Progress FUSE

# FUSE Mediation Router

## Getting Started

Version 1.5
November 2008



PROGRESS SOFTWARE

# Getting Started

Version 1.5

Publication date 10 Mar 2009


***Legal Notices***

# Table of Contents

# List of Figures

# Introducing Mediation Router

*This chapter introduces the Mediation Router architecture and other basic concepts.*

# What is Mediation Router?

**Overview**

Mediation Router is an integration framework and runtime environment. It enables you to quickly define and implement *routes* declarative solutions for specific integration problems. A route defines an integration path between two or more endpoints: a path from an input source to one or more output destinations. Each endpoint in a route whether an input source or output destination is identified by a URL. Mediation Router supports wide variety of endpoint types (and URLs).

Mediation Router works seamlessly with the following FUSE products:

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

You also can use Mediation Router on a standalone basis or in any Spring Framework-hosted application.

**Enterprise integration patterns**

Using routes, you can easily implement enterprise integration patterns (EIPs) using plain old Java objects (POJOs). 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[1] and the Apache Camel Enterprise Integration Patterns Guide[2].

**Domain specific language**

You can use 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

[1] http://www.enterpriseintegrationpatterns.com
[2] http://activemq.apache.org/camel/enterprise-integration-patterns.html

extensive XML configuration. If you change your DSL rules, you must recompile your Java sources.

**Spring Framework XML Configuration**

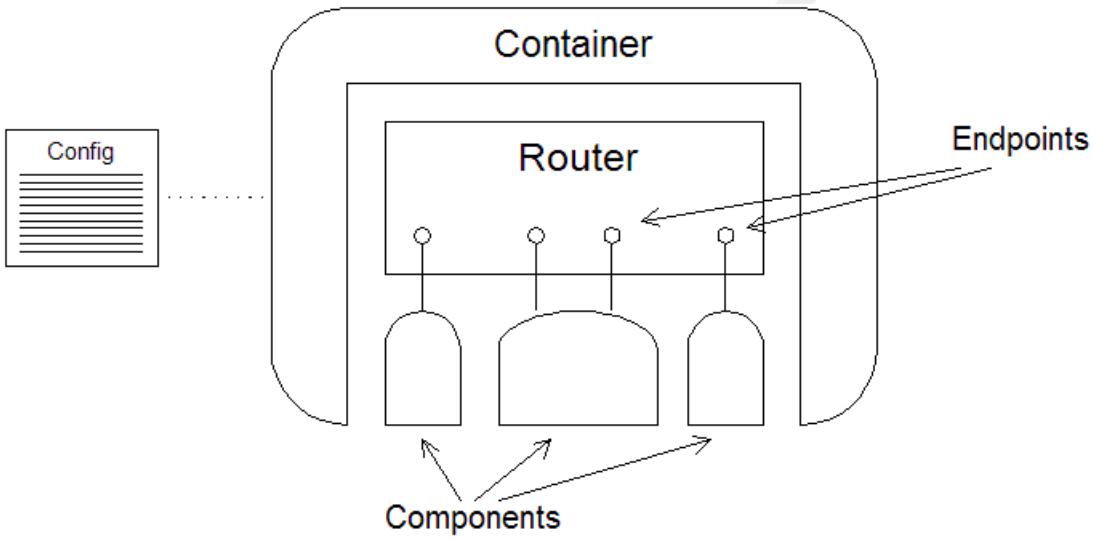
You also can define routing rules with a Spring Framework XML configuration file (which requires no recompilation).

# Architecture

## Overview

Figure 1 on page 10 gives a general overview of the Mediation Router architecture; this architecture enables you to deploy across a wide variety of 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[3].

**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 supports these deployment options:

- *Spring container deployment* You deploy the router application into a Spring container. For this type of deployment, you can use the Spring configuration file to configure components and define routes.
- *Standalone deployment* You 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.



[3] http://activemq.apache.org/camel/components.html

# How to Develop a Router Application

**Outline of the development steps**

To develop a router application, you perform the following high-level steps:

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

*This tutorial describes how to use Apache Maven to create, build, and run a basic router.*

# Prerequisites

## Overview

To follow the steps 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 .

## Java Runtime

Install a suitable Java runtime (JDK 1.5.x). For more information, see the *Install Guide*. After installing, set your `JAVA_HOME` environment variable to point to the root directory of your Java runtime; also set your `PATH` environment variable to include the Java `bin` directory.

## Apache Maven 2

Download and install Apache Maven[1] (2.x), a general purpose tool for building and packaging applications. To download, visit: http://maven.apache.org/download.html.

