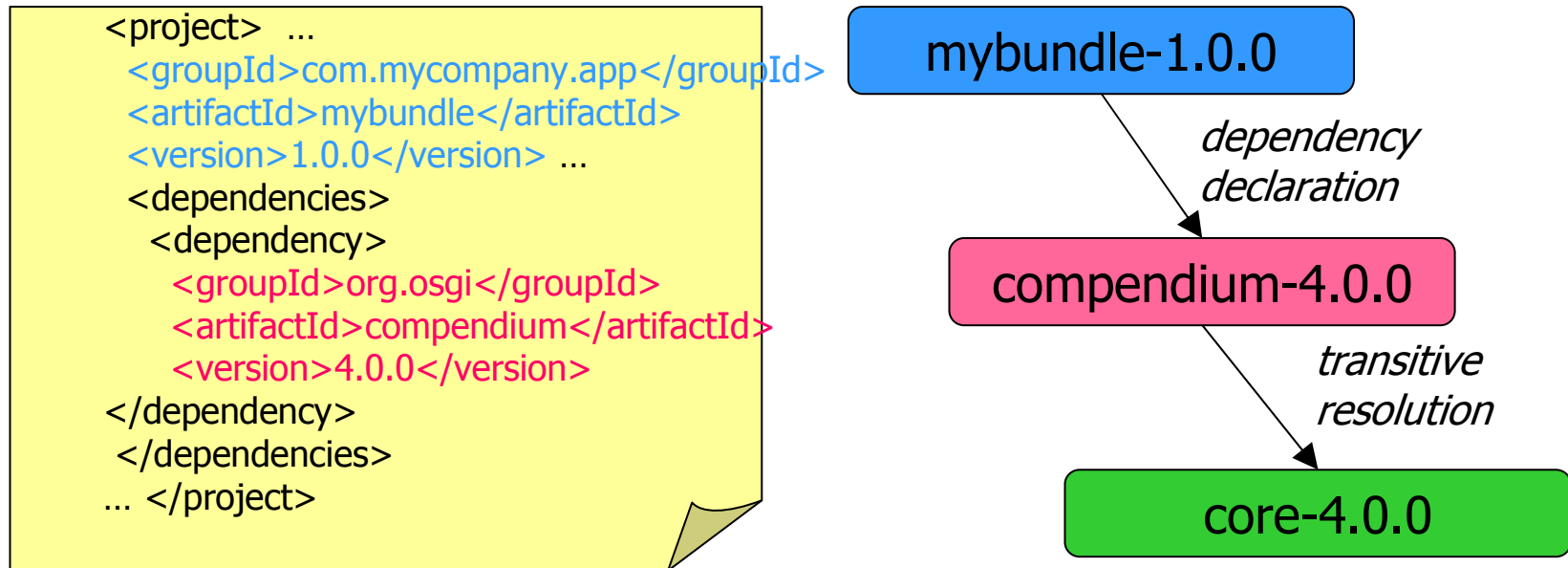
version peut être un intervalle

portée de la dépendance par rapport au cycle
de vie (compile, provided, runtime, test)

la suite bientôt ...

Dependances

- Concerne les artifacts comme les plugins
- Résolution transitive



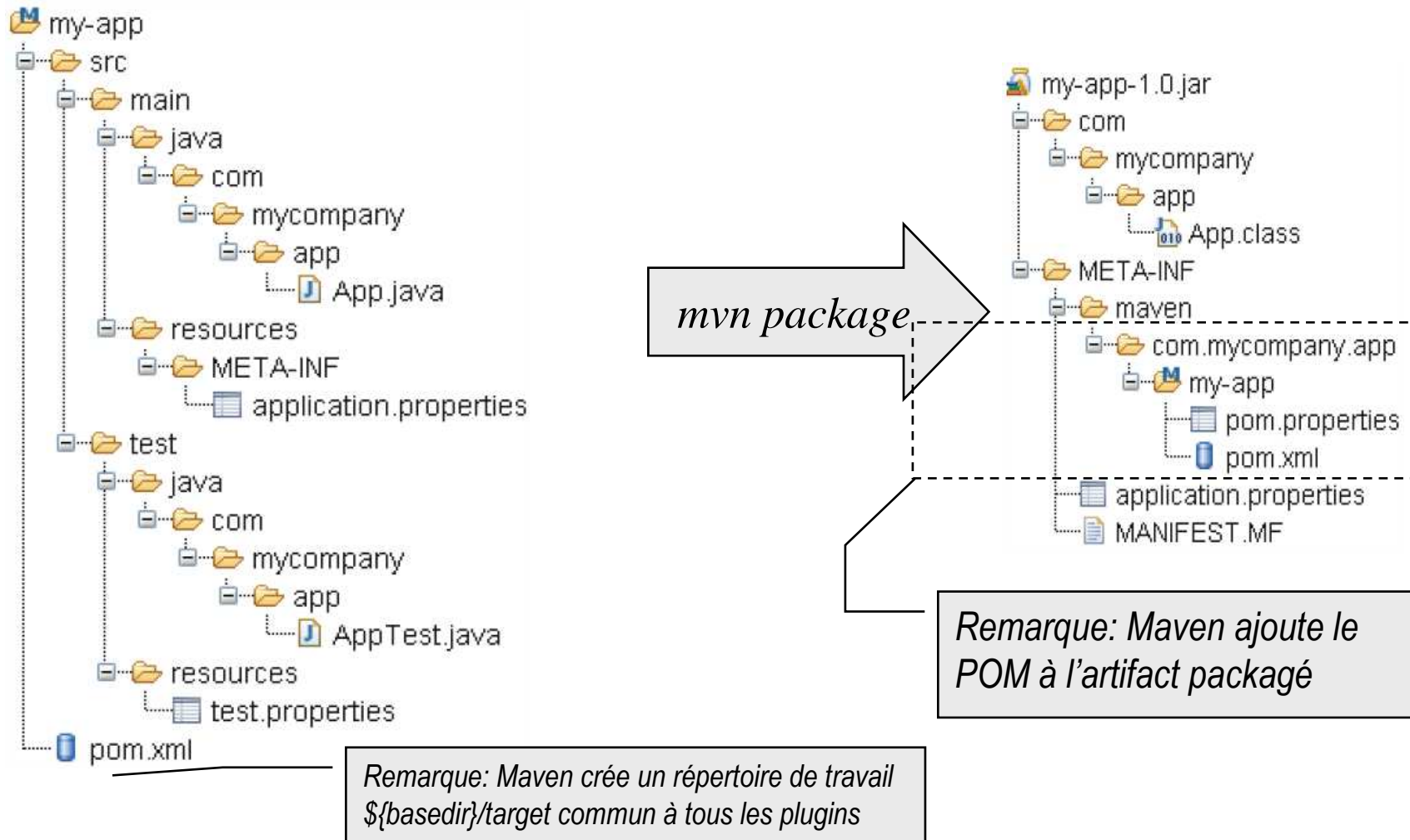
- Sert à constituer le CLASSPATH
 - Pour la compilation, pour les tests, pour l'exécution

