



FUSE Mediation Router

FUSE Mediation Router Getting Started

Version 1.3.6.0
May 2008

Making Software Work Together™

FUSE Mediation Router Getting Started

IONA Technologies

Version 1.3.6.0

Published 04 Jun 2008

Copyright © 1999-2007 IONA Technologies PLC

Trademark and Disclaimer Notice

IONA Technologies PLC and/or its subsidiaries may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this publication. Except as expressly provided in any written license agreement from IONA Technologies PLC, the furnishing of this publication does not give you any license to these patents, trademarks, copyrights, or other intellectual property. Any rights not expressly granted herein are reserved.

IONA, IONA Technologies, the IONA logo, Orbix, High Performance Integration, Artix, FUSE, and Making Software Work Together are trademarks or registered trademarks of IONA Technologies PLC and/or its subsidiaries.

Java and J2EE are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. CORBA is a trademark or registered trademark of the Object Management Group, Inc. in the United States and other countries. All other trademarks that appear herein are the property of their respective owners.

While the information in this publication is believed to be accurate, IONA Technologies PLC makes no warranty of any kind to this material including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. IONA shall not be liable for errors contained herein, or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Copyright Notice

No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, photocopying, recording or otherwise, without prior written consent of IONA Technologies PLC. No third-party intellectual property right liability is assumed with respect to the use of the information contained herein. IONA Technologies PLC assumes no responsibility for errors or omissions contained in this publication. This publication and features described herein are subject to change without notice. Portions of this document may include Apache Foundation documentation, all rights reserved.

Table of Contents

Preface	7
Open Source Project Resources	8
Document Conventions	9
Introducing Mediation Router	11
What is Mediation Router?	12
Architecture	13
How to Develop a Router Application	16
FUSE Mediation Router Tutorial	17
Prerequisites	18
Tutorial Overview	20
Tutorial: Create a New Project	22
Tutorial: Set up the Eclipse IDE	24
Tutorial: Build and Run the Simple Router	30

List of Figures

1. Architecture of the Mediation Router	13
2. Overview of the Tutorial	20
3. Eclipse Open Perspective Dialog	25
4. Eclipse Preferences Dialog	26
5. Eclipse New Variable Entry Dialog	26
6. Eclipse Import Wizard (First Step)	28
7. Eclipse Import Wizard (Second Step)	29
8. Package Explorer View of Simple Router Project	31
9. Icons on the Console Tab in Eclipse	32

Preface

Table of Contents

Open Source Project Resources	8
Document Conventions	9

Open Source Project Resources

Apache Incubator CXF

Web site: <http://incubator.apache.org/cxf/>

User's list: <cxf-user@incubator.apache.org>

Apache Tomcat

Web site: <http://tomcat.apache.org/>

User's list: <users@tomcat.apache.org>

Apache ActiveMQ

Web site: <http://activemq.apache.org/>

User's list: <users@@activemq.apache.org>

Apache Camel

Web site:
<http://activemq.apache.org/camel/enterprise-integration-patterns.html>

User's list: <camel-user@activemq.apache.org>

Apache ServiceMix

Web site: <http://servicemix.org/site/home.html>

User's list: <servicemix-users@geronimo.apache.org>

Document Conventions

Typographical conventions

This book uses the following typographical conventions:

<code>fixed width</code>	<p>Fixed width (Courier font) in normal text represents portions of code and literal names of items such as classes, functions, variables, and data structures. For example, text might refer to the <code>javax.xml.ws.Endpoint</code> class.</p> <p>Constant width paragraphs represent code examples or information a system displays on the screen. For example:</p> <pre>import java.util.logging.Logger;</pre>
<i>Fixed width italic</i>	<p>Fixed width italic words or characters in code and commands represent variable values you must supply, such as arguments to commands or path names for your particular system. For example:</p> <pre>% cd /users/YourUserName</pre>
<i>Italic</i>	Italic words in normal text represent <i>emphasis</i> and introduce <i>new terms</i> .
Bold	Bold words in normal text represent graphical user interface components such as menu commands and dialog boxes. For example: the User Preferences dialog.

