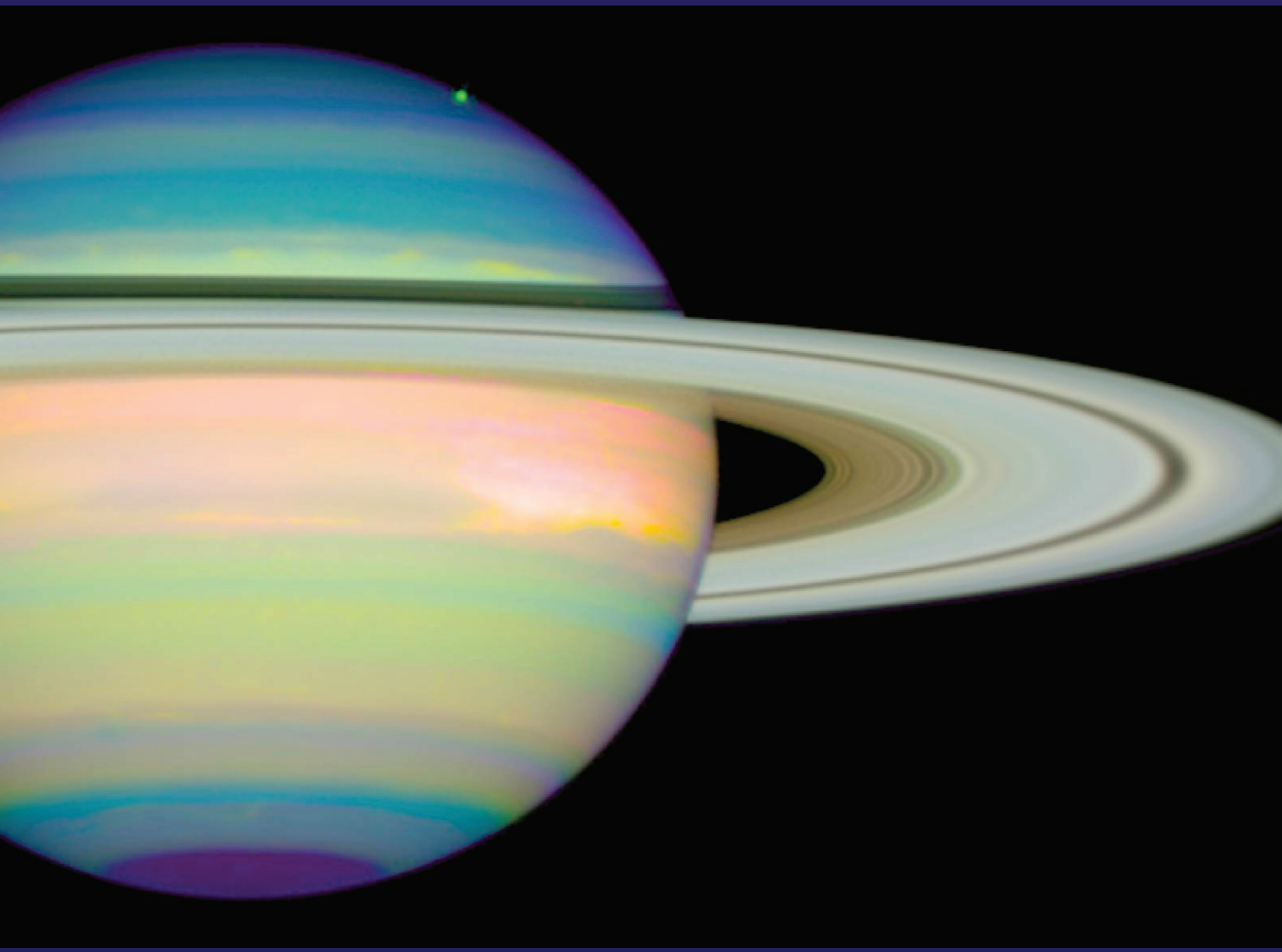
June 2009
Edition 0.4

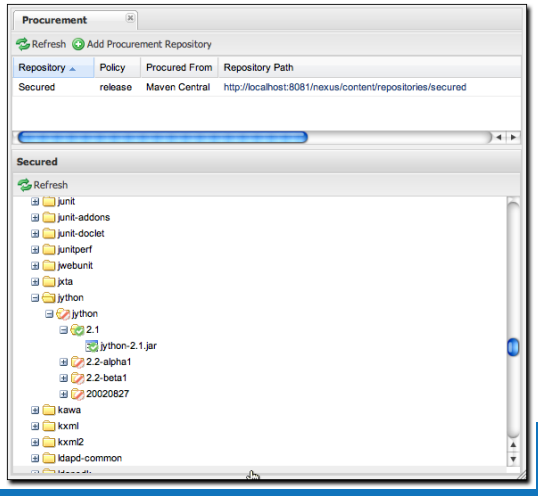


Developing with Eclipse and Maven

A Sonatype Open Book

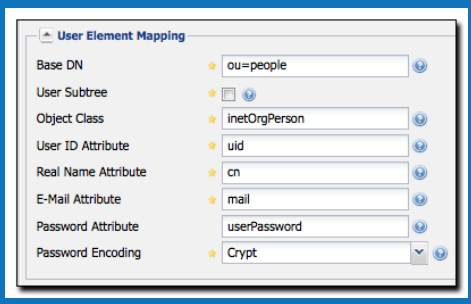
Nexus Professional

Defining Procurement Rules



- Manage Staged Software Releases
- Control Artifact Procurement from Remote Repositories
- Proxy, Host, and Manage P2 Repositories
- Proxy, Host, and Manage Eclipse Update Sites
- Proxy, Host, and Manage OSGi Bundle Repositories
- Scheduled Configuration Backup and Archiving
- Define Global settings.xml templates in Nexus
- Distribute Configuration with the Nexus Maven Plugin
- Effortless Integration with Enterprise LDAP

LDAP User Mapping



Every License Includes One Year of Support

Download a Free Trial Today at:
sonatype.com/products/downloads

http://www.sonatype.com/products/nexus

Nexus Open Source

- Proxy Remote Maven Repositories
- Manage and Host Maven Repositories
- Consolidate Access with Repository Groups
- Rich UI using the amazing ExtJS library
- Advanced task scheduling
- RSS Feeds of new artifacts, system changes, etc

http://nexus.sonatype.org

Sonatype Training

- Covers Maven, Nexus, Eclipse, and Hudson in 2 Days of Hands-on Instruction
- Open Enrollment Training in Chicago, Mountain View, New York, London
- Custom, On Site Training Available on Request

http://www.sonatype.com/training

Copyright

Online version published by Sonatype, Inc., 800 W. El Camino Real, Suite 400, Mountain View, CA, 94040.

Foreword: 0.4-SNAPSHOT

The book you are reading was originally a part of Maven: The Definitive Guide. As Sonatype expanded our coverage of this chapter for the 0.9.7 release we decided to separate the m2eclipse chapter into a stand-alone title.

You are encouraged to check http://books.sonatype.com/m2eclipse-book weekly for any updates to the content. If you would like to receive updates whenever there is a change to this user's manual, please send a quick note to book@sonatype.com, and we will gladly add your email to the update notification list.

We welcome your feedback, please do not hesitate to contact the Sonatype team with any questions or ideas you may have about the product.

Tim O'Brien, Sonatype

February 11, 2009

Edition: 0.4-SNAPSHOT

Chapter 1. Introduction to m2eclipse

1.1. Introduction

The Eclipse IDE is the most widely used IDE for Java development today. Eclipse has a huge amount of plugins (see http://www.eclipseplugincentral.com/) and an innumerable amount of organizations developing their own software on top of it. Quite simply, Eclipse is ubiquitous. The m2Eclipse project, provides support for Maven within the Eclipse IDE, and, in this chapter, we will explore the features it provides to help you use Maven with Eclipse.

1.2. m2eclipse

The m2Eclipse plugins (http://m2eclipse.codehaus.org/) provide Maven integration for Eclipse. m2Eclipse also has hooks into the features of both the Subclipse plugin (http://subclipse.tigris.org/) and the Mylyn plugin (http://www.eclipse.org/mylyn/). The Subclipse plugin provides the m2eclipse plugin with the ability to interact with Subversion repositories, and the Mylyn plugin provides the m2eclipse plugin with the ability to interact with a task-focused interface that can keep track of development context. Just a few of the features m2eclipse provides include:

- Creating and importing Maven projects
- Dependency management and integration with the Eclipse classpath
- Automatic dependency downloads and updates
- Artifact Javadoc and source resolution
- Creating projects with Maven Archetypes
- Browsing and searching remote Maven repositories
- POM management with automatic update to dependency list
- Materializing a project from a Maven POM
- Checking out a Maven project from several SCM repositories
- Adapting nested multi-module Maven projects to the Eclipse IDE
- Integration with Web Tools Project (WTP)
- Integration with AspectJ Development Tools (AJDT)

- Integration with Subclipse
- Integration with Mylyn
- Form-based POM Editor
- Graphical Display of Dependency Graph
- GUI Presentation of Dependency Tree and Resolved Dependencies

There are many more features in m2eclipse beyond the list above and this chapter introduces some of the more impressive features that are currently available. Let’s get started by installing the m2Eclipse plugin.

Chapter 2. Installing m2eclipse

2.1. Introduction

To install the m2Eclipse plugin, you will need to install some prerequisites. You will need to be running Eclipse 3.2 or higher, JDK 1.4 or higher, and you will also need to make sure that Eclipse is running on a JDK and not a JRE. Once you have Eclipse and a compatible JDK, you will need to install two Eclipse plugins: Subclipse and Mylyn.

2.2. Installing Prerequisites

You can install these prerequisites when you install m2eclipse, just add a new remote update site to Eclipse for each of the prerequisite components. To install these prerequisites, go to Help → Software Updates → Find and Install.... Selecting this menu item will load the Install/Update dialog box. Choose the "Search for new features to install" option and click Next. You will then be presented with a list of "Update sites to visit". Click New Remote Site..., and add a new update site for each new prerequisite. Add a new remote site for each plugin and then make sure that the remote site is selected. After you click Finish, Eclipse will then ask you to select plugins components to install. Select the components you want to install and Eclipse will download, install, and configure your plugins.

Note that if you are using a recent build of Eclipse 3.4 (Ganymede) your plugin installation experience may be slightly different. In Ganymede, you will select Help → Software Updates... which will load the "Software Updates and Add-ons" dialog. In this dialog, choose the Available Software panel and click on Add Site... which will load the simple "Add Site" dialog. Enter the URL of the update site you wish to add and click OK. In the "Software Updates and Add-ons" dialog, the available plugins from an update site will appear as soon as the site is added. You can then select the modules you want to install and click the Install... button. Eclipse will then resolve all the dependencies for the selected plugins, and ask you to agree to the plugin license. After Eclipse installs new plugins, it will likely ask you for permission to restart.

2.2.1. Installing Subclipse

To install Subclipse, use the Eclipse plugin update site listed below.

- Subclipse 1.2: http://subclipse.tigris.org/update_1.2.x

For other versions of Subclipse, and for more information about the Subclipse plugin, please see the Subclipse project's web site at http://subclipse.tigris.org/.

2.2.2. Installing Mylyn

To install JIRA integration with Mylyn, add the Mylyn extras Eclipse update URL, you'll want to do this if your organization uses Atlassian's JIRA for issue tracking. To install Mylyn use the following update sites:

- Mylyn (Eclipse 3.3): http://download.eclipse.org/tools/mylyn/update/e3.3
- Mylyn (Eclipse 3.4): http://download.eclipse.org/tools/mylyn/update/e3.4
- Mylyn Extras (JIRA Support): http://download.eclipse.org/tools/mylyn/update/extras

For more information about the Mylyn project, see the Mylyn project's web site at http://www.eclipse.org/mylyn/.

2.2.3. Installing AspectJ Development Tools (AJDT)

If you are installing the 0.9.4 release of m2eclipse, you may also want to install both the Web Tools Platform (WTP) and the AspectJ Development Tools (AJDT). To install the AJDT use one of the following update URLs in Eclipse:

- AJDT (Eclipse 3.3): http://download.eclipse.org/tools/ajdt/33/update
- AJDT (Eclipse 3.4): http://download.eclipse.org/tools/ajdt/34/dev/update

For more information about the AJDT project, see the AJDT project's web site at http://www.eclipse.org/ajdt/.

2.2.4. Installing the Web Tools Platform (WTP)

To install the Web Tools Platform (WTP). Use one of the following update URLs in Eclipse, or just look for the Web Tools Project in the Discovery Site which should already be in your Eclipse remote update sites list.

- WTP: http://download.eclipse.org/webtools/updates/

For more information about the Web Tools Platform, see the Web Tools Platform project's web site at http://www.eclipse.org/webtools/.

2.3. Installing m2eclipse

Once you've installed the prerequisites, you can install the m2eclipse plugin from the following Eclipse update URL:

- m2eclipse Plugin: http://m2eclipse.sonatype.org/update/

If you would like to install the latest snapshot development version of the plugin, you should use the update-dev URL instead of the previous URL:

- m2eclipse Plugin (Development Snapshot): http://m2eclipse.sonatype.org/update-dev/

To install m2eclipse, just add a the appropriate update site for m2eclipse. Go to Help → Software Updates → Find and Install.... Selecting this menu item will load the Install/Update dialog box. Choose the "Search for new features to install" option and click Next. You will then be presented with a list of "Update sites to visit". Click New Remote Site..., and add a new update site for m2eclipse. Add a new remote site for m2eclipse and then make sure that the remote site is selected. After you click Finish, Eclipse will then ask you to select plugins components to install. Select the components you want to install and Eclipse will download, install, and configure m2eclipse.

If you've installed the plugin successfully, you should see a Maven option in the list of preferences options when you go to Window → Preferences....

Chapter 3. Using m2eclipse

3.1. Enabling the Maven Console

Before we begin to examine the features of m2eclipse, let's first enable the Maven console. Open the Console View by going to Window → Show View → Console. Then click on the little arrow on the right-hand side of the Open Console icon and select Maven Console as shown below:

Figure 3.1. Enabling the Maven Console in Eclipse

Maven Console shows the Maven output that normally appears on the console when running Maven from the command line. It is useful to be able to see what Maven is doing and to work with Maven debug output to diagnose issues.

3.2. Creating a Maven Project

When using Maven, project creation takes place through the use of a Maven archetype. In Eclipse, project creation takes place via the new project wizard. The new project wizard inside of Eclipse offers a plethora of templates for creating new projects. The m2eclipse plugin improves upon this wizard to provide the following additional capabilities:

- Checking out a Maven project from a SCM repository
- Creating a Maven project using a Maven archetype
- Creating a Maven POM file

As shown in Figure 3.2, "Creating a New Project with m2eclipse Wizards", all three of these options are important to developers using Maven. Let's take a look at each one.

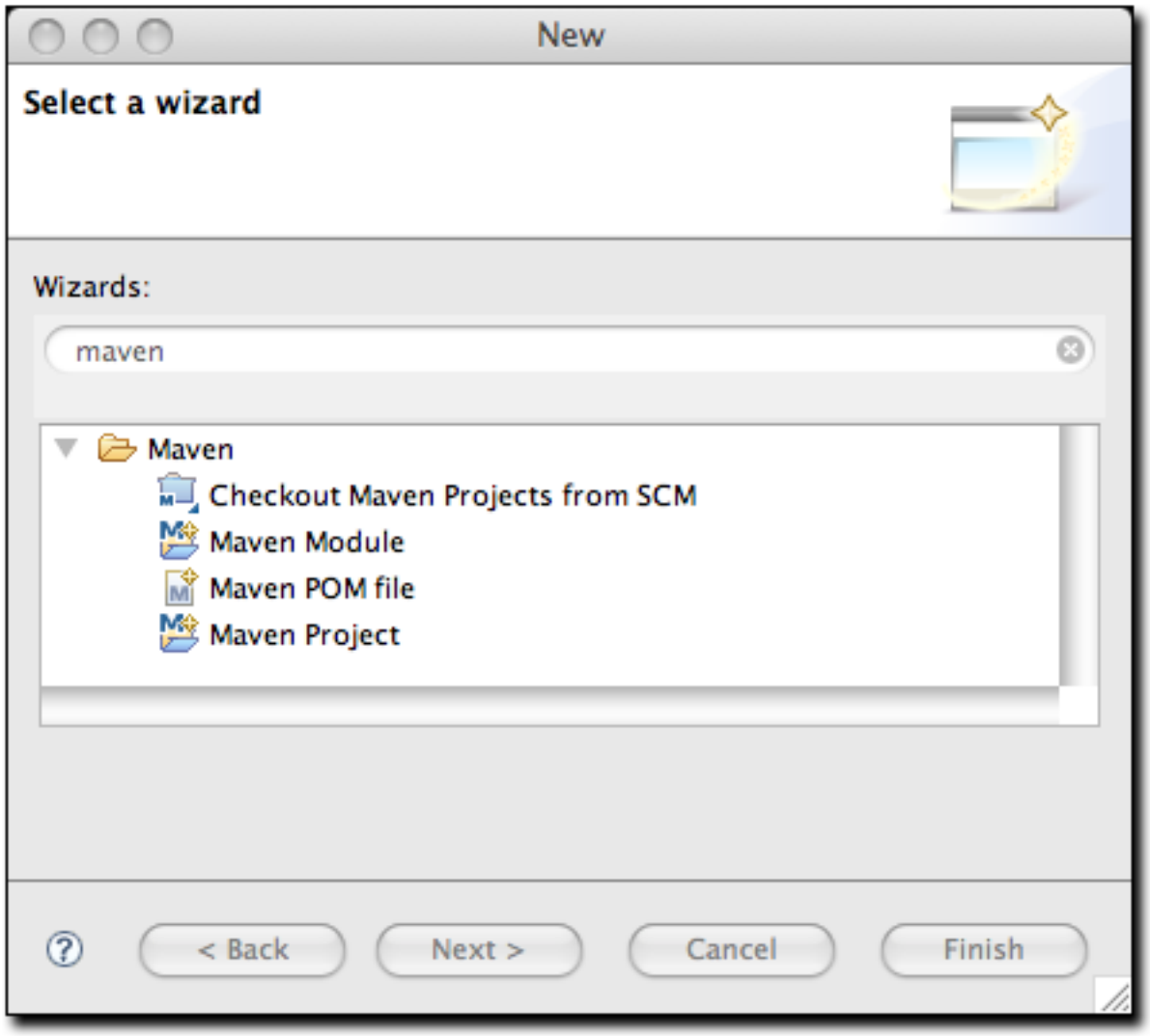


Figure 3.2. Creating a New Project with m2eclipse Wizards

3.2.1. Checking Out a Maven Project from SCM

m2eclipse provides the ability to check out a project directly from a SCM repository. Simply enter the SCM information for a project and it will check it out for you to a location of your choice as shown in Figure 3.3, “Checkout a New Project from Subversion”:

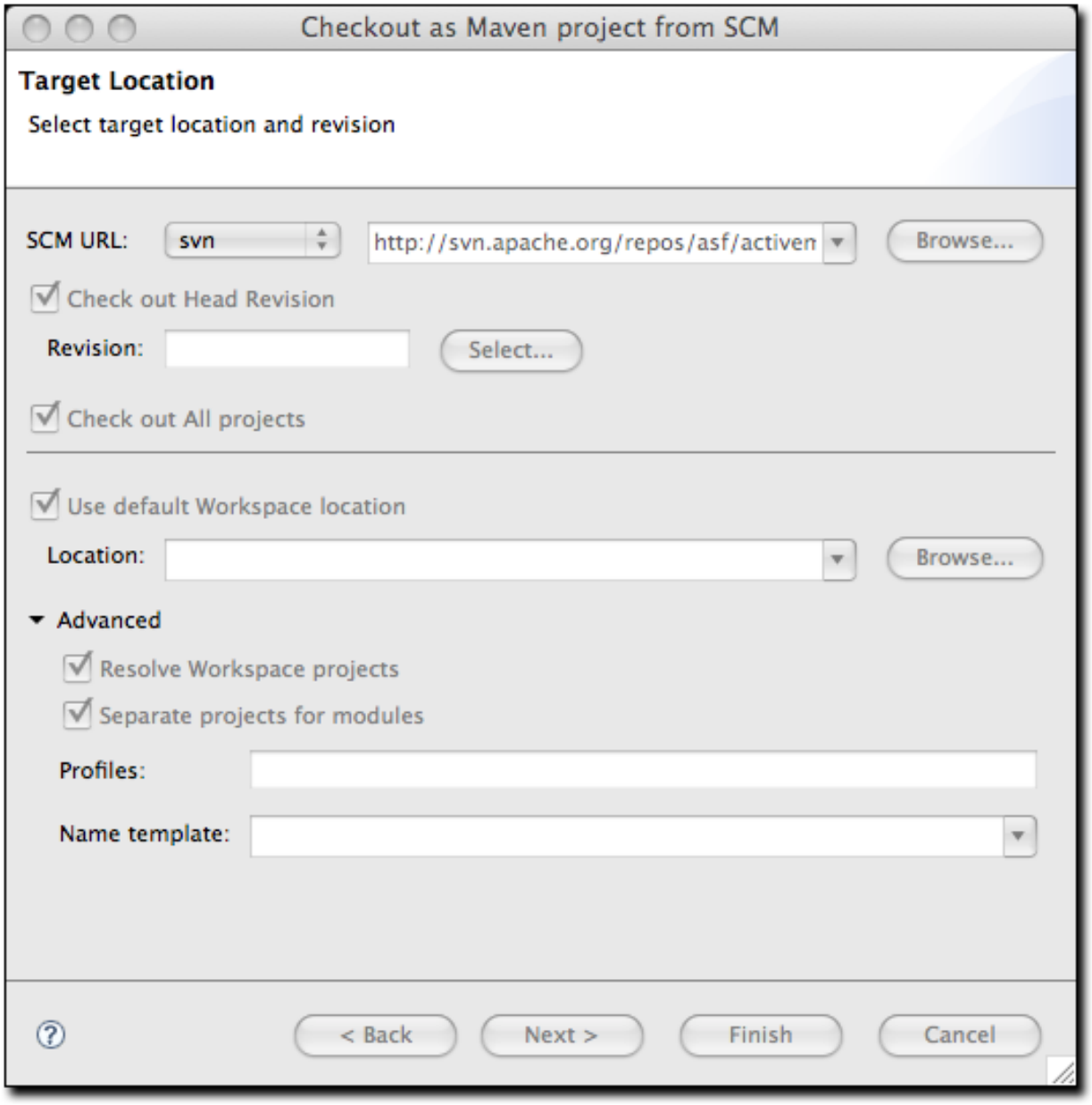


Figure 3.3. Checkout a New Project from Subversion

There are additional options in this dialog for specifying a particular revision by browsing the revisions in a Subversion repository or by simply entering the revision number manually. These features reuse of some of the features in the Subclipse plugin to interact with the Subversion repository. In addition to Subversion, the m2eclipse plugin also supports the following SCM providers:

- Bazaar
- Clearcase

- CVS
- git
- hg
- Perforce
- Starteam
- Subversion
- Synergy
- Visual SourceSafe

3.2.2. Creating a Maven Project from a Maven Archetype

m2eclipse offers the ability to create a Maven project using a Maven Archetype. There are many Maven Archetypes provided in the list that comes with m2eclipse as shown in Figure 3.4, “Creating a New Project with a Maven Archetype”.