Portée des dépendances

- **5 portées possibles par rapport aux classpaths du projet**
 - **compile** (défaut)
 - Disponible dans tous les classpaths
 - Transitive vers les projets dépendants
 - **provided**
 - compilation and test classpaths
 - Not transitive.
 - **runtime**
 - runtime and test classpaths.
 - **test**
 - test compilation and execution phases.
 - **system**
 - similar to provided but the artifact is always available and is not looked up in a repository.
 - **import**
 - only used on a dependency of type pom in the <dependencyManagement> section.
- **Portée transitive**

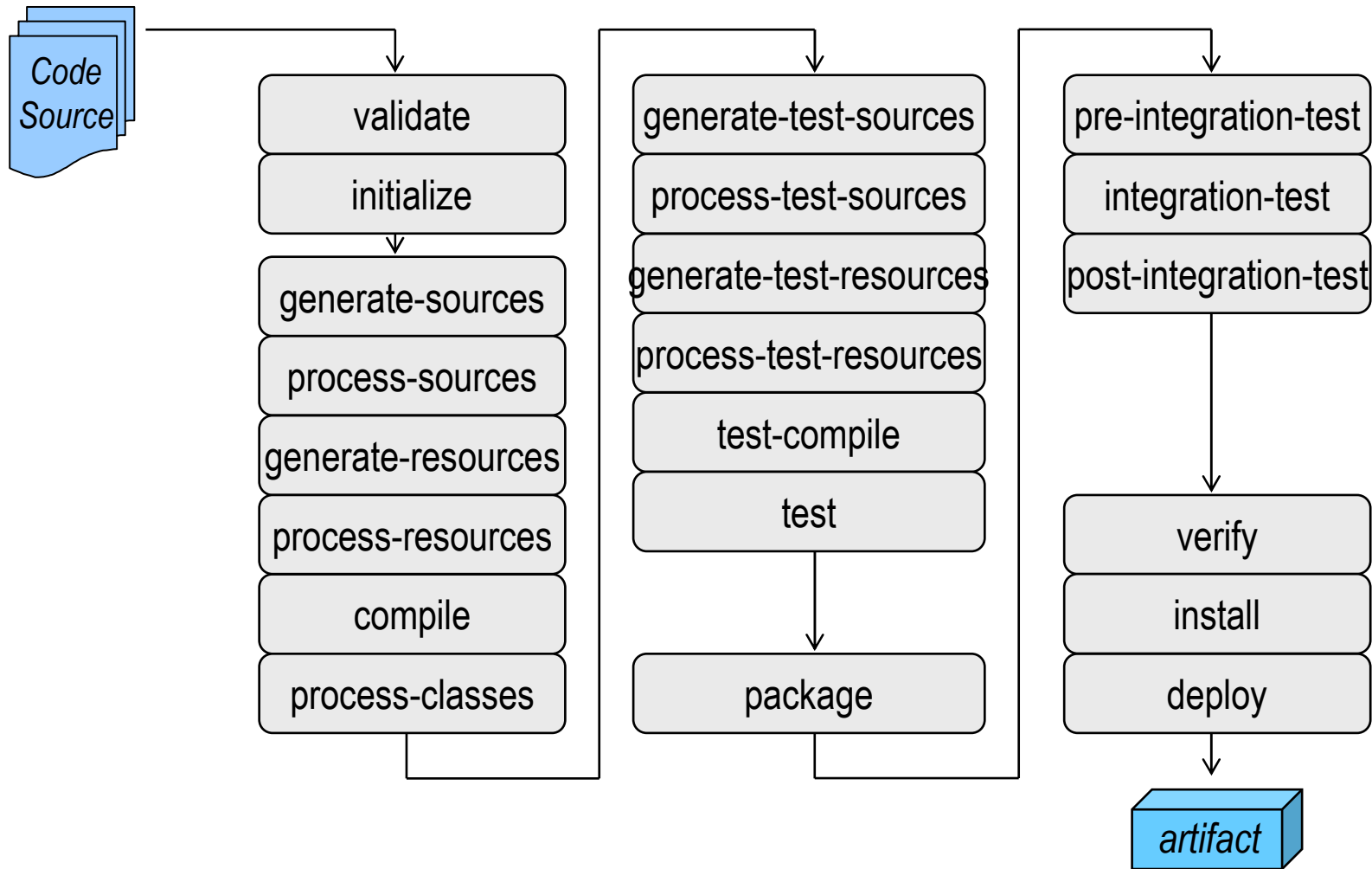
	compile	provided	runtime	test
compile	compile	-	runtime	-
provided	provided	provided	provided	-
runtime	runtime	-	runtime	-
test	test	-	test	-

Structure « standard » d'un projet



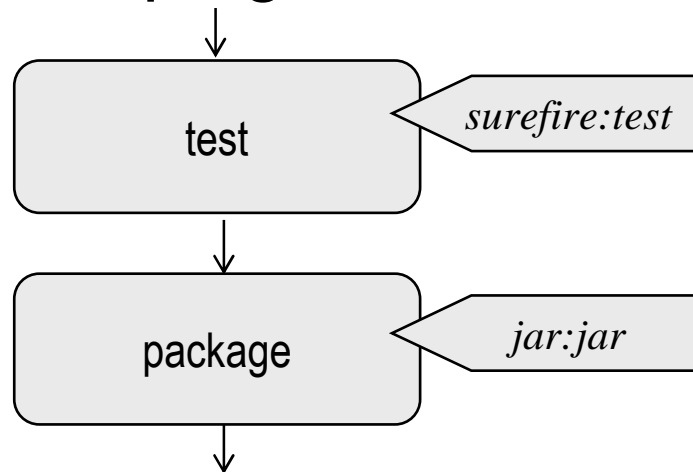
Cycle de vie (par défaut) d'un projet

- Séquence de 21 phases



Phases et Buts (goals)

- A chaque phase est associé un ou plusieurs buts d'un ou de plusieurs plugins



- Remarque

- *mvn resources:resources compiler:compile resources:testResources compiler:testCompile surefire:test jar:jar* est équivalent à *mvn package*

- D'autres cycles de vie ont été définis

- clean = pre-clean → clean → post-clean
- site = pre-site → site → post-site → site-deploy
- ...