Keying conventions






This book uses the following keying conventions:

No prompt	When a command's format is the same for multiple platforms, the command prompt is not shown.
%	A percent sign represents the UNIX command shell prompt for a command that does not require root privileges.
#	A number sign represents the UNIX command shell prompt for a command that requires root privileges.
>	The notation > represents the MS-DOS or Windows command prompt.
...	Horizontal or vertical ellipses in format and syntax descriptions indicate that material has been eliminated to simplify a discussion.
[]	Brackets enclose optional items in format and syntax descriptions.
{ }	Braces enclose a list from which you must choose an item in format and syntax descriptions.

	In format and syntax descriptions, a vertical bar separates items in a list of choices enclosed in {} (braces).
--	-----------------------------------------------------------------------------------------------------------------

Admonition conventions

This book uses the following conventions for admonitions:

	Notes display information that may be useful, but not critical.
	Tips provide hints about completing a task or using a tool. They may also provide information about workarounds to possible problems.
	Important notes display information that is critical to the task at hand.
	Cautions display information about likely errors that can be encountered. These errors are unlikely to cause damage to your data or your systems.
	Warnings display information about errors that may cause damage to your systems. Possible damage from these errors include system failures and loss of data.

Introducing Mediation Router

Summary

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

Table of Contents

What is Mediation Router?	12
Architecture	13
How to Develop a Router Application	16

What is Mediation Router?

Overview

Mediation Router, IONA Technologies' distribution of Apache Camel, is a rule-based routing and process mediation engine that enables you to implement enterprise integration patterns using plain old Java objects (POJOs).

Mediation Router uses generics, annotations and URIs so that it can easily work directly with any kind of transport or messaging model. It is also a small library which has minimal dependencies for easy embedding in any Java application.

You can use Mediation Router as the routing and mediation engine for the following FUSE products:

- FUSE ESB (Apache ServiceMix)
- FUSE Message Broker (Apache ActiveMQ)
- FUSE Services Framework (Apache Incubator CXF)

In addition, you can use it on a standalone basis or in any Spring Framework-hosted application.

Enterprise integration patterns

Enterprise integration patterns are defined in a book of the same name by Gregor Hohpe and Bobby Woolf. The book describes a number of design patterns for the use of enterprise application integration and message-oriented middleware. For more details see Enterprise Integration Patterns [<http://www.enterpriseintegrationpatterns.com>] and the Apache Camel Enterprise Integration Patterns Guide [<http://activemq.apache.org/camel/enterprise-integration-patterns.html>].

Domain specific language

Mediation Router uses a Java domain specific language (DSL) to configure routing and mediation rules. A DSL, also referred to as a *fluent API*, is a programming language designed for a specific kind of task.

The DSL means that Mediation Router can support type-safe smart completion of routing rules in your IDE using regular Java code without you having to use extensive XML configuration.

Spring Framework XML Configuration

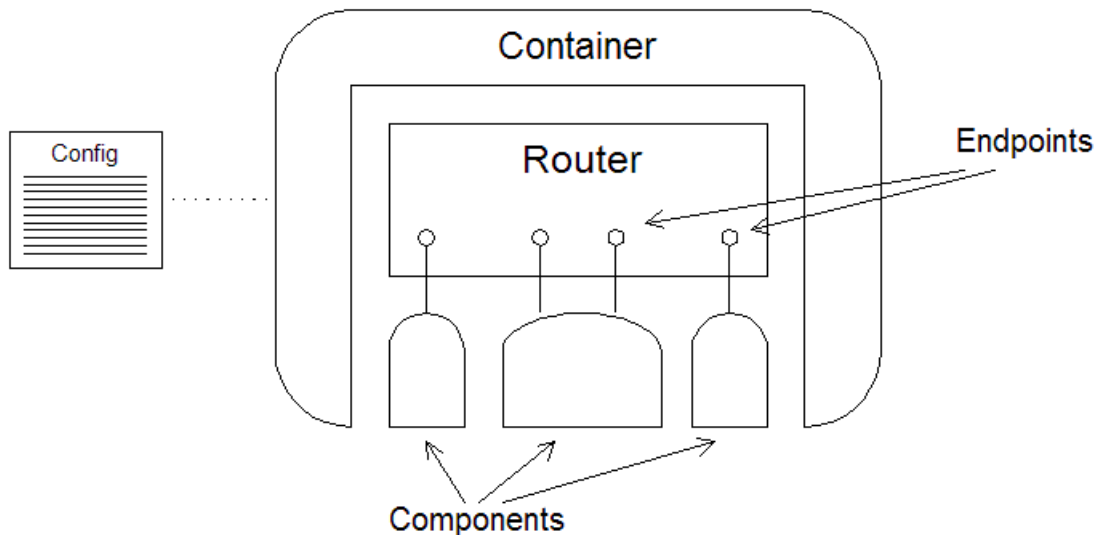
As an alternative to creating routes using the Java DSL, you can define routing rules with a Spring Framework XML configuration file. While the Java-based DSL is a more concise way of creating routes, the Spring XML approach does not require you to recompile your code should the routing information change.

