



Red Hat JBoss Fuse 6.2

Tooling Tutorials

Building Solutions with Red Hat JBoss Fuse Tooling

Red Hat JBoss Fuse 6.2 Tooling Tutorials

Building Solutions with Red Hat JBoss Fuse Tooling

JBoss A-MQ Docs Team

Content Services

fuse-docs-support@redhat.com

Legal Notice

Copyright © 2015 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 Tooling to develop and test applications.

Table of Contents

CHAPTER 1. USING THE FUSE TOOLING RESOURCE FILES	4
PREREQUISITES	4
DOWNLOADING AND INSTALLING THE PREFABRICATED MESSAGE FILES	4
DOWNLOADING AND INSTALLING THE PREFABRICATED CAMEL CONTEXT FILES	4
CHAPTER 2. TO CREATE A NEW ROUTE	6
GOALS	6
PREREQUISITES	6
CREATING THE FUSE PROJECT	8
CREATING THE NEW ROUTING CONTEXT	10
CREATING THE ROUTE	12
CREATING TEST MESSAGES	15
NEXT STEPS	18
FURTHER READING	18
CHAPTER 3. TO RUN A ROUTE	19
GOALS	19
PREREQUISITES	19
RUNNING THE ROUTE	19
VERIFYING THE ROUTE	20
FURTHER READING	21
CHAPTER 4. TO ADD A CONTENT-BASED ROUTER	22
GOALS	22
PREREQUISITES	22
ADDING AND CONFIGURING A CONTENT-BASED ROUTER	22
ADDING AND CONFIGURING LOGGING	24
ADDING AND CONFIGURING MESSAGE HEADERS	24
ADDING AND CONFIGURING AN OTHERWISE BRANCH	25
NEXT STEPS	27
FURTHER READING	28
CHAPTER 5. TO ADD ANOTHER ROUTE TO THE CBR ROUTING CONTEXT	29
GOALS	29
PREREQUISITES	29
RECONFIGURING THE EXISTING ROUTE FOR DIRECT CONNECTION	29
ADDING THE SECOND ROUTE	30
BUILDING AND CONFIGURING THE USA BRANCH OF THE SECOND ROUTE	30
BUILDING AND CONFIGURING THE GREAT BRITAIN BRANCH OF THE SECOND ROUTE	32
BUILDING AND CONFIGURING THE GERMANY BRANCH OF THE SECOND ROUTE	33
BUILDING AND CONFIGURING THE FRANCE BRANCH OF THE SECOND ROUTE	34
SAVING THE NEW ROUTING CONTEXT	35
NEXT STEPS	37
FURTHER READING	38
CHAPTER 6. TO DEBUG A ROUTING CONTEXT	39
GOALS	39
PREREQUISITES	39
SETTING BREAKPOINTS	39
STEPPING THROUGH THE CBRROUTE ROUTING CONTEXT	41
CHANGING THE VALUE OF A VARIABLE	46
NEXT STEPS	51

CHAPTER 7. TO TRACE A MESSAGE THROUGH A ROUTE	52
GOALS	52
PREREQUISITES	52
ACCESSING FUSE INTEGRATION PERSPECTIVE	52
STARTING MESSAGE TRACING	54
DROPPING MESSAGES ON THE RUNNING CBRROUTE PROJECT	55
INITIALIZING AND CONFIGURING MESSAGES VIEW	56
ARRANGING DIAGRAM VIEW	58
STEPPING THROUGH MESSAGE TRACES	58
FINISHING UP	59
NEXT STEPS	60
CHAPTER 8. TO TEST A ROUTE WITH JUNIT	61
OVERVIEW	61
GOALS	61
PREREQUISITES	61
DELETING THE EXISTING TEST CASE	62
CREATING THE NEW TEST CASE	62
MODIFYING THE CAMELCONTEXTXMLTEST FILE	66
MODIFYING THE POM.XML FILE	69
RUNNING THE JUNIT TEST	72
FURTHER READING	73
CHAPTER 9. TO PUBLISH A FUSE PROJECT TO RED HAT JBOSS FUSE	74
GOALS	74
PREREQUISITES	74
DEFINING A RED HAT JBOSS FUSE SERVER	74
CONFIGURING THE PUBLISHING OPTIONS	78
STARTING UP RED HAT JBOSS FUSE SERVER	79
CONNECTING TO THE JBOSS FUSE 6.2 RUNTIME SERVER	80
UNINSTALLING THE CBRROUTE PROJECT	82

CHAPTER 1. USING THE FUSE TOOLING RESOURCE FILES

Abstract

Experienced users may want to focus only on the tutorials that demonstrate the tooling's new features. To do so, you need to download and install the requisite resource files. The prefabricated message files are used by all tutorials, but the prefabricated Camel Context files are specific to particular tutorials. With the exception of [Chapter 2, To Create a New Route](#) and [Chapter 8, To Test a Route with JUnit](#), using these prefabricated resource files enables you to complete most tutorials in any order. Without them, you must complete each tutorial sequentially, as the code generated by one tutorial is the starting point for the next tutorial.