Rappel

Numérotation des versions

- Schéma
 - `<major>.<mini>[.<micro>][-<qualifier>[-<buildnumber>]]`
- Incrément
 - Major : changement majeur
 - pas de retro-compatibilité (descendante) garantie
 - Mini : ajouts fonctionnels
 - retro-compatibilité garantie
 - Micro : maintenance corrective (*bug fix*)
- Qualificateurs
 - SNAPSHOT (Maven) : version en évolution
 - alpha1 : version alpha (très instable et incomplète)
 - beta1, b1, b2 : version beta (instable)
 - rc1, rc2 : release candidate
 - m1, m2 : milestone
 - ea : early access
 - 20081014123459001 : date du build
 - jdk5 : dépendance avec une arch, un os, un langage
- Ordre sur les versions
 - Différent de l'ordre lexicographique
 - 1.1.1 < 1.1.2 < 1.2.2
 - 1.1.1-SNAPSHOT < 1.1.1
 - 1.1.1-alpha1 < 1.1.1-alpha2 < 1.1.1-b1 < 1.1.1-rc1 < 1.1.1-rc2 < 1.1.1
- Remarque (parfois)
 - <mini> pair : release stable
 - <mini> impair : release instable

Versionnement

■ Snapshot

- A **snapshot in Maven** is an artifact which has been prepared using the **most recent** sources available. ... **Specifying a snapshot version** for a dependency means that Maven will look for new versions of that dependency without you having to manually specify a new version.
- mvn -U command line option to force the search for updates.

■ Dépendances

- Spécification d'intervalles de versions

```
<dependency>
  <groupId>org.codehaus.plexus</groupId>
  <artifactId>plexus-utils</artifactId>
  <version>[1.1,)</version>
</dependency>
```

Range	Meaning
(, 1.0]	Less than or equal to 1.0
[1.2, 1.3]	Between 1.2 and 1.3 (inclusive)
[1.0, 2.0)	Greater than or equal to 1.0, but less than 2.0
[1.5,)	Greater than or equal to 1.5
(, 1.1) , (1.1,)	Any version, except 1.1

Quelques plugins usuels

- Core
 - clean, compiler, deploy, install, resources, site, surefire, verifier
- Packaging
 - ear, ejb, jar, rar, war, *bundle (OSGi)*
- Reporting
 - changelog, changes, checkstyle, clover, doap, docck, javadoc, jxr, pmd, project-info-reports, surefire-report
- Tools
 - ant, antrun, archetype, assembly, dependency, enforcer, gpg, help, invoker, one (interop Maven 1), patch, plugin, release, remote-resource, repository, scm
- IDEs
 - eclipse, netbeans, idea
- Autres
 - exec, jdepend, castor, cargo, jetty, native, sql, taglist, javacc, obr ...



<http://maven.apache.org/plugins/>, <http://mojo.codehaus.org/plugins.html>, ...

