



# **Red Hat JBoss Fuse 6.1**

## **Tooling Tutorials**

Building Solutions with Red Hat JBoss Fuse Tooling



# Red Hat JBoss Fuse 6.1 Tooling Tutorials

---

Building Solutions with Red Hat JBoss Fuse Tooling

JBoss Fuse Tooling Docs Team

Content Services

[fuse-docs-support@redhat.com](mailto:fuse-docs-support@redhat.com)

## Legal Notice

Copyright © 2014 Red Hat.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution-Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## Abstract

This guide contains a number of simple tutorials that demonstrate how to use the tooling provided by Red Hat JBoss Fuse ToolingRed Hat JBoss Fuse IDE to develop and test applications.

---

## Table of Contents

<b>ABOUT RED HAT JBOSS FUSE TOOLING</b> .....	<b>4</b>
<b>CHAPTER 1. TO CREATE A NEW ROUTE</b> .....	<b>6</b>
GOALS	6
CREATING THE FUSE PROJECT	6
CREATING THE ROUTE	9
NEXT STEPS	12
FURTHER READING	12
<b>CHAPTER 2. TO RUN A ROUTE</b> .....	<b>14</b>
GOALS	14
PREREQUISITES	14
RUNNING THE ROUTE	14
VERIFYING THE ROUTE	15
FURTHER READING	16
<b>CHAPTER 3. TO ADD A CONTENT-BASED ROUTER</b> .....	<b>17</b>
GOALS	17
PREREQUISITES	17
PROCEDURE	17
NEXT STEPS	20
FURTHER READING	20
<b>CHAPTER 4. TO TRACE A MESSAGE THROUGH A ROUTE</b> .....	<b>21</b>
GOALS	21
PREREQUISITES	21
PROCEDURE	21
NEXT STEPS	24
FURTHER READING	24
<b>CHAPTER 5. TO TEST A ROUTE WITH JUNIT</b> .....	<b>26</b>
OVERVIEW	26
GOALS	26
PREREQUISITES	26
CREATING THE TEST CASE	26
RUNNING THE TEST	28
FURTHER READING	29
<b>CHAPTER 6. TO DEPLOY A CAMEL PROJECT TO RED HAT JBOSS FUSE</b> .....	<b>30</b>
GOALS	30
PREREQUISITES	30
STARTING UP RED HAT JBOSS FUSE 6.X SERVER	30
DEPLOYING A PROJECT TO RED HAT JBOSS FUSE	33
UNINSTALLING YOUR PROJECT'S BUNDLE	34
FURTHER READING	34
<b>CHAPTER 7. TO DEPLOY A CAMEL PROJECT TO A FABRIC PROFILE</b> .....	<b>35</b>
GOALS	35
PREREQUISITES	35
CREATING A FABRIC IN RED HAT JBOSS FUSE	35
CONNECTING TO THE JBOSS FUSE SERVER	36
CONNECTING TO THE FABRIC	37
CREATING A NEW FABRIC PROFILE	37

DEPLOYING YOUR CAMEL PROJECT TO THE NEW FABRIC PROFILE	38
FURTHER READING	39



## ABOUT RED HAT JBOSS FUSE TOOLING

Red Hat JBoss Fuse Tooling provides a set of developer tools that enable you to work with Fuse and Apache versions of ActiveMQ, Camel, CXF, Karaf, and ServiceMix within Red Hat JBoss Development Studio. You can connect and configure Enterprise Integration Patterns to build routes, browse endpoints and routes, drag and drop messages onto running routes, trace message flows, edit running routes, browse and visualize runtime processes via JMX, and deploy your project's code to Red Hat JBoss Fuse and Fabric8 containers, to Apache ServiceMix, and to Apache Karaf.

The Red Hat JBoss Fuse Tooling consists of three feature plugins that you can install separately (via the Eclipse software install mechanism), which enables you to select only the features you need:

- JBoss Fuse Camel Editor Feature (`org.fusesource.ide.camel.editor.feature`)

The JBoss Fuse Camel Editor Feature is the base building block for the remaining features that make up JBoss Fuse Tooling. It provides the tools for creating a Fuse project, including the route editor, with its palette of supported Enterprise Integration Patterns, and the logic for running camel contexts inside the editor.

If you want only to create, test, and visualize routes, but not to debug them, you need install only the JBoss Fuse Camel Editor Feature.



### NOTE

Red Hat JBoss Fuse Service Works tooling uses the JBoss Fuse Camel Editor Feature only.

*Part I, Developing Applications* of Using the JBoss Fuse Tooling describes the functionality that this feature provides.

- JBoss Fuse Runtimes Feature (`org.fusesource.ide.runtimes.feature`)

This feature extends the functionality of the JBoss Fuse Camel Editor Feature, allowing you to debug and monitor camel routes running in JMX-connected containers or as local processes, and to deploy camel routes to JMX-connected containers and Fabric8 containers. It includes the JMX and Fabric8 tooling.

If you want to debug and deploy the routes that you created with JBoss Fuse Camel Editor Feature, you also need to install the JBoss Fuse Runtimes Feature.

*Part II, Debugging Applications* and *Part III, Working with Fabric* of Using the JBoss Fuse Tooling describe how to use the functionality this feature provides.

- JBoss Fuse Server Extensions Feature (`org.fusesource.ide.server.extensions.feature`)

This feature extends the functionality of the JBoss Fuse Camel Editor and the JBoss Fuse Runtimes Feature features to allow you to configure, run, and interact with any of the supported servers via a Karaf command shell inside JBoss Fuse Tooling. For example, with JBoss Fuse installed on your machine, you can start it up and then create a fabric in which to deploy the camel routes you created with the JBoss Fuse Camel Editor Feature.

If you want to run and interact directly with any of the supported servers from inside the JBoss Fuse Tooling, you also need to install the JBoss Fuse Server Extensions Feature.



*Managing servers in Part II, Debugging Applications* of the Using the JBoss Fuse Tooling guide describes how to use the functionality this feature group provides.

Using Red Hat JBoss Fuse Tooling simplifies and streamlines the process of developing integration applications by providing tooling that is specifically designed to work with:

- Red Hat JBoss Fuse
- Red Hat JBoss A-MQ
- Red Hat JBoss Fuse Service Works (works with JBoss Fuse Camel Editor Feature only)
- Apache Camel
- Apache CXF
- Apache Karaf

Using the Red Hat JBoss Fuse Tooling streamlines the process at all stages of application development:

1. Create a Maven project for your application.

The tooling loads all of the relevant Maven archetypes for creating integration projects using the Red Hat supported Apache projects.

2. Add new pieces of logic and functionality to an application.

The tooling has a wizard for creating Apache Camel context files.

3. Edit the integration logic.

The tooling has a visual route editor that makes editing Apache Camel routes as easy as dragging and dropping route components.

4. Test the application.

The tooling includes testing tools that provide the full gamut of testing capabilities including:

- creating JUnit test cases for Apache Camel routes
- JMX analysis of running components
- message tracing through Apache Camel routes

5. Deploy the application.

The tooling can deploy applications to a number of containers.

# CHAPTER 1. TO CREATE A NEW ROUTE

## Abstract

This tutorial walks you through the process of creating a new Fuse project, adding a route to it, and adding two endpoints to the route. It assumes that you have already set up your workspace and that Red Hat JBoss Fuse Tooling is running inside Red Hat JBoss Developer Studio.

## GOALS

In this tutorial you will:

- create a routing project
- add endpoints to a route
- connect two endpoints
- configure the endpoints

## CREATING THE FUSE PROJECT

To create a Fuse project:


1. Open the JBoss perspective.

This is the default perspective when you start up JBoss Developer Studio for the first time.

2. On the Toolbar, select **File** → **New** → **Fuse Project** to open the **New Fuse project** wizard, as shown in [Figure 1.1, “New Fuse project location page”](#) .

Figure 1.1. New Fuse project location page

**New Fuse project**  
Select project location



Use default Workspace location

Location:

Add project(s) to working set


Working set:

3. Click **Next>** to open the **New Fuse Project** details page, as shown in [Figure 1.2, “New Fuse project details page”](#).

