PROGRESS
SOFTWARE

Artix ESB

Java Router, Getting Started

Version 5.5
December 2008

Java Router, Getting Started

Progress Software

Version 5.5

Published 10 Dec 2008

Table of Contents

List of Figures

List of Tables

Introducing Java Router

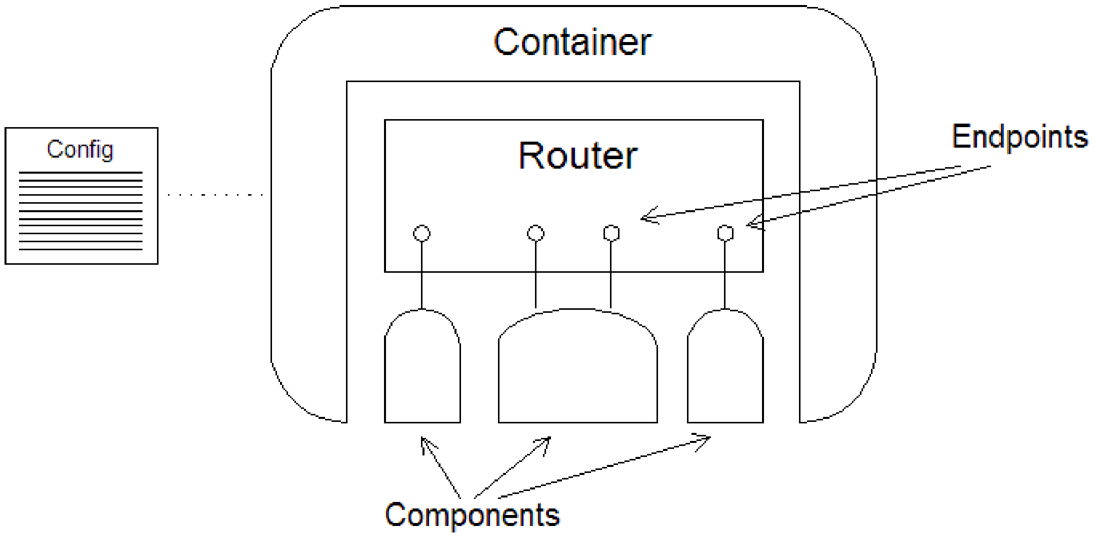
This chapter describes the architecture of the Java Router and introduces some basic concepts that are important for understanding how the router works.

Architecture

Overview

Figure 1 on page 10 gives a general overview of the Java Router architecture. This architecture is designed with the basic requirement in mind that the router should be deployable in a wide variety of different container types.

Figure 1. Architecture of the Java Router



Router

The router service is represented by a *Camel context* object, which encapsulates routing rules (in the form of `RouteBuilder` objects) and components (which enable the router to bind to various network protocols and other resources). The router application itself consists either of Java code or XML configuration, or possibly a combination of the two.

Endpoints

In general, an endpoint is a specific source or a sink of messages, identified by a URI. In practice, this means that an endpoint maps either to a network location or to some other resource that can produce or absorb a stream of messages. Within a routing rule, endpoints are used in two distinct ways: the *source endpoint* appears at the start of a rule (for example, in a `from()` command) and acts as a source of request messages and a sink for reply messages (if any); the *target endpoint* appears at the end of a rule (for example, in a `to()` command) and acts as a sink for request messages and a source of reply messages.

Components

A component is a plug-in that integrates the router core with a particular network protocol or external resource. From the perspective of a router developer, a component appears to be a factory for creating a specific type of endpoint. For example, there is a file component that can be used to create endpoints that read/write messages to and from particular directories or files. There is a CXF component that enables you to create endpoints that communicate with Web services (and related protocols).

Typically, before you can use a particular component, you need to configure it and add it to the Camel context. Some components, however, are embedded in the router core and do not need to be configured. The embedded components are as follows:

- Bean.
- Direct.
- File.
- JMX.
- Log.
- Mock.
- SEDA.
- Timer.
- VM.

For more details about the available components see the *Deployment Guide* and the list of Camel components [http://activemq.apache.org/camel/components.html].

RouteBuilders

The `RouteBuilder` classes encapsulate the routing rules. A router developer defines custom classes that inherit from `RouteBuilder` and adds instances of these classes to the `CamelContext`.

Deployment options

The router architecture is designed to facilitate deploying the router into different kinds of container. The following deployment options are currently supported:

- *Spring container deployment*—the router application is deployed into a Spring container and you can use the Spring configuration file to configure components and define routes.
- *Standalone deployment*—you must write a `main()` method in the application code, which is responsible for creating and registering `RouteBuilder` objects as well as configuring and registering components.

For more details about the deployment options, see the *Deployment Guide*.

Camel context

A `CamelContext` represents a single Camel routing rulebase. It defines the context used to configure routes and details which policies should be used during message exchanges between endpoints.

How to Develop a Router Application

Outline of the development steps

The following steps give a broad outline of what is involved in developing a router application:

1. *Choose a deployment option*—the router architecture is designed to support multiple deployment options. Currently, the following deployment options are supported:
   - Spring container deployment.
   - Standalone deployment.
2. *Define routing rules in Java DSL or in XML*—depending on the deployment option, you define the routing rules either in Java DSL or in XML.
3. *Configure components*—if you need to use components not already embedded in the router core, you must configure the components using either Java code or (in the case of a Spring container) XML.
4. *Deploy the router*—to deploy the router, follow the instructions for the particular container or deployment option that you have chosen. See the *Deployment Guide* for details.

Java Router Tutorial

This tutorial describes how to create, build and run a simple router using Apache Maven and the Eclipse IDE.

Prerequisites

Overview

Before you can follow the steps described in this tutorial, you must have the following:

- An active Internet connection (required by Maven).
- Java runtime on page 16 .
- Apache Maven 2 on page 16 .
- Eclipse on page 16 .

Java runtime

You should already have installed a Java runtime, as described in the *Install Guide*. Both Maven and Eclipse expect that the `JAVA_HOME` environment variable points at the root directory of your Java runtime (JDK 1.5.x) and that the Java `bin` directory is on your `PATH`.

Apache Maven 2

Apache Maven [http://maven.apache.org/] is a general purpose tool for building and packaging applications. You need to install Maven in order to generate the tutorial code. You can download the latest version (Maven 2.x) from the following location: http://maven.apache.org/download.html.

After installing Maven, you need to change the following settings in your operating system environment:

- Set the `M2_HOME` environment variable to point at the Maven root directory.
- Set the `MAVEN_OPTS` environment variable to `-Xmx512M` in order to expand the amount of memory available for Maven builds.
- Add the Maven `bin` directory (`%M2_HOME%\bin` on Windows or `$M2_HOME/bin` on UNIX) to your `PATH`.

Eclipse

Eclipse [http://www.eclipse.org/] is an open source Integrated Development Environment (IDE), which you can use to develop applications in Java (and other languages).In this tutorial, Eclipse is used to build and run the tutorial example. You can download the latest version of Eclipse from the following location: http://www.eclipse.org/downloads/. There are various Eclipse packages available—make sure that the package you download supports Java development. It is recommended that you install Eclipse version 3.3 or higher (the tutorial was tested with version 3.4).

