Maven Project Builder

Maven

- Introduction
- Using Maven (I)
  - Installing the Maven plugin for Eclipse
  - Creating a Maven Project
  - Building the Project
- Understanding the POM
  - Maven Lifecycle
  - Plugins
  - Repositories
- Using Maven (II)
  - A Project example
- Repository Managers

Introduction to Maven

- Maven is a tool for software project management.
  - Based on a model of the project (POM = project object model), Maven can manage a project's build, reporting and documentation from a central piece of information

Introduction to Maven

- Maven…
  - It provides an extensible mean to build the software. Using additional plugins, the build process can include execution of tests, generation of documentation and deploy of application in different servers.

```
mvn exec:java -Dexec.mainClass=kbc.FirstApp
mvn package
```

  - Maven includes support for dependency management. It can download the required libraries and help to developers in the detection of libraries conflicts.

Advantages of Maven

- Builds
- Documentation
- Reporting
- Dependencies
- SCMs
- Releases
- Distribution
- Mailing list

Advantages of Maven

- Maven was originally designed to simplify building processes in Jakarta Turbine project.
- There were several projects and each project contained slightly different ANT build files.
- JARs were checked into CVS.
- Apache group then developed Maven which can build multiple projects together, publish projects information, deploy projects, share JARs across several projects and help in collaboration of teams.

Advantages of Maven

- Simple project setup that follows best practices.
- Consistent usage across all projects.
- Dependency management including automatic updating.
- A large and growing repository of libraries.
- Extensible, with the ability to easily write plugins in java or scripting languages.
- Instant access to new features with little or no extra configuration.
- Model-based builds: Maven is able to build any number of projects into predefined output types such as jar, war, metadata.

Advantages of Maven

Coherent site of project information: Using the same metadata as per the build process, maven is able to generate a website and a PDF including complete documentation.

- Release management and distribution publication: Without additional configuration, maven will integrate with your source control system such as CVS and manages the release of a project.
- Backward Compatibility: You can easily port the multiple modules of a project into Maven 3 from older versions of Maven. It can support the older versions also.

Advantages of Maven

Automatic parent versioning: No need to specify the parent in the sub module for maintenance.

🕛 Parallel builds: It analyzes the project dependency graph and enables you to build schedule modules in parallel. Using this, you can achieve the performance improvements of 20-50%.

🕛 Better Error and Integrity Reporting: Maven improved error reporting, and it provides you with a link to the Maven wiki page where you will get full description of the error.

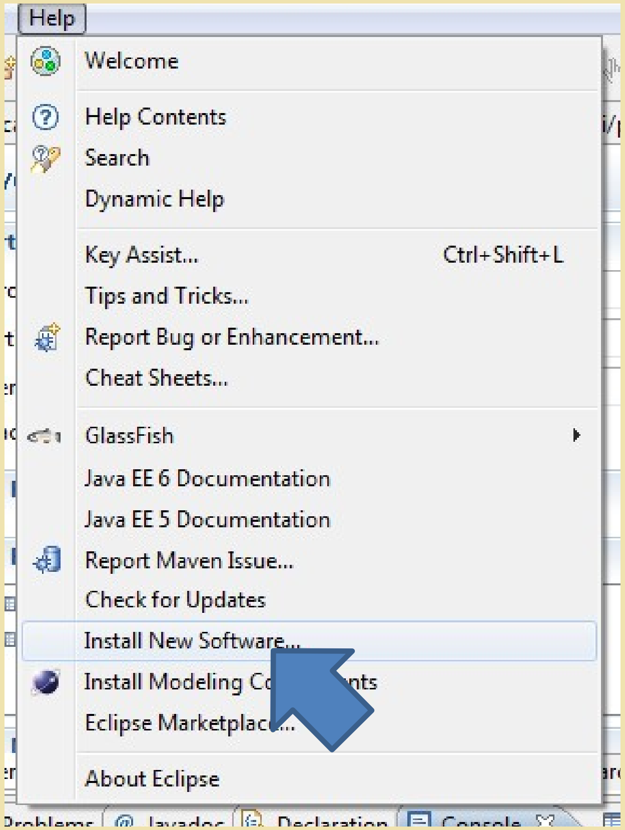
Using Maven (I)

Using Maven

- Installing Maven plugin for Eclipse
  - To install the Sonatype's maven plugin, you can use the traditional installation process.
  - You must use these update sites
    - http://m2eclipse.sonatype.org/sites/m2e
    - http://m2eclipse.sonatype.org/sites/m2e-extras

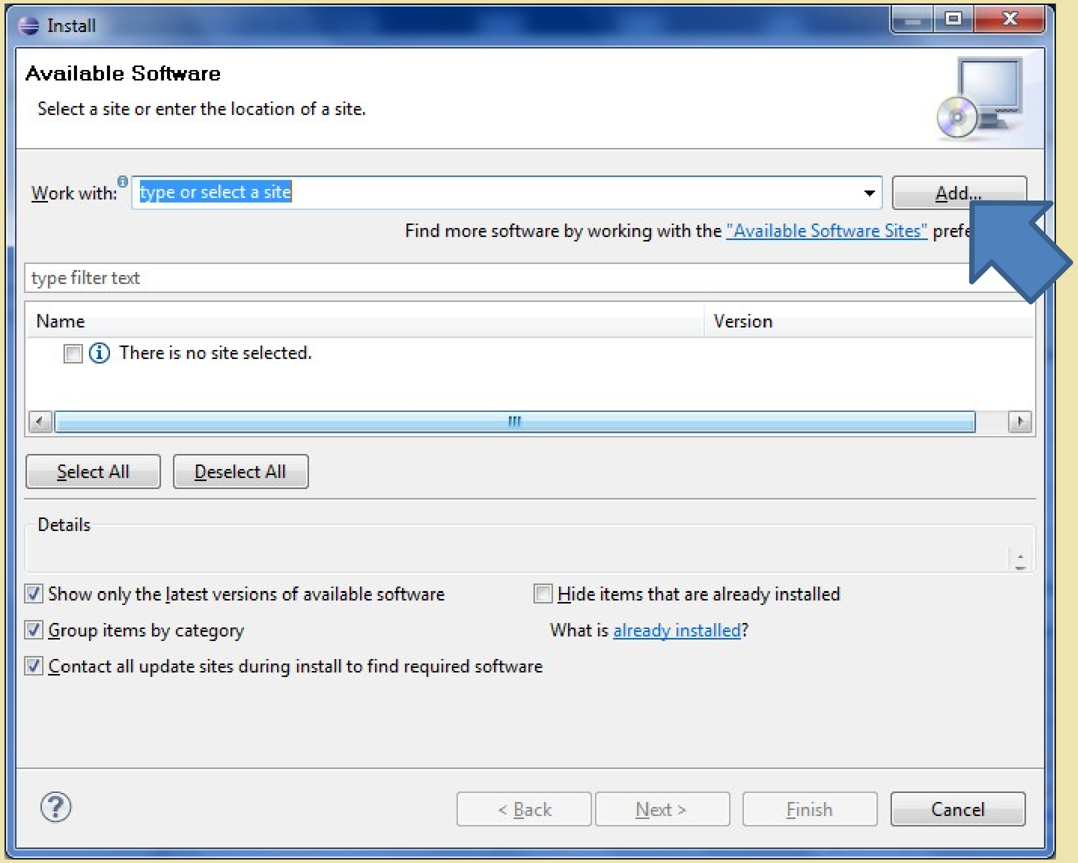
❶Installing Maven plugin for Eclipse

- Select the option “Install New Software” from the Help menu.