Figure 1.2. New Fuse project details page

**New Fuse project**

Select a project archetype to create and specify its details



Filter:

Group ID	Artifact ID	Version
io.fabric8	camel-cxf-code-first-archetype	1.0.0.redhat-312
io.fabric8	camel-cxf-contract-first-archetype	1.0.0.redhat-312
io.fabric8	camel-drools-archetype	1.0.0.redhat-312
io.fabric8	camel-webservice-archetype	1.0.0.redhat-312
org.apache.camel.archetypes	camel-archetype-activemq	2.12.0.redhat-610312
org.apache.camel.archetypes	camel-archetype-blueprint	2.12.0.redhat-610312
org.apache.camel.archetypes	camel-archetype-component	2.12.0.redhat-610312
org.apache.camel.archetypes	camel-archetype-cxf-code-first-...	2.12.0.redhat-610312
org.apache.camel.archetypes	camel-archetype-cxf-contract-fir...	2.12.0.redhat-610312
org.apache.camel.archetypes	camel-archetype-dataformat	2.12.0.redhat-610312
org.apache.camel.archetypes	camel-archetype-java	2.12.0.redhat-610312
org.apache.camel.archetypes	camel-archetype-spring	2.12.0.redhat-610312
org.apache.camel.archetypes	camel-archetype-spring-dm	2.12.0.redhat-610312
org.apache.camel.archetypes	camel-archetype-web	2.12.0.redhat-610312
org.apache.camel.archetypes	camel-archetype-webconsole	2.12.0.redhat-610312
org.apache.cxf.archetype	cxf-jaxrs-service	2.7.0.redhat-610312
org.apache.cxf.archetype	cxf-jaxws-javafirst	2.7.0.redhat-610312


Creates a new Camel project with added Spring DSL support.

Group Id:

Artifact Id:

Version:

Package:

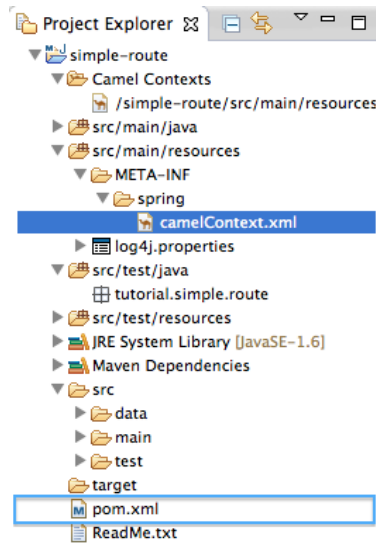


4. Select `camel-archetype-spring`.
5. Enter `tutorial` in the **Group Id**: field.
6. Enter `simple-route` in the **Artifact Id**: field.
7. The **Version**: field defaults to `1.0.0-SNAPSHOT`. To change it, enter a different version identifier.
8. The **Package**: field defaults to `tutorial.simple.route`, the name of the package that contains `camel-archetype-spring`. To include the route in a different package, enter the name of that package.
9. Click **Finish**.

This procedure creates a Fuse project, `simple-route`, in **Project Explorer** that contains everything needed to create and run routes. As shown in [Figure 1.3](#), the files generated for `simple-route` include:

- `simple-route/pom.xml` (Maven project file)
- `simple-route/src/main/resources/META-INF/spring/camel-context.xml` (Spring XML file containing the routing rules)

**Figure 1.3. Generated project files**



## CREATING THE ROUTE

To create the new route:

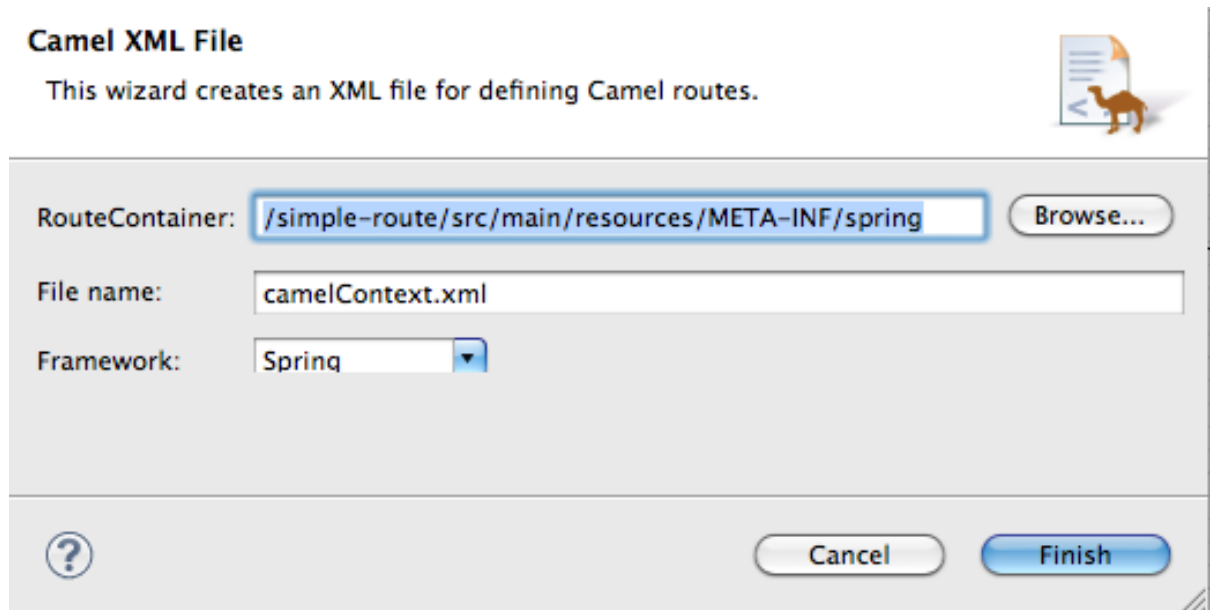
1. In **Project Explorer**, locate `simple-route/src/main/resources/META-INF/spring/camel-context.xml`.


2. Right-click it to open the context menu, then select **Delete**.

You're going to replace the old `camel-context.xml` file with your own to create a new route.

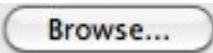
3. In the **Delete** dialog, click **OK** to confirm the operation.
4. In **Project Explorer**, select `simple-route/src/main/resources/META-INF/spring`.
5. Right-click it to open the context menu.
6. Select **New** → **Camel XML File** to open the **Camel XML File** wizard, as shown in [Figure 1.4](#).

Figure 1.4. Camel XML File wizard



7. Check that `/simple-route/src/main/resources/META-INF/spring` appears in the **Container** : field. Otherwise enter it manually, or select it using the  button.

**NOTE**

The  button opens a dialog that displays the folders of all active projects, which you can browse to find and select the files you need.

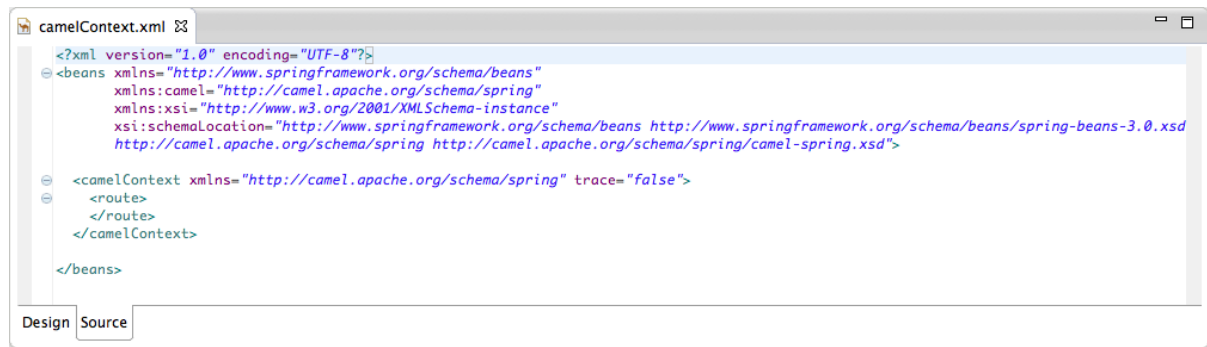
8. Check that `camelContext.xml` appears in the **File Name** : field. Otherwise enter it manually.
9. Check that **Spring** appears in the **Framework** field, or select it from the field's drop-down list.
10. Click **Finish**.


**NOTE**

By default, **Outline** view is located in the upper, right corner of the JBoss perspective. To provide more room for **Design** view to display the graphical representation of your route, drag **Outline** view to the lower, left corner of the workspace, below **Project Explorer**.

11. Click the **Source** tab at the bottom, left of the canvas to open the new `camelContext.xml` file in the route editor's Source view, as shown in [Figure 1.5, "New camelContext file in the route editor's source view"](#).