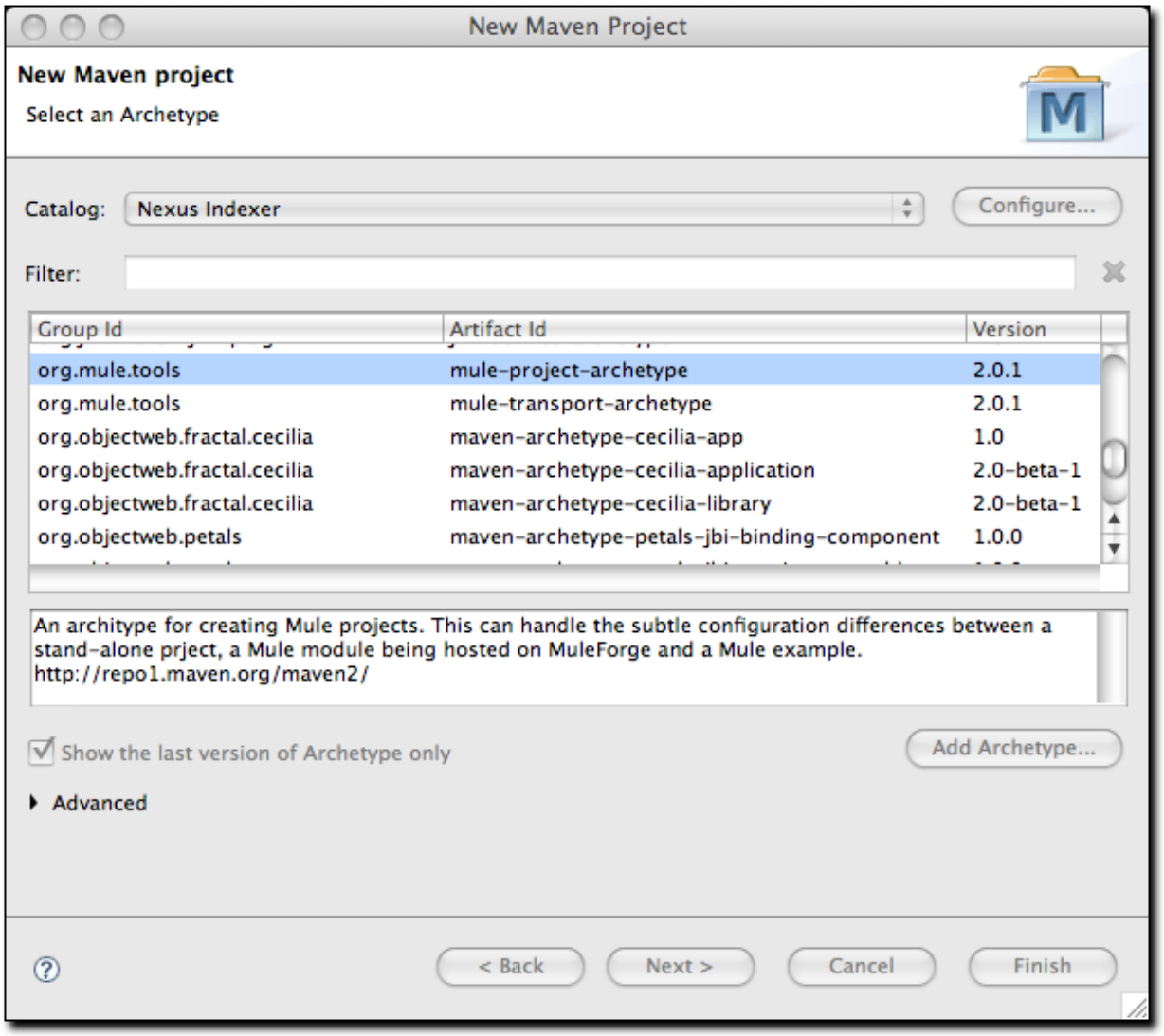


Figure 3.4. Creating a New Project with a Maven Archetype

The list of archetypes in Figure 3.4, “Creating a New Project with a Maven Archetype” is a list generated by something called the Nexus Indexer. Nexus is a repository manager which is introduced in ???. The Nexus indexer is a file which contains an index of the entire Maven repository, and m2eclipse uses it to list all of the available archetypes in the entire Maven repository. When this chapter was last updated, m2eclipse had approximately ninety archetypes in this Archetype dialog. Highlights of this list include:

- Standard Maven Archetypes to create
  - Maven Plugins
  - Simple Web Applications

    - Simple Projects
    - New Maven Archetypes
- Databinder Archetypes (data-driven Wicket Applications) under `net.databinder`
- Apache Cocoon Archetypes under `org.apache.cocoon`
- Apache Directory Server Archetypes under `org.apache.directory.server`
- Apache Geronimo Archetypes under `org.apache.geronimo.buildsupport`
- Apache MyFaces Archetypes under `org.apache.myfaces.buildtools`
- Apache Tapestry Archetypes under `org.apache.tapestry`
- Apache Wicket Archetypes under `org.apache.wicket`
- AppFuse Archetypes under `org.appfuse.archetypes`
- Codehaus Cargo Archetypes under `org.codehaus.cargo`
- Codehaus Castor Archetypes under `org.codehaus.castor`
- Groovy-based Maven Plugin Archetypes (deprecated)[18] under `org.codehaus.mojo.groovy`
- Jini Archetypes
- Mule Archetypes under `org.mule.tools`
- Objectweb Fractal Archetypes under `org.objectweb.fractal`
- Objectweb Petals Archetypes under `org.objectweb.petals`
- ops4j Archetypes under `org.ops4j`
- Parancoe under `org.parancoe`
- slf4j Archetypes under `org.slf4j`
- Springframework OSGI and Web Services Archetypes under `org.springframework`
- Trails Framework Archetypes under `org.trailsframework`

[18]And these were just the archetypes that were listed under the Nexus Indexer Catalog, if you

[18]Don't use the Groovy Maven Plugin in Codehaus' Mojo project. Jason Dillon has moved the Groovy Maven integration to the Groovy project in codehaus. For more information see http://groovy.codehaus.org/GMaven.

switch Catalogs you'll see other archetypes. While your results may vary, the following additional archetypes were available in the Internal Catalog:

- Atlassian Confluence Plugin Archetype under `com.atlassian.maven.archetypes`
- Apache Struts Archetypes under `org.apache.struts`
- Apache Shale Archetypes under `org.apache.shale`

A catalog is simply a reference to a repository index. You can manage the set of catalogs that the m2eclipse plugin knows about by clicking on the Configure... button next to the catalog drop down. If you have your own archetypes to add to this list, you can click on Add Archetype....

Once you choose an archetype, Maven will retrieve the appropriate artifact from the Maven repository and create a new Eclipse project with the selected archetype.

3.2.3. Creating a Maven Module

m2eclipse provides the ability to create a Maven module. Creating a Maven module is almost identical to creating a Maven project as it also creates a new Maven project using a Maven archetype. However, a Maven module is a subproject of another Maven project typically known as a parent project.

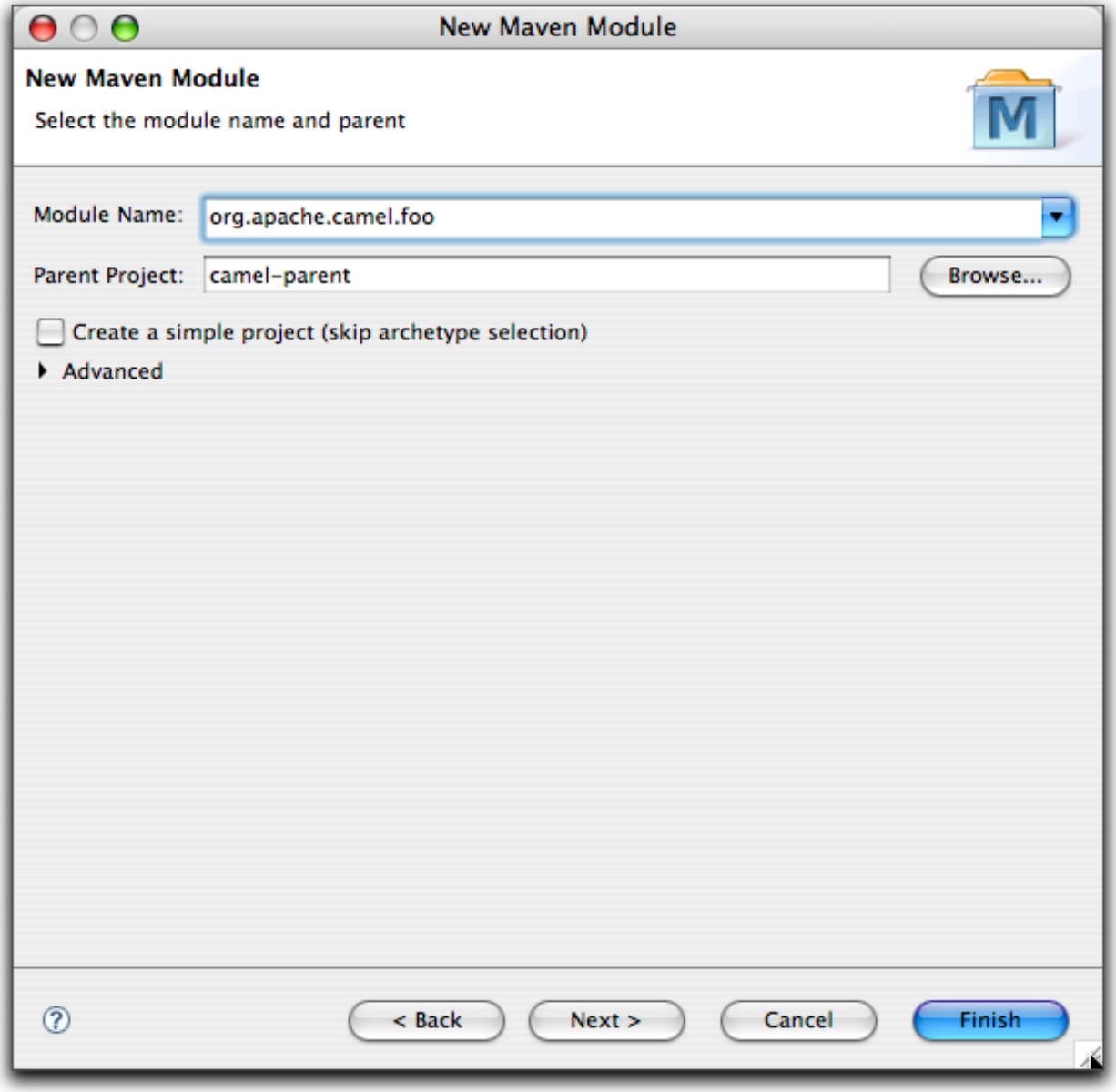


Figure 3.5. Creating a New Maven Module

When creating a new Maven module you must select a parent project that already exists inside of Eclipse. Clicking the browse button displays a list of projects that already exist as shown in Figure 3.6, “Selecting a Parent Project for a New Maven Module”:

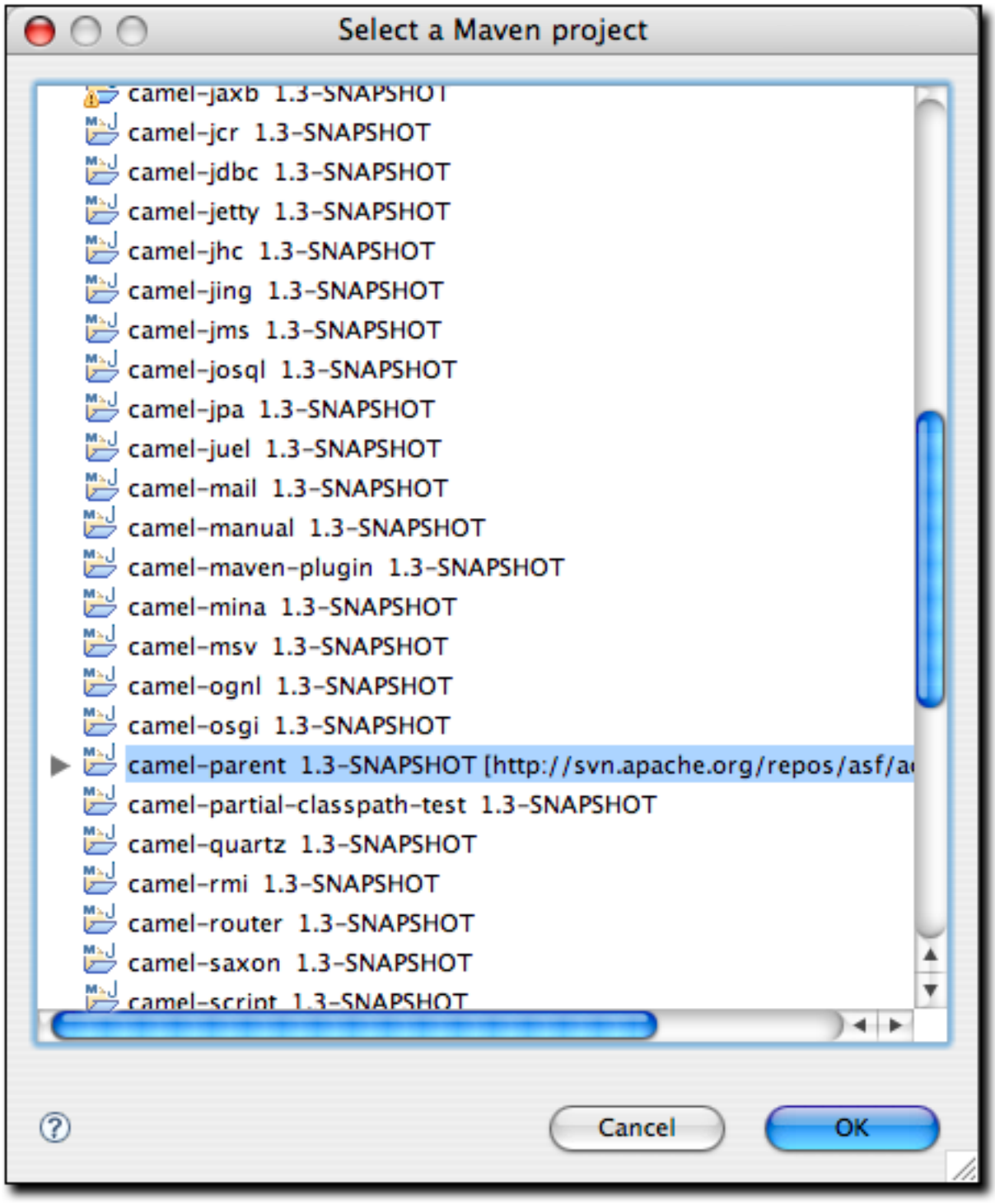


Figure 3.6. Selecting a Parent Project for a New Maven Module

After selecting a parent project from the list, you are returned to the New Maven Module window and the Parent Project field is populated as shown in Figure 3.5, “Creating a New Maven Module”. Clicking Next will then display the standard list of archetypes from

Section 3.2.2, “Creating a Maven Project from a Maven Archetype” so you can choose which one should be used to create the Maven module.

3.3. Create a Maven POM File

Another important feature m2eclipse offers is the ability to create a new Maven POM file. m2eclipse provides a wizard to easily create a new POM file inside of a project that is already in Eclipse. This POM creation wizard is shown in Figure 3.7, “Creating a New POM”:

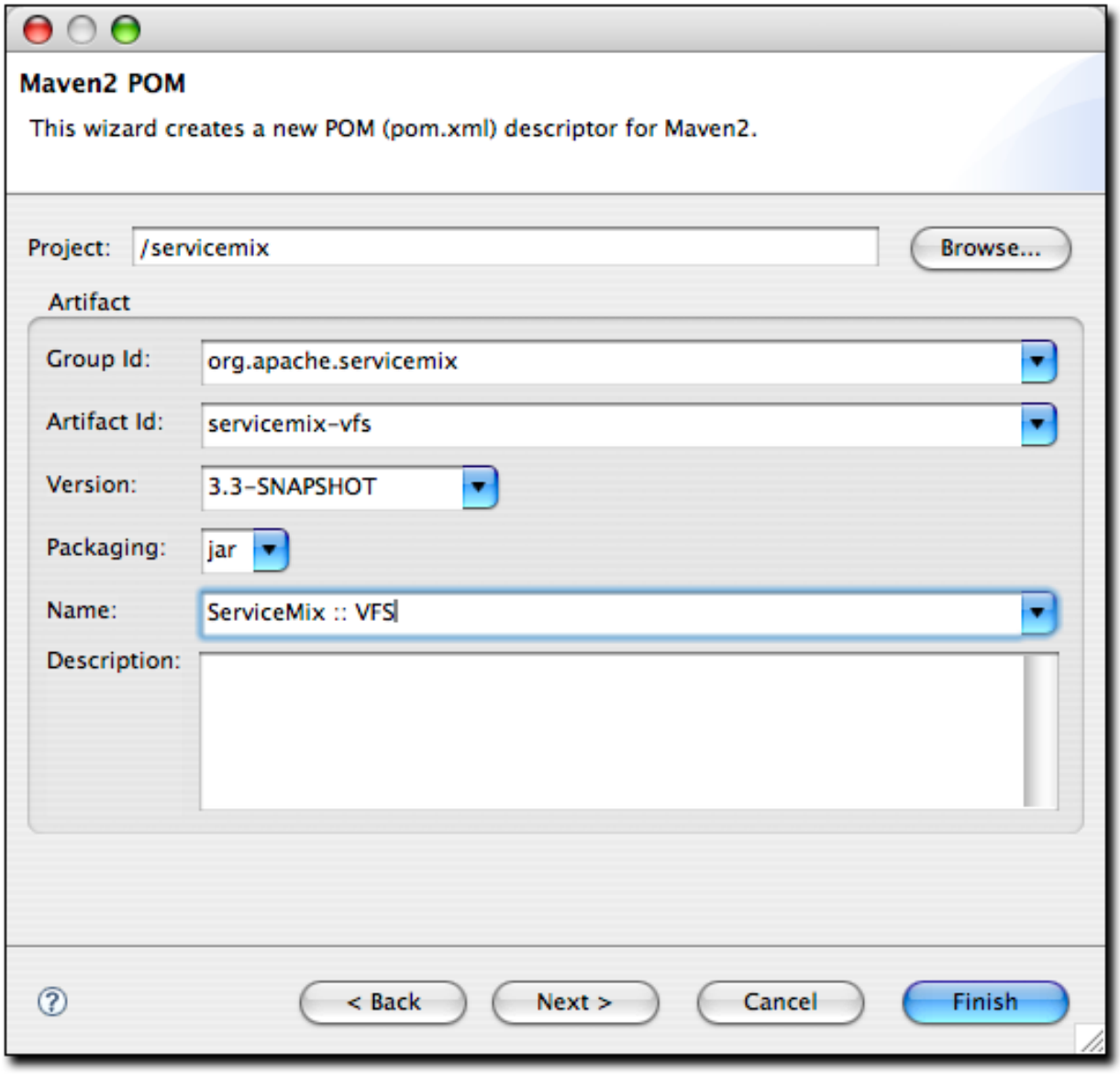


Figure 3.7. Creating a New POM

Creating a new Maven POM is just a matter of selecting a project, entering the Group Id, Artifact

Id, Version, choosing the Packaging type, and providing a Name into the fields provided and m2eclipse. Click the Next button to start adding dependencies.