Configuration des plugins

- Passage de paramètres autre que ceux définis par défaut
- Exemple

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <configuration>
        <source>1.5</source>
        <target>1.5</target>
      </configuration>
    </plugin>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
      <configuration>
        <archive>
          <manifest>
            <mainClass>${artifactId}.Main</mainClass>
            <addClasspath>true</addClasspath>
          </manifest>
        </archive>
      </configuration>
    </plugin>
  </plugins>
</build>
```

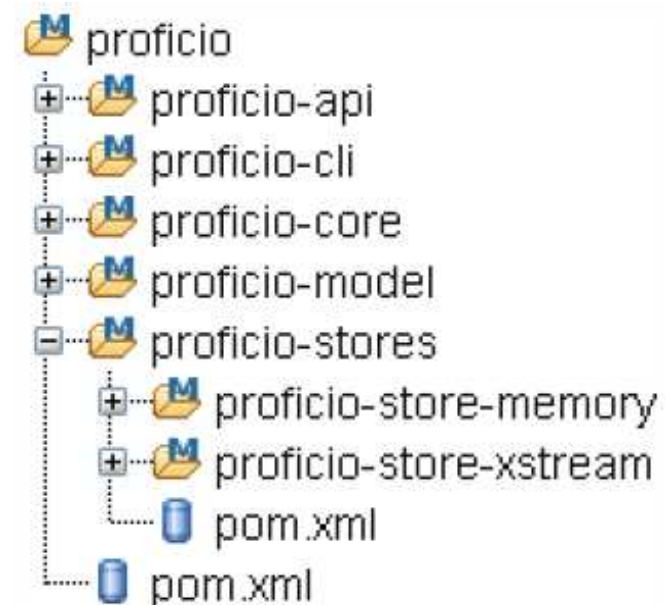
Profils

- Motivation
 - Améliorer la portabilité des projets par rapport aux environnements
 - Différents JVM, versions de Java, serveurs JEE, SGBD, développement versus production
 - Créer des variations (=profils) de projets
- Élément <profile> du build
 - Contient les variations de plugins et entre les plugins
- Activation du profil
 - Profil par défaut
 - En fonction des propriétés (systèmes, version JDK, ...)
 - Par son identifiant
 - `mvn --activate-profiles felix,equinox clean install`

Organisation hiérarchique de projets

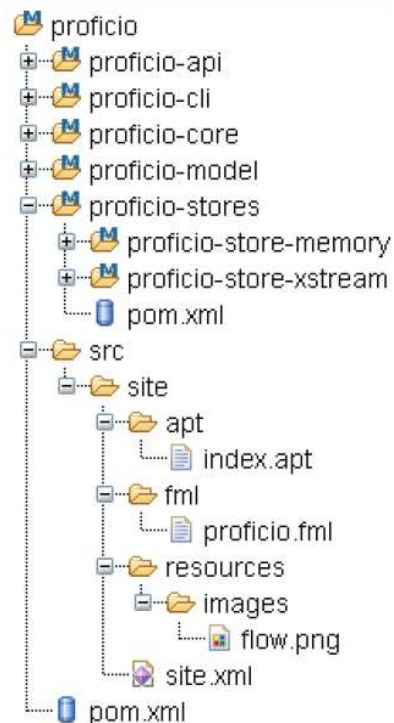
- Motivations
 - Organiser le développement en sous-projets
 - Avec N niveaux ($N \geq 1$)
- Méthode
 - Création d'un super POM (de type pom) par niveau
 - Regroupe les plugins/goals communs du même niveau
 - Les sous-projets (appelé modules) héritent de ce super pom
- Exemple

- Commande
 - `mvn --reactor clean install`
 - Pour la construction globale

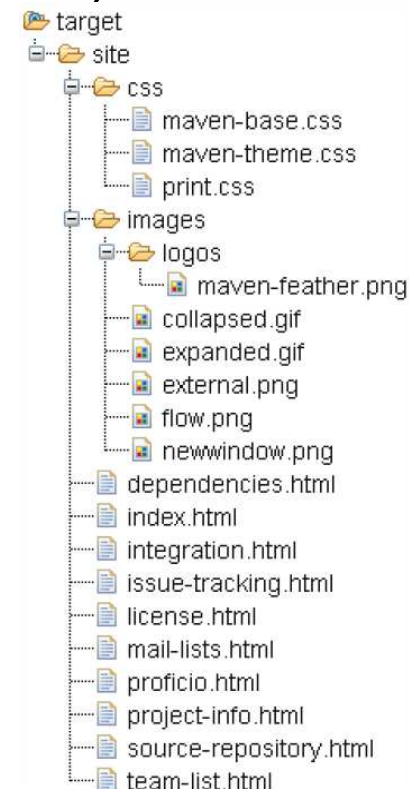


Documentation Web d'un projet

- Transforme plusieurs formats de documentation
 - XDOC, APT (Almost Plain Text), FML (FAQ ML), DocBook Simple, Twiki, Confluence
 - La documentation source peut contenir des variables du projet (`$project.name`, `$reports`, ...)

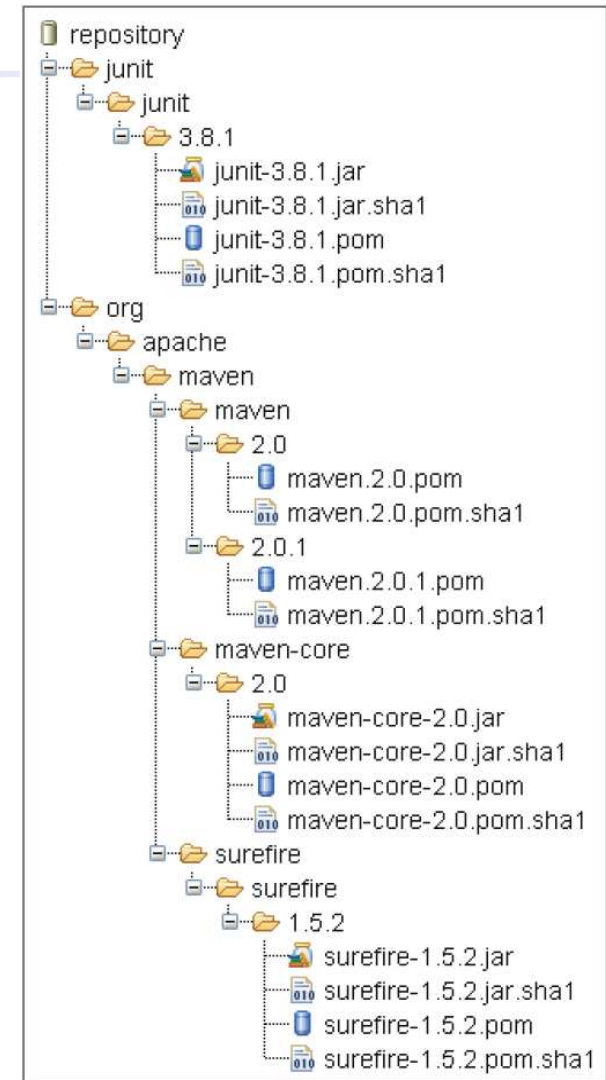


mvn site:site



Dépôts de projets

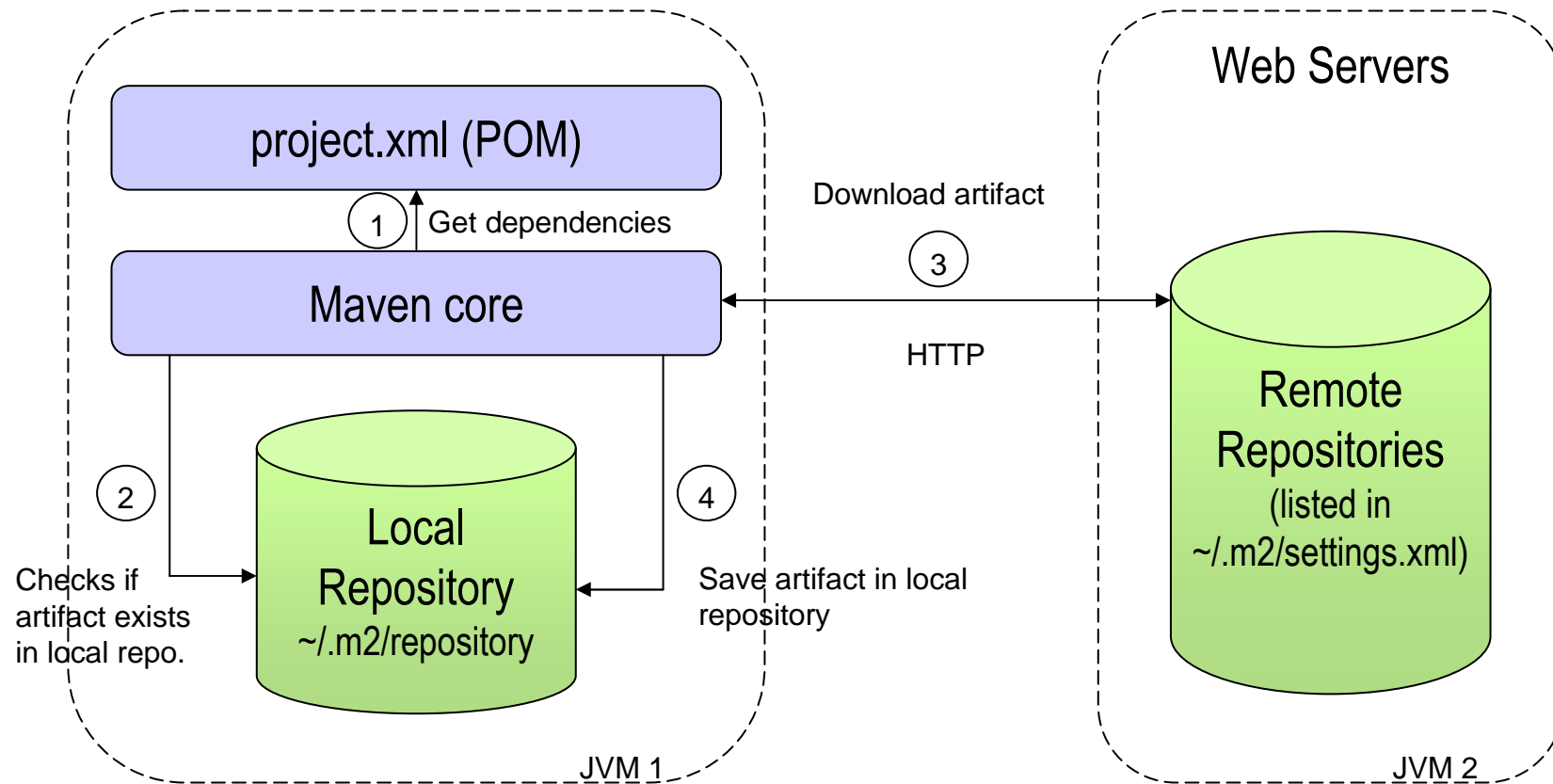
- Local ~/.m2/repository
 - Projets (dont artifacts) installés localement
 - mvn install
 - mvn install:install-file
 - Caches des projets (artifacts) téléchargés depuis les dépôts distants
 - Listés dans les POM et settings.xml
- Distants
 - Dépôts d'entreprise
 - Cache de dépôts
 - Dépôts publiques
- Structure
 - Nommage hiérarchique
 - `${groupId}.replace('.', '/') / ${artifactId} / ${version}`



Dépôts publiques

- Les principaux
 - Apache Maven Central
 - <http://repo1.maven.org/maven2/>
 - Plus de 20000 artifacts décrits (*en 2007*)
 - *Tous en licence ASL v2*
 - CodeHaus
 - <http://www.codehaus.org>
 - Dependance vers d'autres licences (BSD, ...)

Recherche des dépendances



Didier Donsez, 2007-2010, Maven

R1: La mise à jour du dépôt local est journalière (sauf si `mvn -U`)
R2: Les plugins sont recherchés et mis à jour de la même façon

Substitution de variables à la construction

- Motivations
 - Instancier les valeurs des ressources lors de la phase *process-resources*
- Exemple de POM