❶Installing Maven plugin for Eclipse

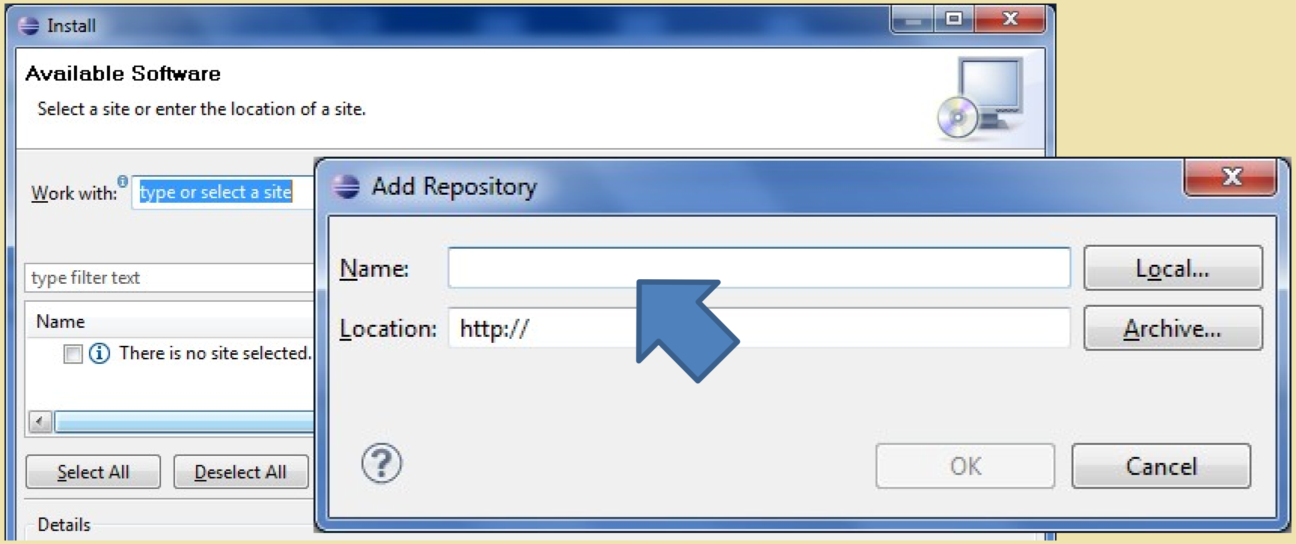
- Add the software update sites into the Eclipse



Install
Available Software
Select a site or enter the location of a site.
Work with: type or select a site
Add...
Find more software by working with the "Available Software Sites" prefe
type filter text
Name
Version
There is no site selected.
Select All
Deselect All
Details
☑ Show only the latest versions of available software
☐ Hide items that are already installed
☑ Group items by category
What is already installed?
☑ Contact all update sites during install to find required software
< Back
Next >
Finish
Cancel

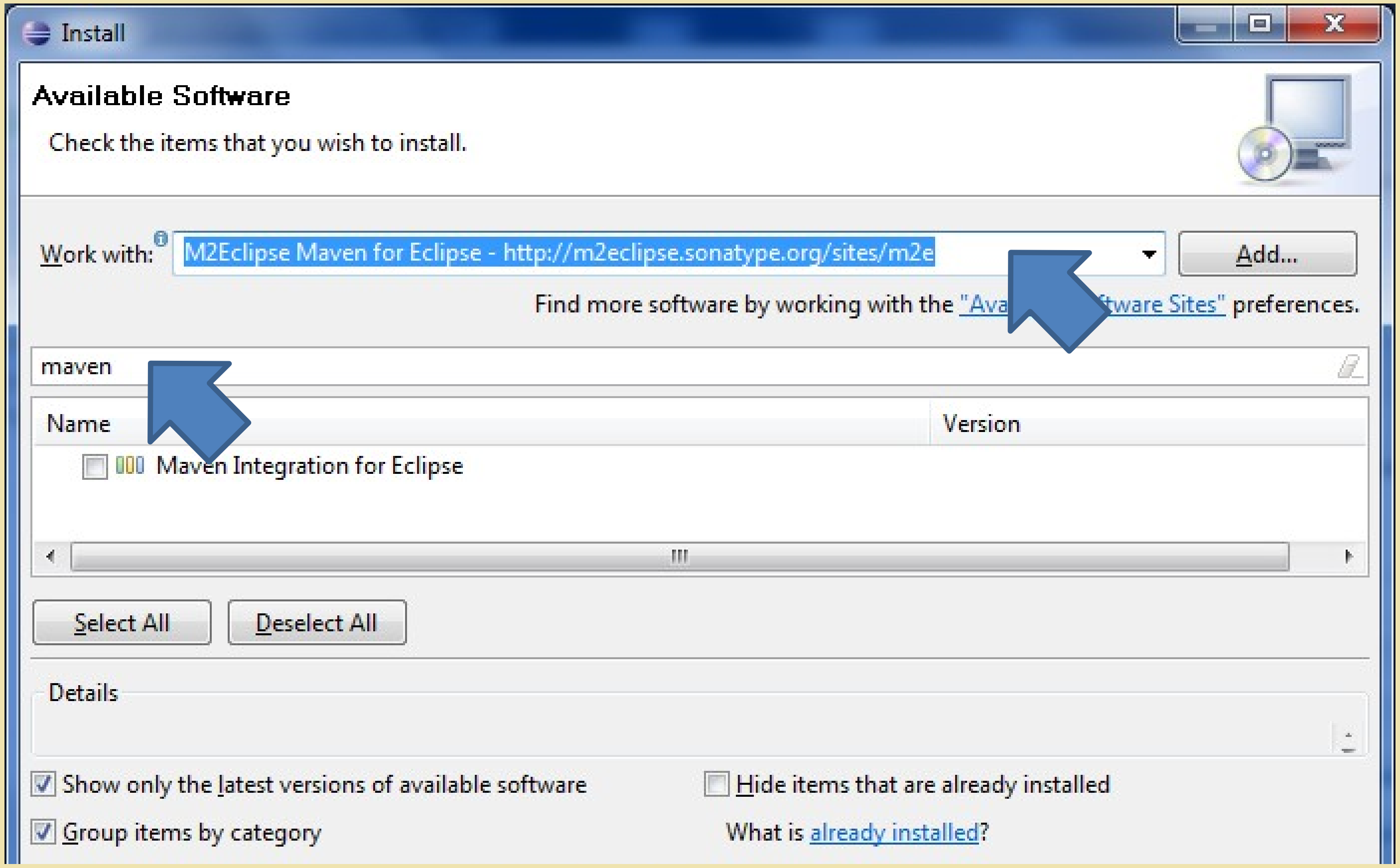
❶Installing Maven plugin for Eclipse

- Define the repositories for the plugin
  - Maven for Eclipse
    - http://m2eclipse.sonatype.org/sites/m2e
  - Maven Extra for Eclipse
    - http://m2eclipse.sonatype.org/sites/m2e-extras



❶Installing Maven plugin for Eclipse

- Search and install the plugin



Install

Available Software

Check the items that you wish to install.