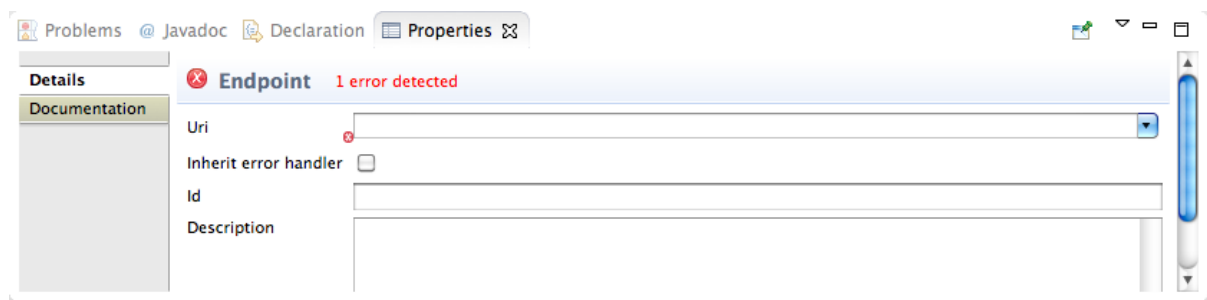
Figure 1.5. New camelContext file in the route editor's source view




12. Click the **Design** tab at the bottom, left of the canvas to return to the route editor's Design view.
13. Drag an **Endpoint** element (  ) from the **PaLETTE** to the canvas.
14. Drag a second **Endpoint** element from the **PaLETTE** to the canvas.
15. Select the first endpoint you dragged onto the canvas.

The **Properties** editor, located below the canvas, displays the endpoint's property fields for editing, as shown in [Figure 1.6, "Endpoint property editor"](#).

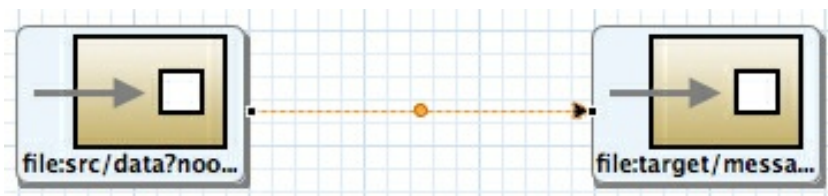
Figure 1.6. Endpoint property editor



16. Enter `file:src/data?noop=true` in the **Uri** field. Leave the other fields blank.
17. Select the second endpoint you dragged onto the canvas.
18. Enter `file:target/messages/others` in the **Uri** field. Leave the other fields blank.
19. On the canvas, select the first endpoint (`file:src/data?noop=true`), and drag its connector arrow (  ) to the second endpoint (`file:target/messages/others`), then release it.

A segmented line connects the two endpoints, as shown in [Figure 1.7](#).

Figure 1.7. Completed route, diagram view





## NOTE

You can drag the line's bendpoint (orange dot) to change the angle of the line's segments. Doing so creates two new bendpoints, one on either side of the original. This behavior enables you to easily adjust your diagram to accommodate increasingly complex routes.

20. To quickly align the connected endpoints, right-click the canvas to open the context menu, and then select **Layout Diagram**.
21. Select **File** → **Save** to save the route.
22. Click the **Source** tab at bottom, left of the canvas.

Source view displays the XML for the route. The `camelContext` element will look like [Example 1.1](#).

### Example 1.1. XML for simple route

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:camel="http://camel.apache.org/schema/spring"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-
                           3.0.xsd
                           http://camel.apache.org/schema/spring
                           http://camel.apache.org/schema/spring/camel-spring.xsd">

  <camelContext trace="false"
    xmlns="http://camel.apache.org/schema/spring">
    <route>
      <from uri="file:src/data?noop=true"/>
      <to uri="file:target/messages/others"/>
    </route>
  </camelContext>

</beans>
```

## NEXT STEPS

After you have created and designed your route, you can run it by deploying it into your Apache Camel runtime, as described in [Chapter 2, To Run a Route](#).

## FURTHER READING

To learn more about:

- using the editor, see Red Hat JBoss Fuse Tooling: JBoss Fuse Tooling User Guide at [https://access.redhat.com/site/documentation/JBoss\\_Fuse/](https://access.redhat.com/site/documentation/JBoss_Fuse/)



- Apache Camel endpoints, see *Red Hat JBoss Fuse: Component Reference* that comes with Red Hat JBoss Fuse.

## CHAPTER 2. TO RUN A ROUTE

### Abstract

This tutorial walks you through the process of running a route.

### GOALS

In this tutorial you will:

- run a route as a local Apache Camel context
- send messages through a route
- examine the messages received by the endpoints

### PREREQUISITES

To complete this tutorial you will need the Apache Camel project created in [Chapter 1, To Create a New Route](#).

### RUNNING THE ROUTE

To run the route:

1. Open the `simple-route` project you created in [the section called “Creating the Fuse project”](#).
2. In **Project Explorer**, select `simple-route/src/main/resources/META-INF/spring/camelContext.xml`.
3. Right-click it to open the context menu, then select **Run As** → **Local Camel Context (without tests)**.

The **Console** panel opens to display messages that reflect the progress of the project's execution. A message similar to the following

```
[INFO] Using org.apache.camel.spring.Main to initiate a CamelContext
[pache.camel.spring.Main.main()] MainSupport INFO Apache Camel
2.12.0.redhat-610062 starting [pache.camel.spring.Main.main()]
SpringCamelContext INFO Apache Camel 2.12.0.redhat-610062
(CamelContext: camel-1) is starting [pache.camel.spring.Main.main()]
ManagedManagementStrategy INFO JMX is enabled
[pache.camel.spring.Main.main()] DefaultTypeConverter INFO Loaded
176 type converters [pache.camel.spring.Main.main()]
SpringCamelContext INFO StreamCaching is not in use. If using
streams then its recommended to enable stream caching. See more
details at http://camel.apache.org/stream-caching.html
[pache.camel.spring.Main.main()] FileEndpoint INFO Endpoint is
configured with noop=true so forcing endpoint to be idempotent as
well [pache.camel.spring.Main.main()] FileEndpoint INFO Using
default memory based idempotent repository with cache max size: 1000
[pache.camel.spring.Main.main()] SpringCamelContext INFO Route:
route1 started and consuming from: Endpoint[file://src/data?
```

```
noop=true] [pache.camel.spring.Main.main()] SpringCamelContext INFO
Total 1 routes, of which 1 is started.
[pache.camel.spring.Main.main()] SpringCamelContext INFO Apache
Camel 2.12.0.redhat-610062 (CamelContext: camel-1) started in 0.354
seconds
```

indicates the route executed successfully.

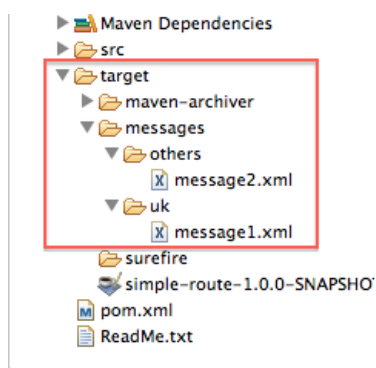
4. To shutdown the route, click the red square (  ) located at the top, right of the **Console** panel.

## VERIFYING THE ROUTE

To verify that the route executed properly:

1. In **Project Explorer**, select **simple-route**.
2. Right-click it to open the context menu, then select **Refresh**.
3. In **Project Explorer**, locate the folder **target/messages/** and expand it, as shown in [Figure 2.1](#).

**Figure 2.1. Target message destinations in Project Explorer tree**



4. Verify that the **target/messages/** subfolders contain these files:
  - **uk/message1.xml**
  - **others/message2.xml**
5. Double-click **message1.xml** to open it in the editor's Design view, then select the **Source** tab at the bottom, left of the canvas to see the xml code.

It's contents should match that shown in [Example 2.1](#).

### Example 2.1. Contents of message1.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<person user="james">
  <firstName>James</firstName>
  <lastName>Strachan</lastName>
  <city>London</city>
</person>
```



## FURTHER READING

To learn more about:

- configuring runtime profiles, see Red Hat JBoss Fuse Tooling: JBoss Fuse Tooling User Guide at [https://access.redhat.com/site/documentation/JBoss\\_Fuse/](https://access.redhat.com/site/documentation/JBoss_Fuse/).
- deploying Apache Camel applications see *Red Hat JBoss Fuse: Deploying into the Container* that comes with Red Hat JBoss Fuse.

## CHAPTER 3. TO ADD A CONTENT-BASED ROUTER

### Abstract

This tutorial walks you through adding a content-based router with logging to a route.

### GOALS

In this tutorial you will:

- add a content based-router to a route
- configure the content-based router
- add a log endpoint to each output branch of the content-based router
- rewire the route to use the content-based router and log its output

### PREREQUISITES

To complete this tutorial you will need the `simple-route` project you created in [Chapter 1, To Create a New Route](#).