```
... <build>
```

```
<filters>
```

```
<filter>src/main/filters/filter.properties</filter>
```

```
</filters>
```

```
<resources>
```

```
<resource>
```

```
<directory>src/main/resources</directory>
```

```
<filtering>>true</filtering>
```

```
</resource>
```

```
</resources>
```

```
</build>
```

```
# src/main/filters/filter.properties
my.filter.value=Hello !
```

```
# src/main/resources/application.properties
message=${my.filter.value}
application.name=${project.name}
application.version=${project.version}
```


Archetype

- Construction initial d'un projet Maven
 - En fonction d'un type de projet T
 - T= **quickstart**, archetype, bundles, j2ee-simple, marmalade-mojo, mojo, plugin, plugin-site, portlet, profiles, simple, site, site-simple, webapp, ...
- Exemple
 - mvn archetype:create
mode interactif
 - mvn archetype:create
 - DgroupId=demo.maven
 - DartifactId=hello
 - Dversion=0.1.0-SNAPSHOT
 - DarchetypeGroupId=org.apache.maven.archetypes
 - DarchetypeArtifactId=maven-archetype-**quickstart**

Archetypes personnalisés

- Possibilité de créer ses propres archetypes
 - → de zero
 - mvn archetype:create
 - DarchetypeGroupId=org.apache.maven.archetypes
 - DarchetypeArtifactId=maven-archetype-**archetype**
 - DgroupId=com.mycompany
 - DartifactId=my-archetype
 - → depuis un archetype existant
- Développement
 - Basé sur des templates Velocity (<http://velocity.apache.org/>)

Développement de plugins

- Plugin = { <goal,MOJO> }
- MOJO = Maven POJO
 - Annotations XDocLet

- Langages
 - Java et Groovy (pour le scripting)
 - D'autres possibles ...

- Déploiement
 - Artifact Maven
 - Utilise les mécanismes de déploiement (version, dépendances, ...)
 - Dépôts de plugins
 - <http://maven.apache.org/plugins/>, <http://repository.codehaus.org/>

Développement de plugins

Exemple (i)

```

package sample.plugin;
import org.apache.maven.plugin.AbstractMojo;
import org.apache.maven.plugin.MojoExecutionException;
/**
 * Says "Hi" to the user.
 * @goal sayhi
 * @phase compile
 */
public class GreetingMojo extends AbstractMojo {
    /** The greeting to display.
     * @parameter alias="message" expression="Hello, world (from ${project.groupId}:${project.artifactId})" */
    private String greeting;

    /** The classpath.
     * @parameter expression="${project.compileClasspathElements}"
     * @required
     * @readonly */
    private List classpathElements;

    public void execute() throws MojoExecutionException {
        getLog().info(greeting);
        getLog().info("Project classpath: " + classpathElements().toString().replace(',', ';'));
    }
}

```

phase et but durant laquelle `execute()` est appelé

paramètre renseigné dans `<configuration>`

Integer, ..., String, List, Properties, Map, Object, File, URL, ...

paramètre issue du pom

Développement de plugins

Exemple (ii)

- Dans le POM

```
<build>
  <plugins>
    <plugin>
      <groupId>sample.plugin</groupId>
      <artifactId>maven-hello-plugin</artifactId>
      <configuration>
        <message>Welcome</message>
      </configuration>
    </plugin>
  </plugins>
</build>
```

- Exécution

```
mvn sample.plugin:maven-hello-plugin:sayhi
```

Plugins et Cycles de vie

- MOJO attaché à une phase du cycle de vie
 - @nnotations doclet

- Cycles de vie personnalisés
 - Surcharge de META-INF/plexus/components.xml,

Appel de tâches ANT dans un projet Maven

- Motivations
 - récupération de projets existants avant conversion
 - Exécution de tâches patrimoniales n'ayant pas de plugins équivalents
 - Remarque: pensez à utiliser la définition de Macro ANT !
- Exemple avec le plugin org.apache.maven.plugins:maven-antrun-plugin

```

<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-antrun-plugin</artifactId>
  <executions>
    <execution>
      <phase>generate-sources</phase>
      <configuration>
        <tasks unless="maven.test.skip">
          <!-- Place any ant task here. You can add anything
          you can add between <target> and </target> in a build.xml.-->
          <echo message="To skip me, just call mvn -Dmaven.test.skip=true"/>
          <exec dir="${basedir}"
                executable="${basedir}/src/main/sh/do-something.sh" failonerror="true">
            <arg line="arg1 arg2 arg3 arg4" />
          </exec>
        </tasks>
      </configuration>
      <goals>
        <goal>run</goal>
      </goals>
    </execution>
  </executions>
</plugin>

```

Conversion d'un projet ANT en projet Maven

- 2 possibilités pour la structure du projet
 - Réorganiser (manuellement, projet ANT si plusieurs projets)
 - src → src/main/java, src/test/java, doc → src/site
 - classes → target/classes, build → target, ...
 - Configurer les paramètres par défaut du POM en fonction de la structure du projet ANT

- Définir les dépendances
 - en fonction du <classpath ...>

Antlib for Maven

- Taches Maven pour projet ANT
 - Manipulation d'artifacts depuis un projet Ant
 - Gestion (transitive) des dépendances
 - scope recognition and SNAPSHOT handling
 - Déploiement des artifacts vers un dépôt Maven
 - Analyse d'un pom.xml

- Exemple

```
<artifact:dependencies pathId="dependency.classpath">  
  <dependency groupId="javax.servlet" artifactId="servlet-api"  
    version="2.4" scope="provided" />
```