PREREQUISITES

You must complete [Chapter 2, To Create a New Route](#), to create the project, the new routing context, and the folder that will hold the test messages. The code generated by this tutorial is used by [Chapter 3, To Run a Route](#) and by [Chapter 4, To Add a Content-Based Router](#).

You must successfully complete [Chapter 8, To Test a Route with JUnit](#) to generate a valid JUnit test case before you can successfully complete [Chapter 9, To Publish a Fuse Project to Red Hat JBoss Fuse](#).

DOWNLOADING AND INSTALLING THE PREFABRICATED MESSAGE FILES

Six prefabricated message files named `message1.xml`, `message2.xml`, ..., `message6.xml` are used in all of the tutorials. These files are provided in a downloadable `Messages.zip` file. Follow the instructions for downloading and installing the messages in [the section called "Creating test messages"](#).

DOWNLOADING AND INSTALLING THE PREFABRICATED CAMEL CONTEXT FILES

Two prefabricated Camel Context files named `camelContext5.xml`, and `camelContext6.xml` are used in one or more of the tutorials. These files are provided in a downloadable `.zip` file.

1. Click [here](#) to download the `camelContexts.zip` file.
2. Unzip the `camelContexts.zip` file in a convenient location external to the CBRroute project's workspace.
3. Delete the existing `camelContext.xml` file from the `CBRroute/src/main/resources/OSGI-INF/blueprint/` folder.
4. Install the `camelContext#.xml` file that corresponds to the tutorial that you want to complete in the vacated `CBRroute/src/main/resources/OSGI-INF/blueprint/` folder.

Use prefabricated Camel Context file:	To complete tutorials:
<code>camelContext5.xml</code>	To Add Another Route to the CBR Routing Context

Use prefabricated Camel Context file:	To complete tutorials:
camelContext6.xml	To Debug a Routing Context To Trace a Message Through a Route To Test a Route with JUnit

5. Rename the `camelContext#.xml` file `camelContext.xml`.
6. In the *Red Hat JBoss Fuse: Tooling Tutorialsguide*, follow the instructions for completing the target tutorial.

CHAPTER 2. 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 Fuse project
- create a new routing context
- create a route
 - add endpoints to the route
 - connect the endpoints
 - configure the endpoints
- create a folder in your project to store test messages that you create for your route
- create the test messages