Architecture

Overview

Figure 1, “Architecture of the Mediation Router” gives a general overview of the Mediation 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 Mediation 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.

FUSE Mediation Router Tutorial

Summary

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

Table of Contents

Prerequisites	18
Tutorial Overview	20
Tutorial: Create a New Project	22
Tutorial: Set up the Eclipse IDE	24
Tutorial: Build and Run the Simple Router	30

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 .
 - Apache Maven 2 .
 - Eclipse .
-

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.

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

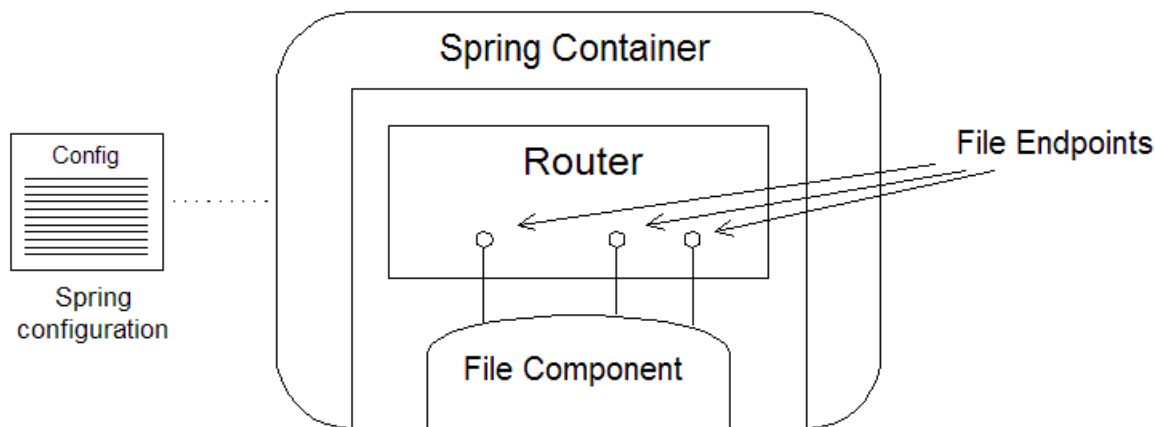
- Add the Eclipse root directory to your `PATH` (the `eclipse` binary is located in the root directory).

Tutorial Overview

Overview

Figure 2, “Overview of the Tutorial” 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, “Overview of the Tutorial” 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 (see Spring [<http://www.springframework.org/>]) that implements sophisticated configuration mechanisms (for example, supporting concepts such as *dependency injection* and *reversion of control*).
- *Spring configuration file*—by default, the Spring wrapper looks for the Spring configuration file, `META-INF/spring/camel-context.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: Create a New Project .
2. Tutorial: Set up the Eclipse IDE .
3. Tutorial: Build and Run the Simple Router .

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 FUSE Mediation Router.