Work with: M2Eclipse Maven for Eclipse - http://m2eclipse.sonatype.org/sites/m2e

Find more software by working with the "Ava tware Sites" preferences.

maven

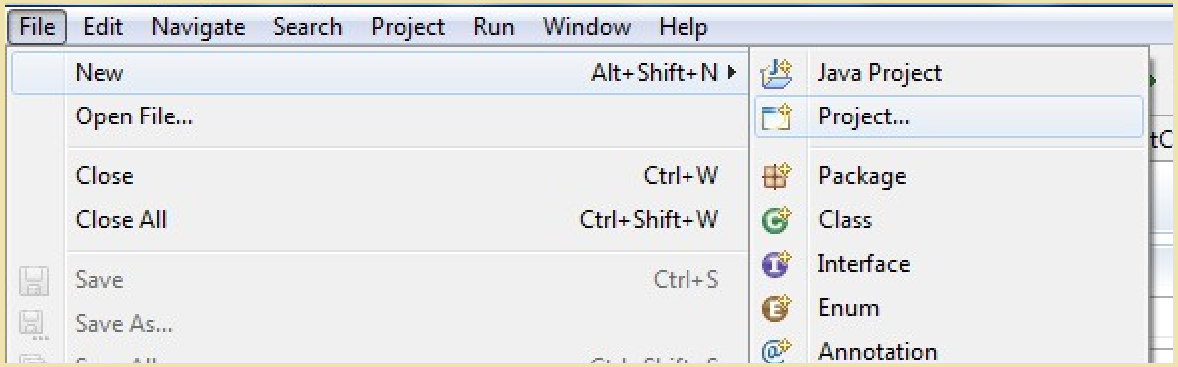
Name	Version
☐ Maven Integration for Eclipse	

Details

☑ Show only the latest versions of available software
☐ Hide items that are already installed
☑ Group items by category
What is already installed?

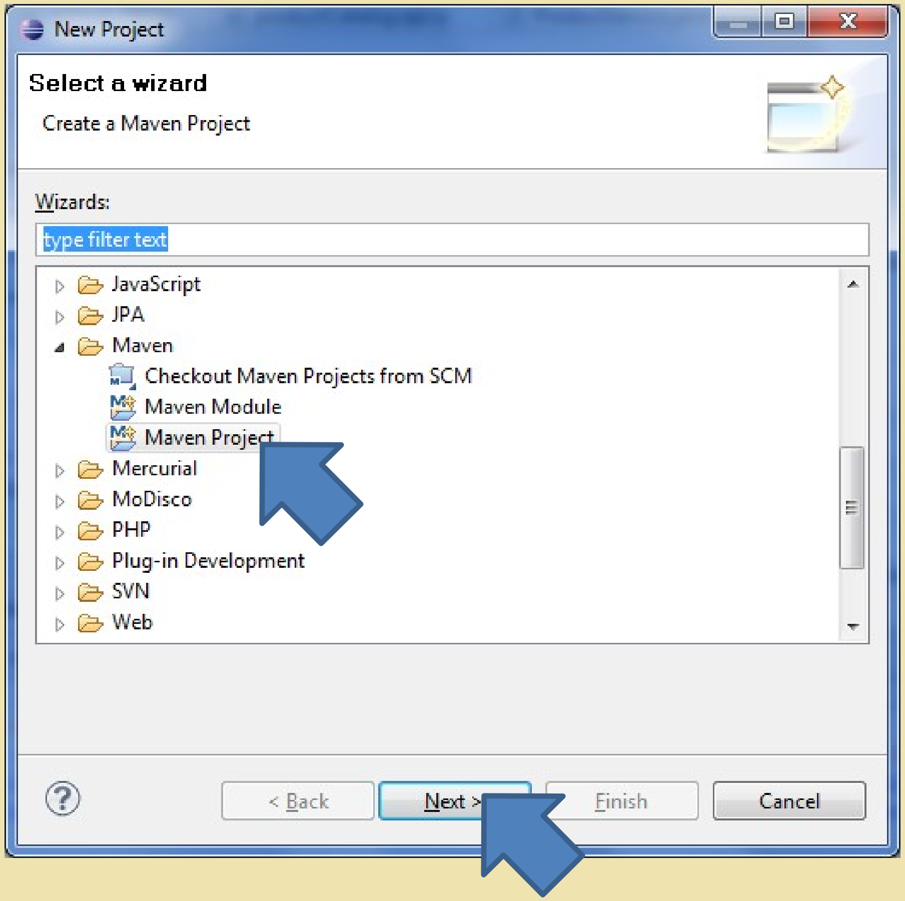
❷ Creating a Project

- Creating a project (using an archetype)
  - Select the option “New” -> “Project…” from the menu “File”



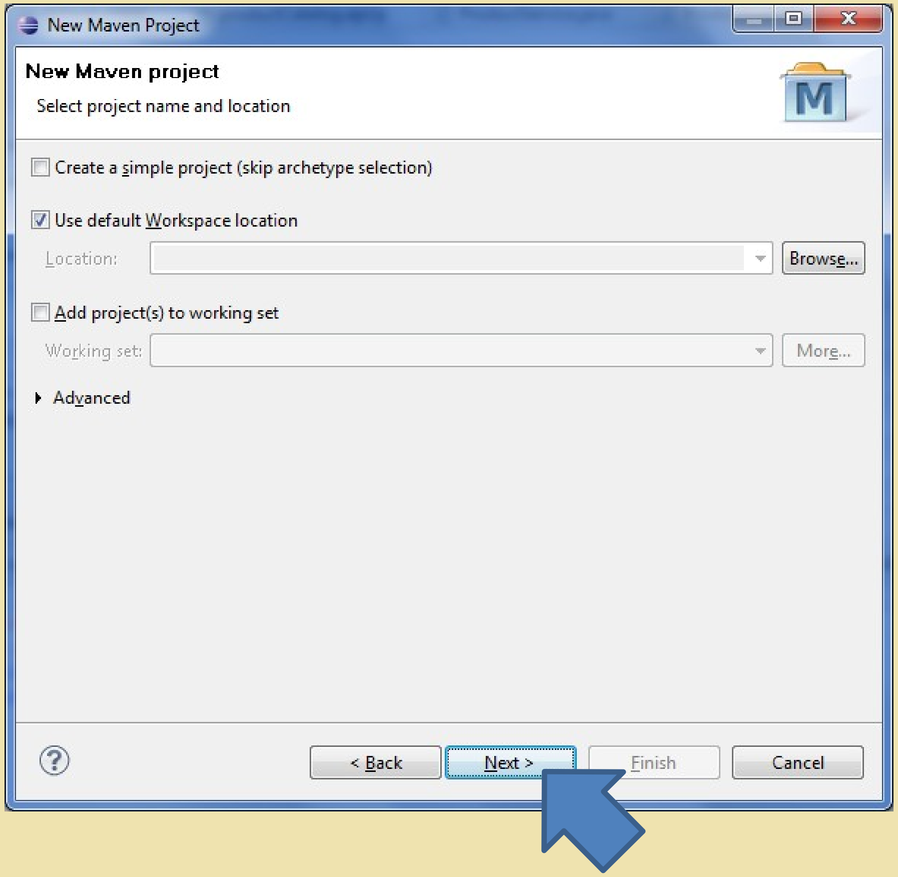
❷ Creating a Project

- Select the “Maven Project” type



❷ Creating a Project

- If it is desired, change the default location and the working set for the new project



New Maven Project

New Maven project

Select project name and location

☐ Create a simple project (skip archetype selection)