### PROCEDURE

To add a content-based router with logging to your route:

1. In **Project Explorer**, double-click `simple-route/src/main/resources/META-INF/spring/camelContext.xml` to open your `simple-route` project.
2. Select the connector joining the two endpoint nodes `file:src/data?noop=true` and `file:target/messages/others`.
3. Right-click it to open the context menu, and select **Edit** → **Remove** to delete the connector.



#### NOTE


Alternatively, you can delete the connector by selecting it, then selecting **Delete** from the toolbar's **Edit** menu.

4. On the canvas, select the terminal endpoint node, `file:target/messages/others`, and drag it out of the way.
5. On the canvas, select the starting endpoint node, `file:src/data?noop=true`, and right-click it to open the context menu.
6. Select **Add** → **Routing** → **Choice**.

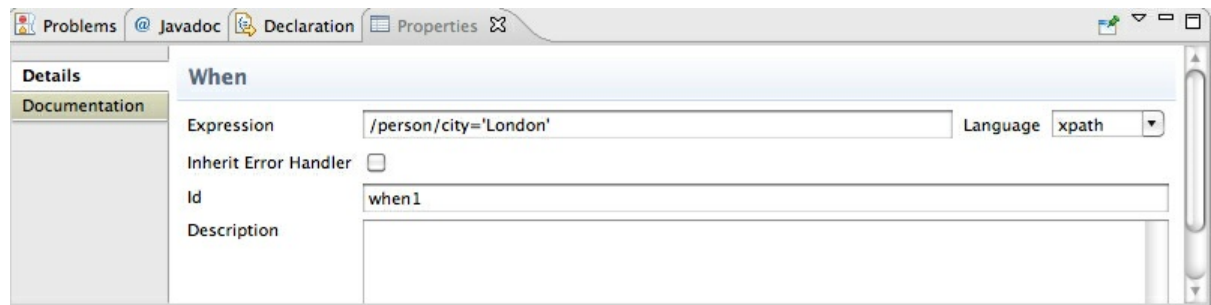
A **choice** node (  ) appears on the canvas connected to the starting endpoint node.

7. On the canvas, select the **choice** node, then right-click it to open the context menu.

8. Select **Add** → **Routing** → **When**.

A **when** node (  ) appears on the canvas connected to the **choice** node. The **Properties** editor opens, displaying the **when** node's property fields for you to edit, as shown in [Figure 3.1](#).

**Figure 3.1. When property editor**

9. In the **Expression** field, enter `/person/city='London'` .


This XPath expression determines which messages will transit this path in the route.

10. From the **Language** drop-down menu, select **xpath**.11. In the **Id** field, enter **when1**.12. On the canvas, reselect the **Choice** node, then right-click it to open the context menu.13. Select **Add** → **Routing** → **Otherwise**.

An **otherwise** node (  ) appears on the canvas, connected to the **choice** node.

The **otherwise** node will eventually route to the terminal endpoint (`file:target/messages/others`) any message that does not match the XPath expression set for the **when1** node.

14. On the canvas, select the **when1** node, and then right-click it to open the context menu.15. Select **Add** → **Endpoints** → **Log**


A **log** node (  ) appears on the canvas, connected to the **when1** node. The **Properties** editor opens, displaying the **log** node's property fields for you to edit.

16. In the **Message** field, enter **uk messages**, and in the **Log Name** field, enter **1**.

### NOTE


In **Fuse Integration** perspective's **Messages View**, the tooling inserts the contents of the log node's **Id** field in the **Trace Node Id** column for message instances, when tracing is enabled on the route (see [Figure 4.4](#)). In the **Console**, it adds the contents of the log node's **Message** field to the log data whenever the route runs.

17. On the canvas, select the **otherwise** node, and then right-click it to open the context menu.18. Select **Add** → **Endpoints** → **Log**

A **log** node (  ) appears on the canvas, connected to the **otherwise** node. The **Properties** editor opens, displaying the **log** node's property fields for you to edit.

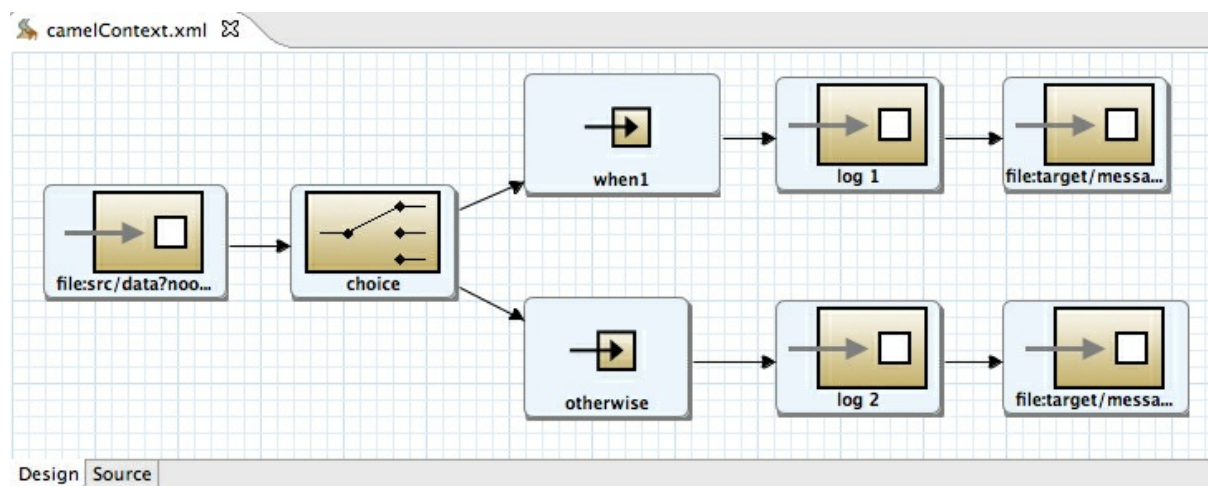
19. In the **Message** field, enter `other` messages, and in the **Log Name** field, enter `2`.
20. On the canvas, select the **log 1** node, and then right-click it to open the context menu.
21. Select **Add** → **Endpoints** → **Endpoint**

An **Endpoint** node (  ) appears on the canvas, connected to the **log 1** node. The **Properties** editor opens, displaying the **Endpoint** node's property fields for you to edit.

22. In the **Uri** field, enter `file:target/messages/uk`, and leave the other property fields blank.
23. On the canvas, select the **log 2** node, and then drag its connector arrow (  ) to the terminal endpoint node, `file:target/messages/others`, and release it.
24. To quickly realign all of the nodes on the canvas, right-click the canvas to open the context menu, and then select **Layout Diagram**.

The route on the canvas should resemble [Figure 3.2](#).

**Figure 3.2. Completed content-based router with logs**



25. On the toolbar, select **File** → **Save** to save the completed route.
26. Click the **Source** tab at the bottom, left of the canvas to display the XML for the route.

The `camelContext` element will look like that shown in [Example 3.1](#).

#### Example 3.1. XML for content-based router

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:camel="http://camel.apache.org/schema/spring"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
```

```
    http://camel.apache.org/schema/spring
    http://camel.apache.org/schema/spring/camel-spring.xsd">

    <camelContext trace="false"
xmlns="http://camel.apache.org/schema/spring">
    <route>
    <from uri="file:src/data?noop=true"/>
    <choice>
    <when id="when1">
    <xpath>/person/city='London'</xpath>
    <log logName="1" message="uk messages"/>
    <to uri="file:target/messages/uk"/>
    </when>
    <otherwise>
    <log logName="2" message="other messages"/>
    <to uri="file:target/messages/others"/>
    </otherwise>
    </choice>
    </route>
</camelContext>