After installing Eclipse, you need to change the following settings in your operating system environment:

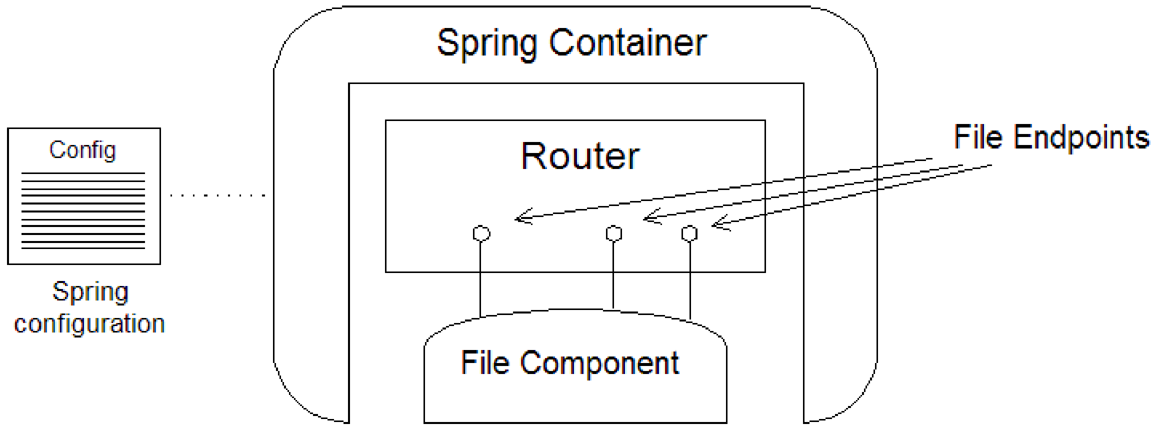
- Add the Eclipse root directory to your `PATH` (the `eclipse` binary is located in the root directory).

Tutorial Overview

Overview

Figure 2 on page 18 gives an overview of the architecture of the router that features in this tutorial.

Figure 2. Overview of the Tutorial



The simple router shown in Figure 2 on page 18 consists of the following parts:

- *Router*—the core component of the simple router. It consists of an instance of `org.apache.camel.builder.RouteBuilder` type, which is responsible for routing messages between component endpoints.

  The `main()` function of the simple router application calls a Spring wrapper class to start up a Spring container.

- *Spring container*—a standard container that enables you to instantiate and configure Java objects using an XML syntax (see Spring [http://www.springframework.org/]) . Spring also supports concepts such as *dependency injection* and *reversion of control*.

- *Spring configuration file*—by default, the Spring wrapper looks for Spring configuration files matching the pattern, `META-INF/spring/*.xml`, on the current classpath.

In this example, the Spring container is configured with the name of a Java package, `tutorial`. The Spring wrapper initializes any router artifacts (for example, instances of `RouteBuilder`) that it finds in the specified Java package.

- *File endpoints*—the `RouteBuilder` instance is responsible for routing messages between different endpoints. In this example, all of the endpoints are *file endpoints*. A file endpoint is used to read or write messages to the file system.

Tutorial stages

The tutorial consists of the following stages:

1. Tutorial: Install m2eclipse into the Eclipse IDE on page 20 .
2. Tutorial: Add Repositories to the Maven Index on page 25 .
3. Tutorial: Add the FUSE Archetypes Catalog on page 31.
4. Tutorial: Create a New Project on page 33.
5. Tutorial: Build and Run the Simple Router on page 38 .

Tutorial: Install m2eclipse into the Eclipse IDE

Overview

The `m2eclipse` plug-in from Sonatype [http://m2eclipse.sonatype.org/] integrates Maven functionality with the Eclipse IDE. After `m2eclipse` is installed, you can use Maven archetypes to generate projects within the Eclipse IDE. The `m2eclipse` plug-in also provides an editor for Maven POM files and support for managing Maven repositories. For a full list of features, see the m2eclipse Wiki [http://docs.codehaus.org/display/M2ECLIPSE/Home].

Plug-in dependencies

The `m2eclipse` plug-in depends on the following Maven plug-ins:

- *Subclipse plug-in*—integrates Eclipse with the Subversion version control system.
- *Mylyn plug-in*—integrates Eclipse with task management software (for example, JIRA).

Instructions on how to install these prerequisite plug-ins are included in the following install instructions.

Steps to install the m2eclipse plug-in

Perform the following steps to install the `m2eclipse` plug-in:

1. Start the Eclipse IDE. Use the Eclipse instance that is provided as part of the Artix installation, in the following directory:

   ```
   ArtixRoot/tools/eclipse
   ```

   Double-click on the `eclipse.exe` executable (Windows) or enter `eclipse` at the command line (UNIX/Linux).

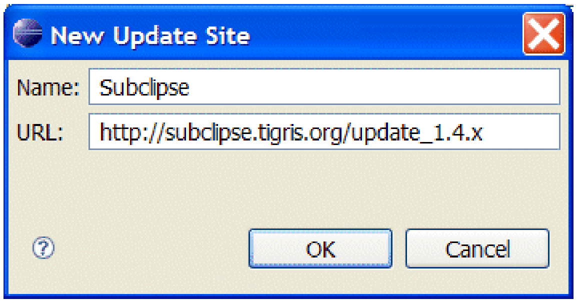
   Note

   The version of Eclipse provided with Artix is 3.3.2. You must have Eclipse version 3.3 or 3.4 in order to be compatible with the `m2eclipse` plug-in.

2. To add the Subclipse update site to the Eclipse site list, choose the menu item, **Help|Software Updates|Find and Install**, which brings up the **Install** wizard. In the **Feature Updates** panel, select the **Search for new features to install** radiobutton and then click **Next**.

3. In the **Update sites to visit** panel, click the **New Remote Site** button to bring up the **New Update Site** dialog. In the **URL** field, insert the URL for the Subclipse update site, `http://subclipse.tigris.org/update_1.4.x`. Enter `Subclipse` in the **Name** field.

   Figure 3. Eclipse New Update Site Dialog

   Name: Subclipse
   URL: http://subclipse.tigris.org/update_1.4.x

   Click **OK** to add the Subclipse site. After performing this step, the Subclipse update site should now be listed in the **Sites to include in search** list in the **Update sites to visit** panel.

4. Add the Mylyn update sites to the Eclipse site list. Follow the procedure described in the preceding step to add *either* of the following sites (depending on whether your Eclipse version is 3.3 or 3.4):

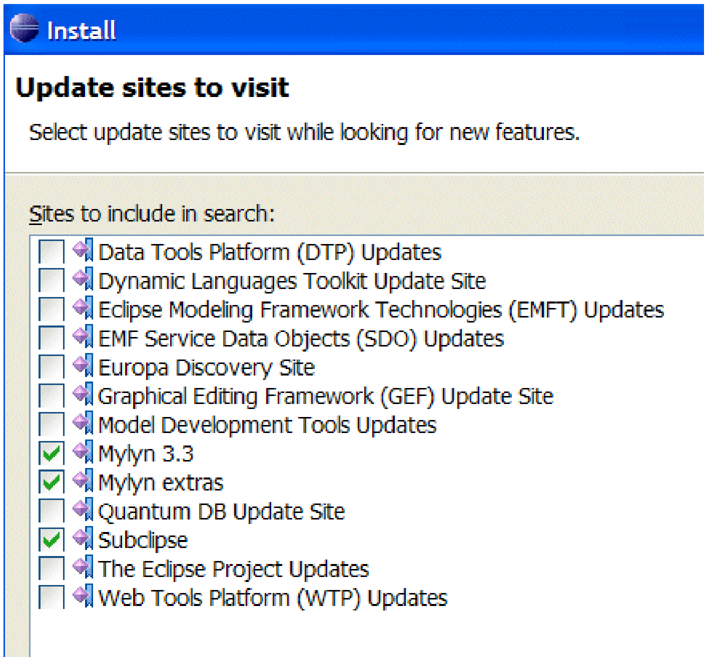
   ```
   http://download.eclipse.org/tools/mylyn/update/e3.3
   http://download.eclipse.org/tools/mylyn/update/e3.4
   ```

   Next, follow the same procedure to add the following site:

   ```
   http://download.eclipse.org/tools/mylyn/update/extras
   ```

5. To search the Subclipse and Mylyn update sites, check the boxes next to the Subclipse and Mylyn sites in the **Sites to include in search** list, as shown in Figure 4 on page 22.