☑ Use default Workspace location

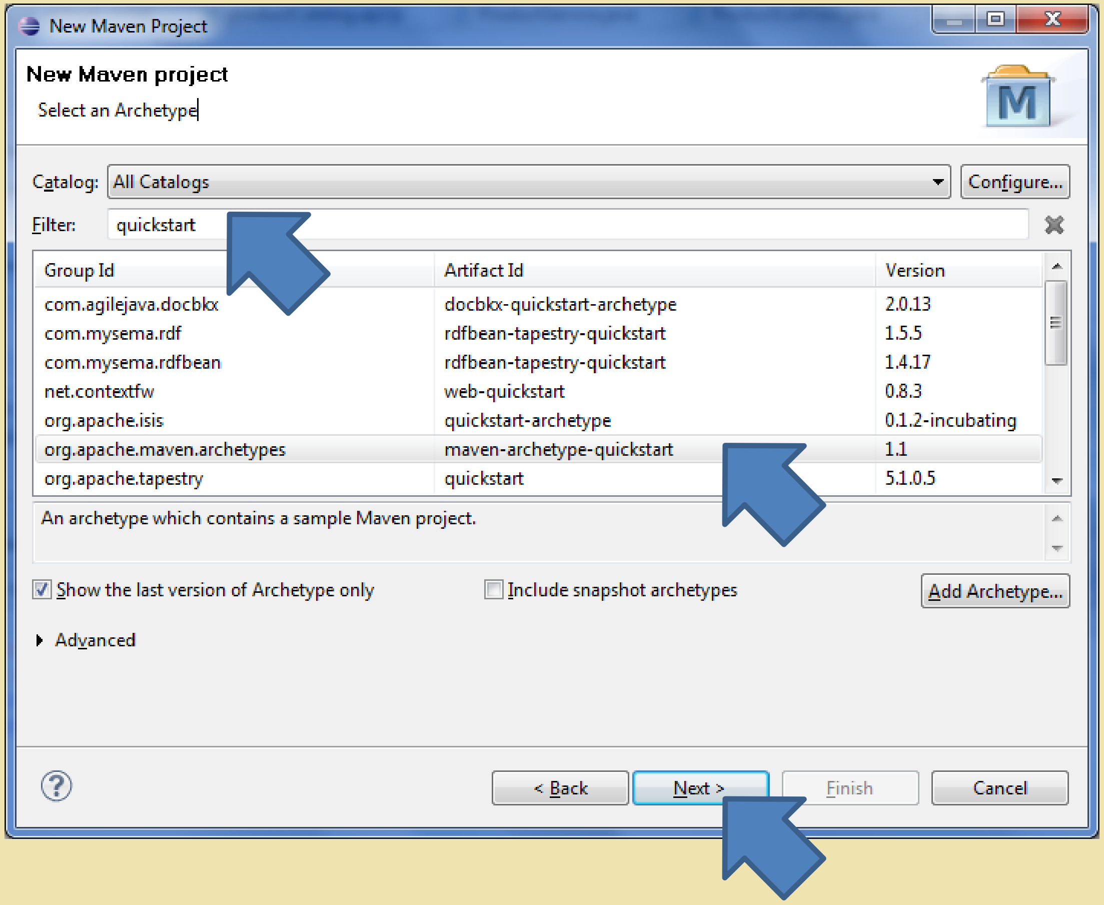
Location:

☐ Add project(s) to working set

Working set:

❷ Creating a Project

- Select the desired archetype. (e.g. maven-archetytpe-quickstart)



Catalog: All Catalogs

Filter: quickstart

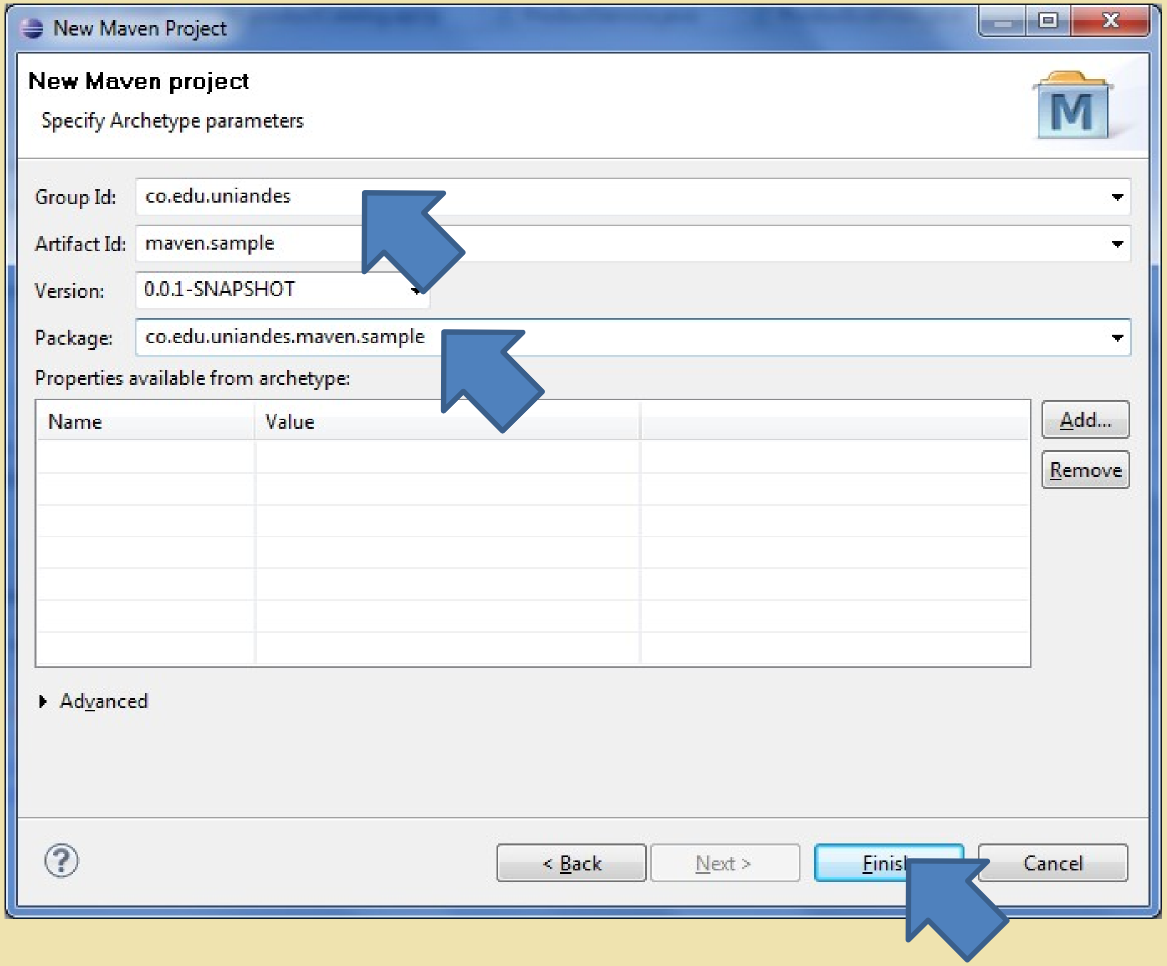
Group Id	Artifact Id	Version
com.agilejava.docbkx	docbkx-quickstart-archetype	2.0.13
com.mysema.rdf	rdfbean-tapestry-quickstart	1.5.5
com.mysema.rdfbean	rdfbean-tapestry-quickstart	1.4.17
net.contextfw	web-quickstart	0.8.3
org.apache.isis	quickstart-archetype	0.1.2-incubating
org.apache.maven.archetypes	maven-archetype-quickstart	1.1
org.apache.tapestry	quickstart	5.1.0.5

An archetype which contains a sample Maven project.