</beans>
```

## NEXT STEPS

You can run the new route as described in [the section called “Running the route”](#).

## FURTHER READING

To learn more about message enrichment see:

- the `when` EIP in *Red Hat JBoss Fuse: Enterprise Implementing Integration Patterns*
- the [Red Hat JBoss Fuse 6.x documentation](#)



# CHAPTER 4. TO TRACE A MESSAGE THROUGH A ROUTE

## Abstract

This tutorial walks you through the process of tracing a message through a route.

## GOALS

In this tutorial you will:

- run a route in the **Fuse Integration** perspective
- enable tracing on your route
- drop messages onto your route and track them through the route's nodes

## PREREQUISITES

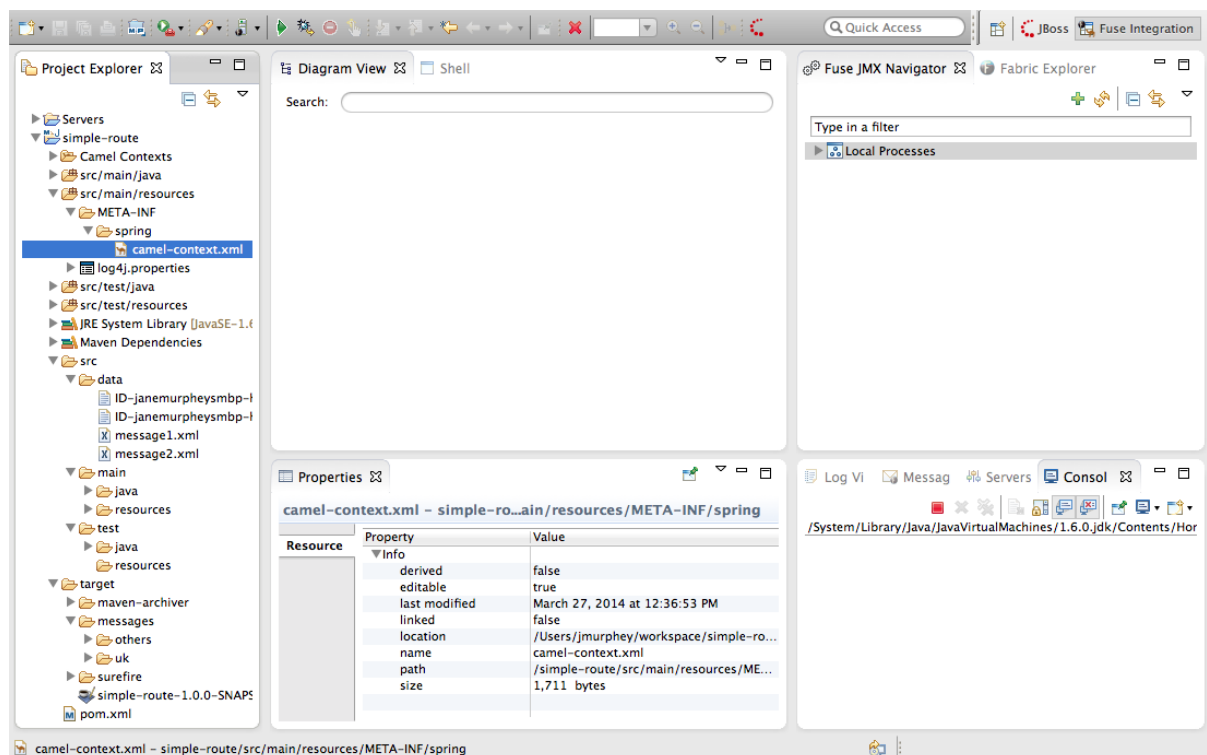
To complete this tutorial you will need the `simple-route` project you updated in [Chapter 3, To Add a Content-Based Router](#).

## PROCEDURE


To trace a message through your route:

1. Select **Window** → **Open Perspective** → **Other...** → **Fuse Integration** to open the **Fuse Integration** perspective as shown in [Figure 4.1, “Fuse Integration perspective”](#).

**Figure 4.1. Fuse Integration perspective**

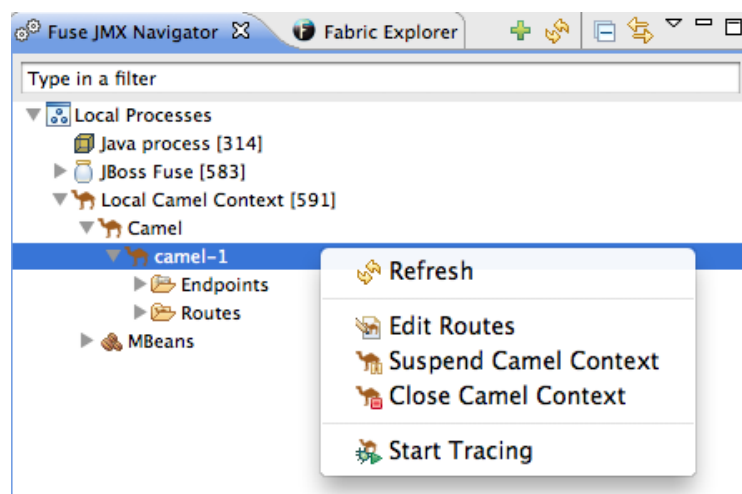


**NOTE**

You can use the **Open Perspective** icon (  ) in the perspectives tab to access the list of available perspectives.

2. In **Project Explorer**, expand the `simple-route` project to expose the `src/main/resources/META-INF/spring/camel-context.xml` file.
3. Select **Run As** → **Local Camel Context (without tests)** from the `camel-context.xml` file's context menu.
4. In **Fuse JMX Navigator**, expand **Local Processes**.
5. Double click **Local Camel Context [id]** to connect to the context and expand the elements of your route as shown in [Figure 4.2, “Route elements in Fuse JMX Navigator”](#) ).

**Figure 4.2. Route elements in Fuse JMX Navigator**



6. In **Fuse JMX Navigator**, select **Start Tracing** from the `camel-1` node's context menu.

The tooling displays a graphical representation of your route in **Diagram View**.

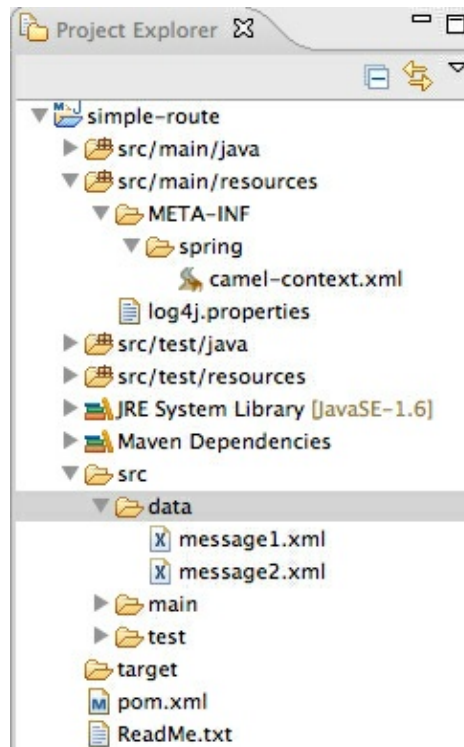
7. In **Diagram View**, drag the nodes to rearrange them, so you can clearly see the route's flow paths.

**NOTE**

You will have to rearrange the nodes in **Diagram View** each time you select a different node in **Fuse JMX Navigator**.

8. In **Project Explorer**, expand `simple-route/src/data`, so you can access the `message1.xml` and `message2.xml` files as shown in [Figure 4.3, “Message files in simple-route project”](#).

Figure 4.3. Message files in simple-route project



9. Drag `message1.xml` and drop it on `.../camel-1/Endpoints/file/src/data?noop=true`.

As the message traverses the route, the tooling traces and records its passage at each step and displays the results in **Messages View**.

10. Repeat [Step 9](#), but this time drag `message2.xml` and drop it on `.../camel-1/Endpoints/file/src/data?noop=true`.
11. In the bottom, right panel, switch from **Console** to **Messages View**.
12. In **Fuse JMX Navigator**, click `camel-1` to populate **Messages View** with the traces of each message.

As shown in [Figure 4.4](#), “[Fuse Integration perspective's message tracing components](#)”, the tooling draws the route in **Diagram View**, tagging paths exiting a processing step with timing and performance metrics. Only the metric **Total exchanges** is displayed in the diagram. Hovering over the displayed metrics reveals additional metrics about message flow:

- mean time the step took to process a message
- maximum time the step took to process a message
- minimum time the step took to process a message

Figure 4.4. Fuse Integration perspective's message tracing components

The screenshot displays the Fuse Integration tooling interface. The **Diagram View** shows a route starting with a `file:src/data?noop=...` endpoint, followed by a `choice` node. The `choice` node branches into two paths: one through a `when /person/city...` node and another through an `otherwise` node. Both paths lead to a `log` node, then a `setHeader/Destina...` node, and finally a `file:target/messa...` endpoint. The **Properties** panel shows message details for a selected instance, including headers like `CamelFileAbsolutePath` and `CamelFileName`. The **Log View** table shows the following data:

Message Body	Trace ID	Trace Node	Destination	Trace Time	Relative TI	Elapse
<?xml versio...	1			Thu Mar...	0	
<?xml versio...	2	choice1		Thu Mar...	0	0
<?xml versio...	3	log1		Thu Mar...	5	5
<?xml versio...	4	setHead1		Thu Mar...	6	1
<?xml versio...	5	to1	UNITED KINGDOM	Thu Mar...	7	1
<?xml versio...	6			Thu Mar...	0	
<?xml versio...	7	choice1		Thu Mar...	0	0
<?xml versio...	8	log2		Thu Mar...	3	3
<?xml versio...	9	setHead2		Thu Mar...	6	3