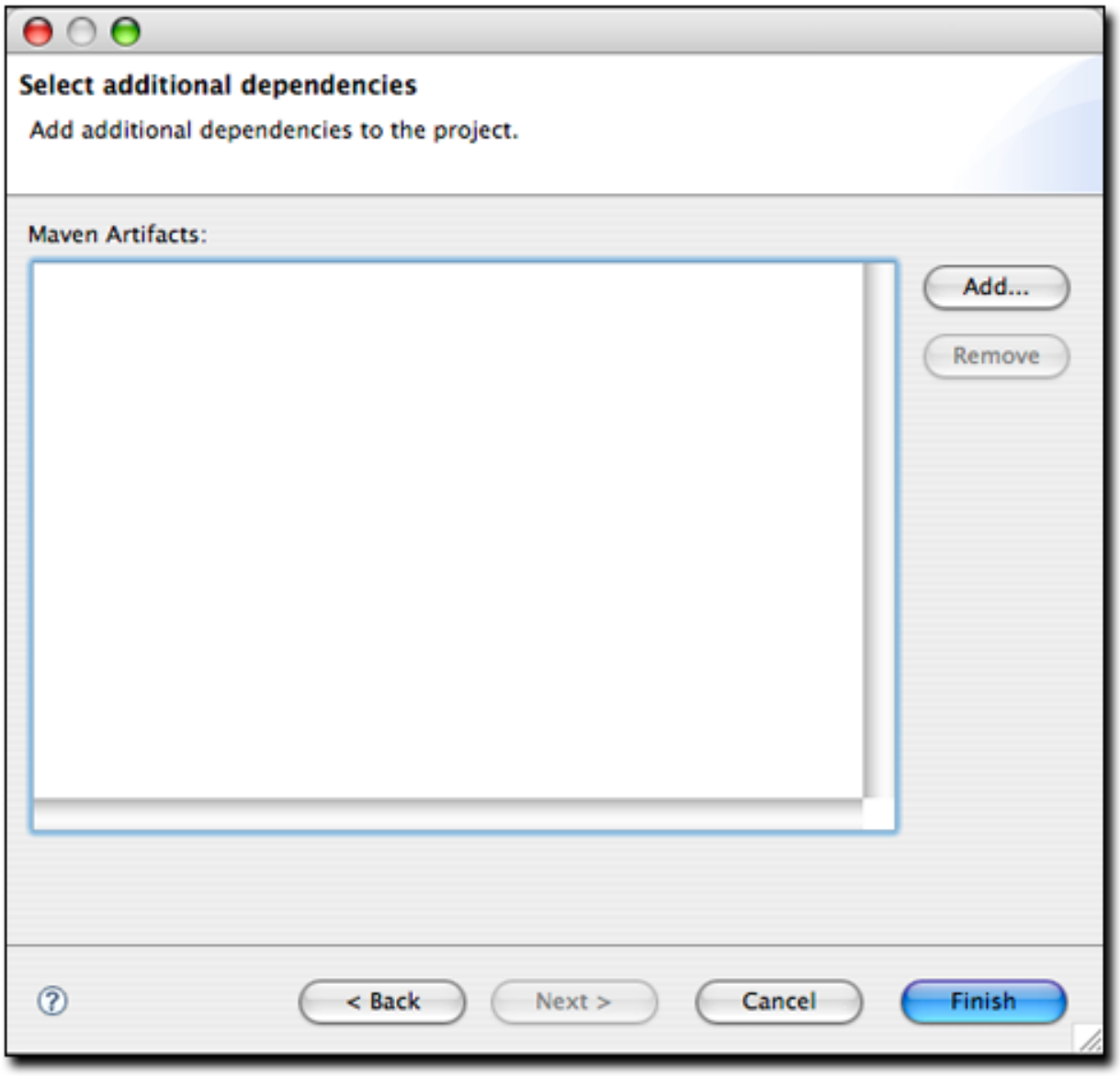


Figure 3.8. Adding Dependencies to a New POM

As you can see in Figure 3.8, “Adding Dependencies to a New POM” here are no dependencies in the POM yet. Just click the Add button to query the central Maven repository for dependencies as shown next in Figure 3.9, “Querying the Central Repository for Dependencies”:

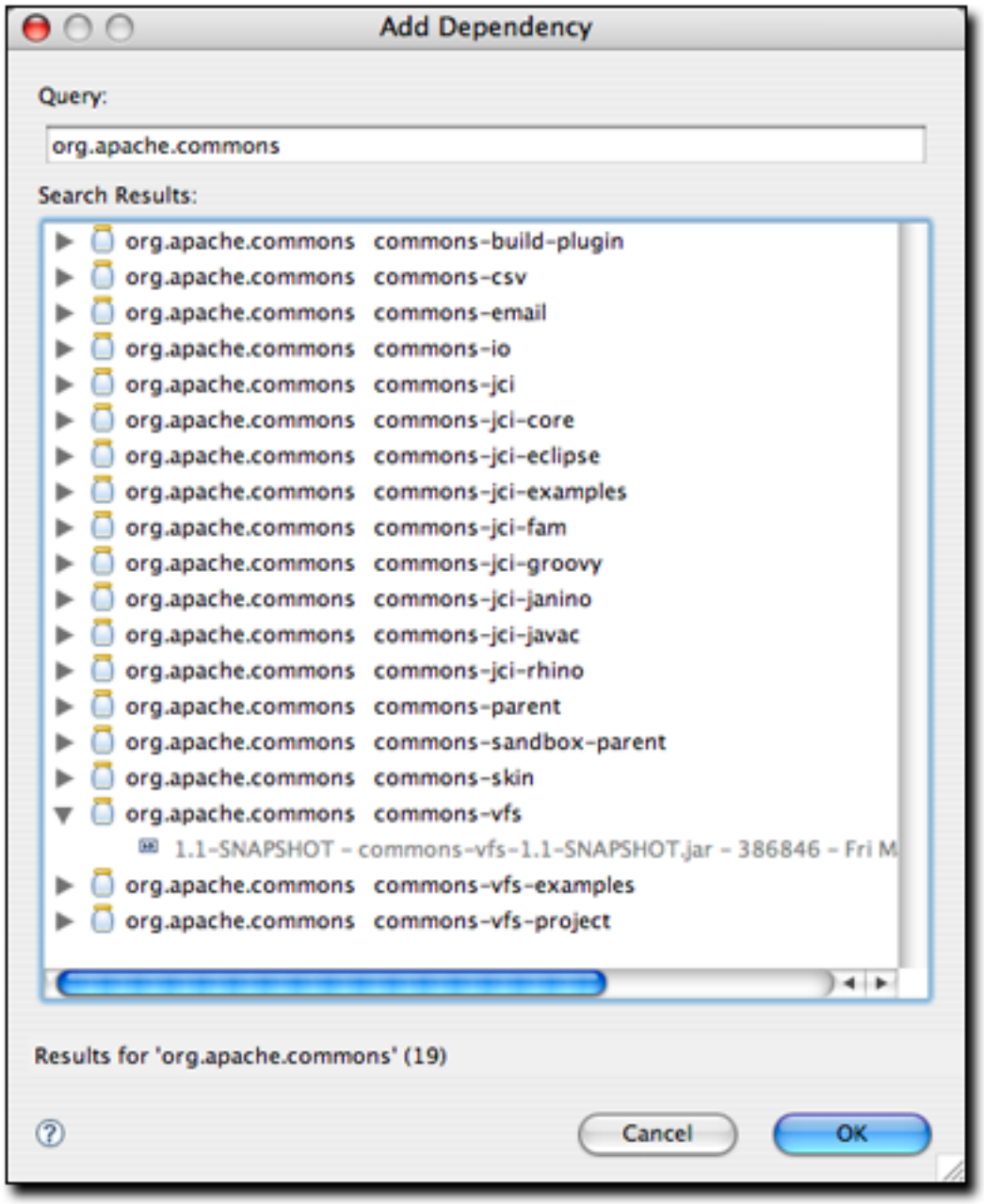


Figure 3.9. Querying the Central Repository for Dependencies

Querying for dependencies is as easy as entering the `groupId` for the artifact you need. Figure 3.9, “Querying the Central Repository for Dependencies” shows a query for `org.apache.commons` with `commons-vfs` expanded to see which versions are available. Highlighting the `1.1-SNAPSHOT` version of `commons-vfs` and clicking OK takes you back to the dependency selection where you can either query for more artifacts or just click finish to create the POM. When you search for dependencies, m2eclipse is making use of the same Nexus repository index that is used in the Nexus Repository Manager from ???.

Now that the you've seen the m2eclipse features for creating a new project, let's look at a similar set of features for importing projects into Eclipse.

3.4. Importing Maven Projects

m2eclipse provides three options for importing a Maven project into Eclipse including:

- Import an existing Maven project
- Check out a Maven project from SCM
- Materialize a Maven project

Figure 3.10, "Importing a Maven Project" shows the wizard for importing projects with the options for Maven provided by m2eclipse:

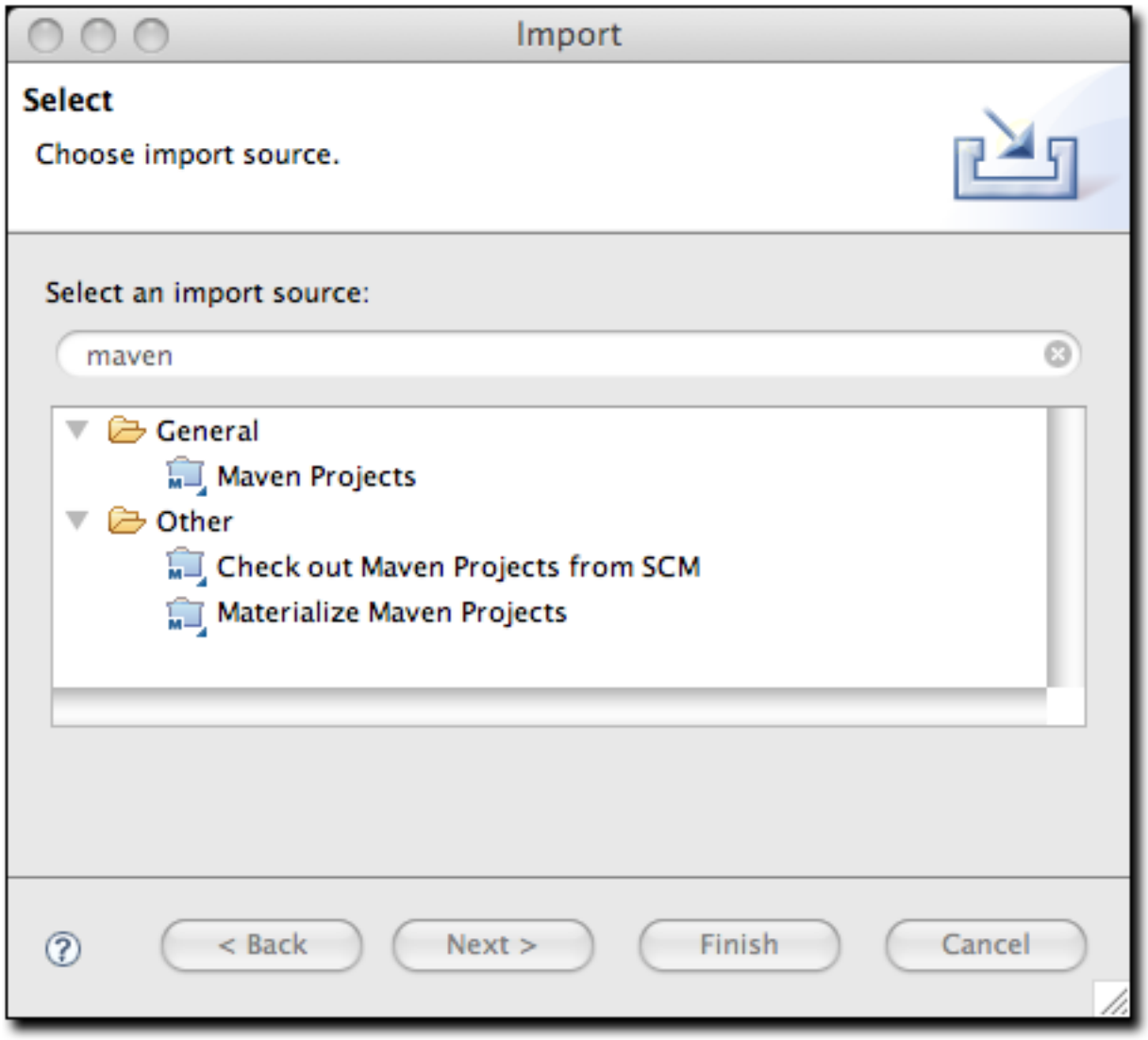


Figure 3.10. Importing a Maven Project

The dialog in Figure 3.10, “Importing a Maven Project” is displayed by using the File → Import command in Eclipse and then filtering the options by entering the word maven in the filter field. As noted above, there are three options available for importing a Maven project into Eclipse including: Maven Projects, Check out Maven Project from Subversion, and Materialize Maven Projects.

Importing a Maven project from Subversion is identical to the creation of a Maven project from Subversion as discussed in the previous section so discussion of it would be redundant. Let’s move on now to review the other two options for importing a Maven project into Eclipse.

3.4.1. Importing a Maven Project

m2eclipse can import a Maven project with an existing `pom.xml`. By pointing at the directory where a Maven project is located, m2eclipse detects all the Maven POMs in the project and provides a hierarchical list of them as shown in Figure 3.11, “Importing a Multi-module Maven Project”.

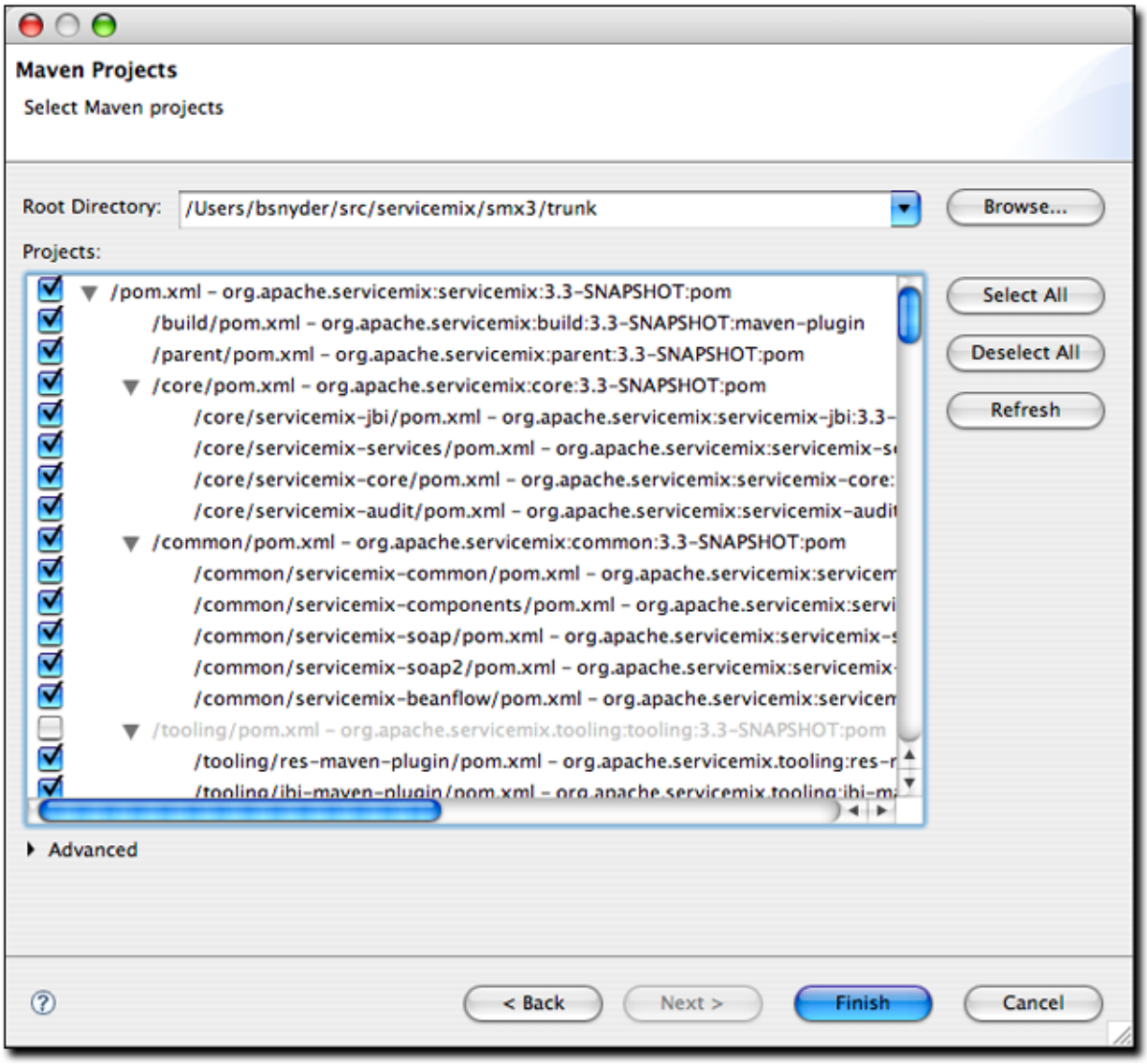


Figure 3.11. Importing a Multi-module Maven Project

Figure 3.11, “Importing a Multi-module Maven Project” displays the view of the project being imported. Notice that all the POMs from the project are listed in a hierarchy. This allows you to easily select which POMs (and therefore which projects) that you want to be imported into Eclipse. Once you select the project you would like to import, m2eclipse will import and build the project(s) using Maven.

3.4.2. Materializing a Maven Project

m2eclipse also offers the ability to "materialize" a Maven project. Materialization is similar to the process of checking out a Maven project from Subversion, but instead of manually entering the URL to the project's Subversion repository, the Subversion URL is discovered from the project's root POM file. You can use this feature to "materialize" projects from nothing more than a POM file if the POM file has the appropriate elements to specify the location of a source repository. Using this feature, you can browse the central Maven repository for projects, and materialize them into Eclipse projects. This comes in handy if your project depends on a third-party open source library, and you need to get your hands on the source code. Instead of tracking down the project web site and figuring out how to check it out of Subversion, just use the m2eclipse project to magically "materialize" the Eclipse project.

Figure 3.12, "Materializing a Maven Project" shows the wizard after choosing to materialize Maven projects:

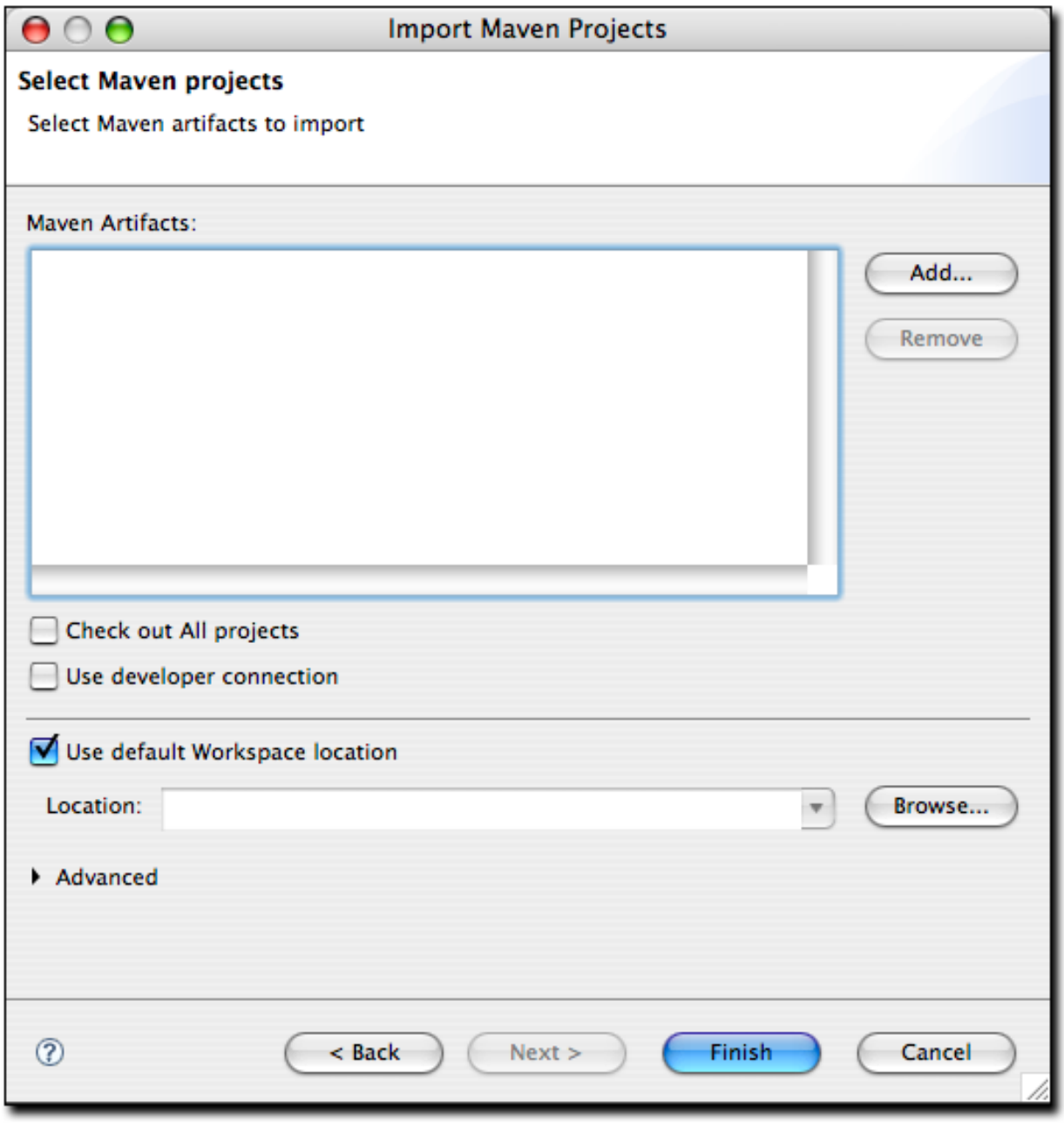


Figure 3.12. Materializing a Maven Project

Notice that the dialog box for Maven artifacts in Figure 3.12, “Materializing a Maven Project” is empty. This is because no projects have been added yet. In order to add a project, you must click the Add button on the right side and select a dependency to add from the central Maven repository. Figure 3.13, “Selecting Artifact to Materialize” shows how to add a project:

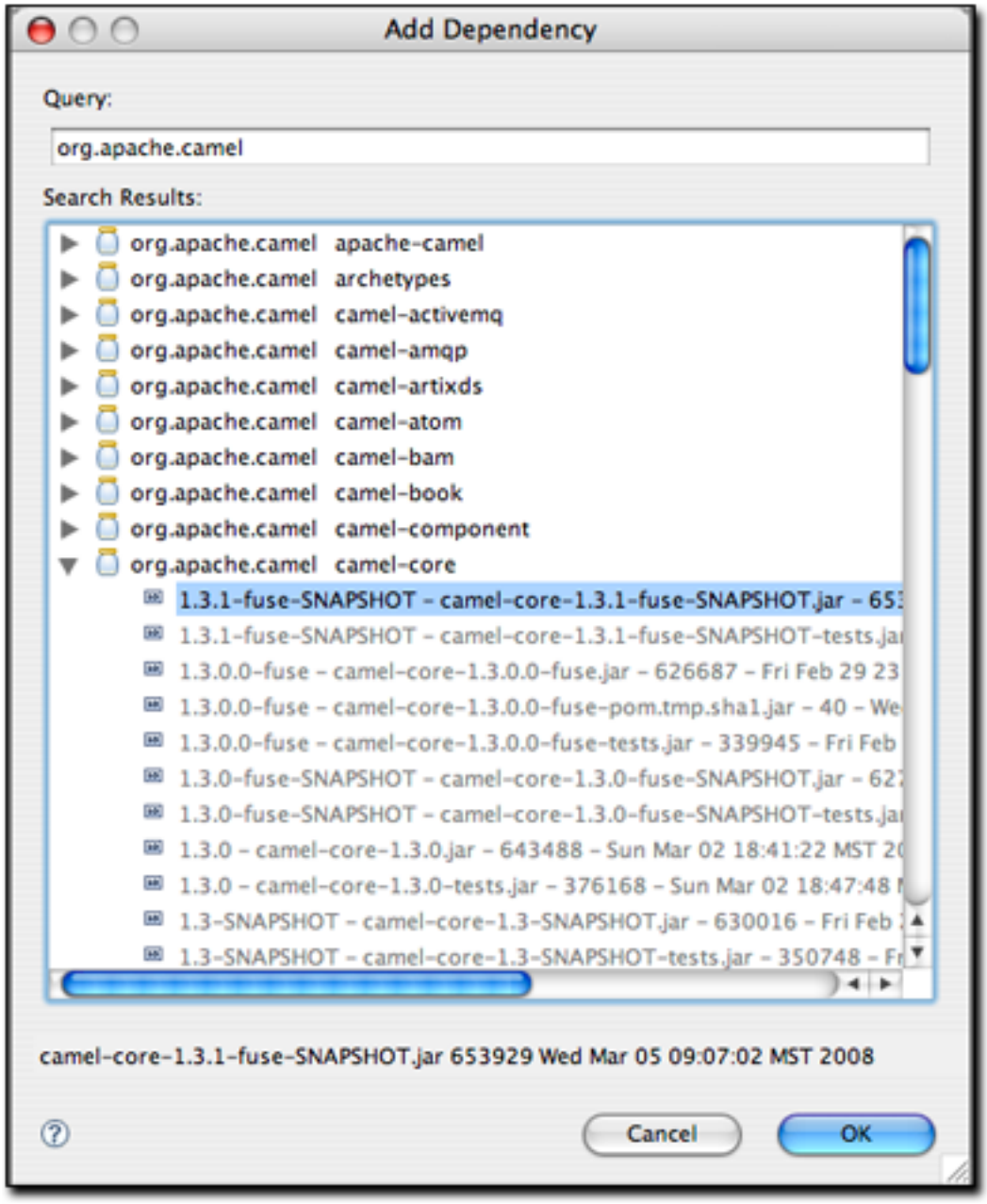


Figure 3.13. Selecting Artifact to Materialize

Upon entering a query, candidate dependencies will be located in the local Maven repository. After a few seconds of indexing the local Maven repository, the list of candidate dependencies appears. Select the dependency to add and click OK so that they are added to the list as shown in Figure 3.14, “Materializing Apache Camel”.