[✓] Show the last version of Archetype only
[ ] Include snapshot archetypes

❷ Creating a Project

- Define the project parameters



New Maven Project

New Maven project

Specify Archetype parameters

Group Id: co.edu.uniandes

Artifact Id: maven.sample

Version: 0.0.1-SNAPSHOT

Package: co.edu.uniandes.maven.sample

Properties available from archetype:

Name	Value	

❷ Creating a Project

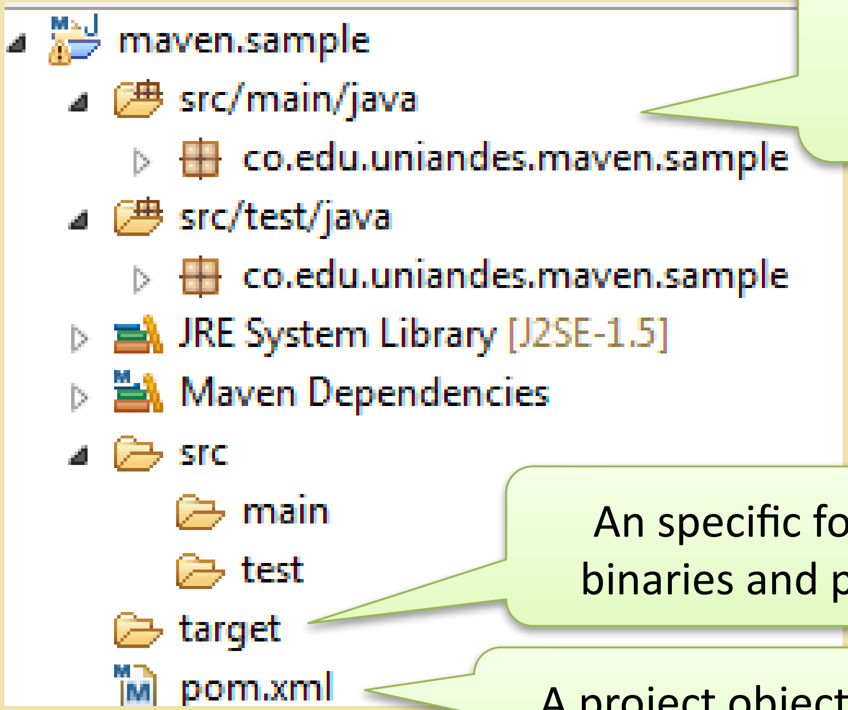
- Project Parameters:
  - **Group Id :** Name of the group of projects. Can be related to the company or application name. Ideally, must be unique (worldwide).
  - **Artifact Id :** Name of the component or application (project)
  - **Version :** Identifier for the version of the component. You can define different versions for the same component.
  - **Package Name :** Name of the java package where the initial source code will be generated.

❸ Reviewing the Project

- Reviewing (Modifying) the project
  - Usually, the archetype is only a skeleton that you can use as foundation for your projects.
  - It can provide source code and files in a maven-standard structure

❸ Reviewing the Project

- Maven-standard structure


maven.sample
src/main/java
co.edu.uniandes.maven.sample
src/test/java
co.edu.uniandes.maven.sample
JRE System Library [J2SE-1.5]
Maven Dependencies
src
main
test
target
pom.xml

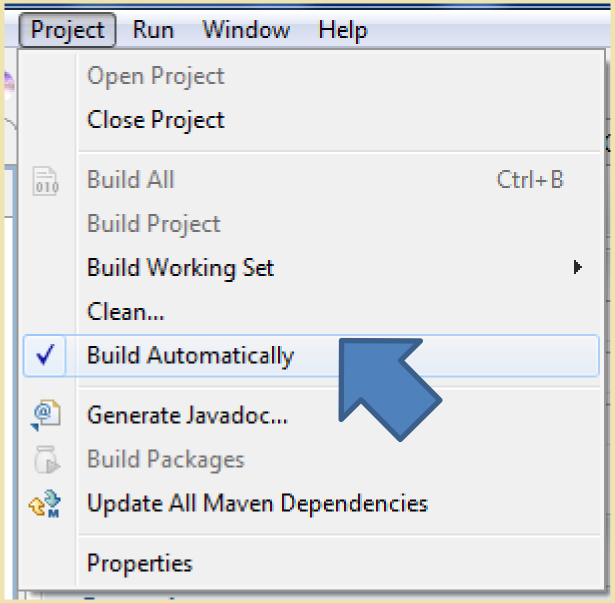
Different folders for application and test source code

An specific folder for binaries and packages

A project object model (POM) specification file

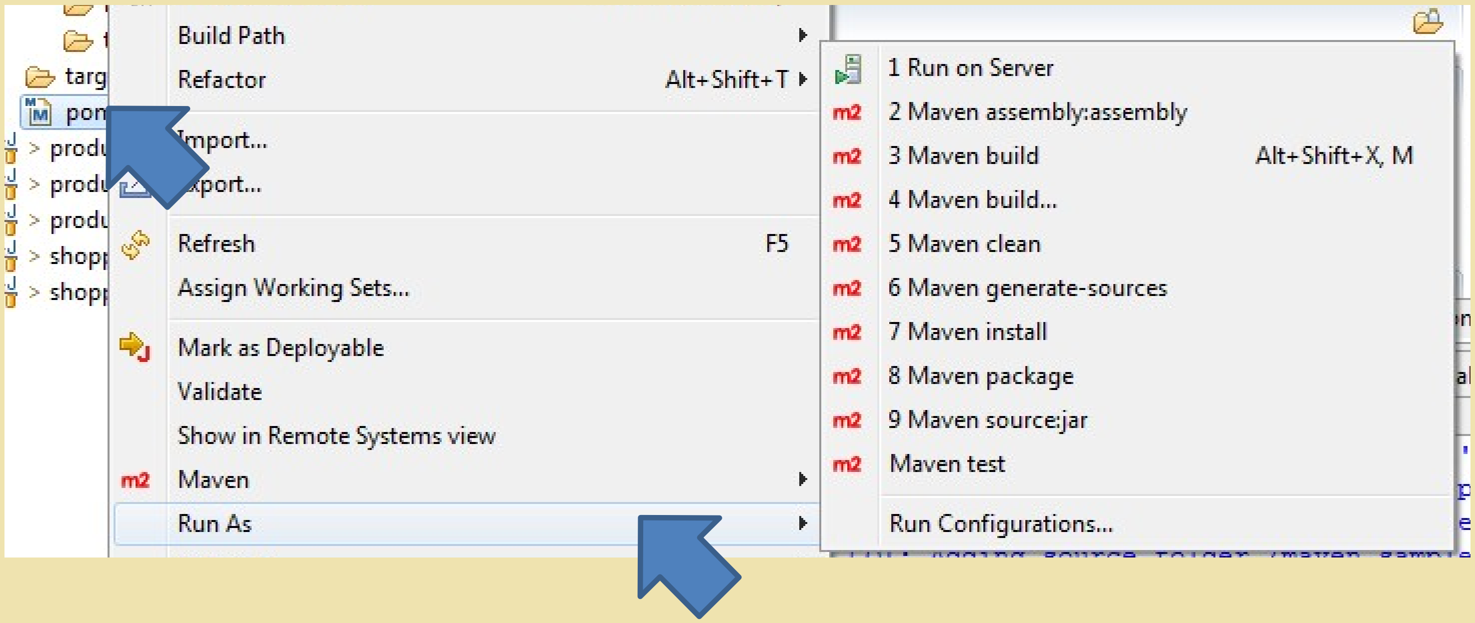
❸ Building the Project

- Building the project
  - If you are using automatic build, the entire project is build after any file modification.



❸ Building the Project

- Building the project
  - However, additional build types can be performed using Maven-specific menu options .



Understanding the POM

Project Object Model

- POM stands for Project Object Model. It is fundamental unit of work in Maven. It is an XML file that resides in the base directory of the project as pom.xml.
- The POM contains information about the project and various configuration detail used by Maven to build the project(s).
- POM also contains the goals and plugins. While executing a task or goal, Maven looks for the POM in the current directory.
- It reads the POM, gets the needed configuration information, and then executes the goal.