### 13. In **Messages View**, click a message instance.

The associated step in the route is highlighted in **Diagram View**. You can step through the message instances to see how a particular message traversed the route and whether it was processed as expected at each step in the route.

The tooling displays the details about a message instance in the top half of the **Properties** panel and the contents of the message instance, including any headers, in the bottom half of the **Properties** panel. So, if your application sets headers at any step within a route, you can check whether they are set as expected.



#### NOTE

You can control columnar layout in all of the tooling's tables. Use the drag method to temporarily rearrange tabular format. For example, drag a column's border rule to expand or contract its width. To hide a column, totally contract its borders. Drag the column header to relocate a column within the table. If you want your arrangement to persist, use the **View Menu** → **Configure Columns...** method instead. To access it, click the  icon on the panel's menu bar.

### 14. When done, switch back to the **Console** and click the stop button ( ) in the upper, right side of its pane.

## NEXT STEPS

You can run the route with a JUnit test case, as described in [Chapter 5, To Test a Route with JUnit](#)

## FURTHER READING

To learn more about message enrichment see:

- the **choice, when, and logEIP** Enterprise Integration Patterns in *Red Hat JBoss Fuse: Implementing Enterprise Integration Patterns* that comes with Red Hat JBoss Fuse.
- the [Red Hat JBoss Fuse 6.x documentation library](#)

## CHAPTER 5. TO TEST A ROUTE WITH JUNIT

### Abstract

This tutorial walks you through the process of using the New Camel Test Case wizard to create a test case for your route and using it test the route.

### OVERVIEW

The New Camel JUnit Test Case wizard generates a boilerplate JUnit test case. This means that when you create or modify a route (for example, adding more processors to it), you'll need to modify the generated test case to add expectations and assertions specific to the new route you've created, so the test is valid for the route.

### GOALS

In this tutorial you will:

- create a Apache Camel test case
- run the route with the test case
- observe the output

### PREREQUISITES

To complete this tutorial you will need:

- the `simple-route` project you created in [Chapter 3, To Add a Content-Based Router](#)

### CREATING THE TEST CASE

To create a new test case:

1. In **Project Explorer**, select `src/test/java`.
2. Right-click it to open the context menu, and then select **New** → **Camel Test Case** to open the **New Camel JUnit Test Case** wizard, as shown in [Figure 5.1, “New Camel JUnit Test Case wizard”](#).

Figure 5.1. New Camel JUnit Test Case wizard

**Camel JUnit Test Case**

Select the name of the new JUnit test case. You have the options to specify the Camel XML file under test and on the next page, to select methods to be tested.

Source folder:

Package:

Camel XML file under test:

Name:

Which method stubs would you like to create?

setUpBeforeClass()  tearDownAfterClass()  
 setUp()  tearDown()  
 constructor

Do you want to add comments? (Configure templates and default value [here](#))  
 Generate comments

3. Make sure the **Source folder** field contains `simple-route/src/test/java`.

**NOTE**

If needed, you can click  to find the proper folder.

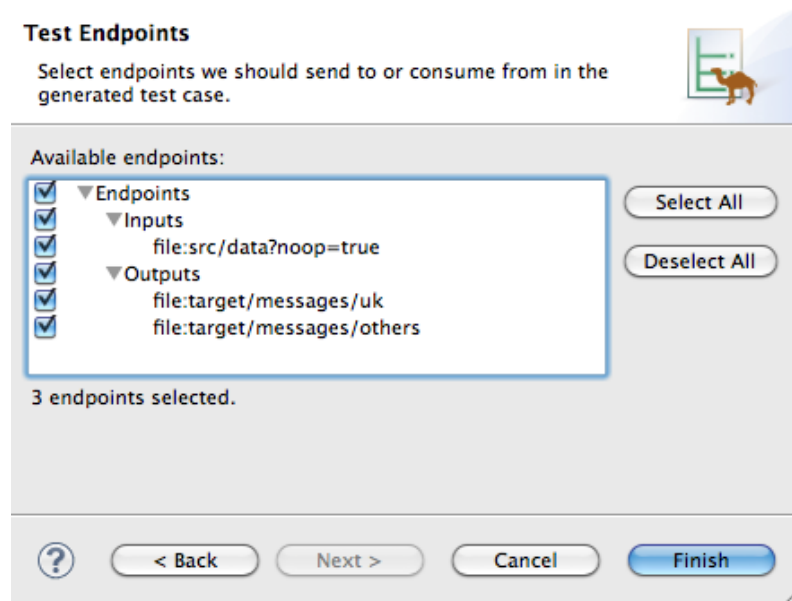
4. The **Package** field defaults to `tutorial.simple-route`. To include the test case in a different package, enter the name of the package.
5. In the **Camel XML file under test** field, enter `src/main/resources/META-INF/spring/camelContext.xml`, or use  to open a file explorer, configured to screen for XML files, to locate the file.

**NOTE**

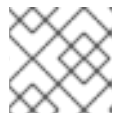
The **Name** field defaults to `CamelContextXmlTest` for the name of the test file.

6. Click **Next>** to open the **Test Endpoints** page, shown in [Figure 5.2, “Test Endpoints page”](#).

Figure 5.2. Test Endpoints page



7. By default, all endpoints are selected and will be included in the test case. You can select or deselect all endpoints by clicking the **Select All** or **Deselect All** button, or you can select and deselect individual endpoints by clicking the check box next to each.
8. Click **Finish**.

**NOTE**

If prompted, add JUnit to the build path.

The artifacts for the test are added to your project, and the class implementing the test case opens in the Java editor.

## RUNNING THE TEST

To run the test:

1. Select the project root, `simple-route`, in the **Project Explorer**.
2. Open the context menu.
3. Select **Run As** → **JUnit Test**.

The **JUnit** view, shown in [Figure 5.3, “JUnit view”](#), opens in the Eclipse sidebar.



Figure 5.3. JUnit view