Steps to create a new project

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

1. Create a new directory, *TutorialRoot*, in a convenient location, to hold the generated tutorial code.
2. Open a command window and change directory to *TutorialRoot*. To generate the `simple-router` project, enter the following command:

```
mvn archetype:create -DarchetypeGroupId=org.apache.camel -  
DarchetypeArtifactId=camel-router -DarchetypeVersion=1.2.0  
-DgroupId=tutorial -DartifactId=simple-router
```



Note

At this point, Maven requires access to the Internet in order to download the necessary JAR files into its local repository.

3. After the previous command has finished executing, change directory to *TutorialRoot/simple-router*. To modify the project's `camel-version`, use your favorite text editor to open the `pom.xml` file. Search for the following lines in `pom.xml`:

```
...  
<properties>  
  <camel-version>1.1-SNAPSHOT</camel-version>  
</properties>  
...
```

Change the contents of the `camel-version` element to `1.3.6.0-fuse`. For example:

```
...  
<properties>  
  <camel-version>1.3.6.0-fuse</camel-version>
```

```
</properties>  
...
```

4. Copy the list of IONA Maven repositories into the `simple-router/pom.xml` file. Open the POM file from the FUSE Mediation Router example directory, `InstallDir/examples/pom.xml`, using your favorite text editor. Copy the entire `repositories` element and the `pluginRepositories` element from the examples POM file. For example, you would copy all of the contents shown below:

```
<repositories>  
  ...  
</repositories>  
  
<pluginRepositories>  
  ...  
</pluginRepositories>
```

Now open the `simple-router/pom.xml` file using your text editor and paste the `repositories` and `pluginRepositories` elements into the `project` element. Save the `pom.xml` file.

Tutorial: Set up the Eclipse IDE

Overview

In this stage of the tutorial, you will set up the Eclipse Integrated Development Environment (IDE), so that you can import the simple router project into Eclipse. Eclipse provides a convenient environment for developing, building, and running FUSE Message Broker projects.

Steps to set up Eclipse

Perform the following steps to set up the Eclipse IDE, so that you can start working on the simple-router project:

1. Open a new command window and change directory to the `TutorialRoot/simple-router` directory.
2. Run the Maven `eclipse` plug-in in order to generate the files you need to work in Eclipse. To run the plug-in, enter the following command in the command window:

```
mvn eclipse:eclipse
```

If the command runs successfully, you should see two new files in the `simple-router` directory, `.project` and `.classpath`, and a new sub-directory, `.settings`.

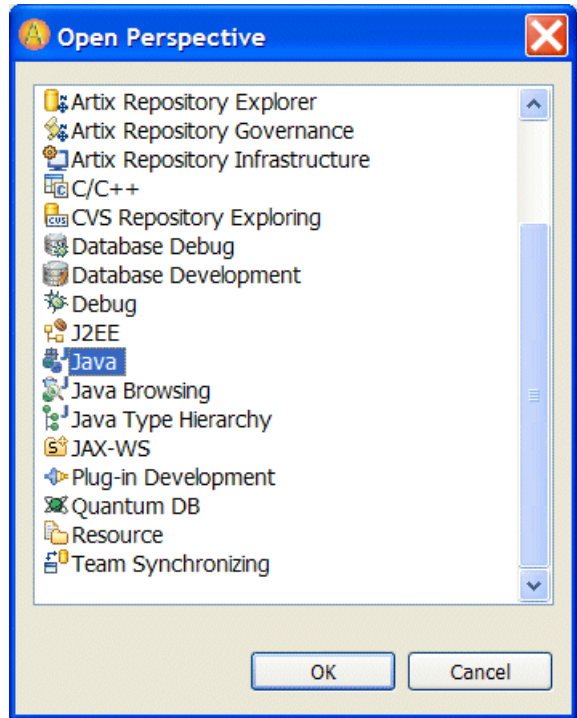
3. To start the Eclipse IDE, enter the following command:

```
eclipse
```

During start-up, Eclipse might present you with a dialog that asks you to specify a workspace directory. It is a good idea to choose a dedicated workspace directory for your FUSE projects.