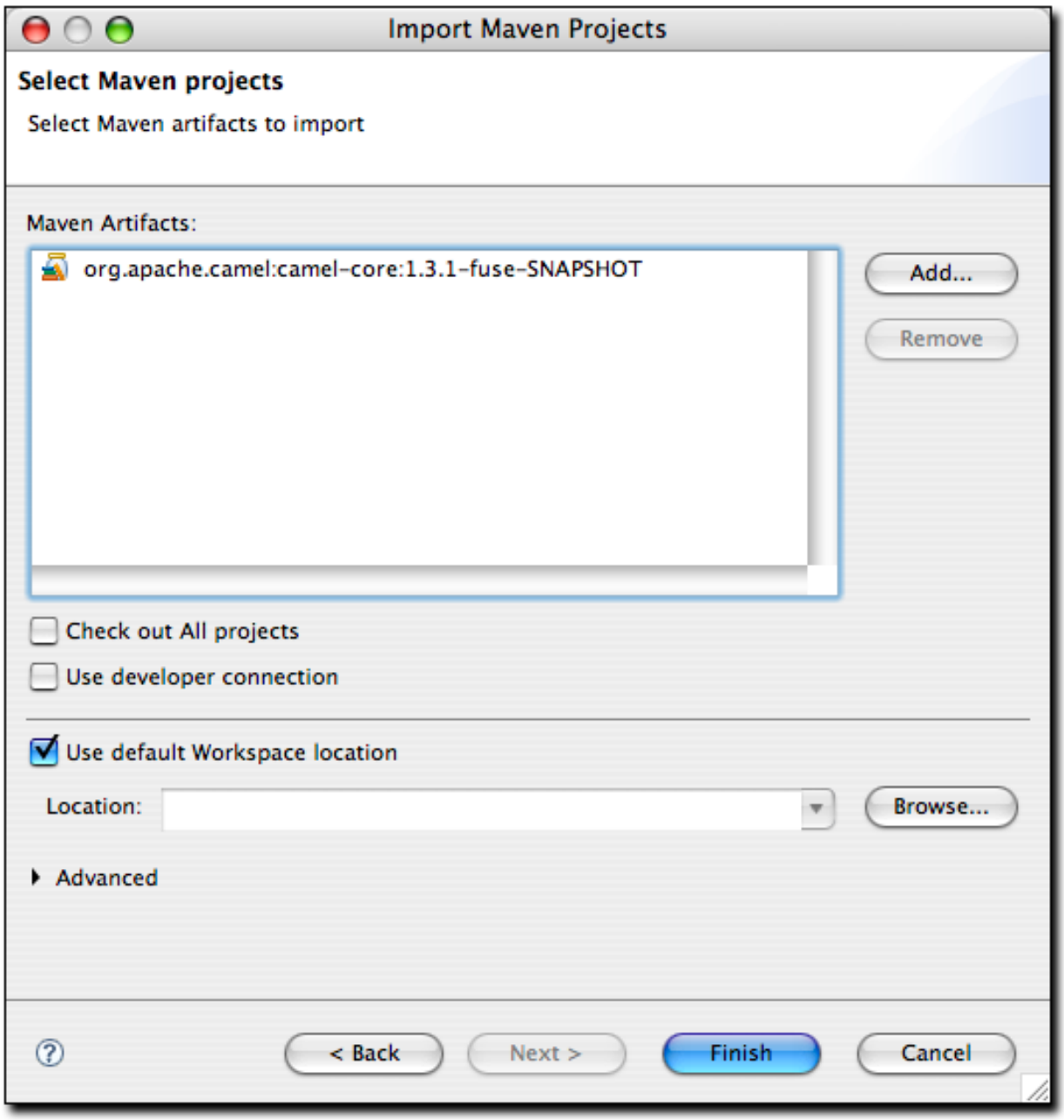


Figure 3.14. Materializing Apache Camel

Upon adding a dependency, you have the option of telling the m2eclipse plugin to check out all projects for the artifact.

3.5. Running Maven Builds

m2eclipse modified the Run As... and Debug As... menus to allow you to run a Maven build within Eclipse. Figure 3.15, “Running an Eclipse build with Run As..” shows the Run As... menu

for an m2eclipse project. From this menu you can run one of the more common lifecycle phases like clean, install, or package. You can also load up the Run configuration dialog window and configure a Maven build with parameters and more options.

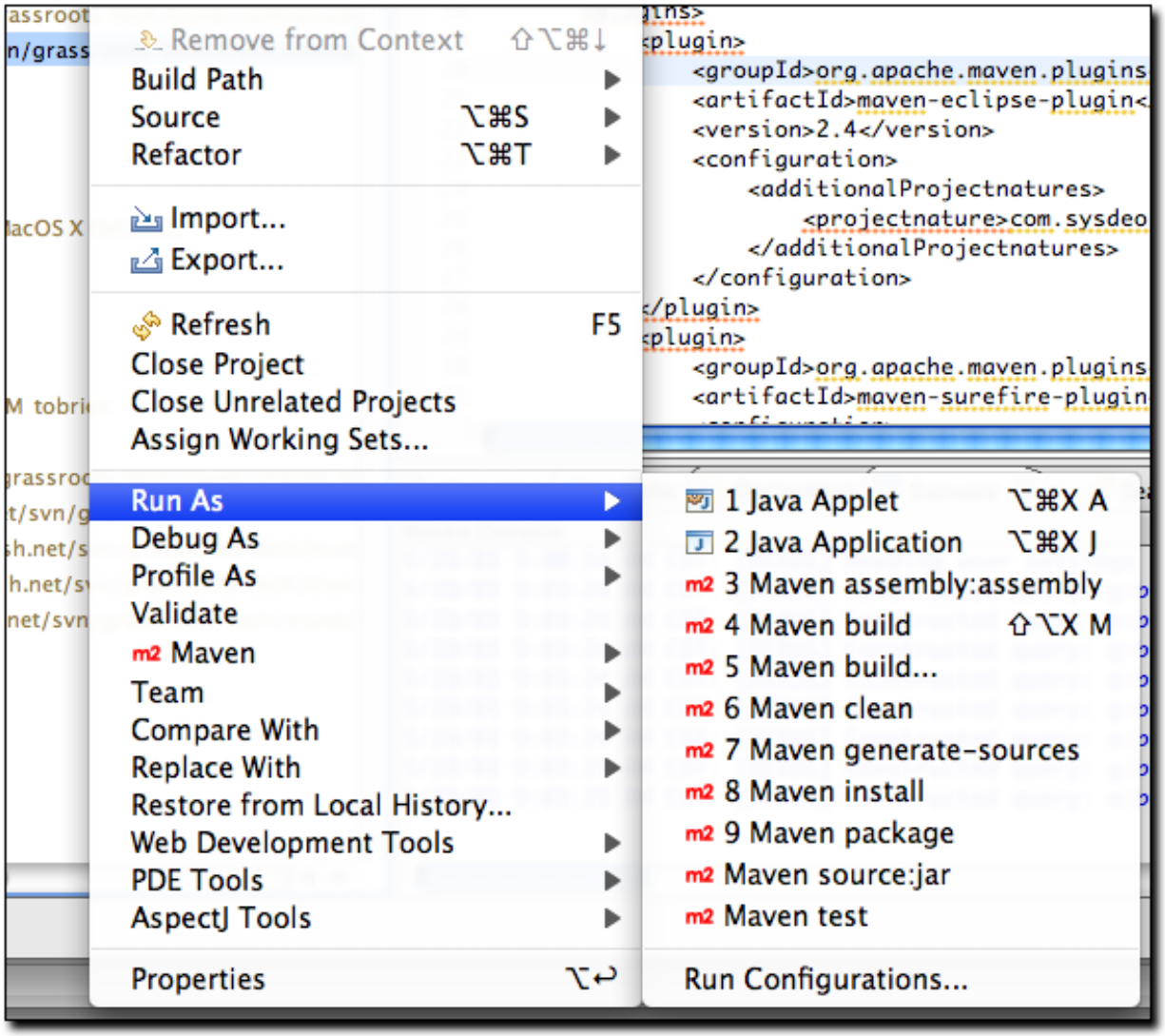


Figure 3.15. Running an Eclipse build with Run As..

If you need to configure a Maven build with more options, you can choose Run Configurations... and create a new Maven build. Figure 3.16, “Configuring a Maven Build as a Run Configuration” shows the Run dialog for configuring a Maven build.

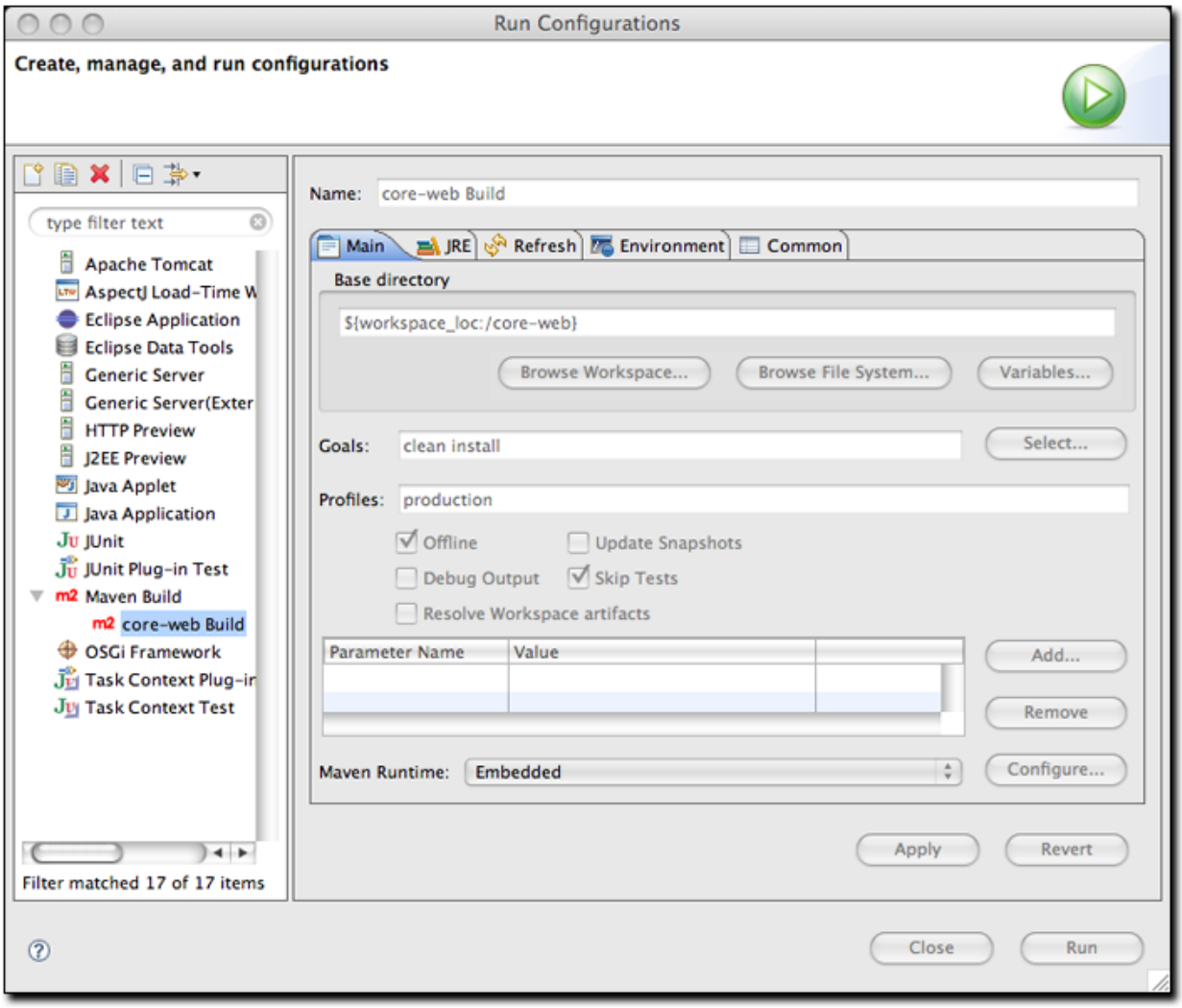


Figure 3.16. Configuring a Maven Build as a Run Configuration

The Run configuration dialog allows you to specify multiple goals and profiles, it exposes options like "skip tests" and "update snapshots", and allows you to customize everything from the project to the JRE to the environment variable. You can use this dialog to support any custom Maven build that you wish to launch with m2eclipse.

3.6. Working with Maven Projects

The m2eclipse plugin also provides a set of features for working with Maven projects once they are inside of Eclipse. There are many features that ease the ability to use Maven in Eclipse so let’s dive right into them. In the previous section, I materialized a Maven project and selected a subproject from the Apache Camel project named camel-core. We’ll use that project to demonstrate these features.

By right-clicking on the camel-core project, and selecting the Maven menu item, you can see the available Maven features. Figure 3.17, “Available Maven Features” shows a screenshot of this:

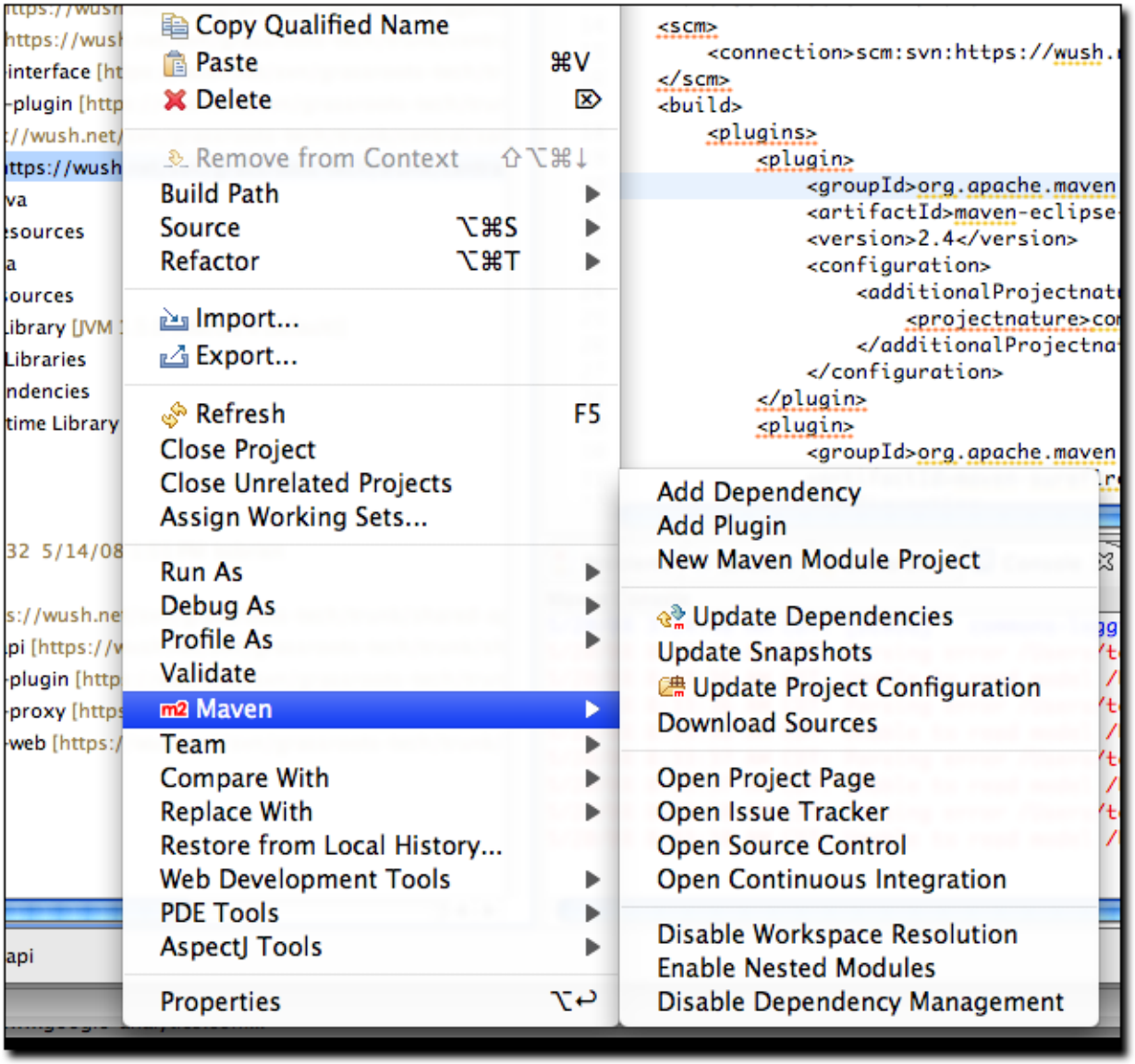


Figure 3.17. Available Maven Features

Notice in Figure 3.17, “Available Maven Features” the available Maven features for the camel-core project, including:

- Adding dependencies and plugins
- Updating dependencies, snapshots and source folders
- Creating a Maven module

- Downloading the source
- Opening Project URLs such as the Project Web Page, Issue Tracker, Source Control, and Continuous Integration tool.
- Enabling /Disabling workspace resolution, nested Maven modules and dependency management

These features are also big time savers so let's review them briefly.

3.6.1. Adding and Updating Dependencies and Plugins

Let's say we'd like to add a dependency or a plugin to the `camel-core` POM. For the sake of demonstration, we're going to add `commons-lang` as a dependency. (Please note that the functionality for adding a dependency or a plugin is exactly the same so we'll demonstrate it by adding a dependency.)

m2eclipse offers two options for adding dependencies to a project. The first option is by manually editing the POM file to type in the XML to add the dependency. The downside to manually editing the POM file to add a dependency is that you must already know the information about the artifact, or use the features discussed in the next section to manually locate the artifact information in the repository indexes. The upside is that after manually adding the dependency and saving the POM, the project's Maven Dependencies container will be automatically updated to include the new dependency. Figure 3.18, "Manually Adding a Dependency to the Project's POM" shows how I added a dependency for `commons-lang` to the `camel-console` POM and the Maven Dependencies container was automatically updated to included it:

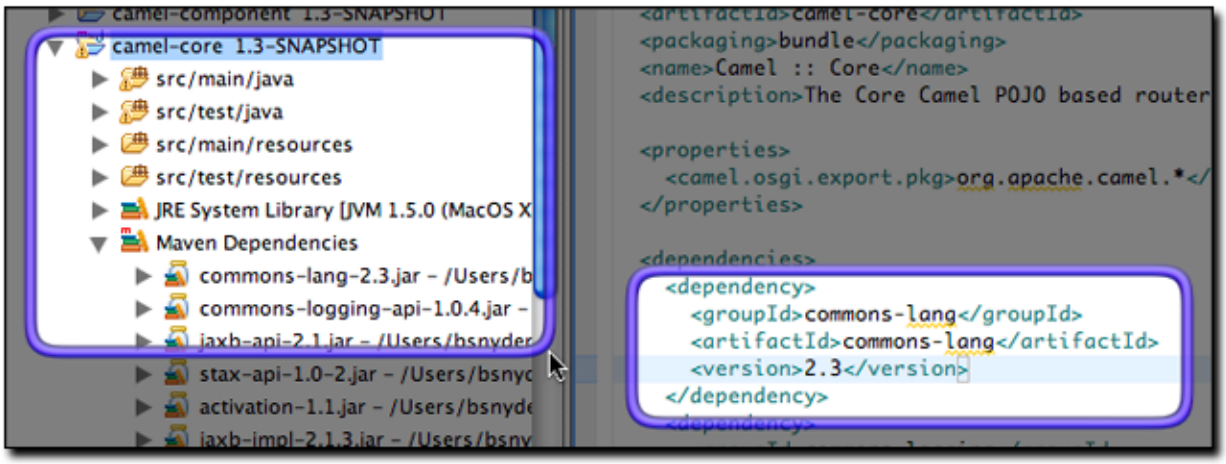


Figure 3.18. Manually Adding a Dependency to the Project's POM

Manually adding a dependency works well but requires more work than the second approach. Upon manually adding the dependency element to the POM, the Eclipse progress in the lower

right-hand corner of the Eclipse workbench reflects the action as shown in Figure 3.19, “Updating Maven Dependencies”:

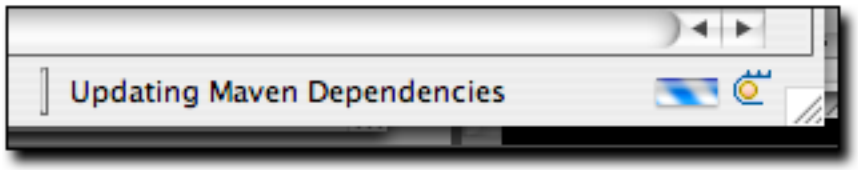


Figure 3.19. Updating Maven Dependencies

The second option for adding a dependency is much easier because you don’t have to know any information about about the artifact other than its `groupId`. Figure 3.20, “Searching for a Dependency” shows this functionality:

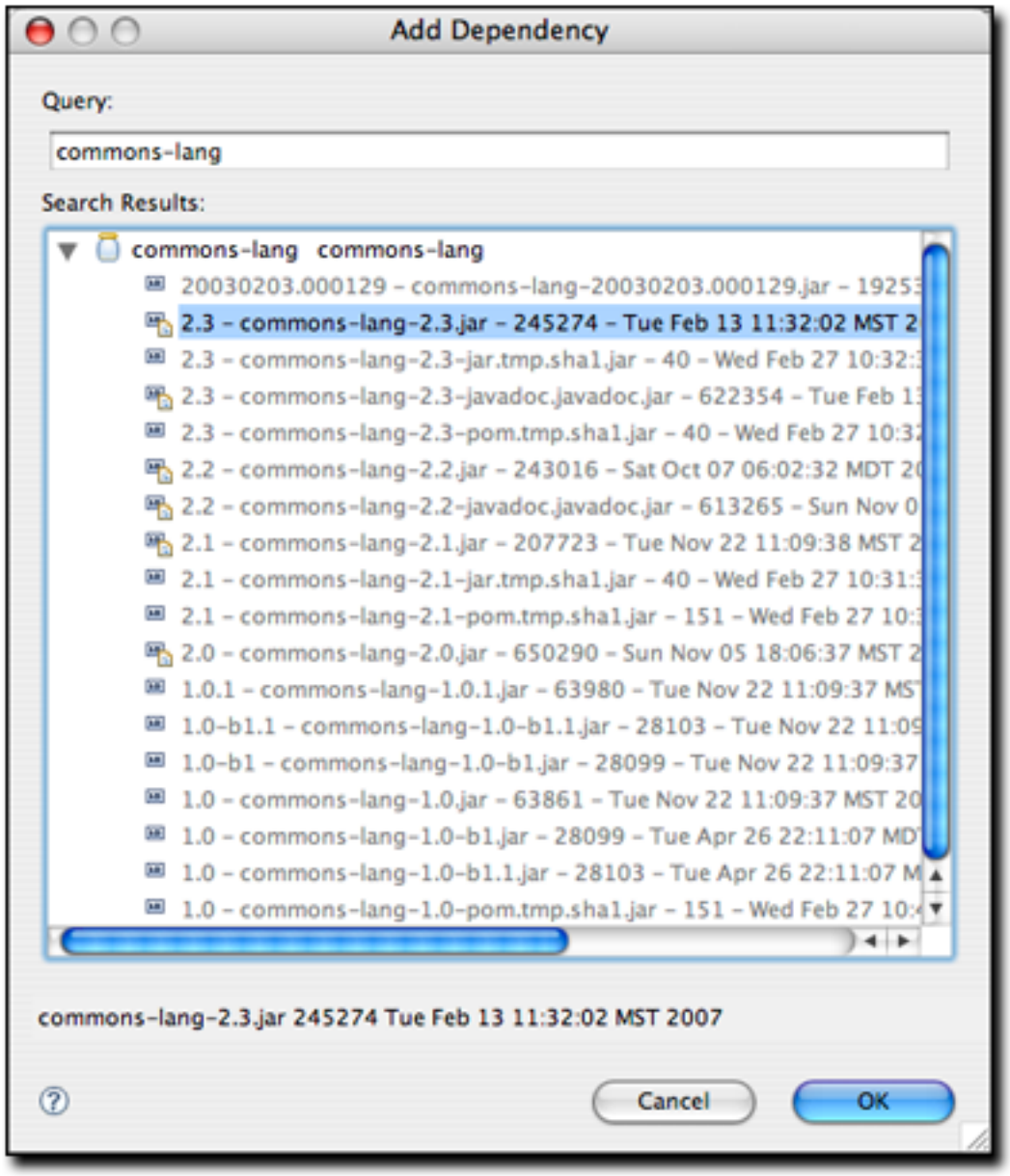


Figure 3.20. Searching for a Dependency

By simply entering a `groupId` into the query field, m2eclipse queries the repository indexes and even shows a version of the artifact that is currently in my local Maven repository. This option is preferred because it is such a tremendous time saver. With m2eclipse, you no longer need to hunt through the central Maven repository for an artifact version.

3.6.2. Creating a Maven Module

m2eclipse makes it very easy to create a series of nested projects in a multi-module Maven project. If you have a parent project, and you want to add a module to the project, just right click

on the project, go the Maven menu, and choose "New Maven Module Project". m2eclipse will walk you through the project creation process to create a new project, then it will update the parent project's POM to include the module reference. Before m2eclipse came along it was very difficult to use a hierarchy of Maven projects within Eclipse. With m2eclipse, the details of the underlying relationships between parent and child projects are integrated into the development environment.

3.6.3. Downloading Source

If the central Maven repository contains a source artifact for a particular project, you can download the source from the repository and expose it to the Eclipse environment. When you are trying to debug a complex issue in Eclipse, nothing can be easier than being able to right click on a third-party dependency and drill into the code in the Eclipse debugger. Select this option, and m2eclipse will attempt to download the source artifact from the Maven repository. If it is unable to retrieve this source artifact, you should ask the maintainers of the project in question to upload the appropriate Maven source bundle to the central Maven repository.

3.6.4. Opening Project Pages

A Maven POM contains some valuable URLs which a developer may need to consult. These are the project's web page, the URL for the source code repository, a URL for a continuous integration system like Hudson, and a URL for an issue tracker. If these URLs are present in a project's POM, m2eclipse will open these project pages in a browser.

3.6.5. Resolving Dependencies

You can configure a project to resolve dependencies from a workspace. This has the effect of altering the way that Maven locates dependency artifacts. If a project is configured to resolve dependencies from the workspace, these artifacts do not need to be present in your local repository. Assume that project-a and project-b are both in the same Eclipse workspace, and that project-a depends on project-b. If workspace resolution is disabled, the m2eclipse Maven build for `project-a` will only succeed if `project-b`'s artifact is present in the local repository. If workspace resolution is enabled, m2eclipse will resolve the dependency via the eclipse workspace. In other words, when workspace resolution is enabled, project's don't have to be installed in the local repository to relate to one another.

You can also disable dependency management. This has the effect of telling m2eclipse to stop trying to manage your project's classpath, and it will remove the Maven Dependencies classpath container from your project. If you do this, you are essentially on your own when it comes to managing your project's classpath.

3.7. Working with Maven Repositories

m2eclipse also provides some tools to make working with Maven repositories a bit easier. These tools provide functionality for:

- Searching for Artifacts
- Searching for Java classes
- Indexing Maven repositories

3.7.1. Searching For Maven Artifacts and Java classes

m2eclipse adds a couple of items to the Eclipse Navigation menu that make searching for Maven Artifacts and Java classes easy work. Each option is available by clicking on the Navigate menu as shown in Figure 3.21, “Searching for Artifacts and Classes”:

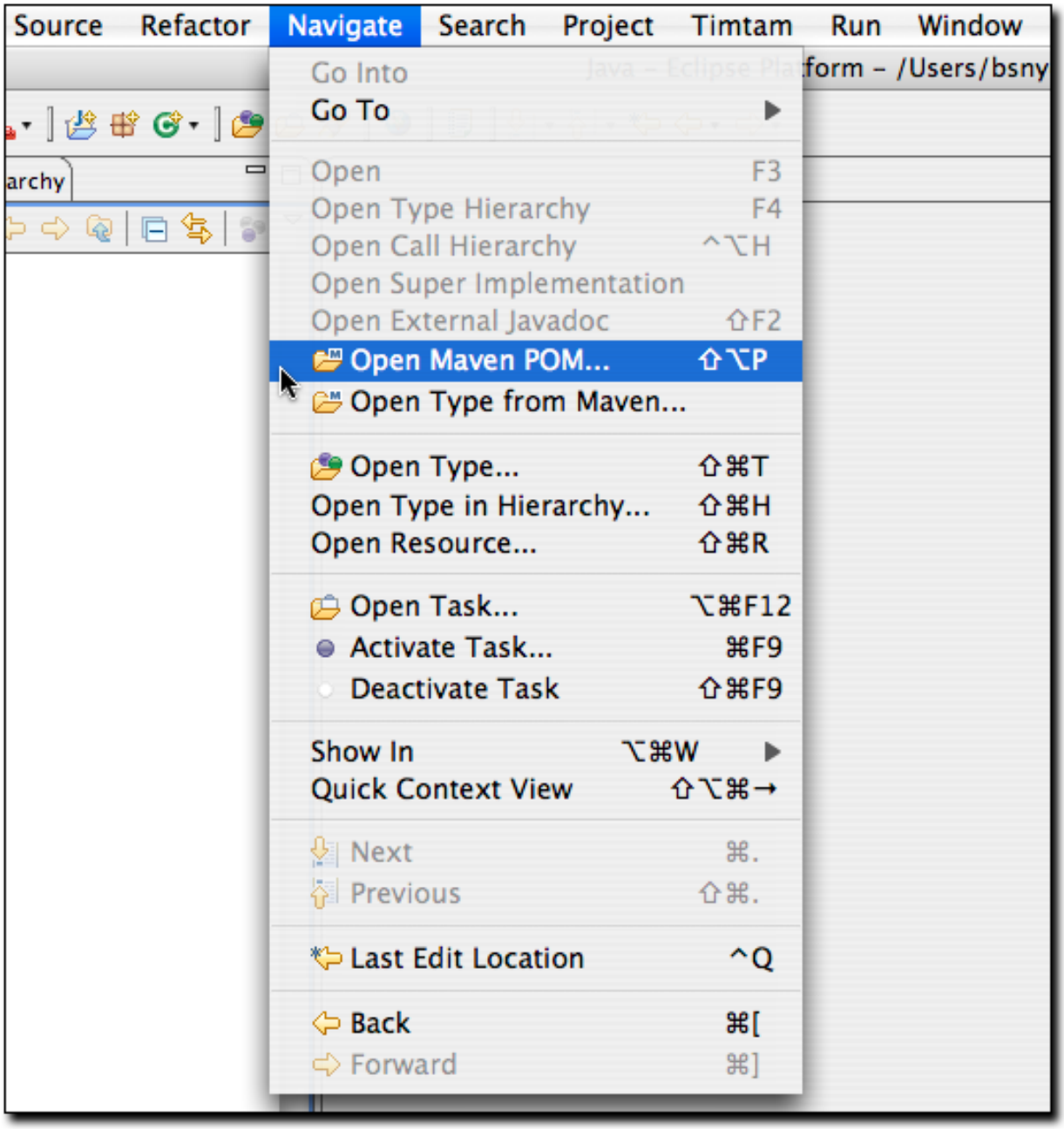


Figure 3.21. Searching for Artifacts and Classes

Notice the available options in Figure 3.21, “Searching for Artifacts and Classes” under the Eclipse Navigate menu named Open Maven POM and Open Type from Maven. The Open Maven POM option allows you to search the Maven repository for a given POM as shown in Figure 3.22, “Searching for a POM”:

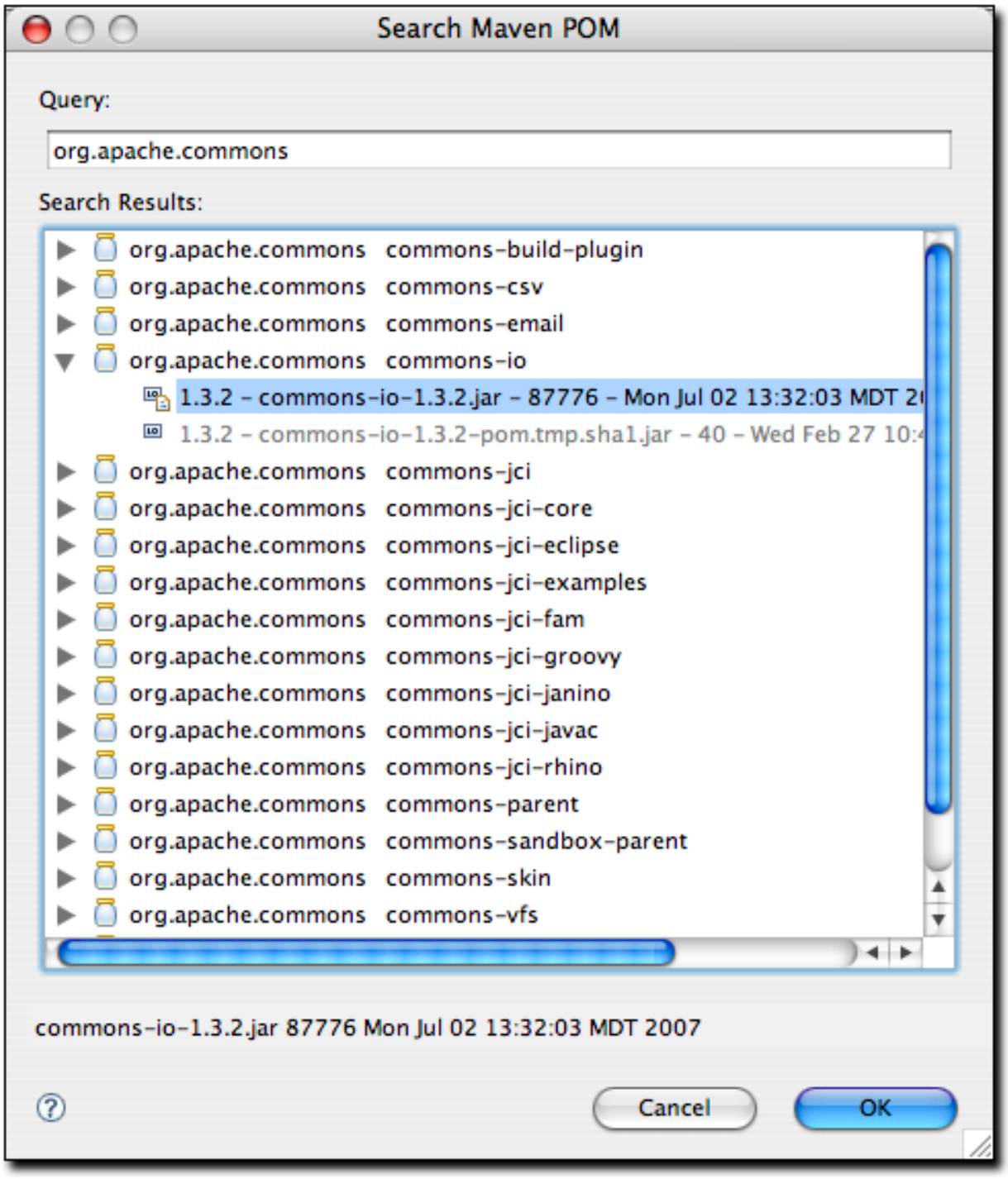


Figure 3.22. Searching for a POM

Upon selecting an artifact and clicking OK, the POM for that artifact is opened in Eclipse for browsing or editing. This is handy when you need to take a quick look at the POM for a given artifact.

The second m2eclipse option in the Navigate menu is named Open Type from Maven. This feature allows you to search for a Java class by name in a remote repository. Upon opening this dialog, simply type ‘factorybean’ and you’ll see many classes with the name `FactoryBean` in them as shown in Figure 3.23, “Searching the Repository for a Class”:

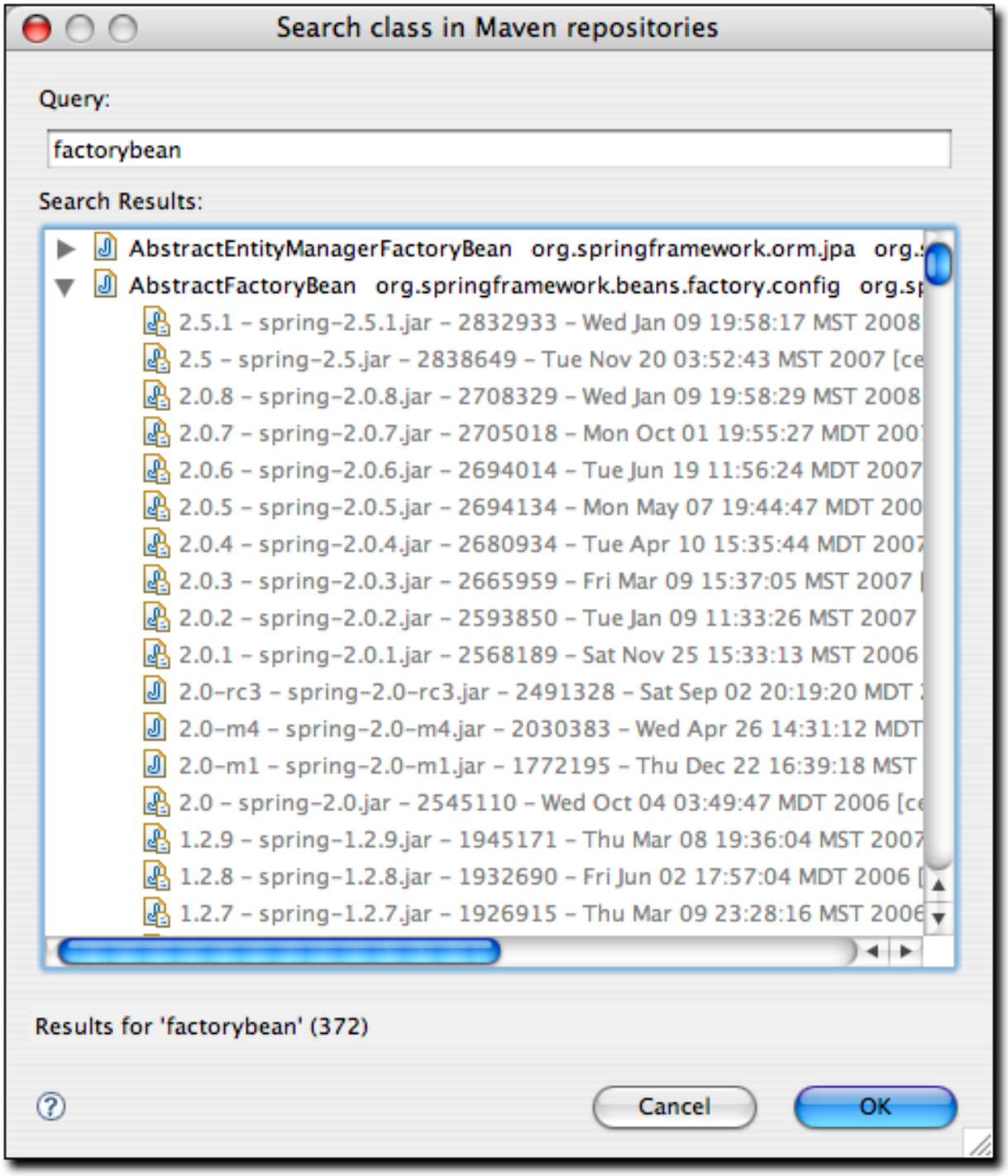


Figure 3.23. Searching the Repository for a Class

This is a big time saving feature because it means that manually searching through artifacts in a Maven repository for a particular class is a thing of the past. If you need to use a specific class, just fire up Eclipse, go to the Navigate menu and search for the class. m2eclipse will show you the list of artifacts in which it appears.