```

tutorial.simple.route.CamelContextXmiTest
  testCamelRoute (4.270 s)
    java.lang.AssertionError: mock://output2 Message with body <something id='2'>expectedBody2</something> was expected but not found in [-<?xml version='1.0' encoding='UTF-8?>
    <person user='hiram'>
    <firstName>Hiram</firstName>
    <lastName>Chirino</lastName>
    <city>Tampa</city>
    </person>
    ]
    at org.apache.camel.component.mock.MockEndpoint.fail(MockEndpoint.java:1318)
    at org.apache.camel.component.mock.MockEndpoint.assertTrue(MockEndpoint.java:1306)
    at org.apache.camel.component.mock.MockEndpoint$8.run(MockEndpoint.java:701)
    at org.apache.camel.component.mock.MockEndpoint.doAssertSatisfied(MockEndpoint.java:383)
    at org.apache.camel.component.mock.MockEndpoint.assertSatisfied(MockEndpoint.java:351)
    at org.apache.camel.component.mock.MockEndpoint.assertSatisfied(MockEndpoint.java:339)
    at org.apache.camel.component.mock.MockEndpoint.assertSatisfied(MockEndpoint.java:177)
    at org.apache.camel.test.junit4.CamelTestSupport.assertMockEndpointsSatisfied(CamelTestSupport.java:653)
    at tutorial.simple.route.CamelContextXmiTest.testCamelRoute(CamelContextXmiTest.java:50)
  
```

4. Examine the output and take action to resolve any test failures.



## NOTE

You may have to run this boilerplate test twice before it runs without a failure.

## FURTHER READING

To learn more about:

- see [JUnit](#)

# CHAPTER 6. TO DEPLOY A CAMEL PROJECT TO RED HAT JBOSS FUSE

## Abstract

This tutorial walks you through the process of deploying a camel project into Red Hat JBoss Fuse. It assumes that you have an instance of Red Hat JBoss Fuse installed on the same machine on which you are running the Red Hat JBoss Fuse Tooling .

## GOALS

In this tutorial you will:

- start up Red Hat JBoss Fuse 6.x Server
- deploy your project into Red Hat JBoss Fuse 6.x Server
- check whether your project was successfully built and deployed
- uninstall your deployed bundle

## PREREQUISITES

To complete this tutorial you will need

- access to a Red Hat JBoss Fuse instance
- the `simple-route` project you updated in [Chapter 3, To Add a Content-Based Router](#)

## STARTING UP RED HAT JBOSS FUSE 6.X SERVER

To start up the server:

1. In `Fuse Integration` perspective, click the `Servers` tab in the lower, right pane to open the `Servers` view.
2. Click the link `No servers are available. Click this link to create a new server...` to open the `Define a New Server` page.
3. Expand the `JBoss Fuse` node to expose the available server options as shown in [Figure 6.1](#).

Figure 6.1. Define a New Server page

**Define a New Server**  
Choose the type of server to create

[Download additional server adapters](#)

Select the server type:

type filter text

- ▶ Apache
- ▶ Basic
- ▶ JBoss Community
- ▶ JBoss Enterprise Middleware
- ▼ JBoss Fuse
  - ▶ Apache Karaf 2.0 Server
  - ▶ Apache Karaf 2.1 Server
  - ▶ Apache Karaf 2.2 Server
  - ▶ Apache Karaf 2.3 Server
  - ▶ Apache ServiceMix 4.0 Server
  - ▶ Apache ServiceMix 4.2 Server
  - ▶ Apache ServiceMix 4.3 Server
  - ▶ Apache ServiceMix 4.4 Server
  - ▶ Apache ServiceMix 4.5 Server
  - ▶ JBoss Fuse 6.0 Server
  - ▶ **JBoss Fuse 6.1 Server**
- ▶ ObjectWeb
- ▶ OpenShift

Server Definition of JBoss Fuse 6.1

Server's host name:

Server name:

? < Back Next > Cancel Finish

4. Click **JBoss Fuse 6.1 Server**.
5. Accept the defaults for **Server 's host name** (*localhost*) and **Server name** (*JBoss Fuse 6.1 Server at localhost*), and then click **Next** to open the **JBoss Fuse Runtime** page.
6. In **Installation directory**, enter the path where the JBoss Fuse 6.1 installation is located, or click the **Browse** to search for it, and then click **Next** to open the **New Server> JBoss Fuse server configuration details** page.

Figure 6.2. New server configuration

**JBoss Fuse**  
Provide JBoss Fuse server configuration details

Host Name: 0.0.0.0  
 Port Number: 8101  
 User Name:  
 Password:

? < Back Next > Cancel Finish

7. Accept the defaults for **Host Name** (*0.0.0.0*) and **Port Number** (*8101*).
8. In **User name**, enter the name used to log into the server.

This is a user name stored in Red Hat JBoss Fuse's *installDir/etc/users.properties* file.

If one has not been set, you can either add one to that file using the format `user=password,role` (for example, `admin=admin,admin`), or you can set one using the `karaf jaas` command set:

- `jaas:realms`—to list the realms
- `jaas:manage --index 1`—to edit the first (server) realm
- `jaas:useradd <username> <password>`—to add a user and associated password
- `jaas:roleadd <username> admin`—to specify the new user's role
- `jaas:update`—to update the realm with the new user information

If a `jaas` realm has already been selected for the server, you can discover the user name by issuing the command `JBossFuse:karaf@root>jaas:users`.

9. In **Password**, enter the password required for **User name** to log into the server.

This is the password set either in Red Hat JBoss Fuse's *installDir/etc/users.properties* file or by the `karaf jaas` commands.

10. Click **Next**, and then click **Finish**.

`jboss-fuse-6.1.0.redhat-xxx [stopped]` appears in **Servers** view.

11. In **Servers** view, right-click `jboss-fuse-6.1.0.redhat-xxx` to open the context menu, and then click **Start**.



## IMPORTANT

If the warning that the remote host identification has changed appears, click **yes** to delete the old key and insert the new key, only if the JBoss Fuse 6.1 server runtime is installed on the same machine where the Red Hat JBoss Fuse Tooling is running! Otherwise click **no** and contact your system administrator.

Wait a few seconds for JBoss Fuse 6.1 Server to start up. When it does, **JBoss Fuse [xxx]** is added to the **Fuse JMX Navigator** tree under the Local Processes node, but you need to expand the Local Processes node to see it.

The JBoss Fuse console also starts up in **Shell** view, as shown in [Figure 6.3](#):

**Figure 6.3. JBoss Fuse console**

```

Shell
SSH: (admin@0.0.0.08159 - CONNECTED) - Encoding: (ISO-8859-1)

  JBoss Fuse
  http://www.redhat.com/products/jbossenterprisemiddleware/fuse/

JBoss Fuse (6.1.0.redhat-314)
http://www.redhat.com/products/jbossenterprisemiddleware/fuse/

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.

Open a browser to http://localhost:8181 to access the management console

Hit '<ctrl-d>' or 'osgi:shutdown' to shutdown JBoss Fuse.

```

12. In **Fuse JMX Navigator**, right-click **JBoss Fuse [xxx]** to open the context menu, and then click **Connect**.
13. In **Fuse JMX Navigator**, expand the **JBoss Fuse [xxx]** node to see its tree structure.

Now you're ready to deploy your project.

## DEPLOYING A PROJECT TO RED HAT JBOSS FUSE

To deploy a camel project to JBoss Fuse:

1. From **Project Explorer**, drag the **simple-route** root project over to **Fuse JMX Navigator** and drop it on **JBoss Fuse [xxx]**.



## NOTE

The **Deploy to...** tool provides an alternative to the drag and drop method of deployment. For details, see *Deploying Projects to a Container* in the *Red Hat JBoss Fuse Tooling: JBoss Fuse Tooling User Guide*.

**Console** view chronicles the process as the tooling builds the **simple-route** project, runs the tests, and then installs the project as a bundle inside **JBoss Fuse [xxx]**.

2. In **Fuse JMX Navigator**, right-click **JBoss Fuse [xxx]** to open the context menu, and then click **Refresh**.



### NOTE

You may have to repeat the **Refresh** operation before your project appears in **Fuse JMX Navigator** under **JBoss Fuse [xxx] → Camel** as `<bundleIdentifier>-camel-#`. (Camel generates and appends an ID number to the end of the node name to ensure that every camel context is uniquely identified.)

In the mean time, you can verify whether your project's bundle was installed by clicking the **Bundles** node under **JBoss Fuse [xxx]** to display, in **Properties** view, all installed bundles. Start typing *tutorial* in **Properties** view's Search tool to quickly determine whether your project's `tutorial.simple-route` bundle is included in the list.

Once your project appears in **Fuse JMX Navigator**, you can start tracing on it, as described in [Chapter 4, To Trace a Message Through a Route](#).

## UNINSTALLING YOUR PROJECT'S BUNDLE

To uninstall your project from JBoss Fuse 6.1 server:

1. Switch to the JBoss Fuse console in **Shell** view.
2. At the JBoss Fuse console command line, type `uninstall <bundleIdentifier>`

## FURTHER READING

To learn more about deploying applications to external containers, see:

- *Deploying Projects to a Container* in the *Red Hat JBoss Fuse Tooling: JBoss Fuse Tooling User Guide*

# CHAPTER 7. TO DEPLOY A CAMEL PROJECT TO A FABRIC PROFILE

## Abstract

This tutorial walks you through the process of deploying a camel project into a fabric profile in Fuse Integration perspective. It assumes that you have an instance of Red Hat JBoss Fuse installed on the same machine on which you are running the Red Hat JBoss Fuse Tooling .

## GOALS

In this tutorial you will:

- create a fabric in JBoss Fuse 6.x Server
- connect to the JBoss Fuse 6.x Server
- connect to the fabric
- create a new fabric profile
- deploy your project to the new fabric profile

## PREREQUISITES

To complete this tutorial you will need:

- access to Red Hat JBoss Fuse
- a user with admin privileges configured in JBoss Fuse's `installDir/etc/users.properties` file, as described in [Starting up Red Hat JBoss Fuse 6.x Server](#) starting at [Step 8](#).
- the `simple-route` project you updated in [Chapter 3, To Add a Content-Based Router](#)

## CREATING A FABRIC IN RED HAT JBOSS FUSE

To create a fabric in Red Hat JBoss Fuse:

1. Open a terminal and `cd` to the JBoss Fuse server's `installDir`.
2. Enter `./bin/fuse` to start up a standalone instance of `jboss-fuse-6.x.x.redhat-xxx`.

Wait a few seconds for the JBoss Fuse 6.x server to start up.

3. At the JBoss Fuse console command line, enter `fabric:create` and press **Enter** to create a fabric.

```
JBossFuse:karaf@root> fabric:create
Waiting for container: root
Using specified zookeeper password:admin
It may take a couple of seconds for the container to provision...
You can use the --wait-for-provisioning option, if you want this
command to block until the container is provisioned.
```

```
JBossFuse:karaf@root>
```

You can use the `fabric:status` command to check whether the fabric has been created and provisioned.