PREREQUISITES



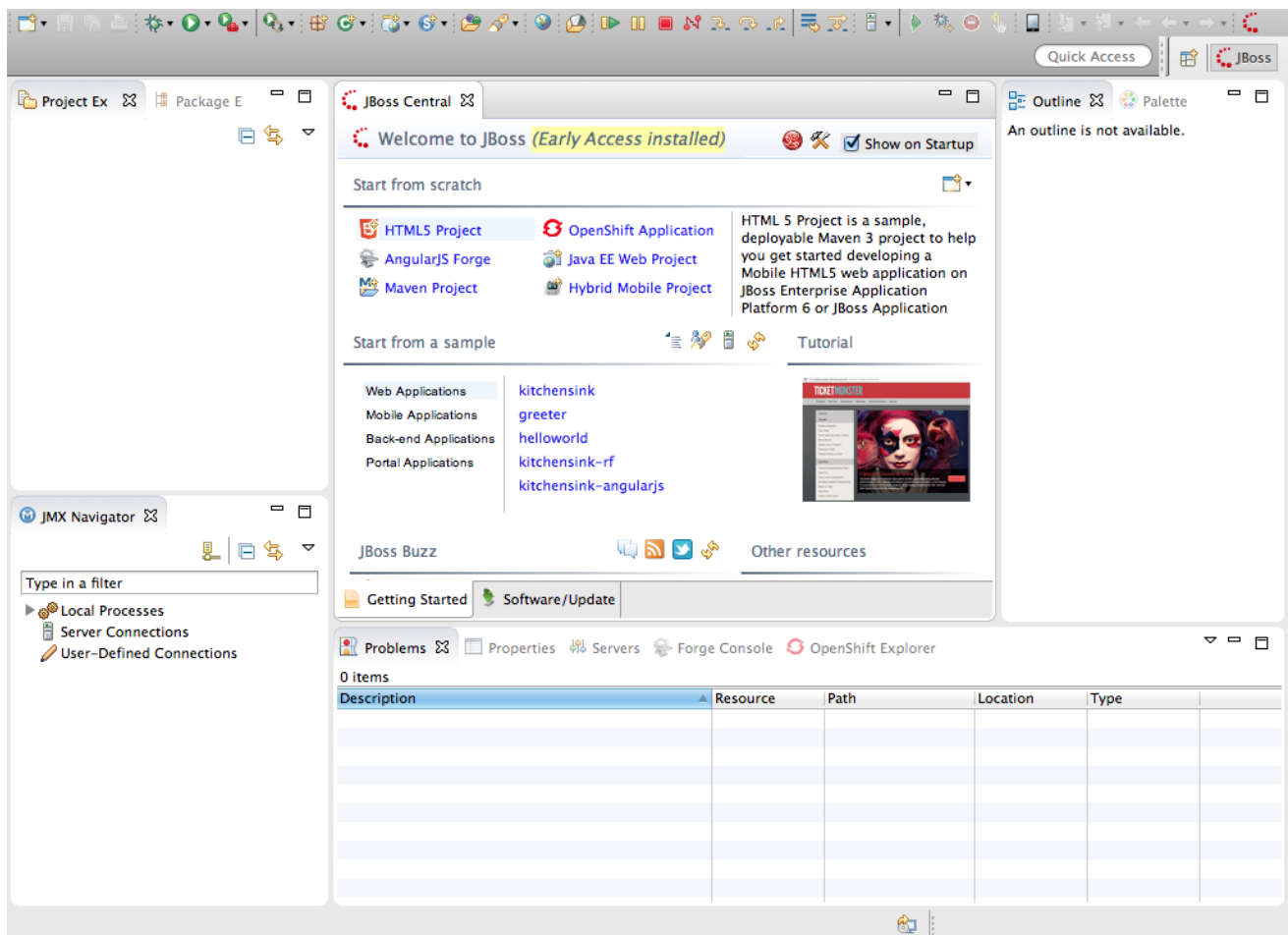
NOTE

You can use **Fuse Integration** perspective to work through all of the tutorials in this guide. However, because **JBoss** perspective provides more room for the route editor's canvas to expand as you build the routing context, this and other tutorials use **JBoss** perspective.

As you proceed through the remaining tutorials, you may find that you prefer using **Fuse Integration** perspective exclusively.

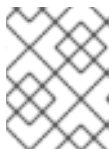
When you first start up JBoss Developer Studio, it opens in **JBoss** perspective, as shown in [Figure 2.1, “JBoss View on initial startup”](#).

Figure 2.1. JBoss View on initial startup



To provide more space for the canvas to expand as you build your projects:

1. Close the **JBoss Central** tab.



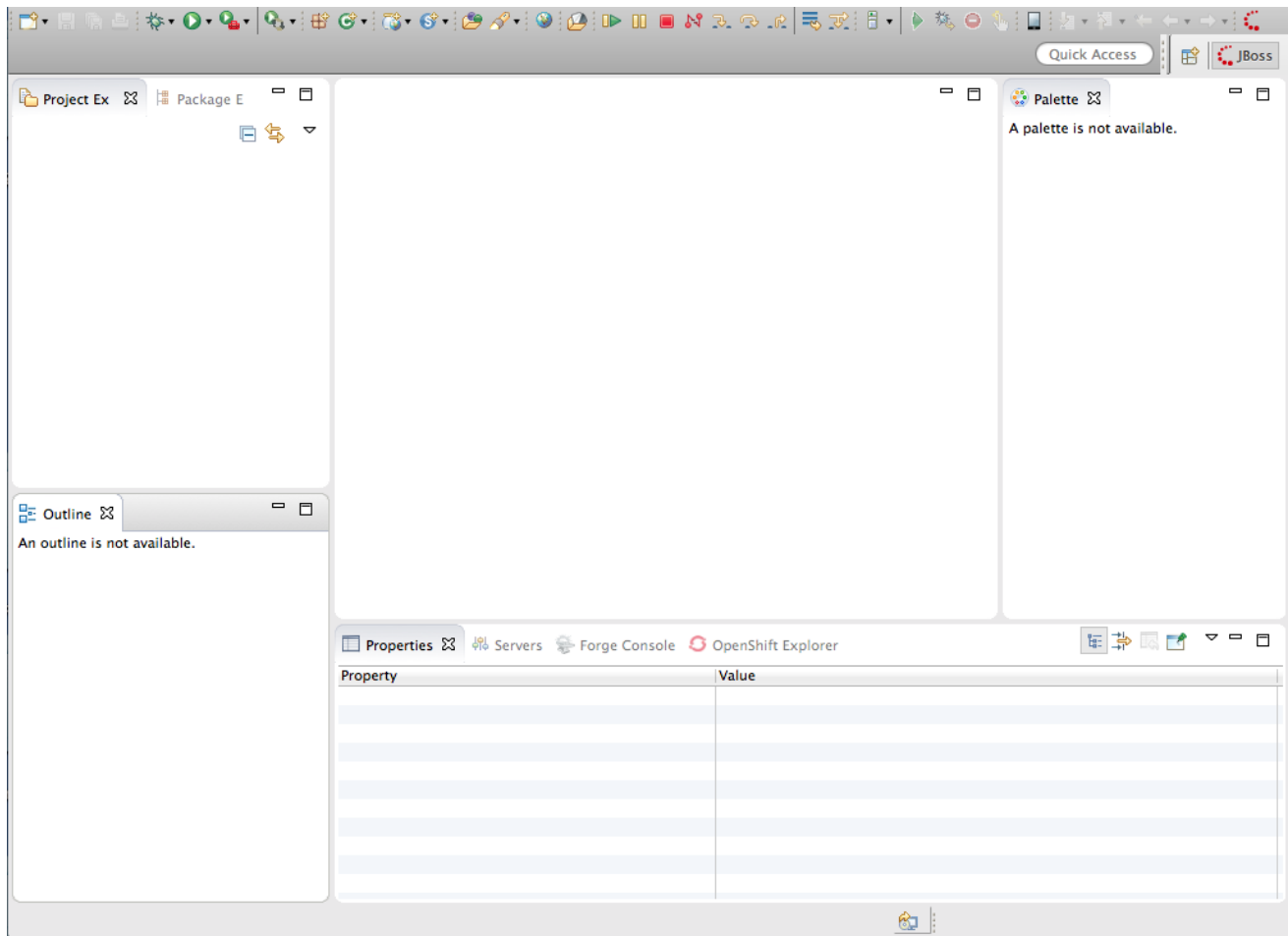
NOTE

You can reopen a view whenever you need it. You can also drag the border of a view or panel to increase or decrease the space it occupies in the workspace.

2. Close the **JMX Navigator** view at bottom, left of the workspace.
3. Drag **Outline** view from top, right of the workspace, and drop it in the spot previously occupied by the **JMX Navigator** view.

Your **JBoss** perspective should now look like that shown in [Figure 2.2, “JBoss View rearranged”](#):

Figure 2.2. JBoss View rearranged

**NOTE**

You can restore an open perspective to its original, default layout at any time by right-clicking the perspective's icon on the menubar to open its context menu, and then clicking **Reset**.

CREATING THE FUSE PROJECT

To create a Fuse project, in **JBoss** perspective:

1. On the Toolbar, select **File** → **New** → **Fuse Project** to open the **New Fuse** project wizard, as shown in [Figure 2.3](#).

Figure 2.3. New Fuse project location page

New Fuse project
Select project location

Project Name:

Use default Workspace location

Location:

Add project(s) to working set

Working set:

2. Enter **CBRoute** in the **Project Name** field.
3. Click **Next>** to open the **New Fuse Project** details page, as shown in [Figure 2.4](#).

Figure 2.4. 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	1.2.0.Beta4
io.fabric8	camel-cxf-contract-first-archetype	1.2.0.Beta4
io.fabric8	camel-drools-archetype	1.2.0.Beta4
io.fabric8	camel-webservice-archetype	1.2.0.Beta4
org.apache.camel.archetypes	camel-archetype-activemq	2.15.0
org.apache.camel.archetypes	camel-archetype-api-component	2.15.0
org.apache.camel.archetypes	camel-archetype-blueprint	2.15.0
org.apache.camel.archetypes	camel-archetype-component	2.15.0
org.apache.camel.archetypes	camel-archetype-cxf-code-first-...	2.15.0
org.apache.camel.archetypes	camel-archetype-cxf-contract-fir...	2.15.0
org.apache.camel.archetypes	camel-archetype-dataformat	2.15.0
org.apache.camel.archetypes	camel-archetype-java	2.15.0
org.apache.camel.archetypes	camel-archetype-scr	2.15.0
org.apache.camel.archetypes	camel-archetype-spring	2.15.0
org.apache.camel.archetypes	camel-archetype-spring-dm	2.15.0
org.apache.camel.archetypes	camel-archetype-war	2.15.0
org.apache.camel.archetypes	camel-archetype-web	2.15.0
org.apache.camel.archetypes	camel-archetype-webconsole	2.15.0
org.apache.cxf.archetype	cxf-jaxrs-service	2.7.11
org.apache.cxf.archetype	cxf-jaxws-javafirst	2.7.11