4. To open the Java perspective in the Eclipse workbench, choose the menu item, `Window | Open Perspective | Other`.

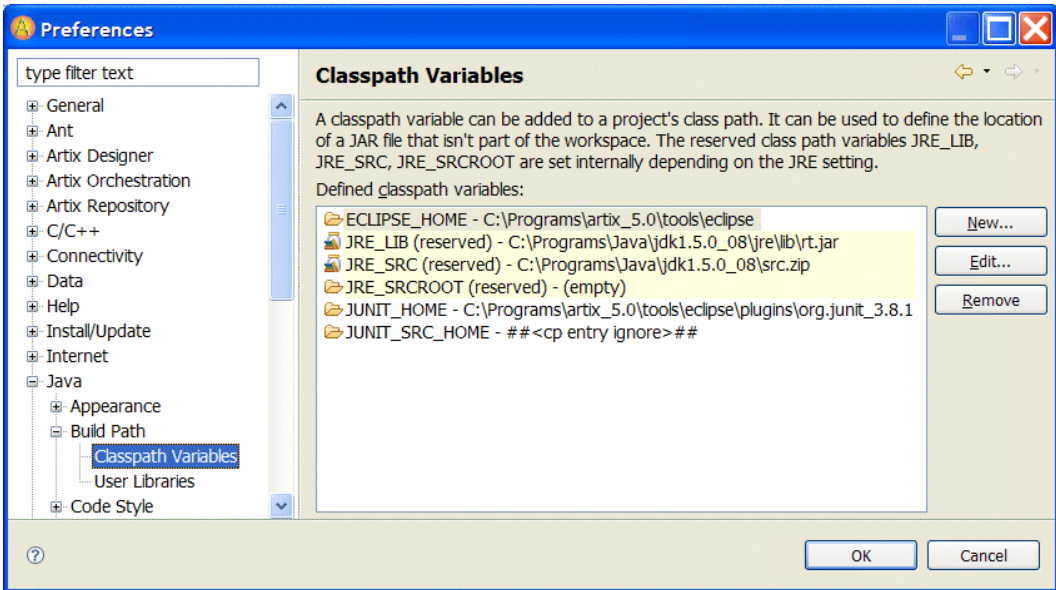
Figure 3. Eclipse Open Perspective Dialog



From the Open Perspective dialog, select Java and click Ok.

5. You need to define a classpath variable for the Maven repository, so that Eclipse knows where to find the JAR files that Maven has downloaded. Select the menu item, Window|Preferences.

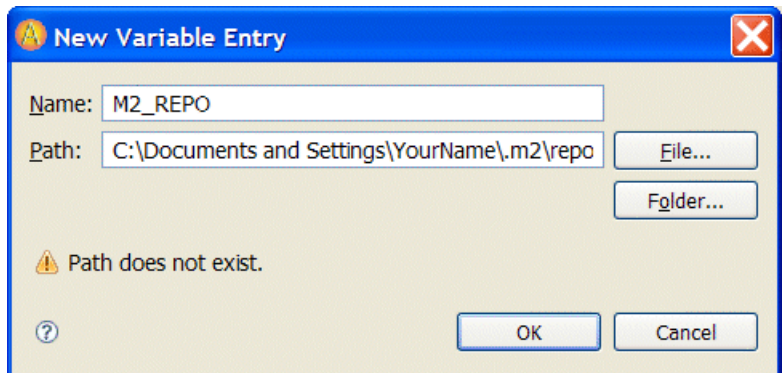
Figure 4. Eclipse Preferences Dialog



In the left-hand pane of the Preferences dialog, drill down to Java|Build Path|Classpath Variables.