Figure 4. Searching the Subclipse and Mylyn Sites



Update sites to visit

Select update sites to visit while looking for new features.

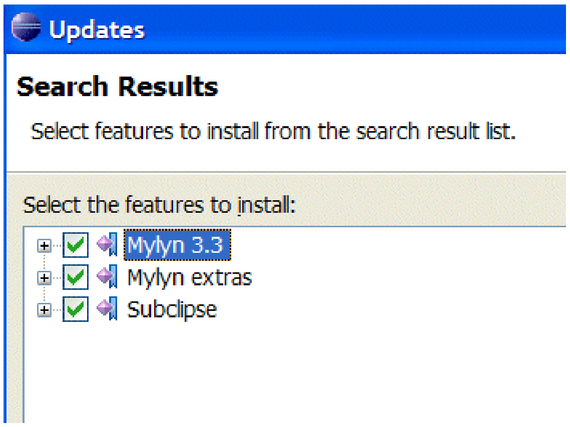
Sites to include in search:

- [ ] Data Tools Platform (DTP) Updates
- [ ] Dynamic Languages Toolkit Update Site
- [ ] Eclipse Modeling Framework Technologies (EMFT) Updates
- [ ] EMF Service Data Objects (SDO) Updates
- [ ] Europa Discovery Site
- [ ] Graphical Editing Framework (GEF) Update Site
- [ ] Model Development Tools Updates
- [✓] Mylyn 3.3
- [✓] Mylyn extras
- [ ] Quantum DB Update Site
- [✓] Subclipse
- [ ] The Eclipse Project Updates
- [ ] Web Tools Platform (WTP) Updates

Click **Finish** to perform a search of the selected update sites.

6. In the **Search Results** pane, select all of the listed featues and clieck **Next**.

Figure 5. Search Results Pane


Updates
Search Results
Select features to install from the search result list.
Select the features to install:
Mylyn 3.3
Mylyn extras
Subclipse

7. In the **Feature License** pane, select the first radio button to accept the license agreements and then click **Next**. Click **Next** again, to accept the **Optional Features**. Click **Finish** to start the installation of the features and follow the on-screen directions. When the installation is finished, restart the Eclipse IDE.

8. After restarting the Eclipse IDE, choose **Help|Software Updates|Find and Install** to bring up the **Install** wizard again.

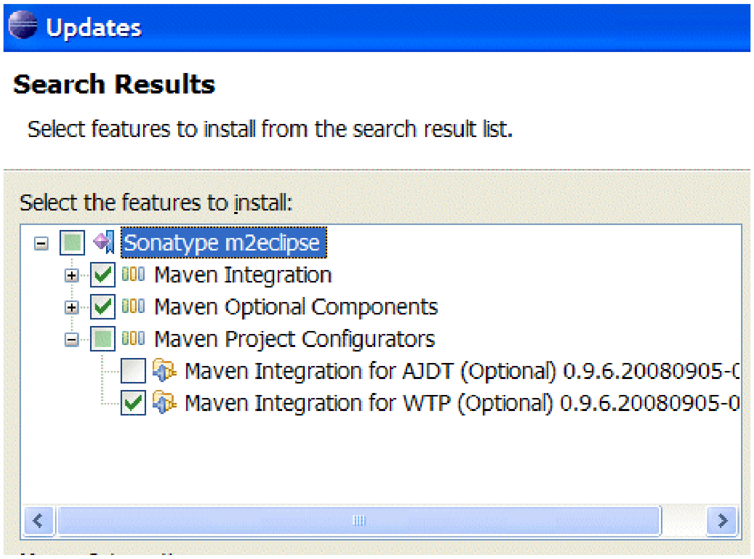
9. Add the `m2eclipse` update site to the update site list. Following the same procedure as before, use the **New Remote Site** button to add the following site:

```
http://m2eclipse.sonatype.org/update/
```

   After performing this step, the `m2eclipse` update site should appear in the site list.

10. Follow the same procedure as before to install the `m2eclipse` plug-in, except that when you get to the **Search Results** pane, you must omit the optional Maven Integration for AJDT feature (in order to avoid a clash with the existing Eclipse configuration).

Figure 6. Search Results for m2eclipse



Updates

Search Results

Select features to install from the search result list.

Select the features to install:

- ☐ Sonatype m2eclipse
  - ✓ Maven Integration
  - ✓ Maven Optional Components
  - ☐ Maven Project Configurators
    - ☐ Maven Integration for AJDT (Optional) 0.9.6.20080905-(
    - ✓ Maven Integration for WTP (Optional) 0.9.6.20080905-0

11. When the installation is finished, restart the Eclipse IDE.

Tutorial: Add Repositories to the Maven Index

Overview

This stage of the tutorial describes how to add the Java Router repositories to the Maven Index in Eclipse. The `m2eclipse` plug-in uses the list of repositories in the Maven Index to discover Maven archetypes and other Maven artifacts.

Java Router repositories

Java Router artifacts are stored in the following repositories:

Table 1. Java Router Repositories

Repository URL	Description
`http://repo.fusesource.com/maven2/`	Repository of stable releases.
`http://repo.fusesource.com/maven2-snapshot/`	Repository of current snapshot release (updated nightly).

Steps to add repositories

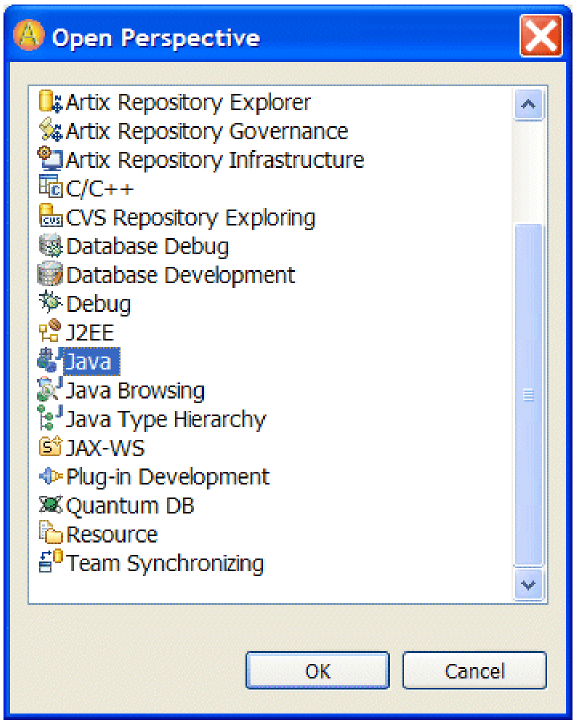
Perform the following steps to add the Java Router repositories to the Maven index:

1. Start the Eclipse IDE, either by double-clicking on the Eclipse icon or by entering the following command:

   ```
   eclipse
   ```

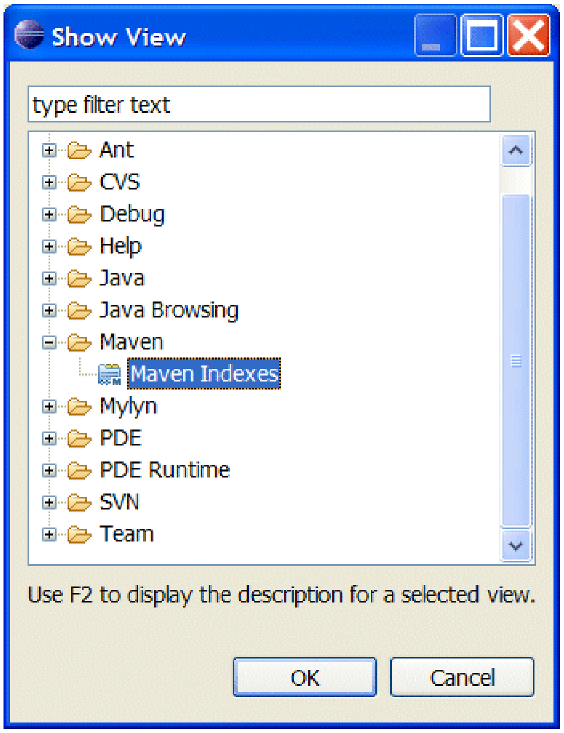
2. To open the Java perspective in the Eclipse workbench, choose the menu item, **Window|Open Perspective|Other**. From the **Open Perspective** dialog, select Java and click **OK**.

Figure 7. Eclipse Open Perspective Dialog



3. To open the Maven Indexes view in the Eclipse workbench, choose the menu item, **Window|Show View|Other**.

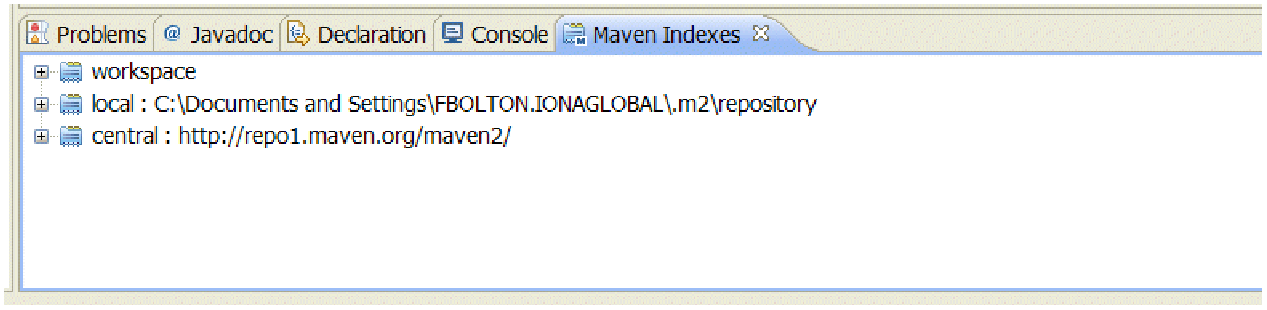
Figure 8. Eclipse Show View Dialog



Use F2 to display the description for a selected view.

From the **Show View** dialog, select the Maven Indexes view and click **OK**. The Maven Indexes tab now appears at the bottom of the Eclipse workbench.

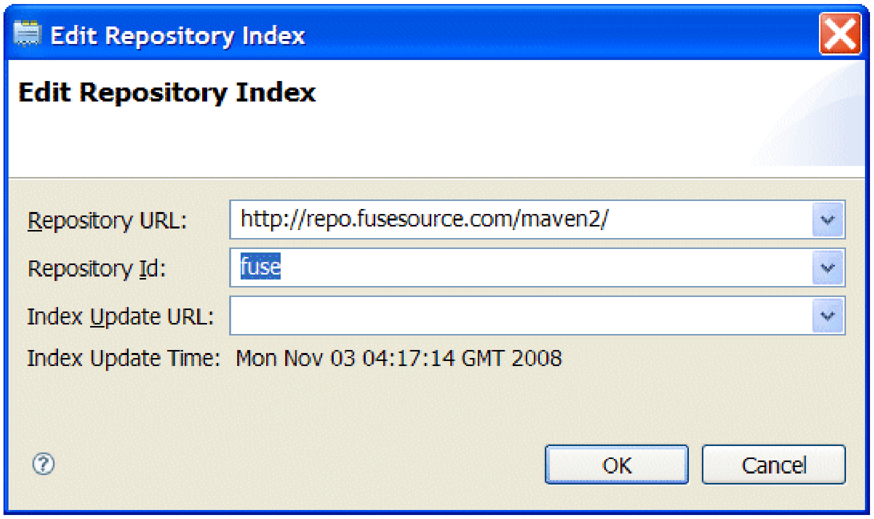
Figure 9. Maven Indexes Tab



workspace
local : C:\Documents and Settings\FBOLTON.IONAGLOBAL\.m2\repository
central : http://repo1.maven.org/maven2/

4. To add the FUSE *release repository* to the Maven Indexes, right click the Maven Indexes view and select **Add Index**. The **Add Repository Index** dialog appears.

Figure 10. Adding the FUSE Release Repository to Maven Indexes

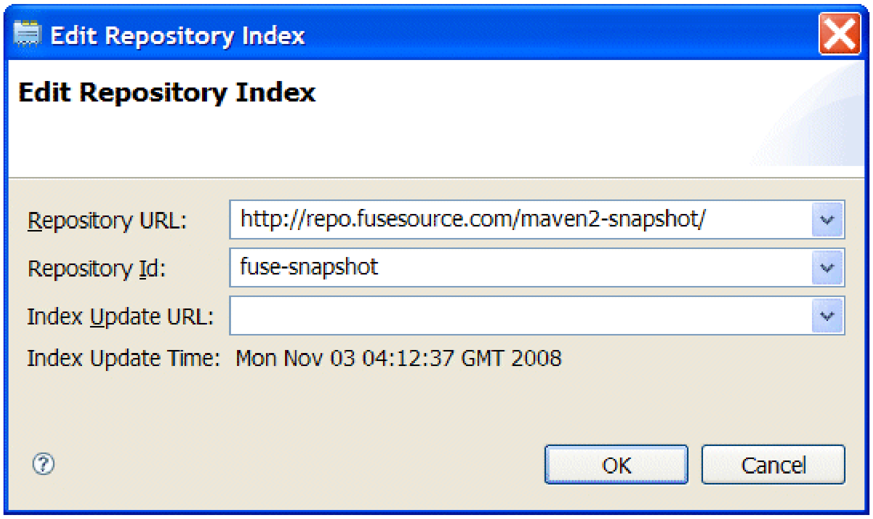


Repository URL: http://repo.fusesource.com/maven2/
Repository Id: fuse
Index Update URL:
Index Update Time: Mon Nov 03 04:17:14 GMT 2008

Enter `http://repo.fusesource.com/maven2/` in the **Repository URL** field, enter `fuse` in the **Repository Id** field, and then click **OK**. The new repository now appears in the Maven Indexes list, but you need to wait for a minute or so before the index contents are fully populated (check the status bar in the bottom right corner). If any errors occur while updating the new index, these will appear in the **Console** tab.