```
  ...
```

```
</artifact:dependencies>
```

```
<javac ...>
```

```
  <classpath refid="dependency.classpath" />
```

```
  ...
```

```
</javac>
```

Maven et autres langages

- Maven est plutôt orienté vers des projets Java
- Projets pour d'autres environnements et langages
 - .NET, ...
 - JNI, C, C++, C#, PHP, JavaScript, GWT, Basic, ..
- Structure du projet
 - src/main/java
 - src/main/c
 - src/main/cpp
 - src/main/cs
 - src/main/php
 - src/main/vb
 - ...
- Plugins
 - maven-antrun-plugin (org.apache.maven.plugins:)
 - pour la récupération de tâches Ant patrimoniales
 - native-maven-plugin (org.codehaus.mojo:)

Misc

- Maven SCM
 - Plugin offrant une API commune vers les principaux SCM
 - Commandes
 - Changelog - command to show the source code revisions
 - Checkin - command for committing changes
 - Checkout - command for getting the source code
 - Diff - command for showing the difference of the working copy with the remote ones
 - Edit - command for starting edit on the working copy
 - Status - command for showing the scm status of the working copy
 - Tag - command for tagging the certain revision
 - UnEdit - command for to stop editing the working copy
 - Update - command for updating the working with the latest changes
 - Validate - validates the scm information on the pom
 - Supported SCM
 - Subversion, CVS, StarTeam, ClearCase, Perforce, Bazaar
- Maven Continuum
 - continuous integration (JEE-based) server for building Java based projects.
 - Schedulable projects: Maven 1, Maven 2, Ant, Shell scripts
 - Notifications : Mail and IM (IRC, Jabber, MSN)
- Maven Archiva
 - Repository manager (search, security, reporting, ...)
- Maven Wagon
 - Outil de transfert des artefacts vers des dépôts (distants ou locaux)
 - File, HTTP, HTTP lightweight, FTP, SSH/SCP, WebDAV, SCM



Misc

- Apache Ivy
 - Gestionnaire de dépendances (pour projet Ant)
 - Sous gestionnaire pour des dépôts Maven (locaux ou distants)

M2Eclipse

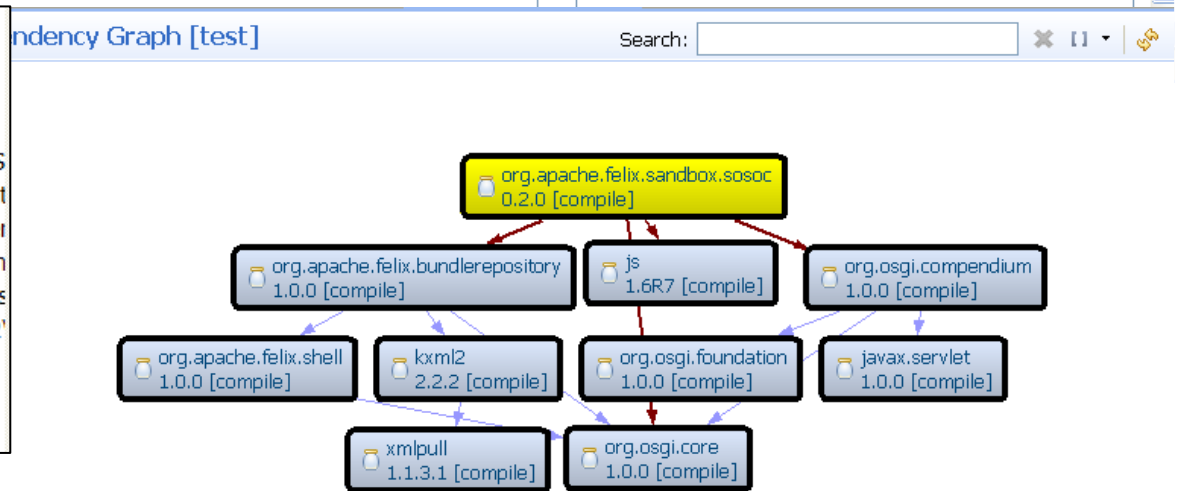
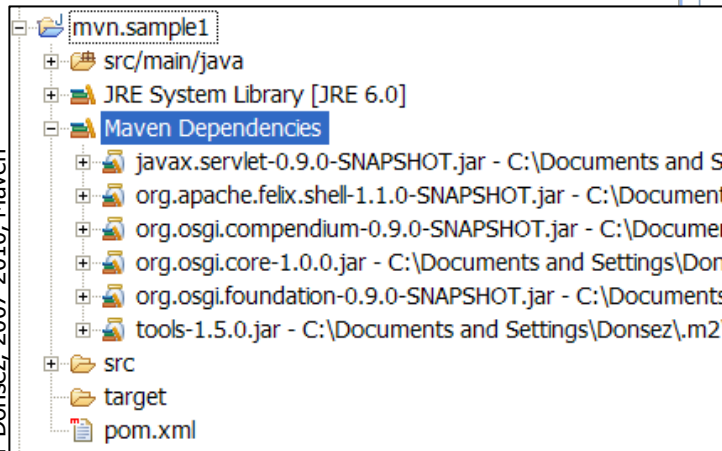
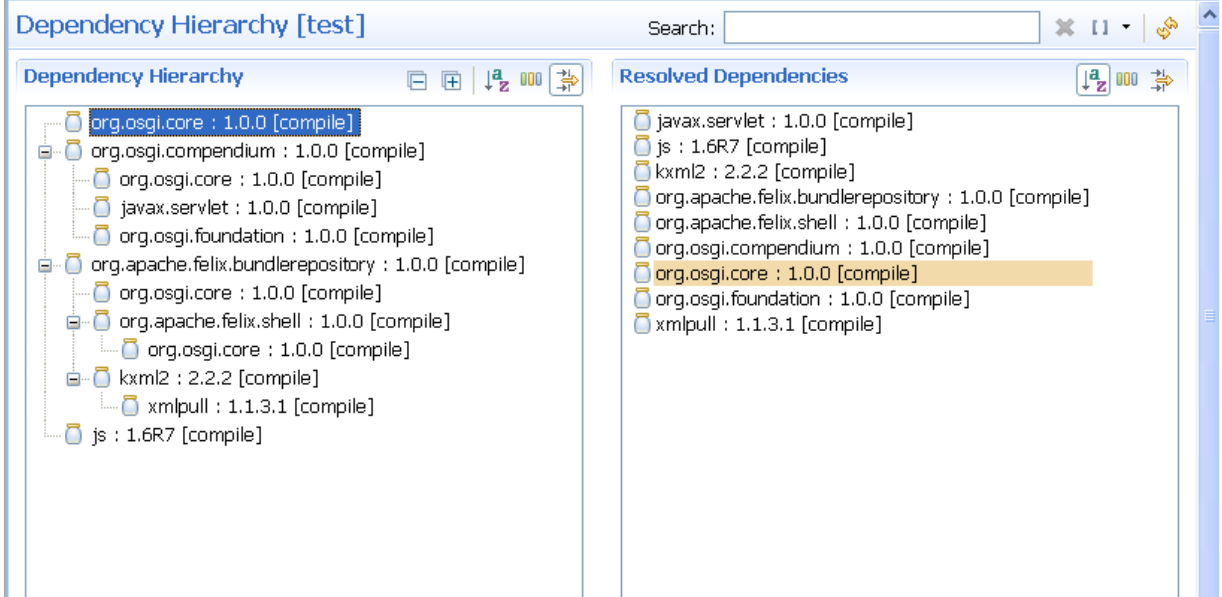
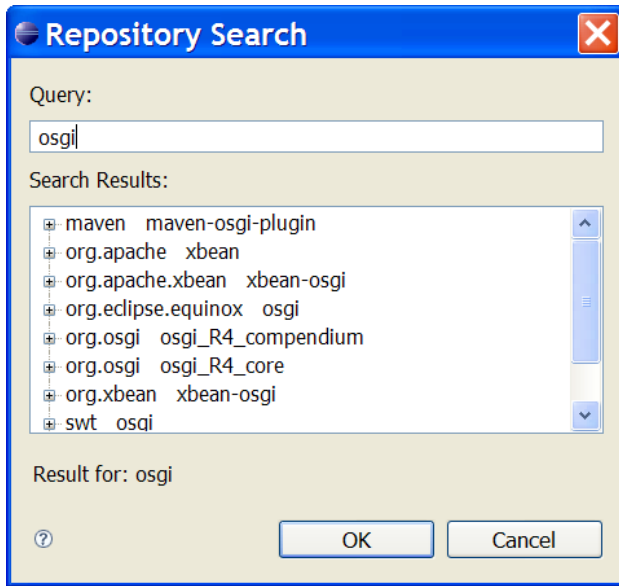
Plugin Eclipse pour Maven

- Création de projets
 - Wizard, Archetypes
- Edition du POM
- Affichage graphique
- Recherche de dépendances
 - Depuis les dépôts local et distants

Ajout des dépendances du POM au .classpath

- `org.maven.ide.eclipse.MAVEN2_CLASSPATH_CONTAINER`
- Exécution des principales phases : clean, test, install, ...
- Livre en ligne
 - <http://www.sonatype.com/m2eclipse/documentation/download-book?file=books/m2eclipse-book.pdf>

M2Eclipse Plugin Eclipse pour Maven



Maven Continuum

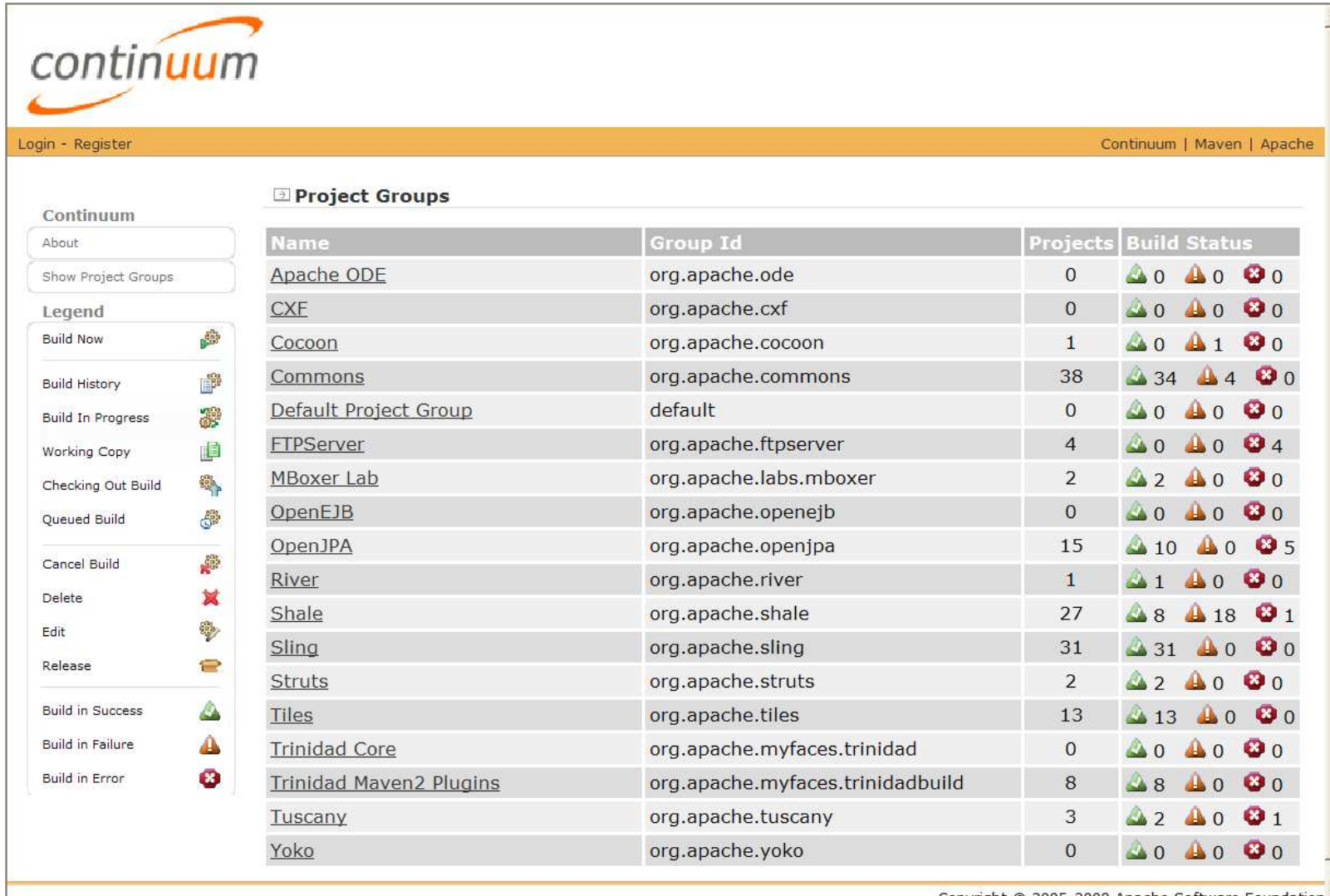


<http://maven.apache.org/continuum/>

- continuous integration (JEE-based) server for building Java based projects.
 - Schedulable projects: Maven 1, Maven 2, Ant, Shell scripts
 - Notifications : Mail and IM (IRC, Jabber, MSN)
 - Release management
 - SCM support
 - CVS, Subversion, Clearcase, Perforce, Starteam, Visual Source Safe, CM Synergy, Bazaar, Mercurial
 - External access with XMLRPC