6. In the Preferences dialog, click New to open the New Variable Entry dialog.

Figure 5. Eclipse New Variable Entry Dialog



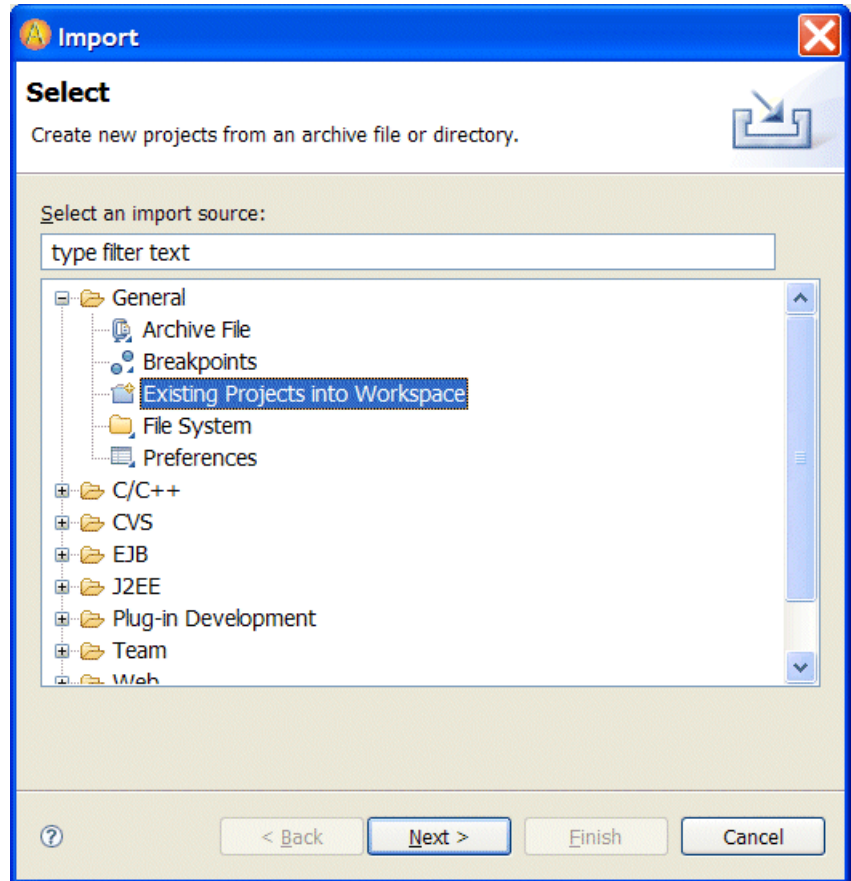
Enter `M2_REPO` in the Name field. In the Path field, you must specify the location of the Maven repository, which depends on the operating system, as follows:

- *Windows*—repository is in `C:\Documents and Settings\UserName\.m2\repository`.
- *UNIX*—repository is in `~/.m2/repository`.

Click Ok to confirm the new classpath variable.

7. Now you are ready to import the `simple-router` project into the Eclipse IDE. Select the menu item, File|Import.

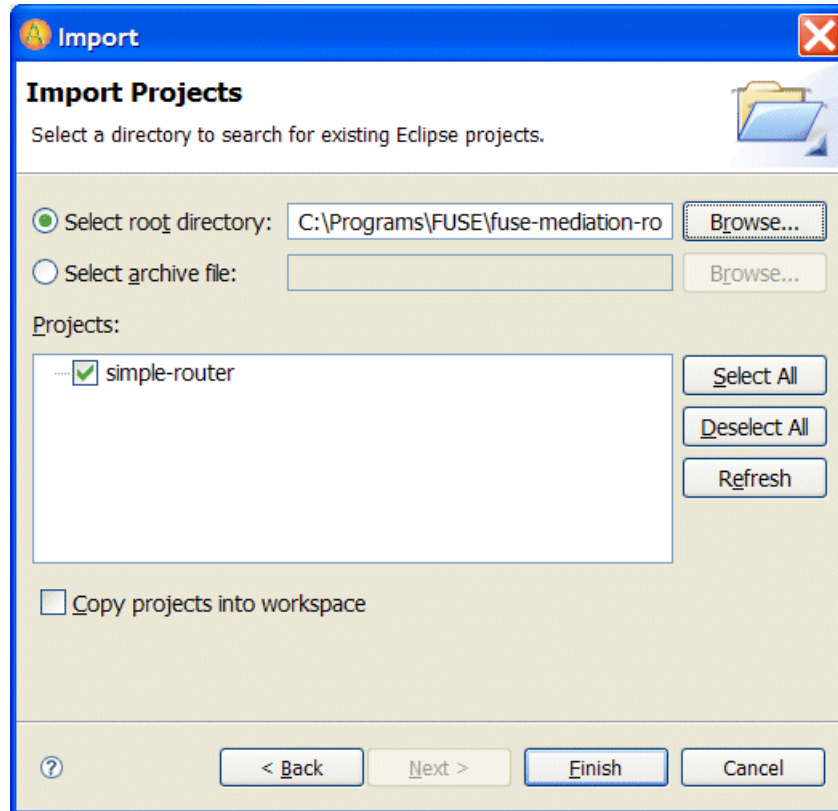
Figure 6. Eclipse Import Wizard (First Step)