3.7.2. Indexing Maven Repositories

The Maven Indexes View allows you to manually navigate to POMs in a remote repository and open them in Eclipse. To see this View, go to View → Show View → Other, type the word "maven" into the search box and you should see a view named Maven Indexes as shown in Figure 3.24, “Show Maven Indexes View”:

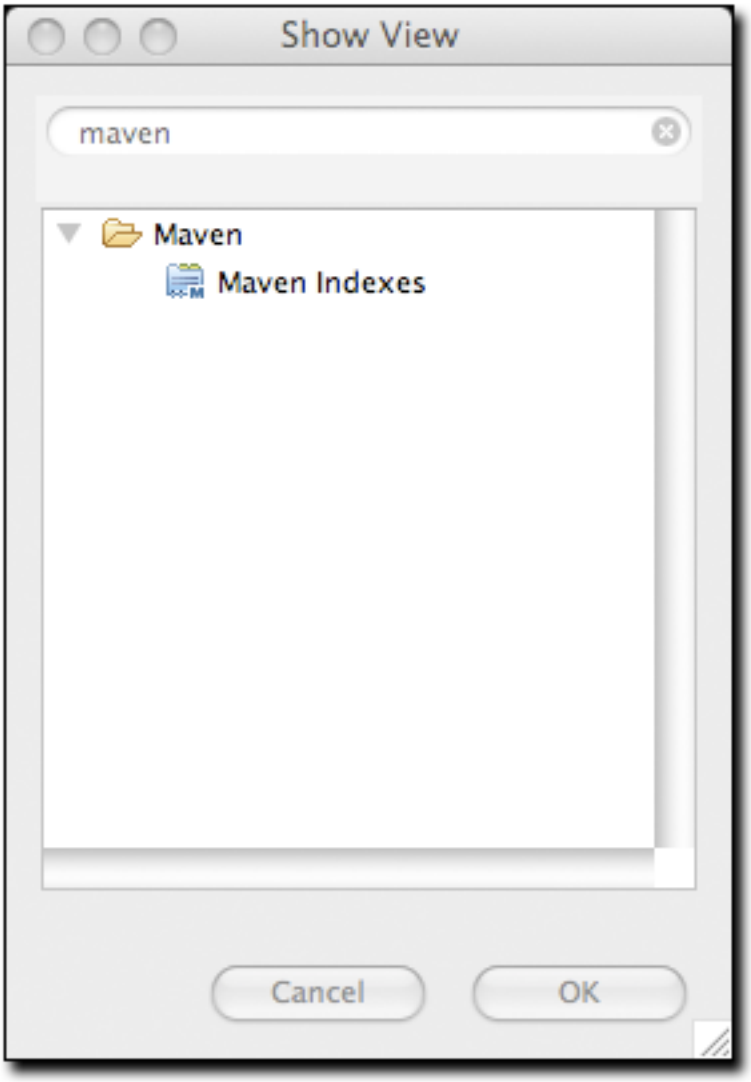


Figure 3.24. Show Maven Indexes View

Select this View and click OK. This will show the Maven Indexes View as shown in Figure 3.25,

“Maven Indexes View”:

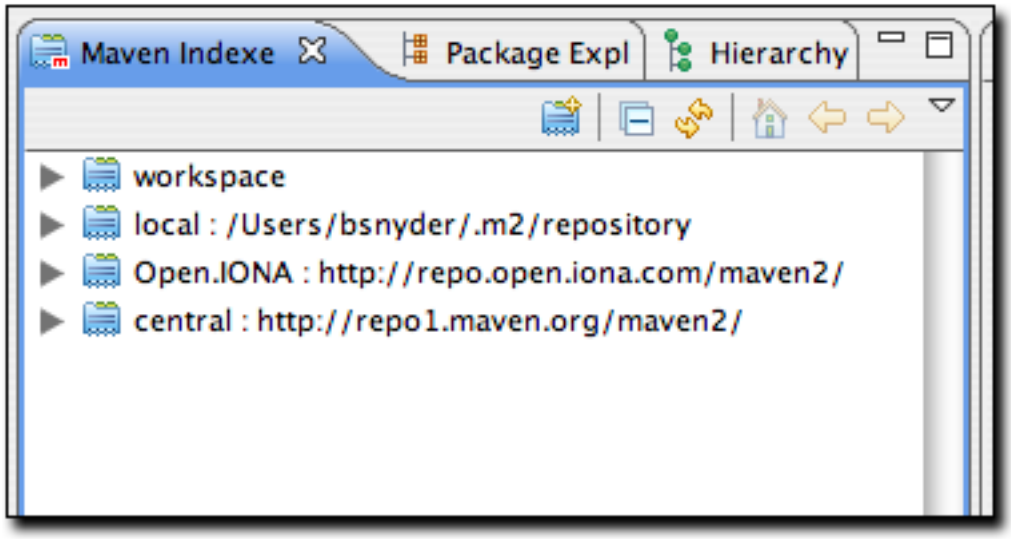


Figure 3.25. Maven Indexes View

Additionally, Figure 3.26, “Locating a POM from the Indexes View” shows the Maven Indexes View after manually navigating to locate a POM:

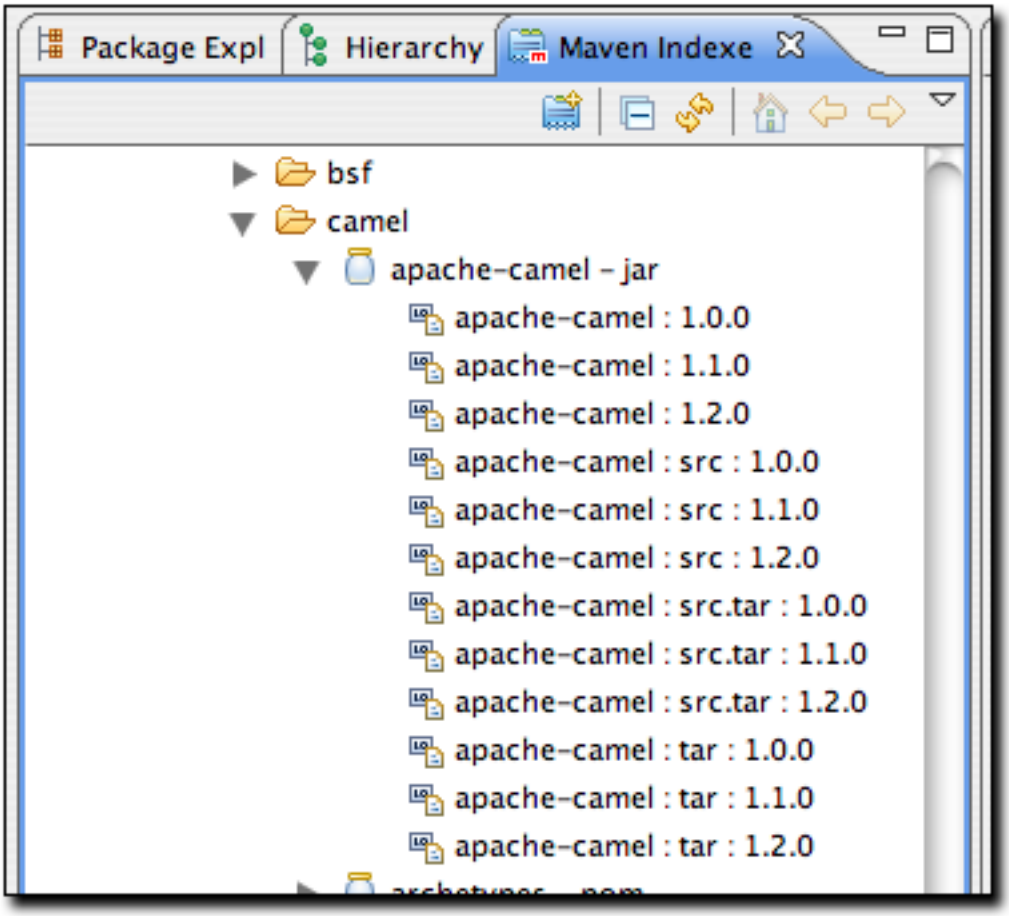


Figure 3.26. Locating a POM from the Indexes View

After finding the `apache-camel` artifact, double-clicking on it will open it up in Eclipse for browsing or editing.

These features make working with remote repositories from inside of Eclipse so much easier and faster. After all the hours you may have spent doing these types of tasks by manually over the last few years - visiting repositories through a web browser, downloading artifacts and grepping through them for classes and POMs - you'll find that m2eclipse is a welcome change for the better.

3.8. Using the Form-based POM Editor

The latest release of the m2eclipse plugin has a form-based POM editor which allows you to edit every part of a project's `pom.xml` with an easy-to-use GUI interface. To open the POM Editor, click on a project's `pom.xml` file. If you've customized the editors for a pom.xml file, and the POM Editor is not the default editor, you may need to right-click on the file and choose "Open With... / Maven POM Editor". The POM Editor will then display the Overview tab as shown in

Figure 3.27, “Overview Tab of POM Editor for idiom-core”.

One common complaint about Maven is that it forces a developer to confront large and often overwhelming XML documents in a highly complex multi-module project build. While the authors of this book believe this is a small price to pay for the flexibility of a tool like Maven, the graphical POM editor is a tool that makes it possible for people to use Maven without ever having to know about the XML structure behind a Maven POM.

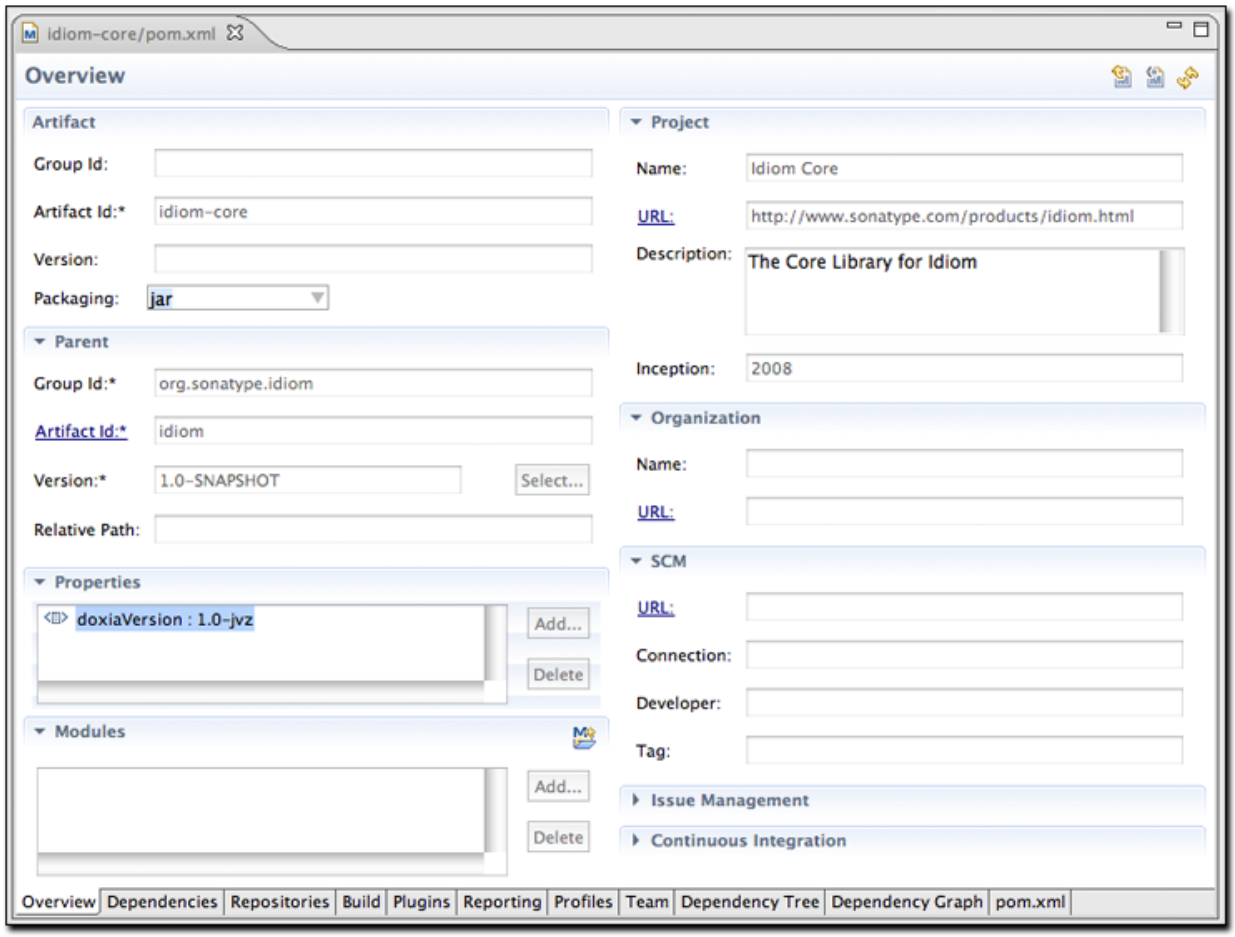


Figure 3.27. Overview Tab of POM Editor for idiom-core

The project shown in Figure 3.27, “Overview Tab of POM Editor for idiom-core” is a project with an `artifactId` of of `idiom-core`. You'll notice that most of the fields in this `idiom-core` project are blank. There is no `groupId` or `version` and there is no SCM information supplied in the POM editor. This is due to the fact that `idiom-core` inherits most of this information from a parent project named `idiom`. If we open the `pom.xml` for the parent project in the POM Editor we would see the Overview tab shown in Figure 3.28, “Overview Tab of POM Editor for idiom Parent Project”.

That “open folder” icon on the various list entries throughout the POM editor indicate that the corresponding entry is present in the Eclipse workspace and “jar” icon indicates artifacts which

are referenced from the Maven repository. You can double-click on those entries in order to open its POM in the POM editor. This works for modules, dependencies, plugins and other elements that have corresponding Maven artifacts. Underlined labels in several POM editor sections represent hyperlinks which can be used to open the POM editor for corresponding Maven artifact.

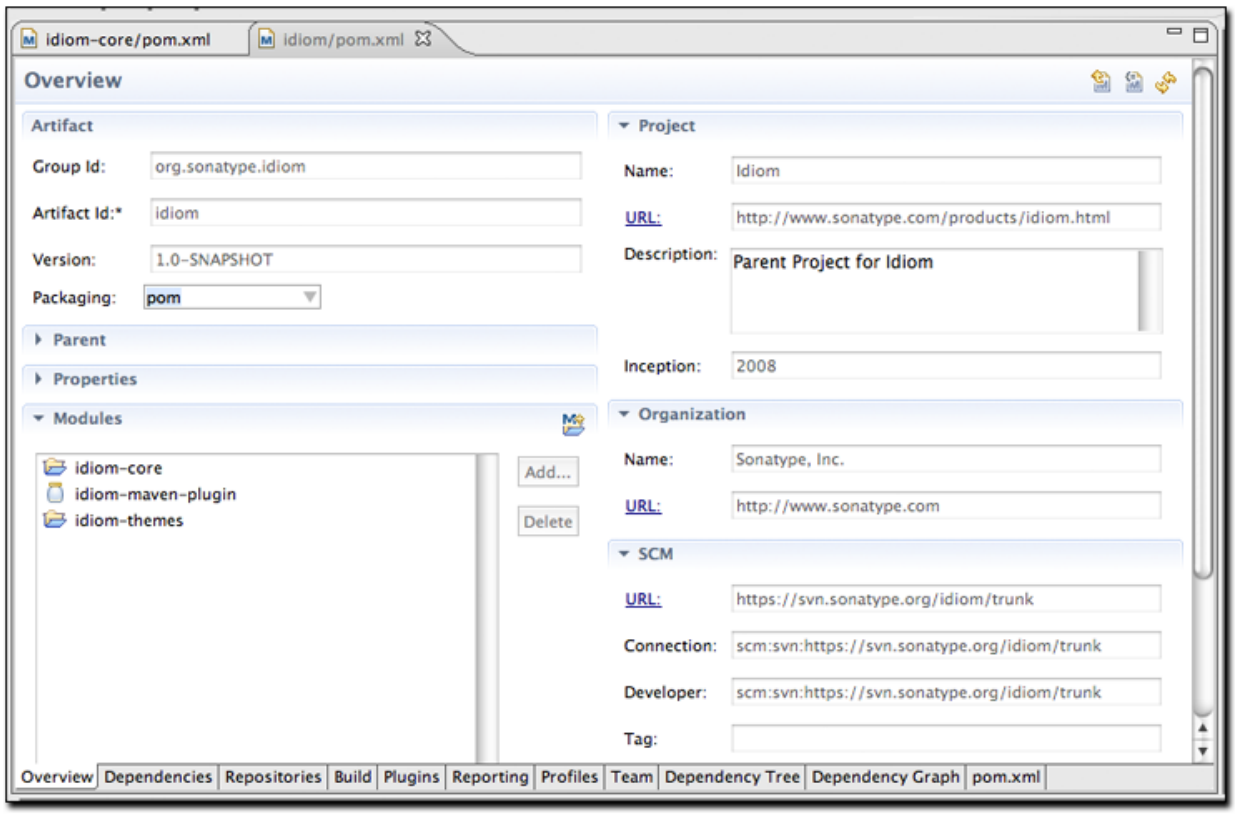


Figure 3.28. Overview Tab of POM Editor for idiom Parent Project

In this parent POM, we see that the `groupId` and `version` are defined and that the parent POM supplies much of the information which was missing in the `idiom-core` project. The POM editor is going to show you the contents of the POM that you are editing, and it will not show you any of the inherited values. If you wanted to look at the `idiom-core` project's effective POM in the POM editor, you can use “Show Effective POM” action from the tool-bar in the upper right-hand corner of the POM editor, which shows a left bracket and an equals sign on a page with a blue M. It will load the effective POM for `idiom-code` in the POM Editor as shown in Figure 3.29, “Effective POM for idiom-core”.

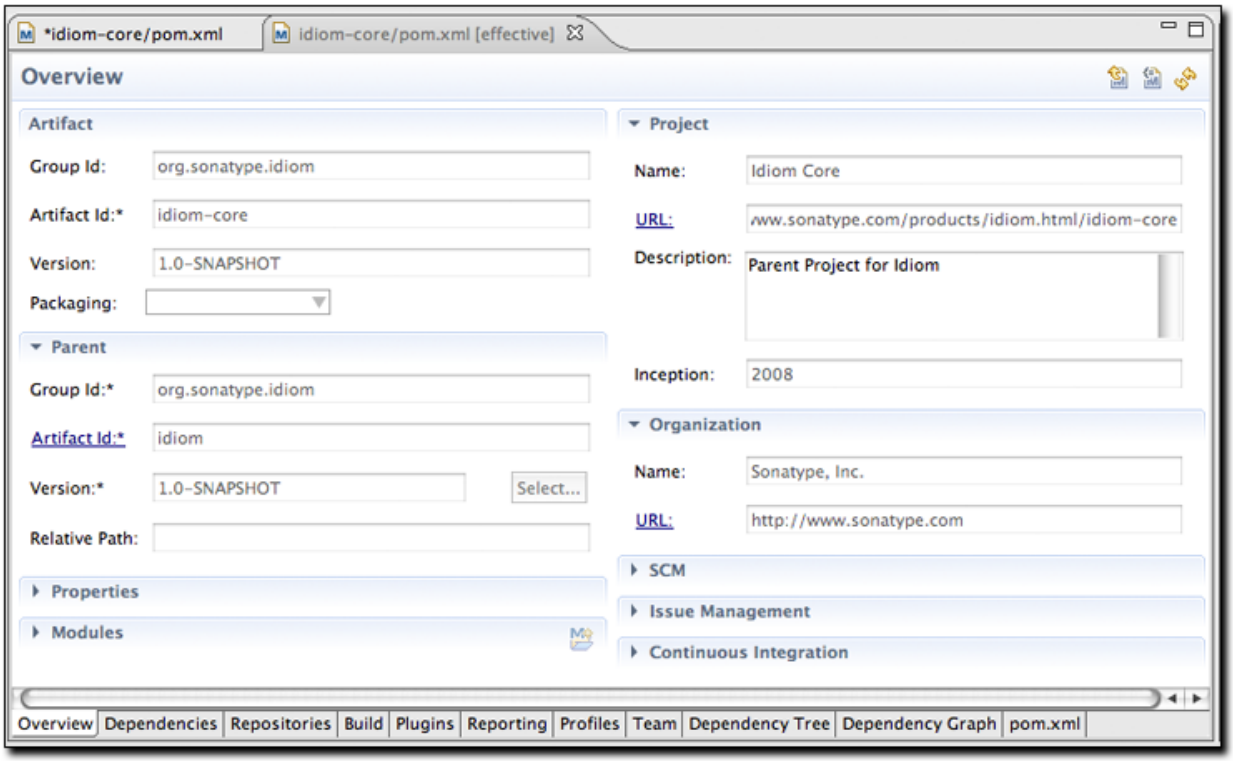


Figure 3.29. Effective POM for idiom-core

This effective view of the POM merges the `idiom-core` POM with the ancestor POMs (the parent, the grandparent, etc.), similarly to “mvn help:effective-pom” command and displays the POM editor with the effective values. Because the POM editor is display a composite view of many different merged POMs, this effective POM Editor is read-only, and you will not be able to update any of the fields in this effective POM view.

If you were looking at the POM editor for the idiom-core project as shown in Figure 3.27, “Overview Tab of POM Editor for idiom-core”, you can also navigate to the parent POM using, ”Open Parent POM” action from the POM editor tool-bar in the upper right-hand of the POM editor.

The POM editor shows a number of showing various information from the POM. The final tab exposes the `pom.xml` as an XML document. There is a dependencies tab shown in Figure 3.30, “Dependencies Tab of the POM Editor” which exposes an easy-to-use interface for adding and editing dependencies to your project, as well as editing the `dependencyManagement` section of the POM. This dependency management screen is also integrated with the artifact searching facilities in the m2eclipse plugin. You can use actions from the editor sections, as well as Ctrl-Space typing assistance for the fields in “Dependency Details” section.

If you need to know more about one of the artifacts, you can use “Open Web Page” action from “Dependency Details” section tool-bar to check the project web page.

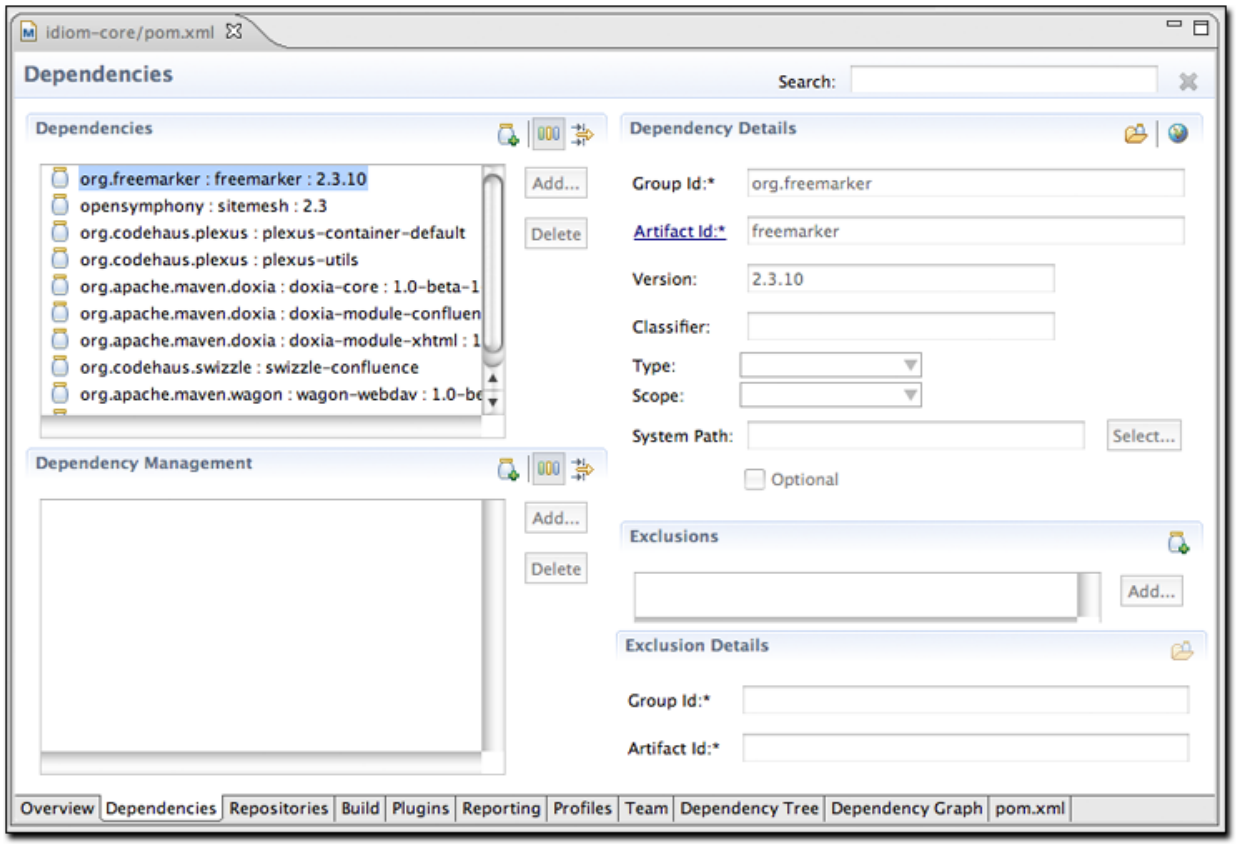


Figure 3.30. Dependencies Tab of the POM Editor

The build tab shown in Figure 3.31, “Build Tab of the POM Editor” provides access to the contents of the build element. From this tab you can customize source directories, add extensions, change the default goal name, and add resources directories.