5. To add the FUSE *snapshot repository* to the Maven Indexes, right click the Maven Indexes view and select **Add Index**. The **Add Repository Index** dialog appears.

Figure 11. Adding the FUSE Snapshot Repository to Maven Indexes

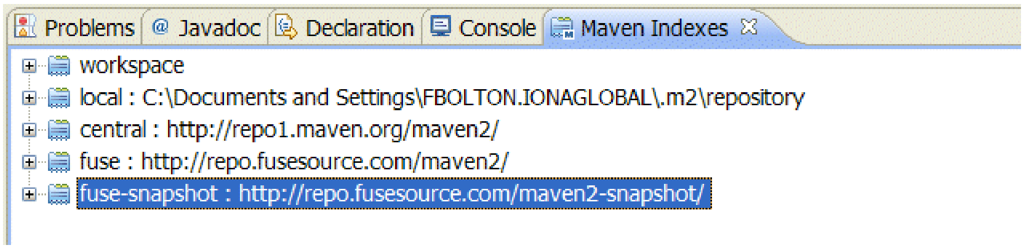


Repository URL: http://repo.fusesource.com/maven2-snapshot/
Repository Id: fuse-snapshot
Index Update URL:
Index Update Time: Mon Nov 03 04:12:37 GMT 2008

Enter `http://repo.fusesource.com/maven2-snapshot/` in the **Repository URL** field, enter `fuse-snapshot` in the **Repository Id** field, and then click **OK**.

6. After performing the preceding steps, your Maven Index view should look like the following:

Figure 12. Maven Indexes View with FUSE Repositories



workspace
local : C:\Documents and Settings\FBOLTON.IONAGLOBAL\.m2\repository
central : http://repo1.maven.org/maven2/
fuse : http://repo.fusesource.com/maven2/
fuse-snapshot : http://repo.fusesource.com/maven2-snapshot/

Tutorial: Add the FUSE Archetypes Catalog

Overview

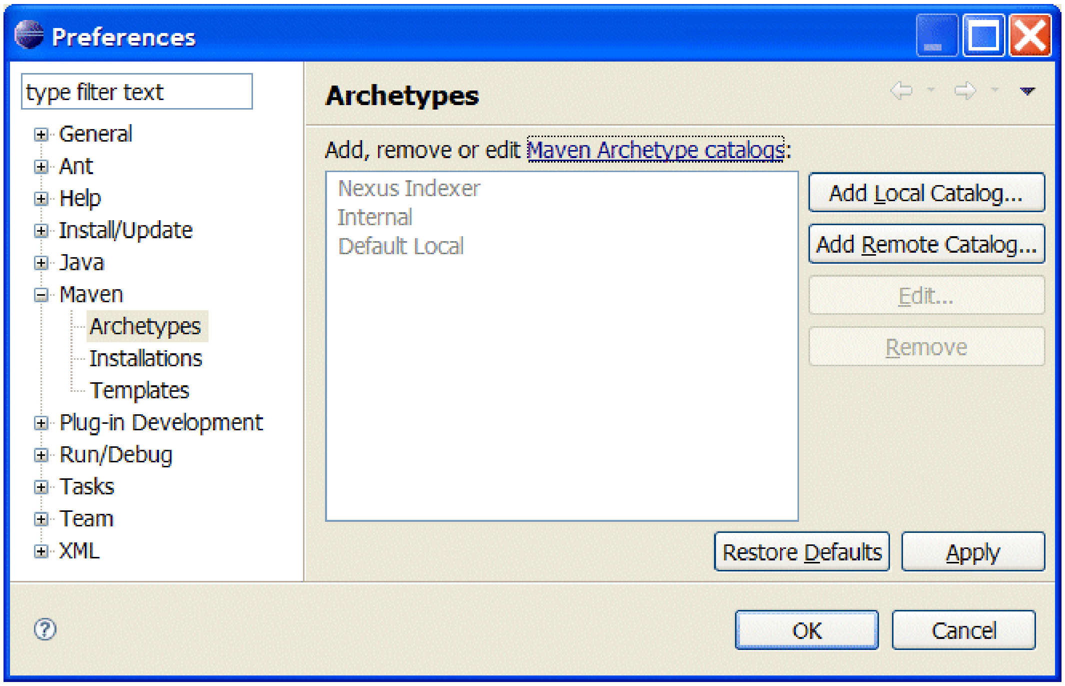
This stage of the tutorial explains how to add the FUSE archetypes catalog to the list of catalogs accessible from `m2eclipse`. This ensures that you can access the FUSE archetypes when you run the *Create a Maven Project* in Eclipse (see Tutorial: Create a New Project on page 33).

Steps to add the FUSE archetypes catalog

Perform the following steps to add the FUSE archetypes catalog:

1. Choose the menu item, **Windows|Preferences**, to open the **Preferences** dialog and drill down to the **Maven|Archetypes** preferences, as shown.

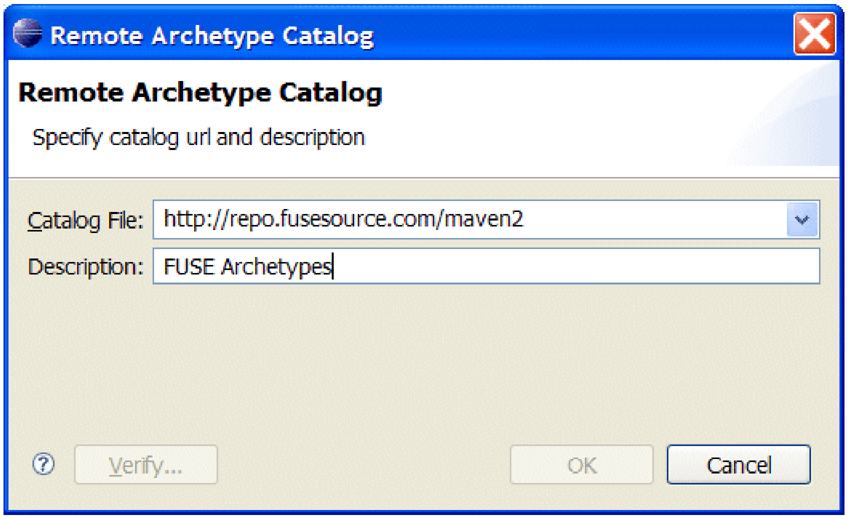
Figure 13. Maven Archetypes Preferences



2. Click **Add Remote Catalog** to bring up the **Remote Archetype Catalog** dialog. In the **Catalog File** text box, enter

`http://repo.fusesource.com/maven2` and in the **Description** text box, enter `FUSE Archetypes`. Click **OK**.

Figure 14. Remote Archetype Catalog Dialog



Catalog File: http://repo.fusesource.com/maven2
Description: FUSE Archetypes

3. In the **Preferences** dialog, click **Apply** and then click **OK**.

Tutorial: Create a New Project

Overview

In this stage of the tutorial, you will use the Maven build tool to generate code for a simple router application based on Java Router.