After installing Maven:

- Set your `M2_HOME` environment variable to point to the Maven root directory.
- Set your `MAVEN_OPTS` environment variable to `-Xmx512M`. (This setting increases memory available for Maven builds.)
- Set your `PATH` environment variable to include the Maven `bin` directory (`%M2_HOME%\bin` on Windows or `$M2_HOME/bin` on UNIX).

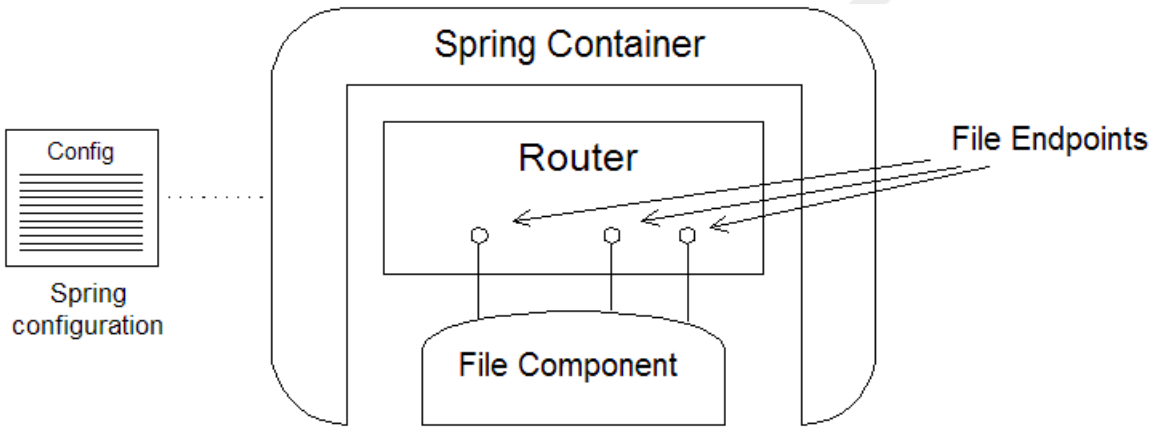
[1] http://maven.apache.org/

# Tutorial Overview

**Overview**

Figure 2 on page 17 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 17 consists of the following parts:

- *Router*—the core component of the simple router. It consists of an instance of type `org.apache.camel.builder.RouteBuilder`, 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[2]) 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.

[2] http://www.springframework.org/

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. Stage 1: Create a New Project on page 19.
2. Stage 2: Examining the Sample Code on page 21.
3. Stage 3: Build and Run the Sample Project on page 23.

# Stage 1: Create a New Project

**Overview**

In this stage, you create a Maven project (`simple-router`) that contains a sample application.

**Steps**

To create the project:

1. Create a new directory, *`ProjectRoot`*.
2. In a command window, change to the *`ProjectRoot`* directory.
3. Enter the following command to create the `simple-router` project:

```
mvn archetype:create
-DremoteRepositories=http://repo.open.iona.com/maven2
-DarchetypeGroupId=org.apache.camel.archetypes
-DarchetypeArtifactId=camel-archetype-java
-DarchetypeVersion=1.5.4.0-fuse
-DgroupId=tutorial
-DartifactId=simple-router
```

**Note**

Maven accesses the Internet to download JARs and store them in its local repository.

As a result of this command, Maven creates the following directories and files:

```
ProjectRoot/simple-router
ProjectRoot/simple-router/pom.xml ❶
ProjectRoot/simple-router/ReadMe.txt
ProjectRoot/simple-router/src
ProjectRoot/simple-router/src/data
ProjectRoot/simple-router/src/data/message1.xml ❷
ProjectRoot/simple-router/src/data/message2.xml ❸
ProjectRoot/simple-router/src/main
ProjectRoot/simple-router/src/main/java
ProjectRoot/simple-router/src/main/java/tutorial
ProjectRoot/simple-router/src/main/java/tutorial/MyRouteBuilder.java ❹
ProjectRoot/simple-router/src/main/resources
ProjectRoot/simple-router/src/main/resources/log4j.properties
ProjectRoot/simple-router/src/main/resources/META-INF
ProjectRoot/simple-router/src/main/resources/META-INF/spring
```

```
ProjectRoot/simple-router/src/main/resources/META-INF/spring/camel-context.xml ❺
```

Some of the project artifacts are described below:

❶ `pom.xml` is a Maven project file.

❷ `message1.xml` is an input message in XML format.