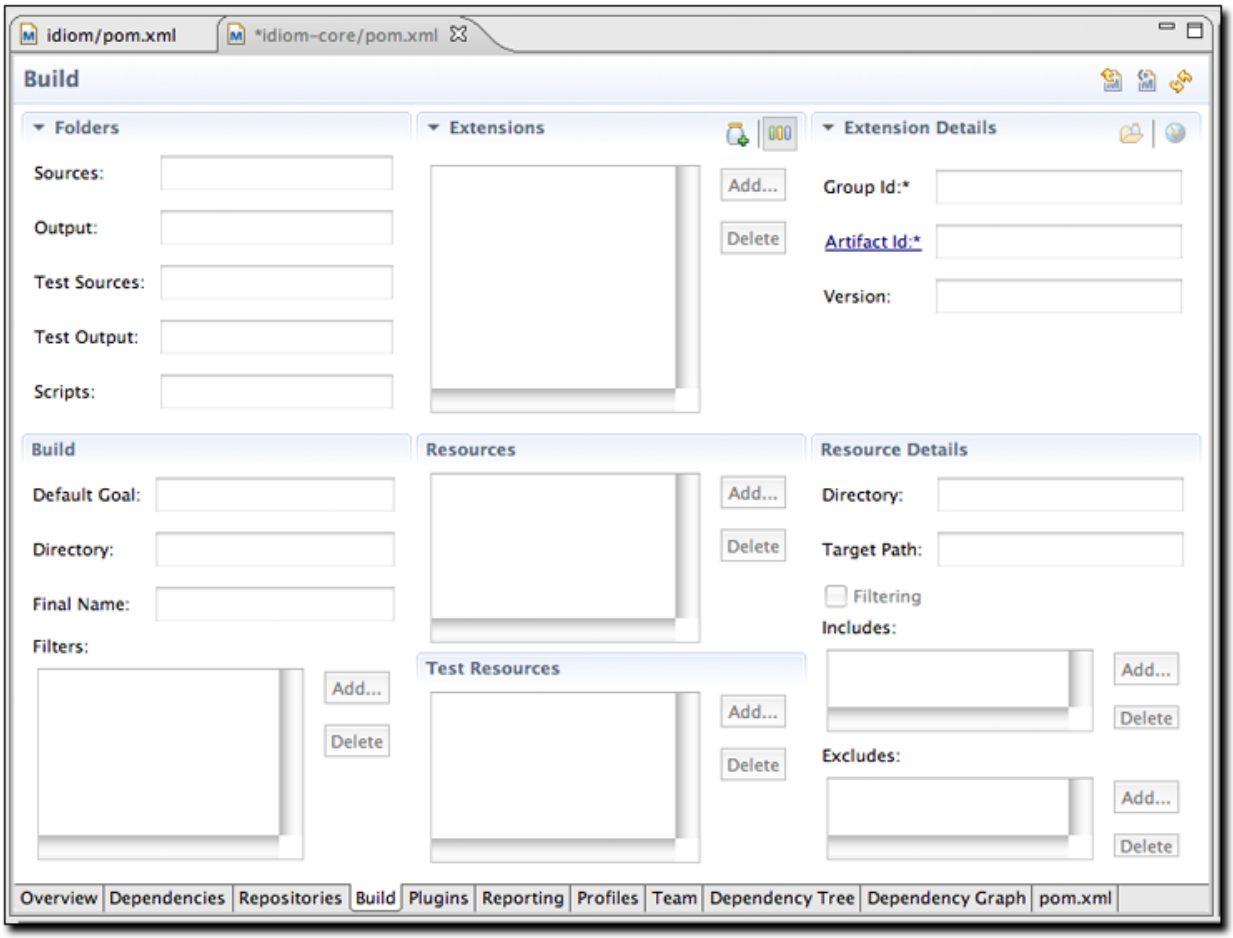


Figure 3.31. Build Tab of the POM Editor

We only showed a small subset of the POM editor. If you are interested in seeing the rest of the tabs, please download and install the m2eclipse plugin.

3.9. Analyzing Project Dependencies in m2eclipse

The latest release of m2eclipse contains a POM editor which provides some dependency analysis tools. These tools promise to change the way people maintain and monitor a project's transitive dependencies. One of the main attractions to Maven is the fact that it manages a project's dependencies. If you are writing an application which depends on the Spring Framework's Hibernate3 integration, all you need to do is depend on the `spring-hibernate3` artifact from the Central Maven Repository. Maven then reads this artifact's POM and adds all of the necessary transitive dependencies. While this is a great feature that attracts people to using Maven in the first place, it can often become confusing with a project starts to depend on tens of dependencies, each with tens of transitive dependencies.

Problems start to happen when you depend on a project with a poorly crafted POM which fails to

flag dependencies as optional, or when you start encountering conflicts between transitive dependencies. If one of your requirements is to exclude a dependency like `commons-logging` or the `servlet-api`, or if you need to find out why a certain dependency is showing up under a specific scope you will frequently need to invoke the `dependency:tree` and `dependency:resolve` goals from the command-line to track down the offending transitive dependencies.

This is where the POM editor in m2eclipse comes in handy. If you open a project with many dependencies, you can open the Dependency Tree tab and see a two-column display of dependencies as shown in Figure 3.32, “Dependency Tree Tab of the POM Editor”. The left-side of the panel displays a tree of dependencies. The first level of the tree consists of direct dependencies from your project, and each subsequent level lists the dependencies of each dependency. The left-hand side is a great way to figure out how a specific dependency made its way into your project's resolved dependencies. The right-hand side of this panel displays the resolved dependencies. This is the list of effective dependencies after all conflicts and scopes have been applied, and it is the effective list of dependencies that your project will use for compilation, testing, and packaging.

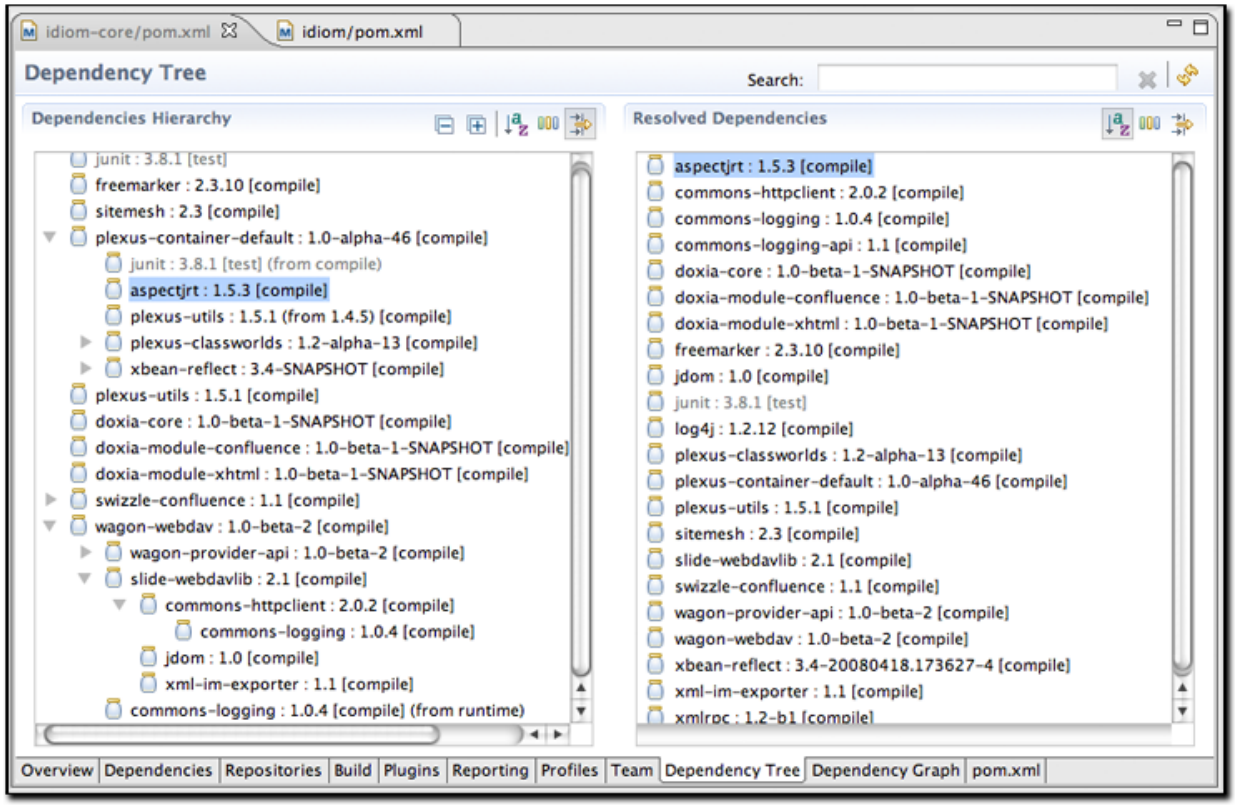


Figure 3.32. Dependency Tree Tab of the POM Editor

The feature which makes the Dependency Tree tab so valuable is that it can be used as an investigative tool to figure out how a specific dependency made it into the list of resolved dependencies. Searching and filtering functionality available in the editor makes it really easy to

search and browse trough the project dependencies. You can use “Search” entry field from the editor tool-bar and “Sort” and “Filter” actions from “Dependency Hierarchy” and “Resolved Dependencies” sections to navigate trough dependencies. Figure 3.33, “Locating Dependencies in the Dependency Tree” shows what happens when you click on commons-logging in the "Resolved Dependencies" list. When filtering is enabled in “Dependencies Hierarchy” section, clicking on a resolved dependency filters the hierarchy on the left-hand side of the panel to show all of the node which contributed to the resolved dependency. If you are trying to get rid of a resolved dependency, you can use this tool to find out what dependencies (and what transitive dependencies) are contributing the artifact to your resolved dependencies. In other words, if you are trying to get rid of something like `commons-logging` from your dependency set, the Dependency Tree tab is the tool you will likely want to use.

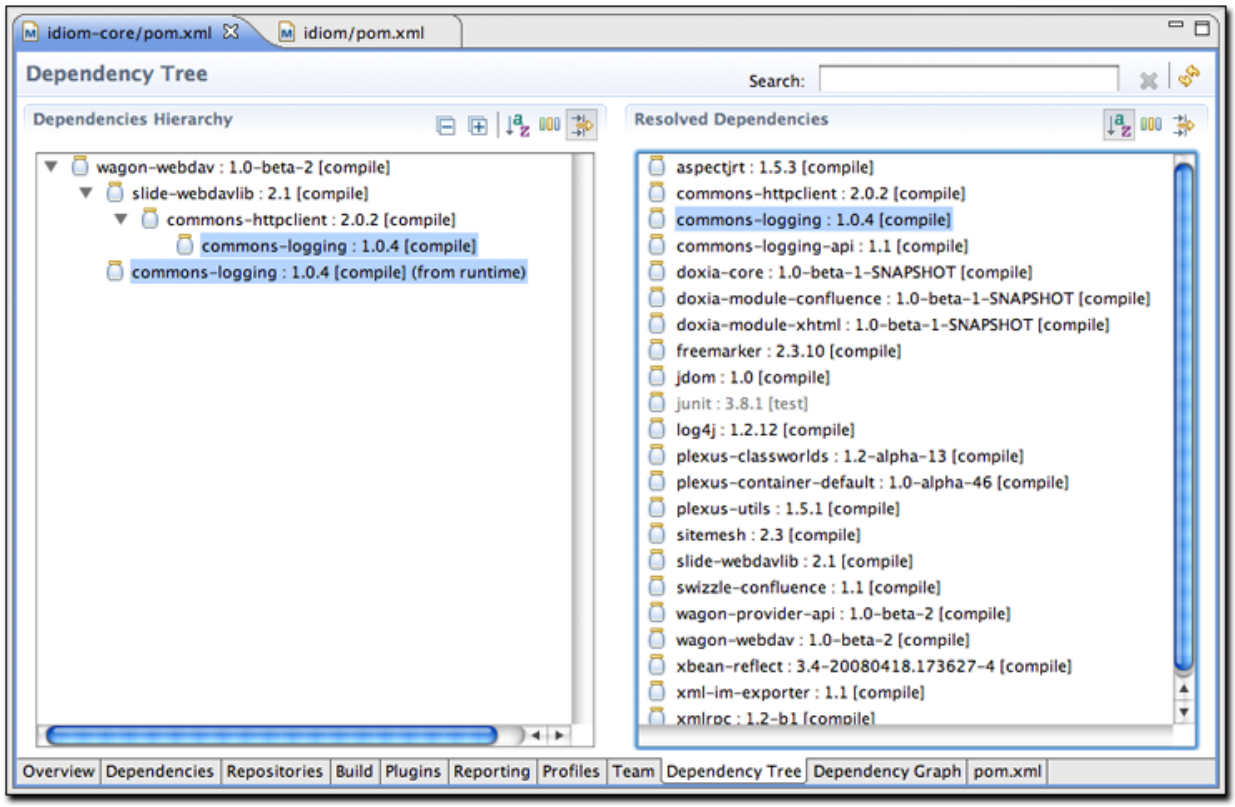


Figure 3.33. Locating Dependencies in the Dependency Tree

m2eclipse also provides you with the ability to view your project's dependencies as a graph. Figure 3.34, “Viewing the Dependencies of a Project as a Graph” shows the dependencies of `idiom-core`. The top-most box is the `idiom-core` project and the other dependencies are shown below it. Direct dependencies are linked from the top box and the transitive dependencies are linked from those. You can select a specific node in the graph to highlight the linked dependencies, or you can use the Search field at the top of the page to find matching nodes.

Note that “open folder” icon on each graph node indicates that the corresponding artifact is present in the Eclipse workspace and “jar” icon indicates that the node's artifact is referenced

from the Maven repository.

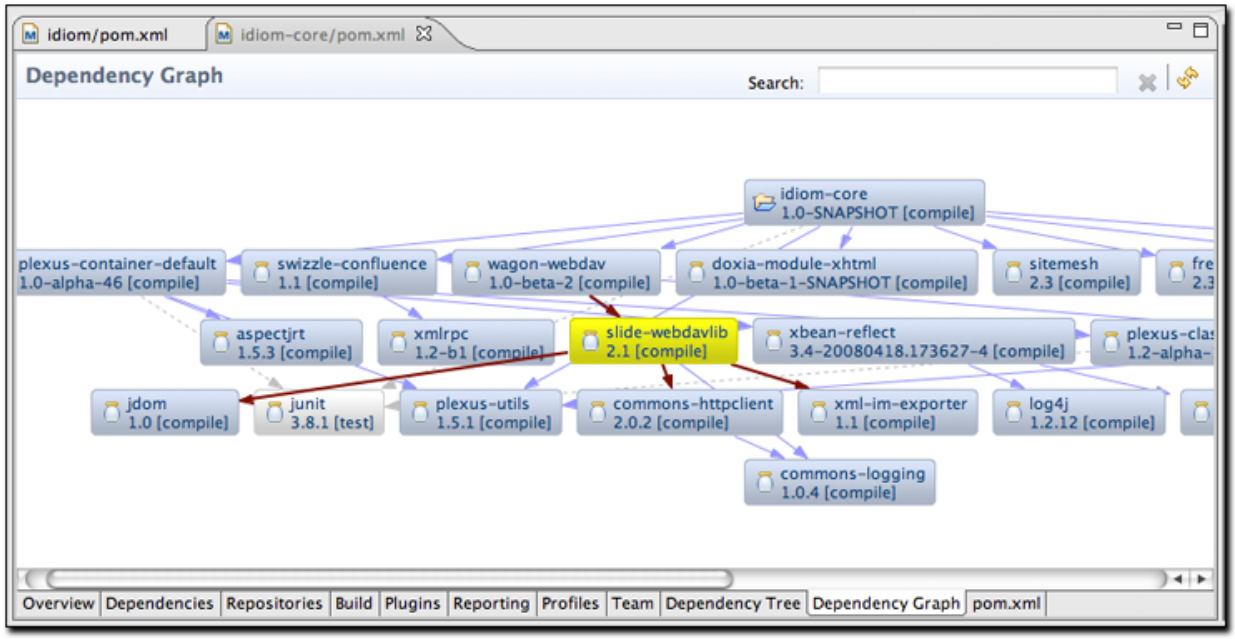


Figure 3.34. Viewing the Dependencies of a Project as a Graph

The graph presentation can be changed by right clicking in the editor. You can choose to show artifact ids, group ids, versions, scopes, or if you want to wrap node text or show icons. Figure 3.35, “Radial Layout of Dependency Graph” shows the same graph from Figure 3.34, “Viewing the Dependencies of a Project as a Graph” with a radial layout.

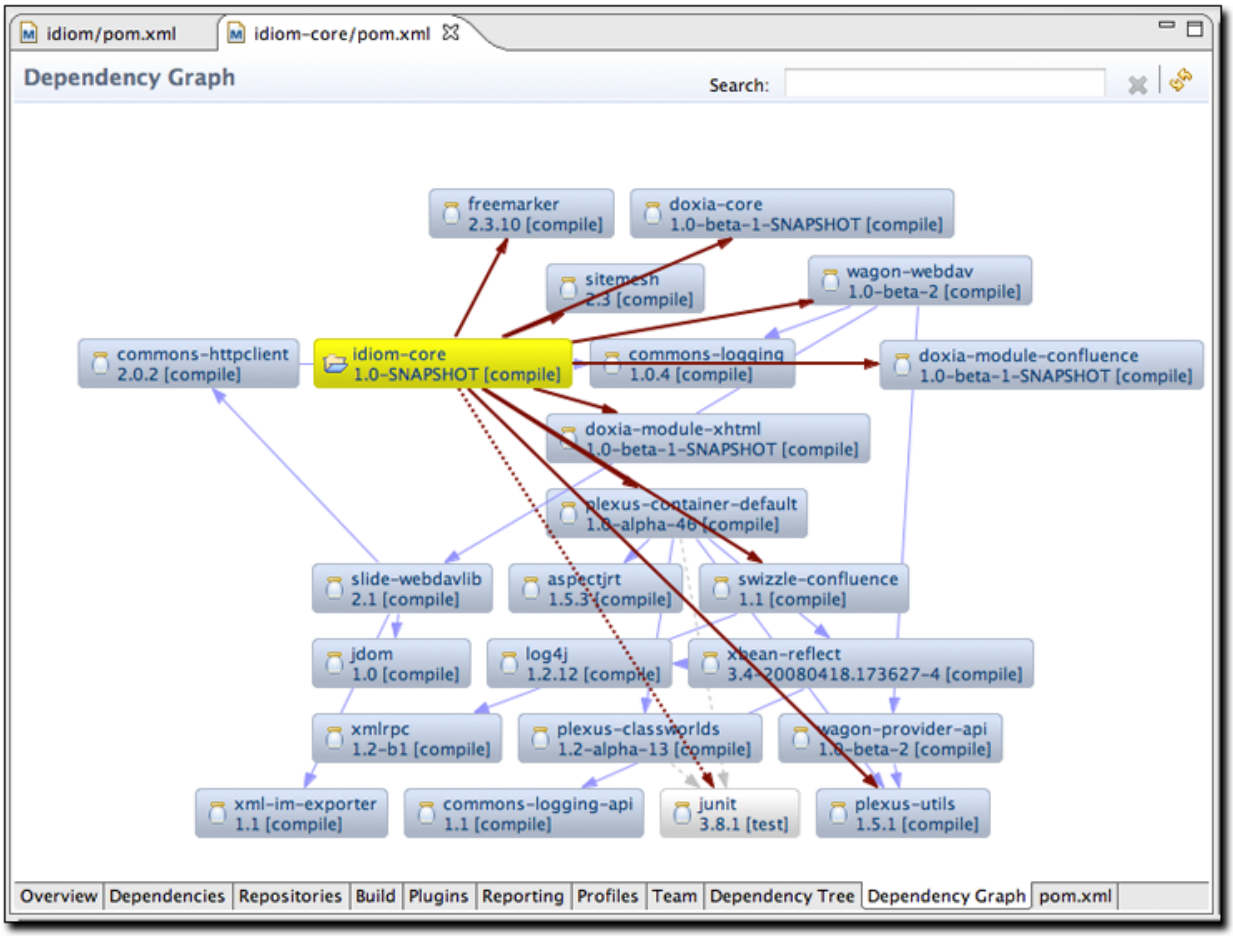


Figure 3.35. Radial Layout of Dependency Graph

3.10. Maven Preferences

The ability to adjust the Maven preferences and some Maven options is an important aspect of developing with Maven and m2eclipse offers the ability to tweak these items via the Maven preferences page inside of Eclipse. Typically when using Maven on the command line, such preferences and options are available from files in your `~/.m2` directory and as command line options. m2eclipse provides access to some of the most important preferences and options from the Eclipse IDE. Figure 3.36, “Maven Preferences for Eclipse” shows the Maven preferences page in Eclipse:

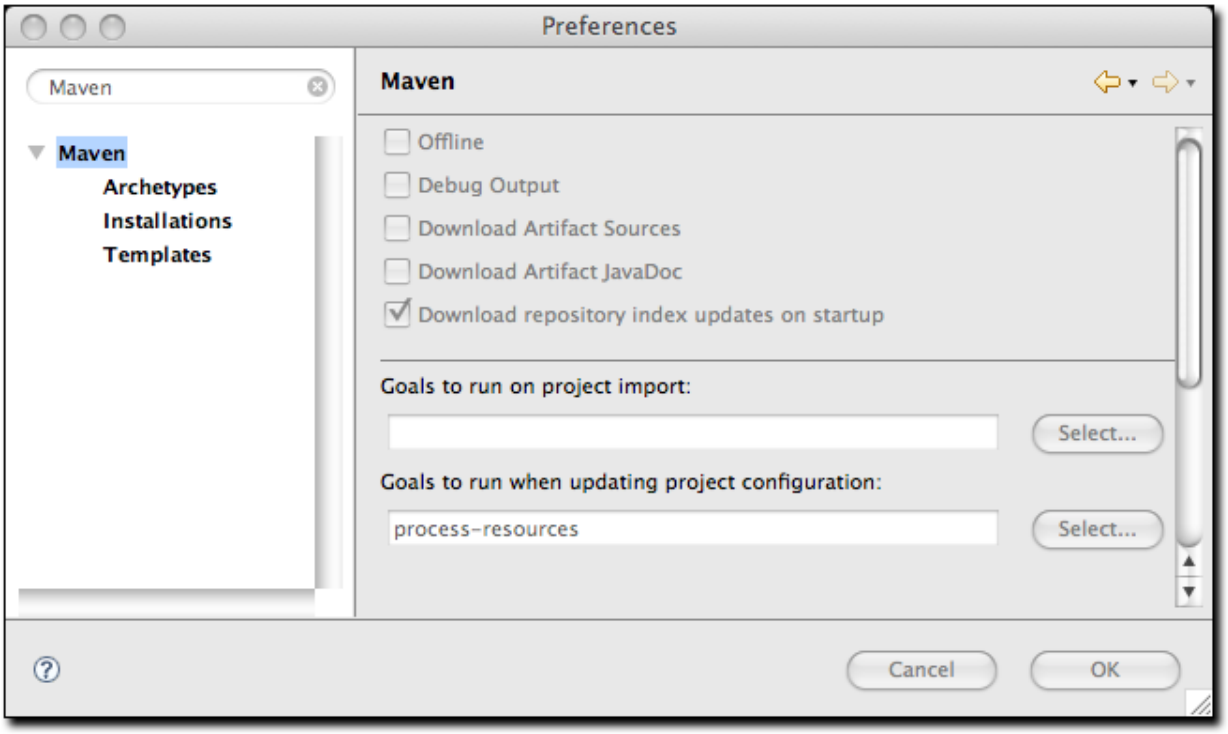


Figure 3.36. Maven Preferences for Eclipse

The check boxes in the top section provide the ability to:

- Run Maven in Offline mode, disabling any downloads from remote repositories
- Enable Debug output in the Maven Console
- Download Source jars for artifacts from remote Maven repositories
- Download JavaDoc jars for artifacts from remote Maven repositories
- Download and Update local indexes for remote repositories on startup

The next section offers a pop-up menu to select which goal you'd like to be executed when a project is imported and when the source folders for a given project are updated. The default goal is named `process-resources` which copies and process the resources for the project into the destination directory to make the project ready for packaging. Customizing this list of goals can come in handy if you need to run any custom goals which process resources or generate supporting configuration.