Camel archetypes

A Maven archetype is analogous to a new project wizard: it generates the outline of a sample project, where the generated project follows the standard Maven directory layout. Java Router provides the following Maven archetypes to generates basic projects:

Table 2. Java Router Archetypes

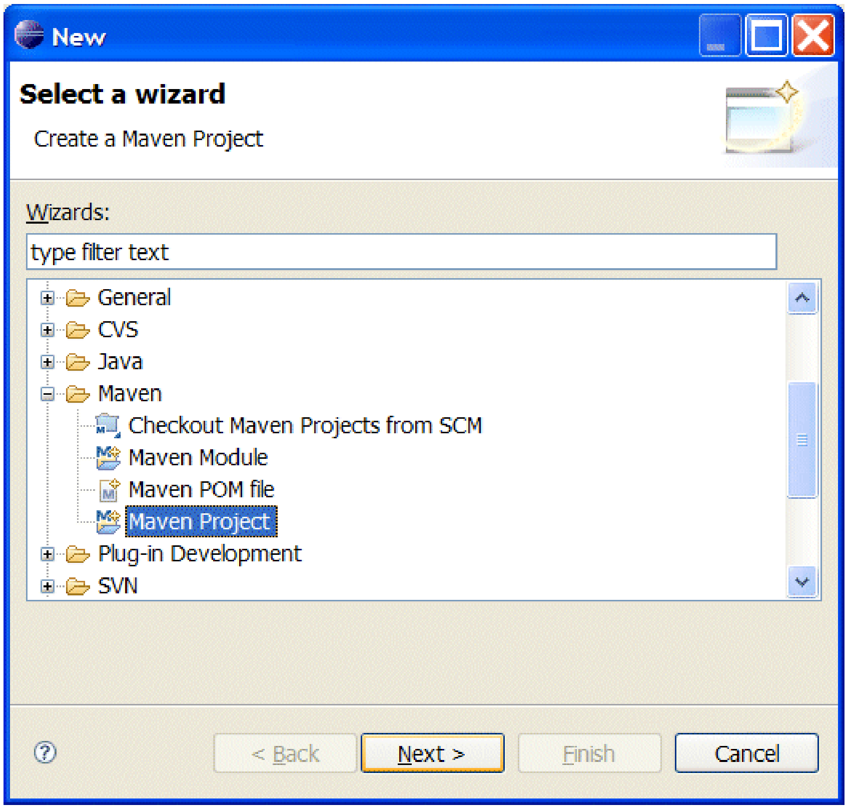
Archetype	Description
`camel-archetype-java`	Generates an example of a simple route written in Java DSL.
`camel-archetype-spring`	Generates an example of a simple route written in Spring XML.
`camel-archetype-scala`	Generates an example of a simple route written in Scala.
`camel-archetype-activemq`	Generates an example of a simple Spring XML route featuring the FUSE Message Broker (*Enterprise ActiveMQ*).
`camel-archetype-component`	Generates a starting point for writing your own Java Router component. For more details, see *Implementing a Component* in the *Java Router, Programmer's Guide*.

Steps to create a new project

Perform the following steps to generate a new Java Router project:

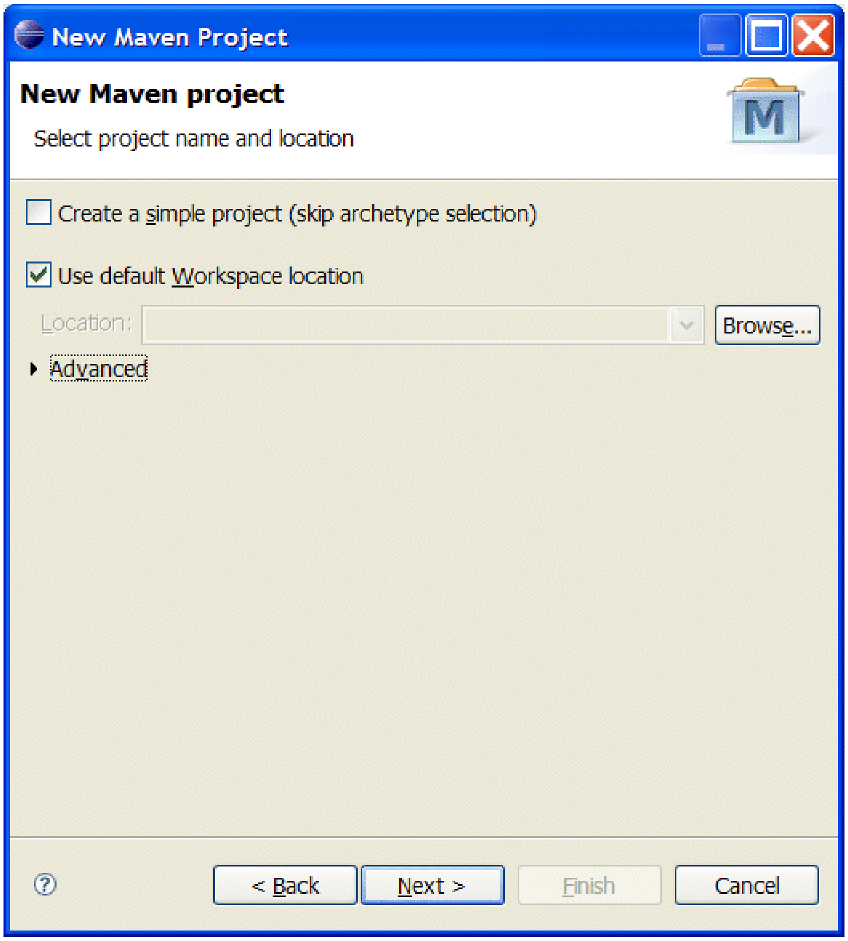
1. Choose the menu item, **File|New|Other**, to open the **Select a wizard** dialog. From the scrollbox, select Maven Project and then click **Next**.

Figure 15. Select a Wizard Dialog



2. The next wizard step allows you to customize the project location, if desired. Click **Next**.

Figure 16. New Maven Project - Project Location


New Maven Project

New Maven project

Select project name and location

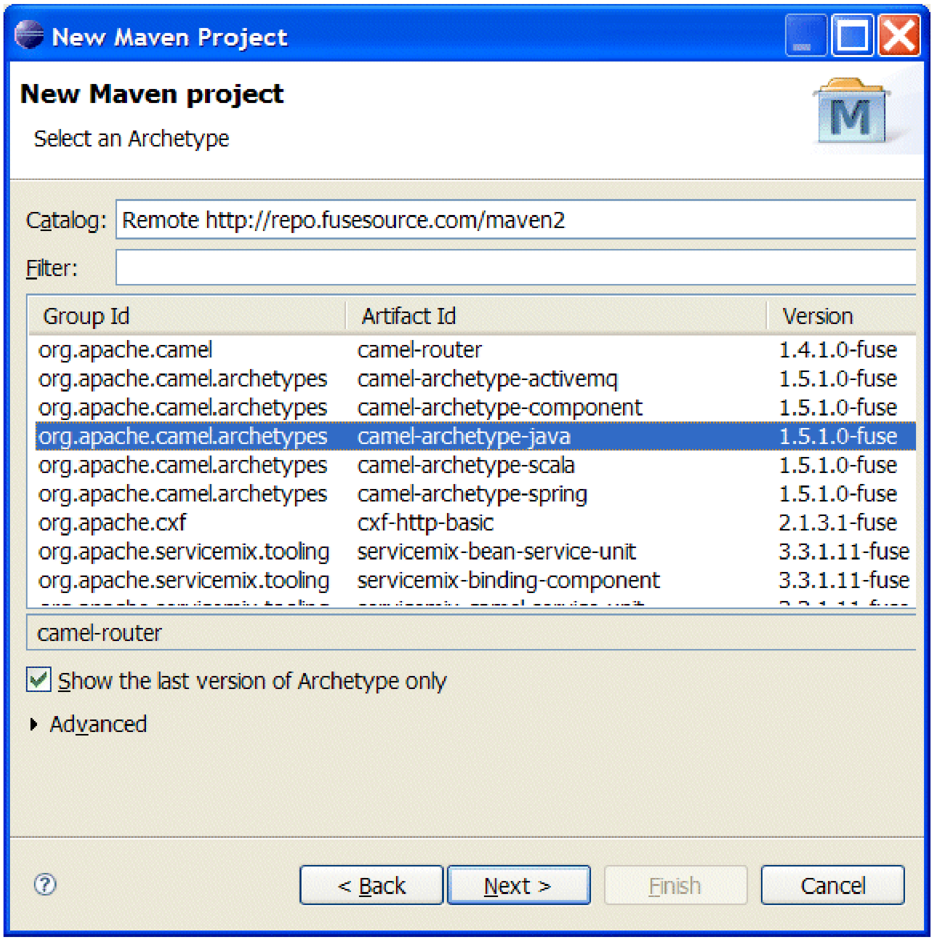
☐ Create a simple project (skip archetype selection)