❸ `message2.xml` is an input message in XML format.

❹ `MyRouteBuilder.java` is a Java source file that implements the sample route.

❺ `camel-context.xml` is a Spring configuration file.

# Stage 2: Examining the Sample Code

**Overview**

In this stage, you examine the sample code in *ProjectRoot*/simple-router/src/main/java/tutorial/MyRouteBuilder.java.

This sample implements a typical EIP pattern, a content-based router, and illustrates how easily and concisely you can solve integration problems using Mediation Router.

**Sample Code**

The following code shows how the sample route is implemented:

```
...
package tutorial; ❶

import org.apache.camel.Exchange;
import org.apache.camel.Processor;
import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.spring.Main;

import static org.apache.camel.builder.xml.XPathBuilder.xpath;

/**
* A Camel Router
*
* @version $
*/
public class MyRouteBuilder extends RouteBuilder {

     /**
     * A main() so we can easily run these routing rules in our IDE
     */
     public static void main(String... args) {
  Main.main(args);
     }

     /**
     * Lets configure the Camel routing rules using Java code...
     */
     public void configure() {

  // TODO create Camel routes here.

  // here is a sample which processes the input files
  // (leaving them in place - see the 'noop' flag)
  // then performs content based routing on the message
  // using XPath
```

```
  from("file:src/data?noop=true"). ❷
      choice().
      when(xpath("/person/city = 'London'")).to("file:target/messages/uk"). ❸
  otherwise().to("file:target/messages/others");

     }
}
```

This sample has several notable features. First, observe how the Java DSL uses a series of method calls to create an English-like expression. This technique makes the intent of the code clear: the sample reads input messages from a directory, applies an XPath expression to each message's XML content, and, based on the result, chooses a different route for the output messages.

Note also the following:

❶ Maven automatically creates a package name based on the value of the `-DgroupId` argument to the `mvn archetype:create` command.

❷ The `from()` method call takes a file URL as its argument. This URL provides information that Mediation Router uses to interpret and execute the route. The `file:` prefix in the URL indicates that a file endpoint is required, which means the file component is responsible for creating the endpoint. (The file component, like other Mediation Router components, serves as an endpoint factory.) The option in the URL, `?noop=true`, indicates that the files in `src/data` should be left in place and not consumed. (This option is one of many available; like other components, the file component provides numerous options.)

❸ The `when()` method call specifies an XPath expression, which is applied to each input. If the expression evaluates to true, output is routed to the `uk` subdirectory; if false, to the `others` subdirectory.

# Stage 3: Build and Run the Sample Project

**Overview**

In this stage, you use Maven to build and run the sample project.

**Steps**

To build and run the sample project:

1. In a command window, change to the *ProjectRoot*/simple-router directory.
2. Enter the following command to build the project:

```
mvn install
```

Maven builds the project and creates a target directory for the build artifacts:

```
ProjectRoot/simple-router/target
ProjectRoot/simple-router/target/simple-router-1.0-SNAPSHOT.jar ❶
ProjectRoot/simple-router/target/classes
ProjectRoot/simple-router/target/classes/log4j.properties  ❷
ProjectRoot/simple-router/target/classes/META-INF
ProjectRoot/simple-router/target/classes/META-INF/spring
ProjectRoot/simple-router/target/classes/META-INF/spring/camel-context.xml
ProjectRoot/simple-router/target/classes/tutorial
ProjectRoot/simple-router/target/classes/tutorial/MyRouteBuilder.class  ❸
ProjectRoot/simple-router/target/maven-archiver
ProjectRoot/simple-router/target/maven-archiver/pom.properties
```

Some of the project artifacts are described below:

❶ simple-router-1.0-SNAPSHOT.jar is the deployment JAR.

❷ log4j.properties is a properties file used to control logging levels.

❸ MyRouteBuilder.class is the class file compiled from MyRouteBuilder.java.

3. Enter the following command to run the project:

```
mvn camel:run
```

When FUSE Mediation Router starts, it prints lines like the following to the console:

```
23-Feb-2009 16:51:04 org.apache.camel.spring.Main doStart
INFO: Apache Camel 1.5.4.0-fuse starting
23-Feb-2009 16:51:04 org.springframework.context.support.Ab
stractApplicationContext prepareRefresh
...
```

The sample application runs until you stop it. It routes messages from *ProjectRoot*/simple-router/src/data to either *ProjectRoot*/simple-router/target/messages/uk or *ProjectRoot*/simple-router/target/messages/others.