From the Import wizard, drill down to General | Existing Projects into Workspace. Click Next.

8. In the second step of the Import wizard (see Figure 7, "Eclipse Import Wizard (Second Step)"), use the Browse button to specify the location of the `simple-router` directory.

Figure 7. Eclipse Import Wizard (Second Step)



Click Finish to import the `simple-router` project into Eclipse.

Tutorial: Build and Run the Simple Router

Overview

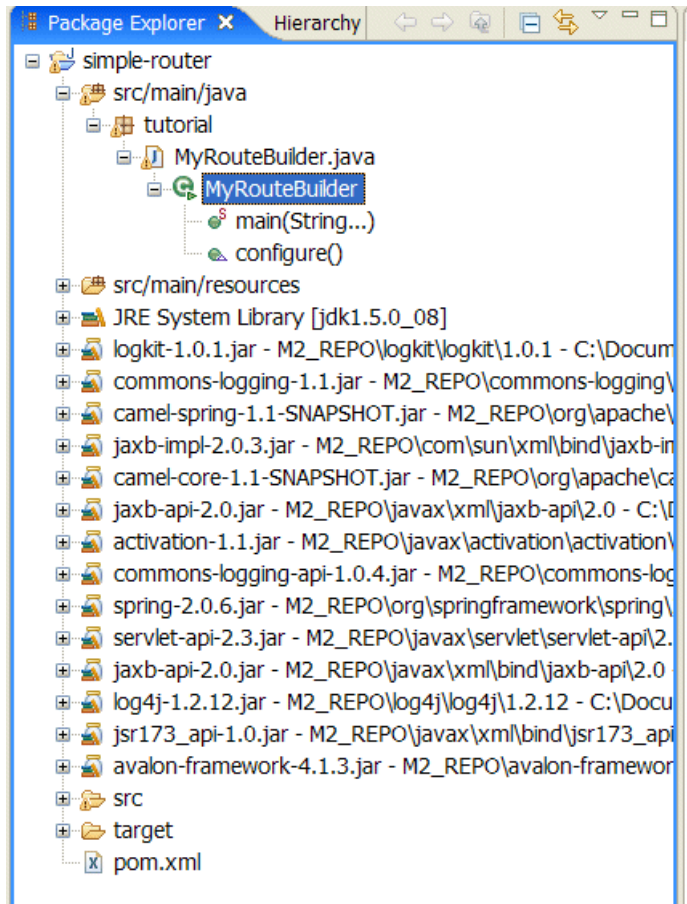
In this stage of the tutorial, you will learn how to use the Eclipse IDE to build and run the simple router application.

Steps to build and run the simple router

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

1. 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 8. Package Explorer View of Simple Router Project



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

2. 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).

3. To run the simple router application, right-click `MyRouteBuilder.java` in the Package Explorer tab (see Figure 8, “Package Explorer View of Simple Router Project”) and select Run As|Java Application.
4. When the router starts up, you should see the following line appear in the Console tab (normally located at the bottom of the Eclipse workbench):

```
[           main] Main
      INFO
Apache Camel 1.3.6.0-fuse starting
```

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.

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 9, “Icons on the Console Tab in Eclipse” .

Figure 9. Icons on the Console Tab in Eclipse



Click on the red square (leftmost icon shown in Figure 9, “Icons on the Console Tab in Eclipse”) to shut down the router application.