```
JBossFuse:karaf@root> fabric:status
[profile]                [instances]  [health]
fabric                    1            100%
fabric-ensemble-0000-1   1            100%
jboss-fuse-full           1            100%
JBossFuse:karaf@root>
```

- Once the fabric is running, enter `fabric:container-list` and press **Enter** to list the new fabric's default container (`root*`) and its status.

```
JBossFuse:karaf@root> fabric:container-list
[id] [version] [alive] [profiles]
[provision status]
root* 1.0      true   fabric,fabric-ensemble-0000-1,jboss-fuse-
full  success
JBossFuse:karaf@root>
```

## CONNECTING TO THE JBOSS FUSE SERVER

To connect the Fuse Tooling to the JBoss Fuse server:

- If necessary, reopen **Fuse Integration** perspective.
- In **Fabric Explorer**, right-click **Fabrics** to open the context menu, and then click **Add Fabric details** to open the **Fabric Details** wizard.

Figure 7.1. Fabric Details wizard

- In **Name**, enter the name of the fabric to which you want to connect. The name you enter identifies the fabric whose location you specify in **Jolokia URL**, and this name will appear in **Fabric Explorer**.

The default **Name** is **Local Fabric**.

- In **Jolokia URL**, enter the url, in the form `http://hostname:port/jolokia/`, of the fabric to which you want to connect. This URL specifies the location of a *fabric registry agent*, whose default port is *8181*.



The default URL is `http://localhost:8181/jolokia`.

5. In **User name**, enter the name used to log into the specified fabric.

This is the user name specified when the fabric was created, has `admin` privileges, and is stored in Red Hat JBoss Fuse's `installDir/etc/users.properties` file. In that file, user information is specified using this format: `user=password,role` (for example, `admin=admin,admin`).

You can also discover the user name by issuing the command `JBossFuse:karaf@root>jaas:users`, if the Jaas realm has been selected for the fabric.

6. In **Password**, enter the password required for **User name** to log into the specified fabric.

This is the password specified for **User name** when the fabric was created and is stored in Red Hat JBoss Fuse's `installDir/etc/users.properties` file.

7. In **Zookeeper Password**, enter the password required for logging into the specified fabric's zookeeper registry.

This is the password that was specified when the fabric was created, or it is the password of the first user defined in Red Hat JBoss Fuse's `installDir/etc/users.properties` file.

You can also discover the Zookeeper password by issuing the command `JBossFuse:karaf@root>fabric:ensemble-password`.

8. Click **OK**.

The fabric's name appears in **Fabric Explorer** as a node beneath **Fabrics**.

## CONNECTING TO THE FABRIC

In **Fabric Explorer**, right-click **Local Fabric** to open the context menu, and then click **Connect** to connect to the new fabric.

Now you're ready to create a new fabric profile into which you'll deploy your camel project.

## CREATING A NEW FABRIC PROFILE

To create a new fabric profile:

1. In **Fabric Explorer**, expand **Local Fabric** → **Versions** → **1.0** to reveal the fabric's top-level profiles.
2. Further expand the profiles tree to find the `example-quickstarts-jms` profile nested under `default/karaf/feature-camel/feature-camel-jms/`.
3. Right-click `example-quickstarts-jms` to open the context menu, and then click **Create a new Profile**.
4. In **Profile name**, enter `myCamelRoute`, and then click **OK**.

The new profile `myCamelRoute` appears under its parent profile, `example-quickstarts-jms`, in **Fabric Explorer**.

5. Click the `myCamelRoute` profile to populate the **Details** tab's **Profiles** page with its information.
6. In the **Parents** pane, you can see that `example-quickstarts-jms` is the new profile's immediate parent. Leave `example-quickstarts-jms` selected.

Now you're ready to deploy your camel project to the profile `myCamelRoute`.

## DEPLOYING YOUR CAMEL PROJECT TO THE NEW FABRIC PROFILE

To deploy your camel project to the new `myCamelRoute` fabric profile:

1. From **Project Explorer**, drag the `simple-route` root project over to **Fabric Explorer** and drop it on **Local Fabric** → **Versions** → **1.0** → **default** → **karaf** → **feature-camel** → **feature-camel-jms** → **example-quickstarts-jms** → **myCamelRoute**.

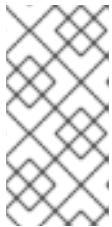


### NOTE

The **Deploy to...** tool provides an alternative to the drag and drop method of deployment. For details, see *Deploying a Project to a Fabric Container* in *Red Hat JBoss Fuse Tooling: JBoss Fuse Tooling User Guide*.

**Console** view chronicles the process as the tooling builds the `simple-route` project, runs the tests, installs the project as a bundle in the `myCamelRoute` profile, and then uploads the profile to the fabric's internal Maven repository.

2. In **Fabric Explorer**, click the `myCamelRoute` profile to populate **Properties** view with its properties and profile information.

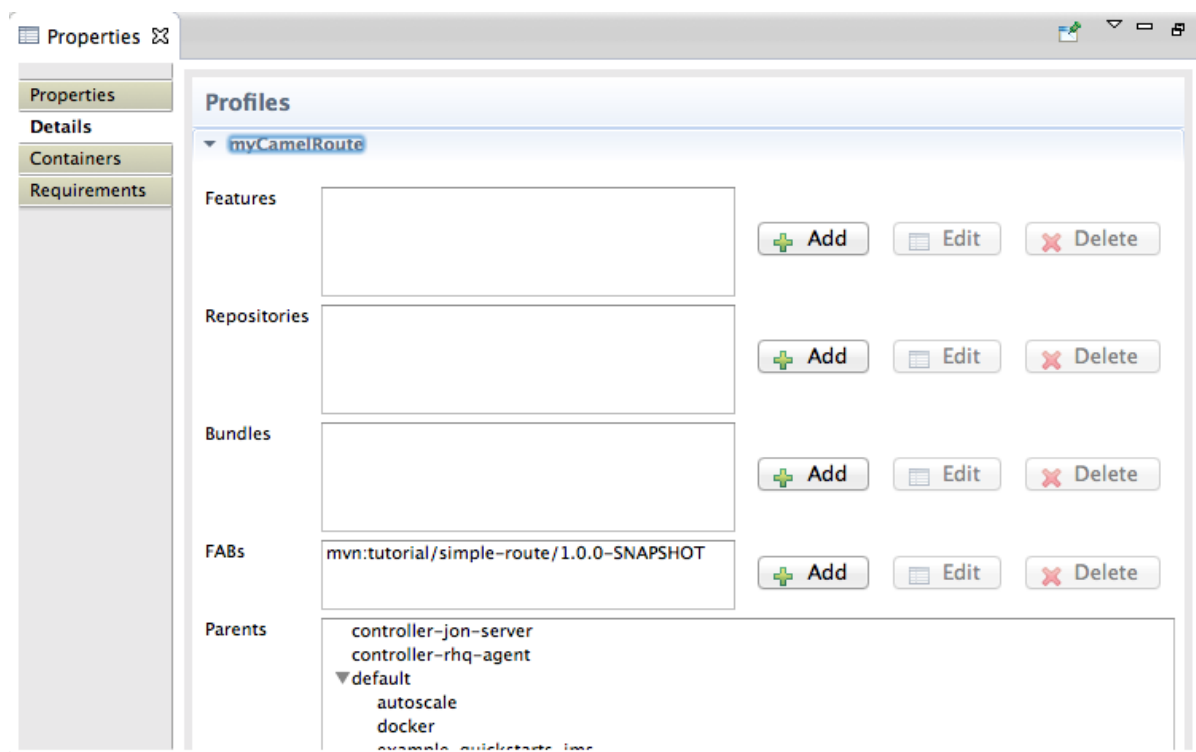


### NOTE

It can take some time for the tooling to build the project, run the tests, and install the project bundle. The `simple-route` bundle will appear in the **FABs** field on the **Profiles** page only when the process has finished. You can use **Fabric Explorer**'s **Refresh** button to trigger an update of the **Profiles** page.

3. On the **Profiles** page, check that the `simple-route` bundle appears in the **FABs** field, as shown in [Figure 7.2, “simple-route bundle deployed”](#).

Figure 7.2. simple-route bundle deployed



Now you can deploy your camel project to the fabric by creating one or more containers on the fabric and assigning the `myCamelRoute` profile to them. Once the containers are started, you can start tracing on the deployed projects, as described in [Chapter 4, To Trace a Message Through a Route](#).

## FURTHER READING

To learn more about deploying applications to a fabric, see in *Deploying a Project to a Fabric Container in Red Hat JBoss Fuse: Tooling User Guide* on the [Red Hat Customer Portal](#) :

- *Deploying Projects to a Container*
- *Working with Fabric Containers*
- *Working with Fabric Profiles*
- *Working with Versions*