If you need help selecting a goal, click the Select... button to see the "Goals" dialog. The dialog on the left-hand side ofFigure 3.37, "Maven Goal Dialogs" shows the Goals dialog with a list of all the phases in the default Maven lifecycle.

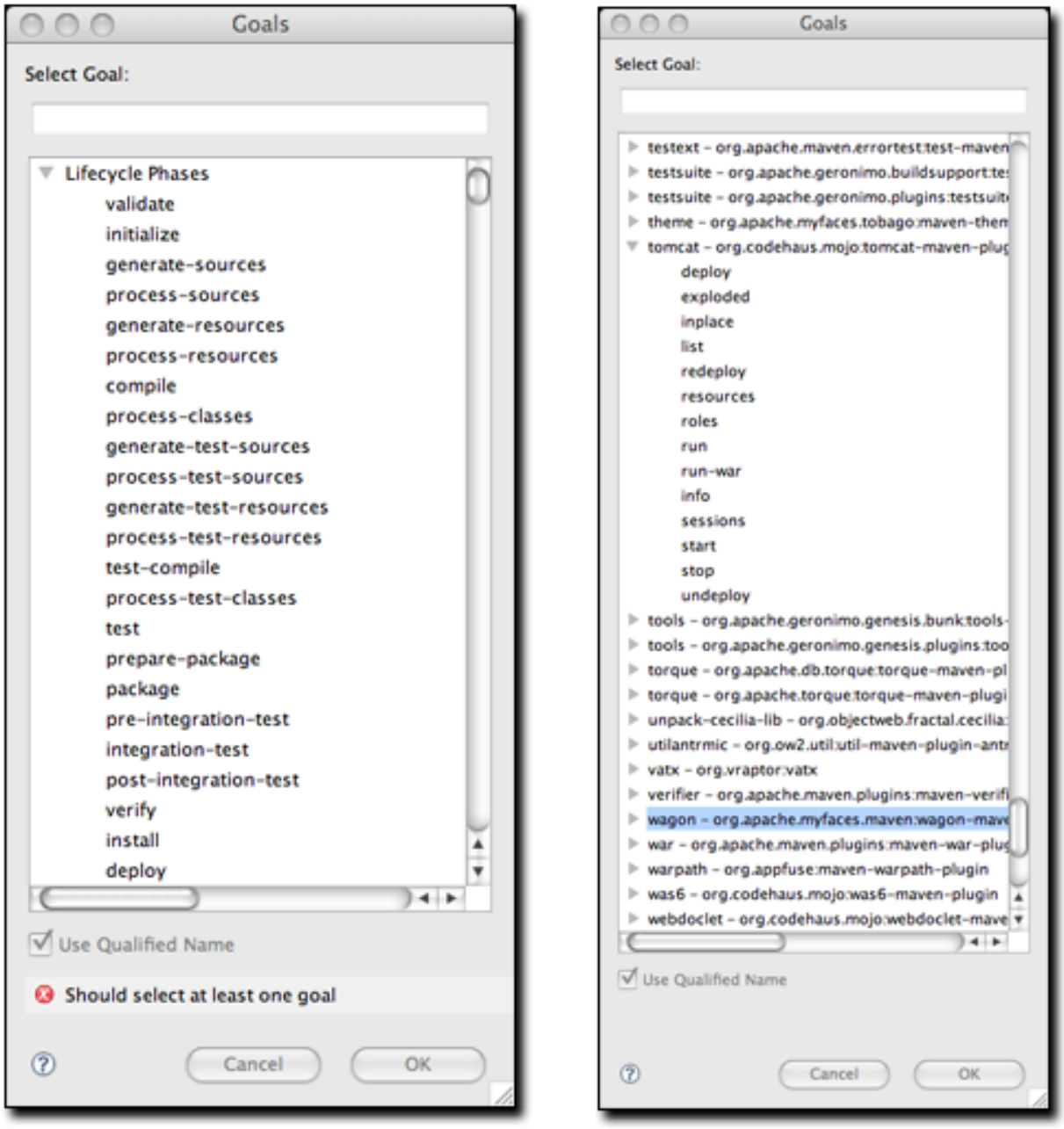


Figure 3.37. Maven Goal Dialogs

When you see the Goals dialog for the first time, there's a chance you might be overwhelmed by the number of goals it lists. There are literally hundreds of Maven plugins for everything from generating a database, to running integration tests, to performing static analysis, to generating web services with XFire. There are over two hundred plugins with selectable goals in the Goals dialog, the dialog on the right-hand side ofFigure 3.37, “Maven Goal Dialogs” shows the "Goals" dialog with the Tomcat Maven plugin's goals highlighted. You can always narrow the list of goals shown in this dialog by typing in some text to the search dialog, as you type in text,

m2eclipse is going to narrow the list of available goals to goals which contain the text in the search field.

Another Maven preference page is the Maven Installations configuration page shown in Figure 3.38, “Maven Installations Preference Page”:

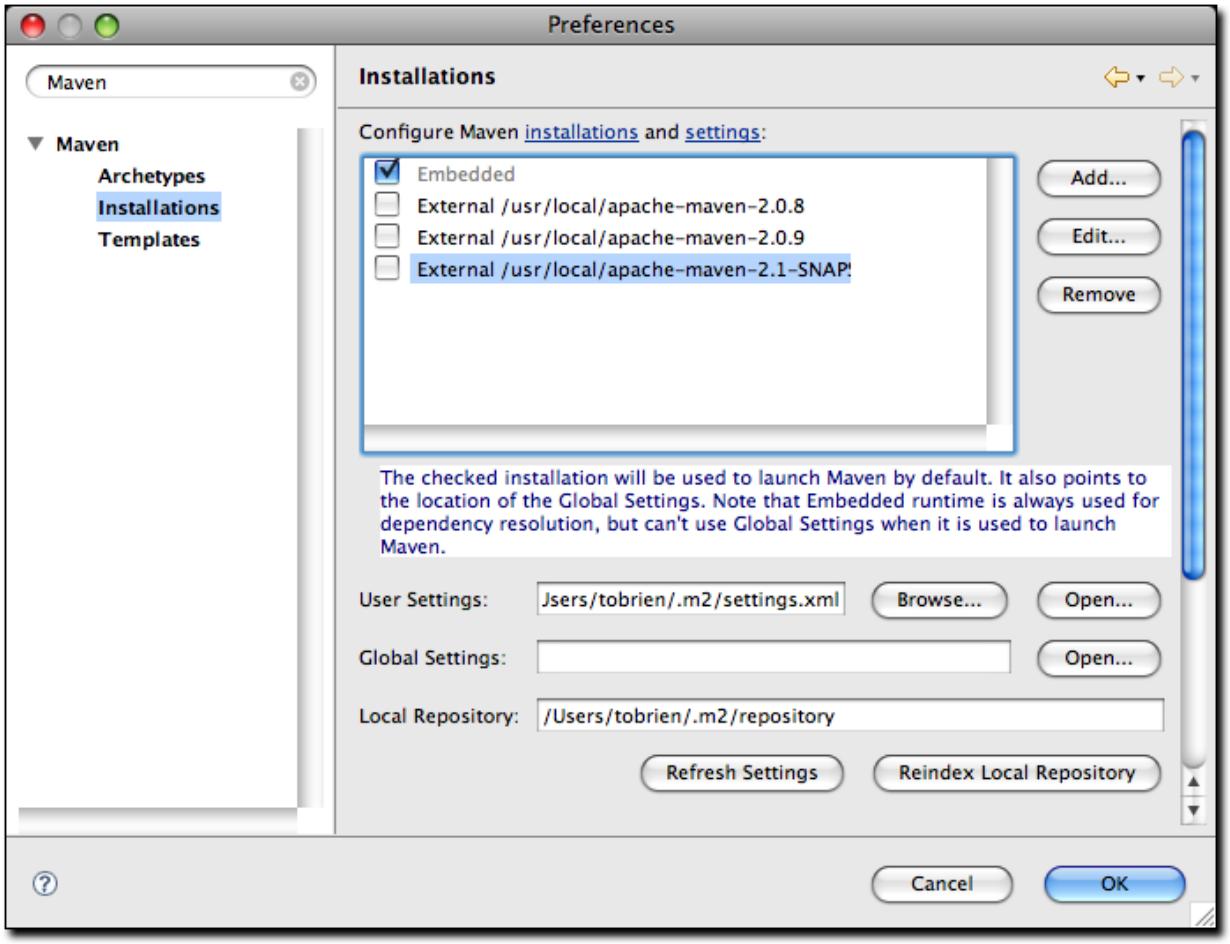


Figure 3.38. Maven Installations Preference Page

This page allows you to add other Maven installations to the Eclipse environment. If you want to use a different version of Maven with the m2eclipse plugin you can configure multiple installations of Maven from this configuration page, this is very similar to the ability to add more than one Java Virtual Machine to be run inside of Eclipse. An embedded version of the Maven known as the Maven Embedder is already specified. This is what is used to execute Maven inside of Eclipse. If you have another installation of Maven which you would like to use instead of the Maven Embedder, you can add another Maven runtime by clicking on the Add.. button. Figure 3.38, “Maven Installations Preference Page” shows a configuration page that lists the Maven Embedder, Maven 2.0.9, and an installation of Maven 2.1-SNAPSHOT.

The Installations configuration page also allows you to specify the location of the global Maven settings file. If you do not specify the location of this file on this configuration page, Maven will

use the default global settings file found in `conf/settings.xml` of the selected Maven installation. You can also customize the location of your user settings file from the default location of `~/.m2/settings.xml`, and you can customized the location of your local Maven repository from the default location of `~/.m2/repository`.

Also available in the Eclipse preferences is the ability to enable a decorator named the Maven Version Decorator. This preference provides a given project's current version on the Eclipse Package Explorer and is shown in Figure 3.39, "Enabling the Maven Version Decorator".

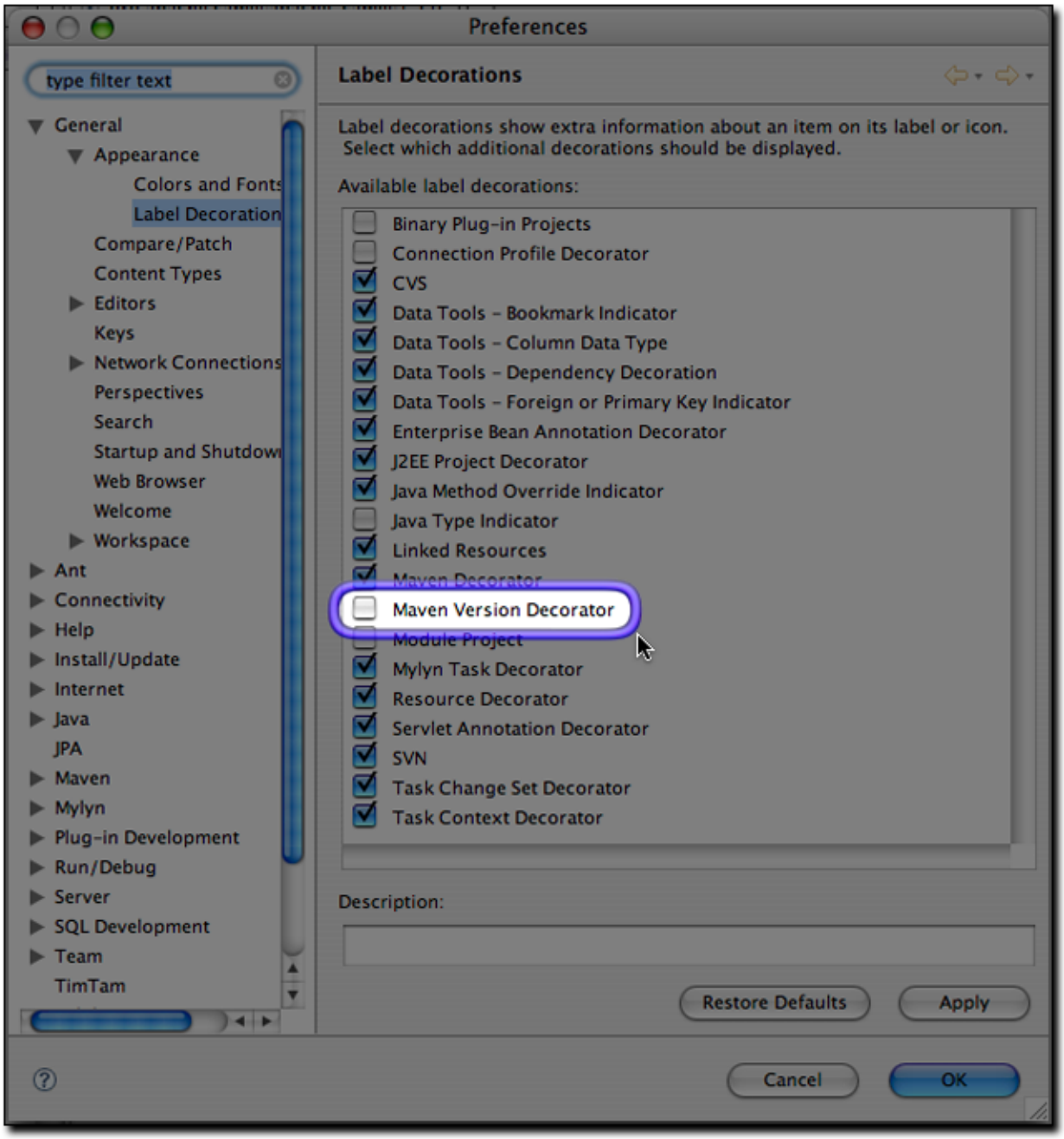


Figure 3.39. Enabling the Maven Version Decorator

To enable this preference, simply check the Maven Version Decorator option that is highlighted in Figure 3.39, “Enabling the Maven Version Decorator”. If the Maven Version Decorator is not enabled, a project will only list it’s name and relative path in the Package Explorer as shown in Figure 3.40, “Package Explorer without Maven Version Decorator”:

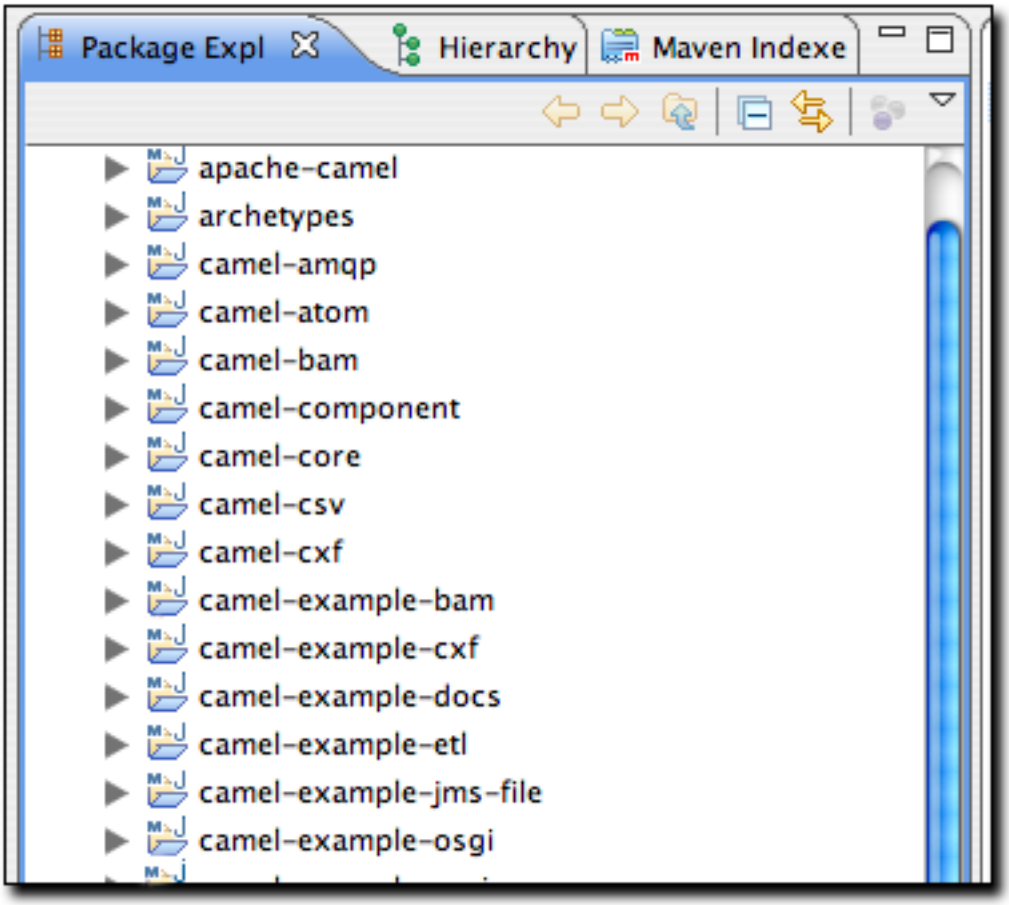


Figure 3.40. Package Explorer without Maven Version Decorator

Upon enabling the Maven Version Decorator, the project name will include the current project version as shown in Figure 3.41, “Package Explorer with Maven Version Decorator Enabled”:

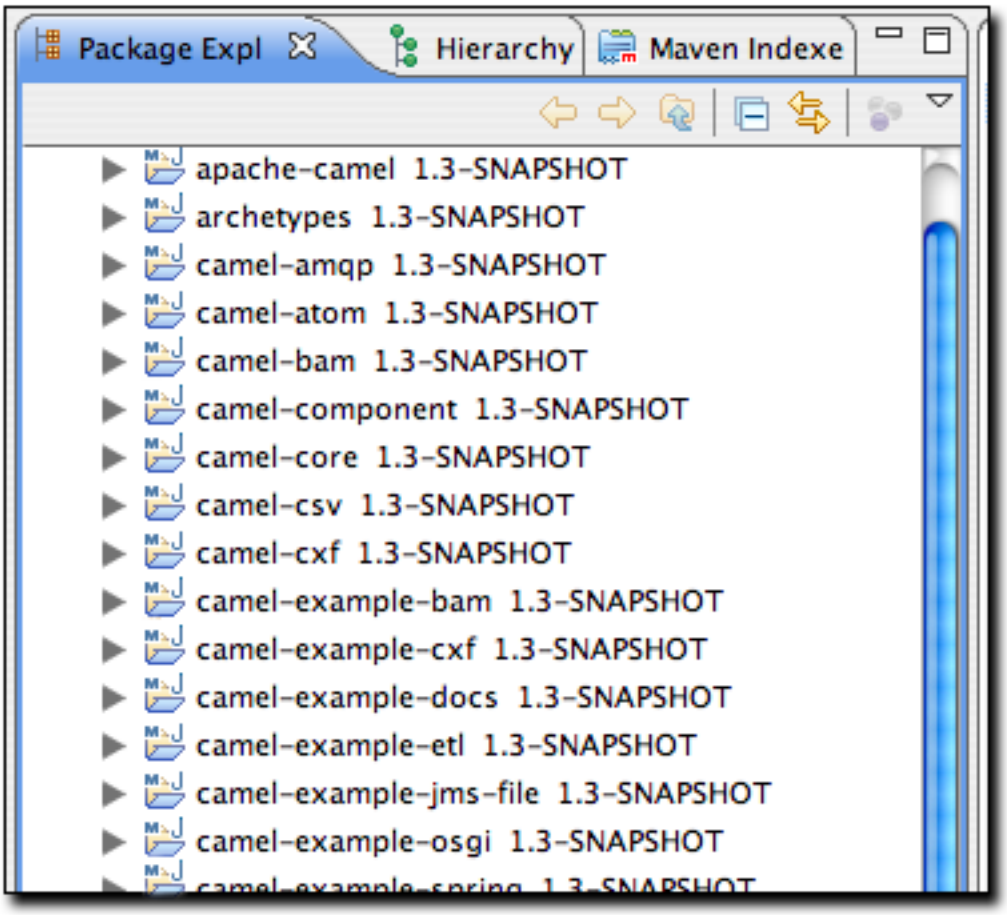


Figure 3.41. Package Explorer with Maven Version Decorator Enabled

This is a helpful feature that provides the project version at a glance instead of being required to open the POM to locate the version element.

3.11. Summary

m2eclipse is more than just a simple plugin which adds Maven support to Eclipse, it is a comprehensive integration that will make everything from creating new projects to locating third-party dependencies orders of magnitude easier. m2eclipse is the first step toward an IDE that is aware of the rich semantic treasure that is the central Maven repository. As more people come to use m2eclipse, more projects are going to be releasing Maven Archetypes, and more projects are going to see value in publishing source artifacts to the Maven repository. If you've tried to use Eclipse and Maven together without a tool that can comprehend the hierarchical project relationships that are central to any multi-module Maven project, you will know that the ability to work with nested projects is essential to smooth integration between the Eclipse IDE and Maven.

Index