Maven Continuum

■ Console



The screenshot displays the Maven Continuum console interface. At the top left is the 'continuum' logo. Below it is a navigation bar with 'Login - Register' on the left and 'Continuum | Maven | Apache' on the right. The main content area is titled 'Project Groups' and contains a table with the following columns: Name, Group Id, Projects, and Build Status. The Build Status column uses icons to represent the number of projects in different states: green for success, yellow for failure, and red for error.

Name	Group Id	Projects	Build Status
Apache ODE	org.apache.ode	0	0 Success 0 Failure 0 Error
CXF	org.apache.cxf	0	0 Success 0 Failure 0 Error
Cocoon	org.apache.cocoon	1	0 Success 1 Failure 0 Error
Commons	org.apache.commons	38	34 Success 4 Failure 0 Error
Default Project Group	default	0	0 Success 0 Failure 0 Error
FTPServer	org.apache.ftpserver	4	0 Success 0 Failure 4 Error
MBoxer Lab	org.apache.labs.mboxer	2	2 Success 0 Failure 0 Error
OpenEJB	org.apache.openejb	0	0 Success 0 Failure 0 Error
OpenJPA	org.apache.openjpa	15	10 Success 0 Failure 5 Error
River	org.apache.river	1	1 Success 0 Failure 0 Error
Shale	org.apache.shale	27	8 Success 18 Failure 1 Error
Sling	org.apache.sling	31	31 Success 0 Failure 0 Error
Struts	org.apache.struts	2	2 Success 0 Failure 0 Error
Tiles	org.apache.tiles	13	13 Success 0 Failure 0 Error
Trinidad Core	org.apache.myfaces.trinidad	0	0 Success 0 Failure 0 Error
Trinidad Maven2 Plugins	org.apache.myfaces.trinidadbuild	8	8 Success 0 Failure 0 Error
Tuscany	org.apache.tuscany	3	2 Success 0 Failure 1 Error
Yoko	org.apache.yoko	0	0 Success 0 Failure 0 Error

On the left side of the console, there is a sidebar with various actions and build status indicators:

- Continuum**
 - About
 - Show Project Groups
- Legend**
 - Build Now
 - Build History
 - Build In Progress
 - Working Copy
 - Checking Out Build
 - Queued Build
 - Cancel Build
 - Delete
 - Edit
 - Release
 - Build in Success
 - Build in Failure
 - Build in Error

At the bottom right of the console, there is a copyright notice: Copyright © 2005-2008 Apache Software Foundation.

Divers

- Recherche d'artifacts
 - <http://www.mvnrepository.com/>

Misc

- Tree Surgeon (Maven pour .NET ?)
 - <http://confluence.public.thoughtworks.org/display/TREE/Tree+Surgeon>
 - « *Tree Surgeon est un outil Open Source édité par la société de Martin Fowler Thoughtworks. "It is a tool that automates the process of establishing a directory structure with source code stubs and supporting infrastructure in a consistent manner. (...) It supports tools like NAnt and NUnit by generating build files and unit tests as part of the automated process. . Même si Tree Surgeon (au nom peu invocateur) est loin d'atteindre le niveau fonctionnel de son homologue Java Maven, il constitue une excellente base pour industrialiser la génération de squelettes de projets .NET. »*

Good & Best Practices

- **Beginners**
 - KISS (Keep It Simple, Stupid)
 - Start from scratch
 - No Copy/Paste
 - Use only what you need
 - Filtering, Modules, Profiles, ...