☑ Use default Workspace location

Location:

3. The next wizard step lets you select the archetype to generate your Maven project. From the **Catalog** drop-down list, select the **FUSE Archetypes** catalog. Then use the scrollbar to locate the `camel-archetype-java` archetype, version `1.5.1.0-fuse`, and select the archetype. Click **Next**.

Figure 17. New Maven Project - Select an Archetype



Catalog: Remote http://repo.fusesource.com/maven2

Filter:

Group Id	Artifact Id	Version
org.apache.camel	camel-router	1.4.1.0-fuse
org.apache.camel.archetypes	camel-archetype-activemq	1.5.1.0-fuse
org.apache.camel.archetypes	camel-archetype-component	1.5.1.0-fuse
org.apache.camel.archetypes	camel-archetype-java	1.5.1.0-fuse
org.apache.camel.archetypes	camel-archetype-scala	1.5.1.0-fuse
org.apache.camel.archetypes	camel-archetype-spring	1.5.1.0-fuse
org.apache.cxf	cxf-http-basic	2.1.3.1-fuse
org.apache.servicemix.tooling	servicemix-bean-service-unit	3.3.1.11-fuse
org.apache.servicemix.tooling	servicemix-binding-component	3.3.1.11-fuse

camel-router

[✓] Show the last version of Archetype only

Advanced

Note

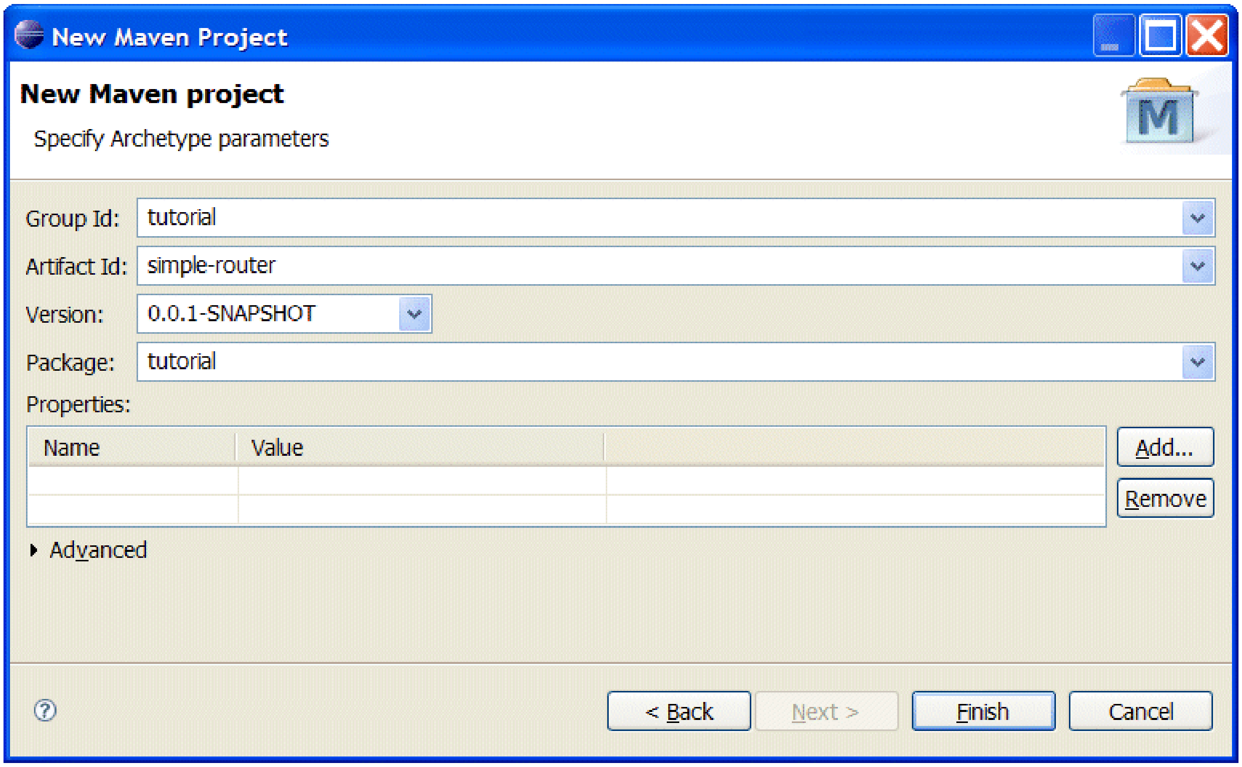
If you cannot find the current version of the archetype in the preceding list, your Maven indexes might be out of date. To update the indexes, right-click inside the **Maven Indexes** view, and select **Update Index**.

Note

If you want to generate an older version of an archetype, uncheck the box beside **Show the last version of Archetype only** to display the *full* list of archetypes in the scroll box.

4. The next wizard step prompts you to supply the basic parameters of the archetype. Enter `tutorial` in the Group Id field, enter `simple-router` in the Artifact Id field, and enter `tutorial` in the Package field. Click **Finish** to generate the `simple-router` project.

Figure 18. New Maven Project - Specify Archetype Parameters



New Maven Project

New Maven project

Specify Archetype parameters

Group Id: tutorial

Artifact Id: simple-router

Version: 0.0.1-SNAPSHOT

Package: tutorial

Properties:

Name	Value

Tutorial: Build and Run the Simple Router

Overview

In this stage of the tutorial, you will learn how to use the Eclipse IDE to build and run the simple router application.

Steps to build and run the simple router

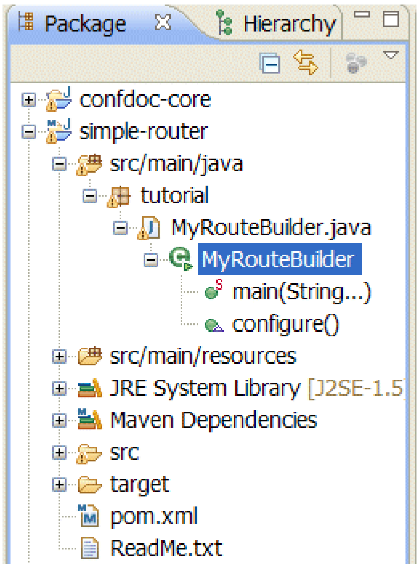
Perform the following steps to build and run the simple router:

1. Normally, you do not have to do anything to build the simple router project. By default, Eclipse is configured to build projects automatically (for example, the `simple-router` project would be built as soon as it is imported into Eclipse).