Components of pom.xml

- project dependencies
- plugins
- goals
- build profiles
- project version
- developers
- mailing list

Scope in pom.xml

- **Compile**
  - The default scope, used if none is specified. Compile dependencies are available in all class paths of a project. Furthermore, those dependencies are propagated to dependent projects.
- **Provided**
  - This is much like compile, but indicates you expect the JDK or a container to provide the dependency at runtime. Java EE APIs to scope provided because the web container provides those classes. This scope is only available on the compilation and test classpath, and is not transitive

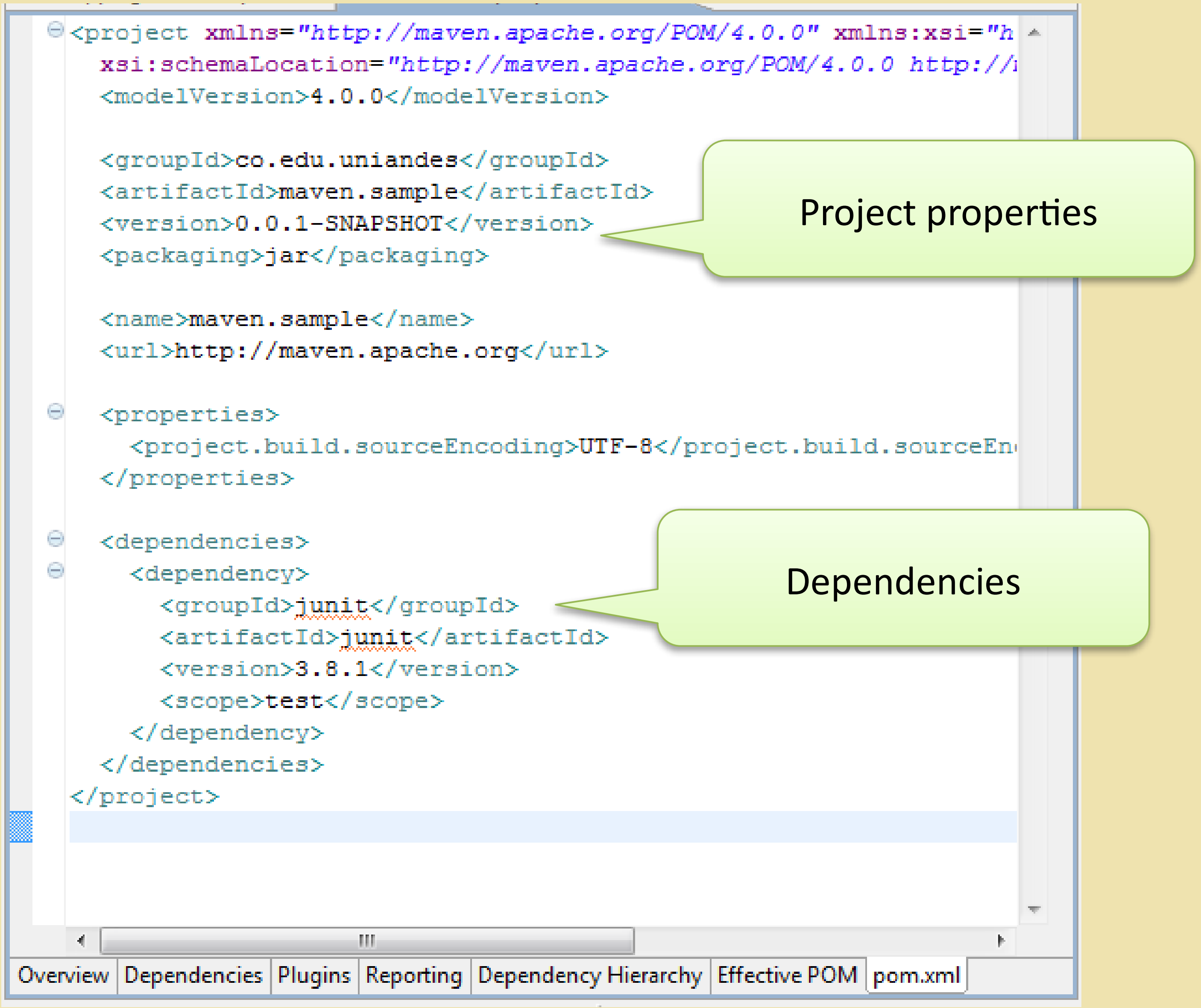
Scope in pom.xml

- **test**
  - This scope indicates that the dependency is not required for normal use of the application, and is only available for the test compilation and execution phases.
- **runtime**
  - This scope indicates that the dependency is not required for compilation, but is for execution. It is in the runtime and test classpaths, but not the compile classpath.

Scope in pom.xml

- **import(only available in Maven 2.0.9 or later)**
  - This scope is only used on a dependency of type pom in the section. It indicates that the specified POM should be replaced with the dependencies in that POM's section. Since they are replaced, dependencies with a scope of import do not actually participate in limiting the transitivity of a dependency.
- **system**
  - This scope is similar to provided except that you have to provide the JAR which contains it explicitly. The artifact is always available and is not looked up in a repository