- **Bad practices**
 - Ignore maven conventions
 - Different versions in sub modules
 - Too many inheritance levels
 - AntRun (OK for integration test)
 - Plugins without versions
 - ...

Maven 3.x

- Any-source POM (json, groovy, ...)
- Versionless parent elements
- Mixin : a composition of POMs
- Better IDE integration
- Error reporting
 - Codes d'erreur commune
- Lifecycle extension points
- Plugin extension points
 - example: war extends jar
- Incremental build support
- Queryable lifecycle
 - comportement en fonction du lifecycle calculé au démarrage
- Extensible reporting
 - Continuous integration (sonar, ...)
- New tools
 - tycho : OSGi, Eclipse
 - Integration continu
 - Transféré vers Eclipse ?
 - mvnsh : shell (performance)
- New IoD : Guice (annotation ?)

Bibliographie et Webographie

- Web
 - Site Maven, <http://maven.apache.org>
- Exemples et exercices
 - <http://www-adele.imag.fr/users/Didier.Donsez/cours/tpmvn>
- Complément
 - <http://www-adele.imag.fr/users/Didier.Donsez/cours/coursjavaoutil.pdf>

Bibliographie et Webographie

<http://maven.apache.org/articles.html>

■ Ouvrages

- *Maven: The Definitive Guide*
 - <http://www.sonatype.com/book/maven-user-guide.pdf>
 - En français <http://www.maven-definitive-guide.fr/>
- John Casey, Vincent Massol, Brett Porter, Carlos Sanchez, Jason van Zyl, *Better Builds with Maven*, Publisher Mergere Library Press, March 2006
 - (PDF gratuit en ligne)
 - Vraiment bien, mais c'est préférable de connaître Maven auparavant !
- Vincent Massol, Tim O'Brien, *Maven: A Developer's Notebook*, Publisher O'Reilly, July 2005
- Une FAQ en français
 - <http://java.developpez.com/faq/maven/>
- Livre M2Eclipse en ligne
 - <http://www.sonatype.com/m2eclipse/documentation/download-book?file=books/m2eclipse-book.pdf>

Exercice Développement d'un plugin

- Développement d'un plugin pour Velocity
 - et DVSL (Declarative Velocity Style Language)
- Questions
 - A quelles phases peut être appliqué ce plugin ?
 - ...

Rappel sur Apache Velocity

- Langage de templates (VTL)
 - Syntaxe proche des macros CPP
 - Macros #set, #foreach() ... #end, #if () ...#elseif () ...#else ...#end, #include(...), #parse(...)
 - DVSL (Declarative Velocity Style Language) #match() ... #end
 - Variables \$var ou \${var}
- Usage
 - Génération de pages Web
 - Génération de codes sources (generative programming), ...
- Exemple VTL

```
// generated at $date
package ${pkgName};
public interface ${itfName}MBean {
#foreach ($attribute in ${attributesList})
  /** setter for the attribute ${attribute} */
  public void set${attribute}(String new${attribute});
  /** getter for the attribute ${attribute} */
  public String get${attribute}();
#end
  /** reset all the attributes */
  public void reset();
}
```

```
public static void main(String [] args) {
  Velocity.init();
  VelocityContext vc = new VelocityContext();
  vc.put("date", new Date());
  vc.put("itfName", "Config"); ...
  Template template
    = Velocity.getTemplate(args[0]);
  OutputStreamWriter osw =
    new StringWriter(System.out);
  template.merge(vc, osw);
}
```