   If the project is not yet built, however, you can either enable automatic building (select **Project|Build Automatically**) or force the build manually (select **Project|Build All**). A prerequisite for a successful build is that all of the project's Maven dependencies are satisfied (in other words, Maven can locate and download all of the artifacts it needs for this project). If you have trouble with Maven dependencies, see Troubleshooting Maven dependencies on page 40.

2. To view the source code for the simple router application, click on the **Package Explorer** tab on the left of the Eclipse workbench and then drill down to **simple-router|src/main/java|tutorial|MyRouteBuilder.java**.

Figure 19. Package Explorer View of Simple Router Project



Double-click on `MyRouteBuilder.java` to open the Java source file for the simple router.

3. To run the simple router application, right-click `MyRouteBuilder.java` in the **Package Explorer** tab (see Figure 19 on page 39) and select **Run As|Java Application**.

4. When the router starts up, you should see lines like the following appear in the **Console** tab (normally located at the bottom of the Eclipse workbench):

```
28-Aug-2008 11:32:55 org.apache.camel.spring.Main doStart
INFO: Apache Camel 1.5.1.0-fuse starting
28-Aug-2008 11:32:55 org.springframework.context.support.Ab
stractApplicationContext prepareRefresh
...
```

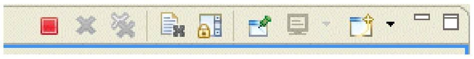
When the router is running, it reads messages from the `*TutorialRoot*/simple-router/src/data` directory and routes them to the `*TutorialRoot*/simple-router/target/messages` directory. To see whether the router is running correctly, check that the messages have arrived under the `target/messages` directory.

Note

If you want to inspect the `target/messages` directory from within the Eclipse environment, you will need to right-click the `target` directory and select **Refresh**.

5. To shut down the simple router, click on the **Console** tab and, to the right of the **Console** tab, look for the icons shown in Figure 20 on page 40 .

   Figure 20. Icons on the Console Tab in Eclipse

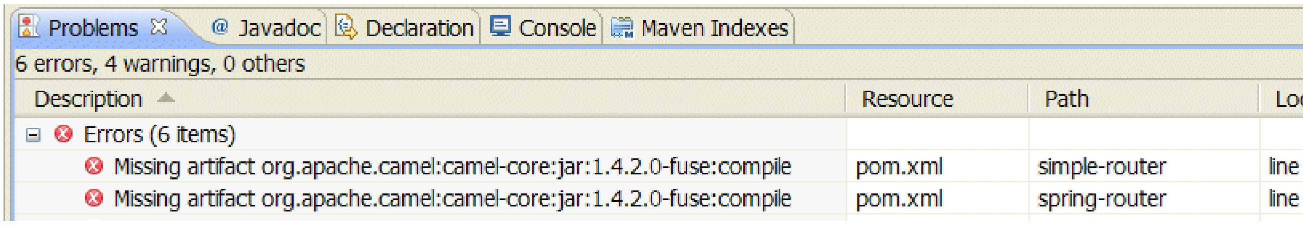
   

   Click on the red square (leftmost icon shown in Figure 20 on page 40) to shut down the router application.

Troubleshooting Maven dependencies

If Maven fails to locate some of the dependencies required for your project, you will see errors like the following, in the **Problems** tab at the bottom of the Eclipse workbench:

Figure 21. Missing Artifact Errors



6 errors, 4 warnings, 0 others

Description	Resource	Path	Lo
Errors (6 items)			
Missing artifact org.apache.camel:camel-core:jar:1.4.2.0-fuse:compile	pom.xml	simple-router	line
Missing artifact org.apache.camel:camel-core:jar:1.4.2.0-fuse:compile	pom.xml	spring-router	line

Missing Maven dependencies give rise to *Missing artifact* errors. The **Type** column clearly identifies this as a Maven problem: to resolve it, check the following points:

1. If you are working behind a HTTP proxy, it is necessary to configure Maven with the proxy details, otherwise Maven will not be able to download

artifacts. For details, see the Maven guide to using proxies [http://maven.apache.org/guides/mini/guide-proxies.html].

2. Make sure that the **Maven Indexes** view list all of the repositories you need. Your list should include at least the repositories, `workspace`, `local`, `central`, `fuse`, and `fuse-snapshot`. The central Maven repository, `central`, contains a huge range of third-party software, but occasionally you might find that you need to add a new repository to access a particular third-party product.

   For instructions on how to add repositories to the Maven Indexes list, see Tutorial: Add Repositories to the Maven Index on page 25.

3. Make sure that the Maven indexes cache is up to date. Right-click inside the **Maven Indexes** view, and select **Update Index**.

4. The repositories must be listed in your project's POM file, `pom.xml`. To check the list of repositories accessible to your project, double-click the `pom.xml` file (see Figure 19 on page 39) to open the file in the POM editor. Click the **Repositories** tab at the bottom of the editor pane to see the project's current repository list. To edit the repository list, click the **pom.xml** tab and edit the XML source directly. Save the `pom.xml` file (Ctrl-S) to make the changes effective.

   ### Note

   In the version of `m2eclipse` tested for this tutorial, the editing features in the **Repositories** tab of the POM editor were not functioning properly. You must edit from the **pom.xml** tab instead.

   The minimum set of repositories required for a Java Router project are:

   ```
   http://repo.fusesource.com/maven2
   http://repo.fusesource.com/maven2-snapshot
   ```

   To include these repositories in your proejct, add the following `repository` elements to the `pom.xml` file:

   ```
    <repositories>
      <repository>
        <id>fusesource.m2</id>
        <name>FUSE Open Source Community Release Reposit
   ory</name>
   ```

```
     <url>http://repo.fusesource.com/maven2</url>
     <snapshots>
       <enabled>false</enabled>
     </snapshots>
     <releases>
       <enabled>true</enabled>
     </releases>
   </repository>
   <repository>
     <id>fusesource.m2-snapshot</id>
     <name>FUSE Open Source Community Snapshot Reposit
ory</name>
     <url>http://repo.fusesource.com/maven2-snapshot</url>

     <snapshots>
       <enabled>true</enabled>
     </snapshots>
     <releases>
       <enabled>false</enabled>
     </releases>
   </repository>
  </repositories>
```

5. You also need to ensure that your project's `pom.xml` file specifies all of the requisite Maven dependencies. The minimum set of dependencies for a Java Router project is:

```
  <dependencies>
    <dependency>
      <groupId>org.apache.camel</groupId>
      <artifactId>camel-core</artifactId>
      <version>${camel-version}</version>
    </dependency>
    <dependency>
      <groupId>org.apache.camel</groupId>
      <artifactId>camel-spring</artifactId>
      <version>${camel-version}</version>
    </dependency>
  </dependencies>
```

   However, a typical project could have many additional dependencies. In particular, most of the Java Router components are packaged as separate Maven artifacts. Each time you add another Java Router component to your project, you will usually need to add a dependency to your `pom.xml` file (the exceptions are the few components that belong to the `camel-core` artifact).

6. When you are finished checking the previous points, select **Project|Update All Maven Dependencies** to download the missing artifacts to your local Maven repository.