Understanding the POM



```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="h
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
  <modelVersion>4.0.0</modelVersion>

  <groupId>co.edu.uniandes</groupId>
  <artifactId>maven.sample</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>maven.sample</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEn
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

pom.xml

- First decide the project group (groupId), its name (**artifactId**) and its version as these attributes help in uniquely identifying the project in repository.
- All POM files require the project element and three mandatory fields: **groupId**, **artifactId**, **version**.

pom.xml

Node	Description
Project Root	This is project root tag. You need to specify the basic schema settings such as apache schema and w3.org specification.
Model Version	Model version should be 4.0.0
groupId	This is an Id of project's group. This is generally unique amongst an organization or a project. For example, a banking group com.company.bank has all bank related projects.
artifactId	This is an Id of the project. This is generally name of the project. For example, consumer-banking. Along with the groupId, the artifactId defines the artifact's location within the repository.
version	This is the version of the project. Along with the groupId, It is used within an artifact's repository to separate versions from each other. For example: com.company.bank:consumer-banking:1.0 com.company.bank:consumer-banking:1.1.

Understanding Maven

- Maven Lifecycle
  - Compile
  - Test
  - Package
  - Install
  - Deploy

Understanding Maven

- Additional tasks for Maven
  - Clean
  - Assembly:assembly
  - Site
  - Site-deploy
  - …
- Maven plugins can include additional tasks

Using Maven (II)

Using Maven

- A simple project
  - This first example uses a basic project from Cupi2.
  - Typically, a Cupi2 project includes a set of unit tests. These tests can be integrated into the maven process in order to perform the unit tests and generate reports of the results.

Using Maven

- A more-complex example
  - Several projects can be organized for components (assets) and products.
  - A product can be defined including combinations and dependencies to several components (assets)

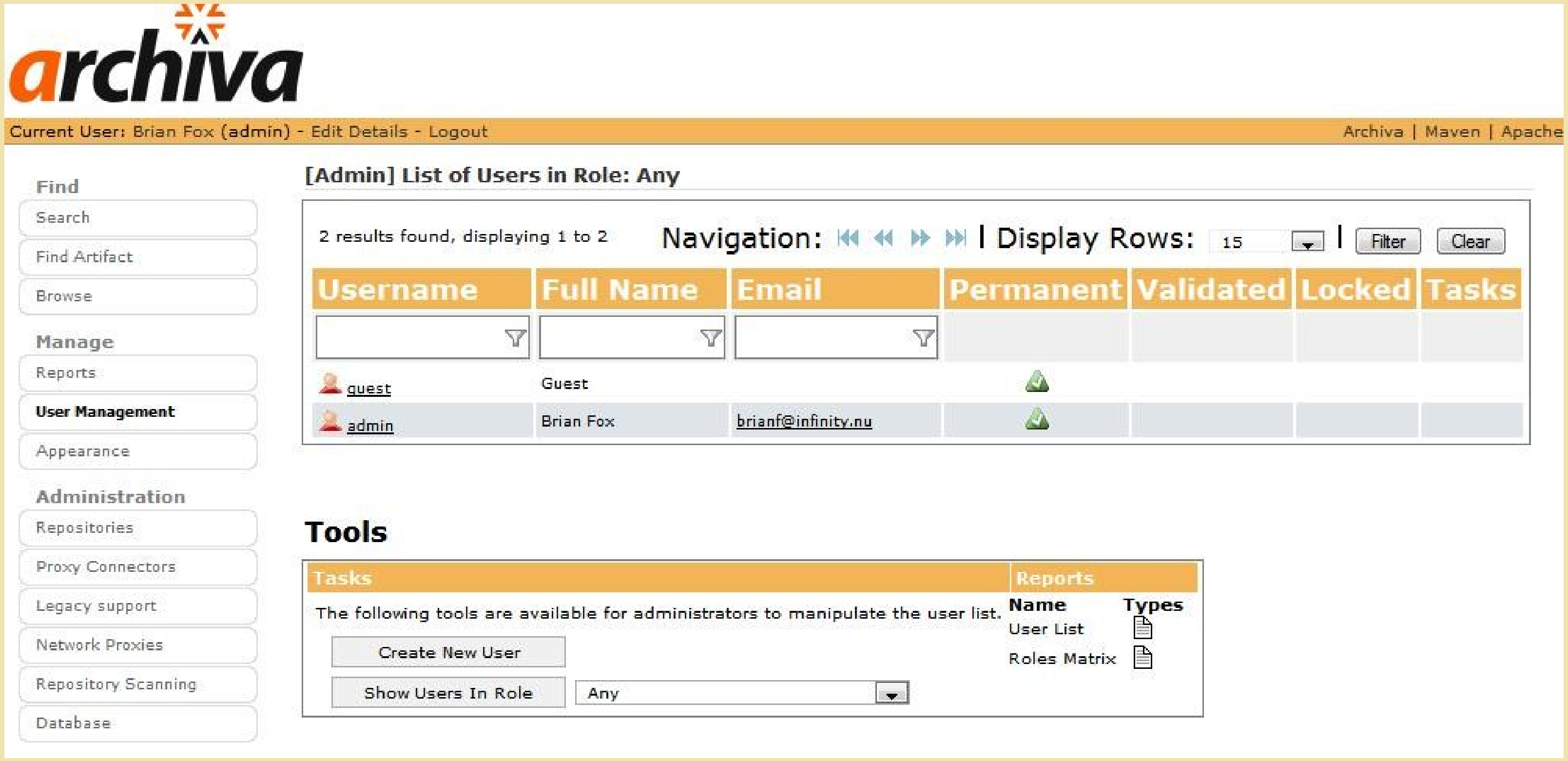
Repository Managers

Repository Managers

- Repository Managers
  - A local maven repository can be configured using SVN.
  - For additional functionality, software for repository management can be also used in a company for its private components and products.

Repository Managers

- Apache Archiva (http://archiva.apache.org)



Current User: Brian Fox (admin) - Edit Details - Logout

[Admin] List of Users in Role: Any

2 results found, displaying 1 to 2 Navigation: | Display Rows: 15 | Filter Clear

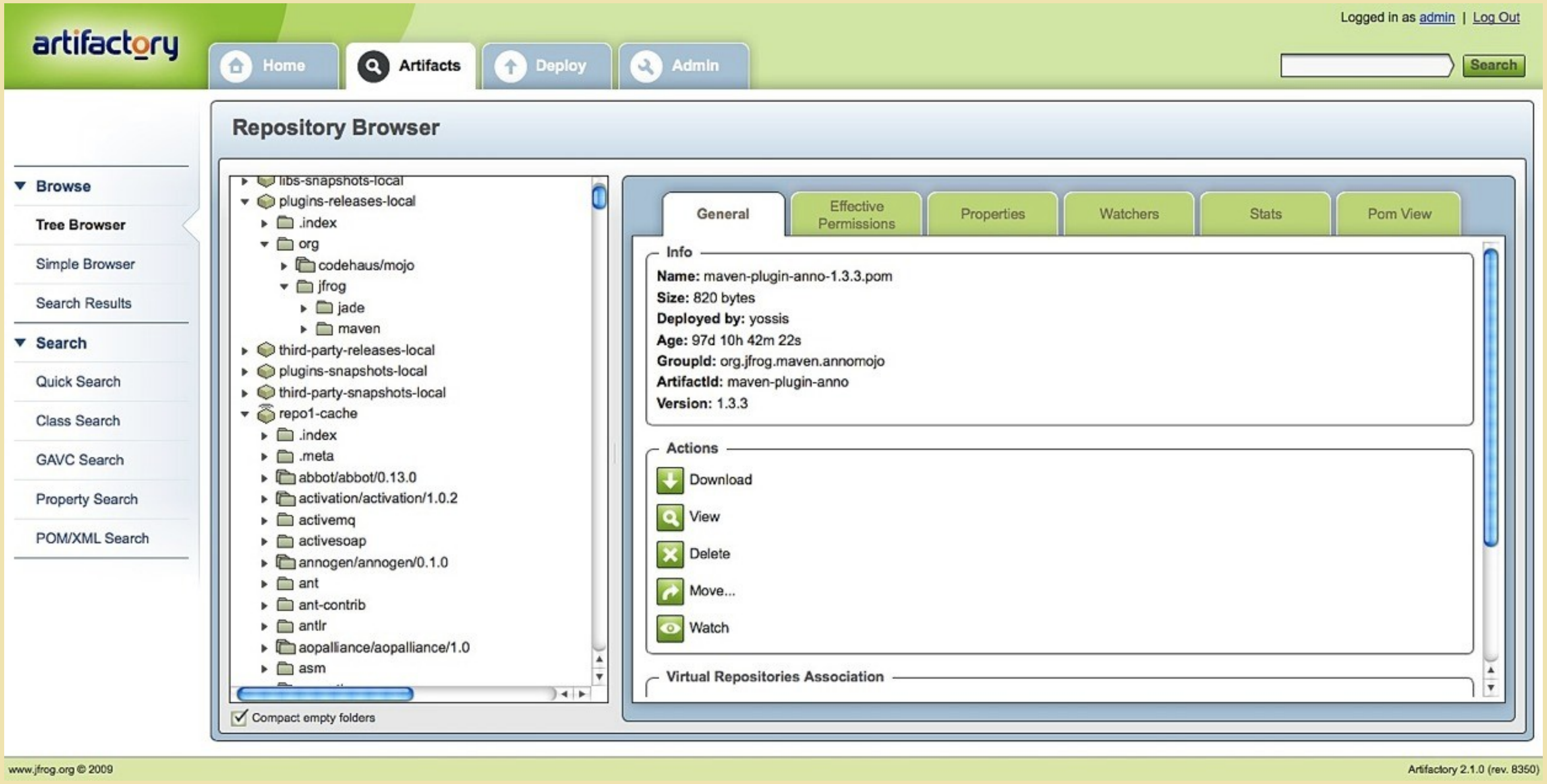
Username	Full Name	Email	Permanent	Validated	Locked	Tasks
guest	Guest			✓		
admin	Brian Fox	brianf@infinity.nu		✓		

Tools

Tasks	Reports	
The following tools are available for administrators to manipulate the user list.	Name	Types
Create New User	User List	🗎
Show Users In Role: Any	Roles Matrix	🗎

Repository Managers

- Artifactory (http://sourceforge.net/projects/artifactory/)

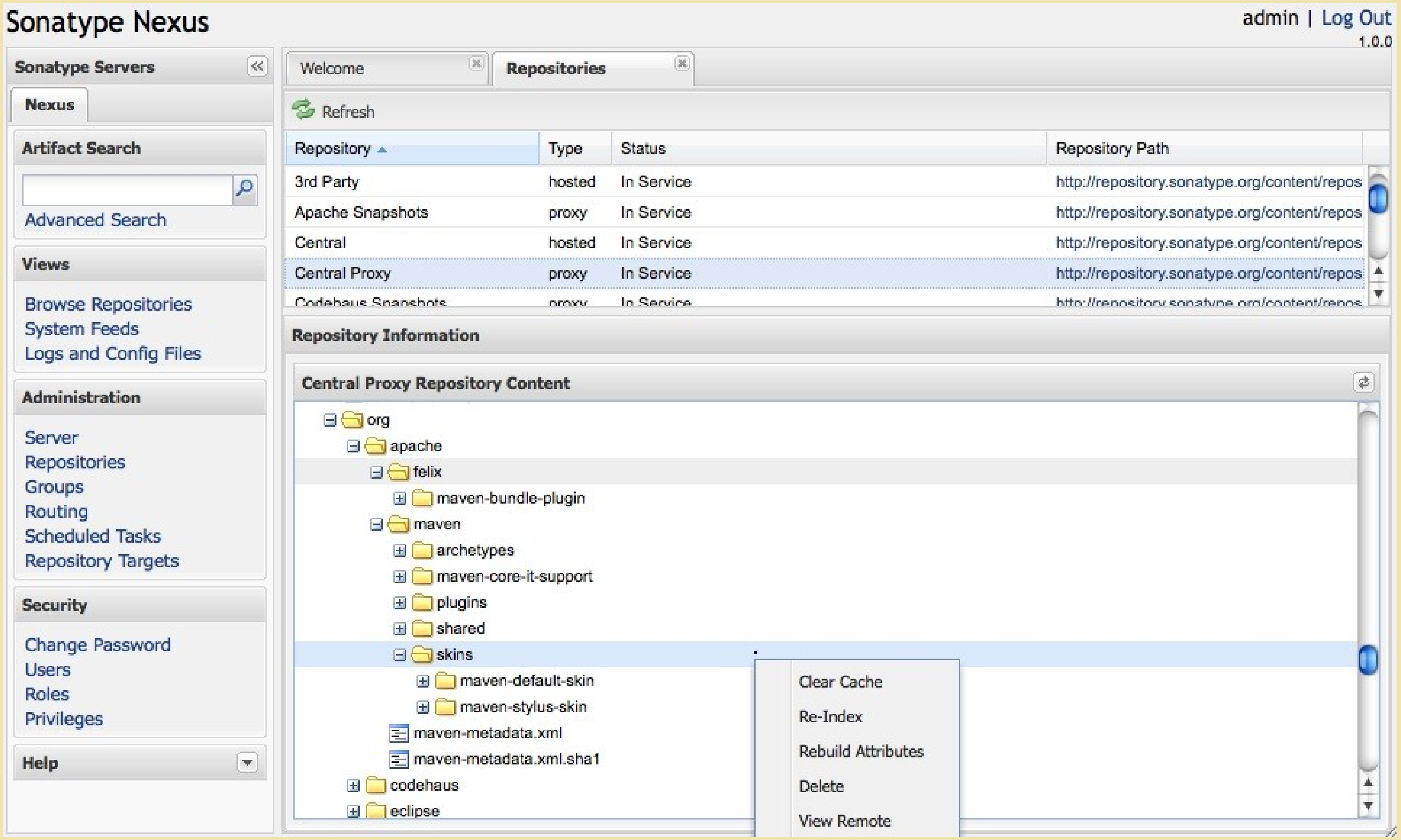


Name: maven-plugin-anno-1.3.3.pom
Size: 820 bytes
Deployed by: yossis
Age: 97d 10h 42m 22s
GroupId: org.jfrog.maven.annomojo
ArtifactId: maven-plugin-anno
Version: 1.3.3

Repository Managers

Sonatype Nexus

https://www.sonatype.com/products/repository-oss?topnav=true



Repository	Type	Status	Repository Path
3rd Party	hosted	In Service	http://repository.sonatype.org/content/repos
Apache Snapshots	proxy	In Service	http://repository.sonatype.org/content/repos
Central	hosted	In Service	http://repository.sonatype.org/content/repos
Central Proxy	proxy	In Service	http://repository.sonatype.org/content/repos
Codehaus Snapshots	proxy	In Service	http://repository.sonatype.org/content/repos

```
⊟ org
  ⊟ apache
    ⊟ felix
      ⊞ maven-bundle-plugin
    ⊟ maven
      ⊞ archetypes
      ⊞ maven-core-it-support
      ⊞ plugins
      ⊞ shared
      ⊟ skins
        ⊞ maven-default-skin
        ⊞ maven-stylus-skin
      maven-metadata.xml
      maven-metadata.xml.sha1
  ⊞ codehaus
  ⊞ eclipse
```

Super pom

- A POMs inherit from a parent or default (despite explicitly defined or not). This base POM is known as the Super POM, and contains values inherited by default.
- An easy way to look at the default configurations of the super POM is by running the following command: **mvn help:effective-pom**.
- We've created a pom.xml in C:\MVN\project folder. Now open the command console, go to the folder containing pom.xml and execute the following mvn command.

```
C:\MVN\project>mvn help:effective-pom
```

Maven Build Life Cycle

- A POMs inherit from a parent or default (despite explicitly defined or not). This base POM is known as the Super POM, and contains values inherited by default.
- An easy way to look at the default configurations of the super POM is by running the following command: **mvn help:effective-pom**.
- We've created a pom.xml in C:\MVN\project folder. Now open the command console, go to the folder containing pom.xml and execute the following mvn command.

```
C:\MVN\project>mvn help:effective-pom
```

Maven Build Life Cycle

- A Build Lifecycle is a well-defined sequence of phases, which define the order in which the goals are to be executed.
- Here phase represents a stage in life cycle.
- A typical Maven Build Life cycle consists of the following sequence of phases.
- There are always **pre** and **post** phases to register goals, which must run prior to, or after a particular phase.

Phase	**Handles**	**Description**
prepare-resources	Resource copying	Resource copying can be customized in this Phase.
validate	Validating the information	Validates if the project is correct and if all necessary information is available.
compile	compilation	Source code compilation is done in this phase
test	testing	Tests the compiled source code suitable for testing framework
package	packaging	This phase creates the JAR/WAR package as mentioned in the packaging in POM.xml
install	installation	This phase installs the package in local/remote maven repository
deploy	deploying	Copies the final package to the remote repository

Maven Build Life Cycle

- When Maven starts building a project, it steps through a defined sequence of phases and executes goals, which are registered with each phase.
- A goal represents a specific task which contributes to the building and managing of a project. It may be bound to zero or more build phases.
- A goal not bound to any build phase could be executed outside of the build lifecycle by direct invocation.
- The order of execution depends on the order in which the goal(s) and the build phase(s) are invoked.
- The clean and package arguments are build phases while the dependency:copy-dependencies is a goal.

- Here the clean phase will be executed first, followed by the dependency:copy-dependencies goal, and finally package phase will be executed

Clean Lifecycle

- When we execute mvn post-clean command, Maven invokes the clean lifecycle consisting of the following phases.
- 🕐 pre-clean
- 🕐 clean
- 🕐 post-clean
- Maven clean goal (clean:clean) is bound to the clean phase in the clean lifecycle. Its clean:cleangoal deletes the output of a build by deleting the build directory. Thus, when mvn clean command executes, Maven deletes the build directory. We can customize this behavior by mentioning goals in any of the above phases of clean life cycle.
- In the following example, We'll attach **maven-antrun-plugin:run** goal to the pre-clean, clean, and post-clean phases. This will allow us to echo text messages displaying the phases of the clean lifecycle.
- C:\MVN\project>mvn post-clean