Creates a new Camel project with OSGi blueprint support. Ready to be deployed in OSGi.

Group Id:

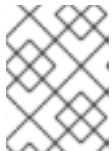
Artifact Id:

Version:

Package:

4. Select **camel-archetype-blueprint**.

5. Enter `tutorial` in the **Group Id:** field.
6. Enter `cbr-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.cbr.route`, the name of the package that contains `camel-archetype-blueprint`. To include the route in a different package, enter the name of that package.
9. Click **Finish**.



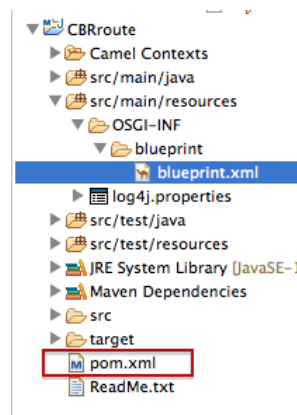
NOTE

Click **No** when the **Open Associated Perspective?** dialog asks whether you want to open the **Fuse Integration** perspective now.

This procedure creates a Fuse project, **CBRoute**, in **Project Explorer** that contains everything needed to create and run routes. As shown in [Figure 2.5](#), the files generated for **CBRoute** include:

- **CBRoute/pom.xml** (Maven project file)
- **CBRoute/src/main/resources/OSGI-INF/blueprint/blueprint.xml** (Blueprint XML file containing the routing rules)

Figure 2.5. Generated project files



NOTE

When you create a new project, the Fuse Tooling downloads from the Maven repository all of the files it needs to build the project. This operation can take several minutes.

CREATING THE NEW ROUTING CONTEXT

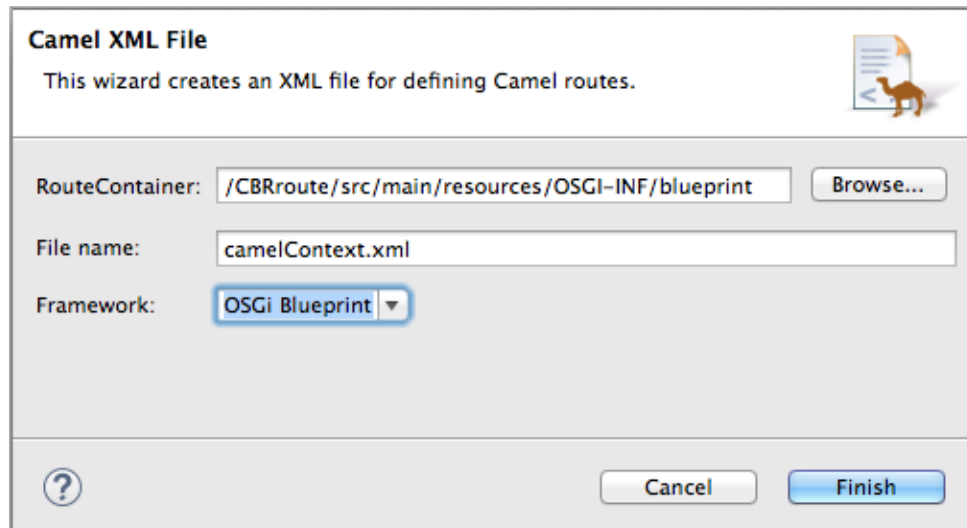
To create the new routing context:

1. In **Project Explorer**, locate **CBRoute/src/main/resources/OSGI-INF/blueprint/blueprint.xml**.
2. Right-click it to open the context menu, then select **Delete**.

You're going to replace the old `blueprint.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 `CBRroute/src/main/resources/OSGI-INF/blueprint`.
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 2.6](#).

Figure 2.6. Camel XML File wizard



7. Check that `/CBRroute/src/main/resources/OSGI-INF/blueprint` appears in the **Container:** field. Otherwise enter it manually, or select it using the **Browse...** button.



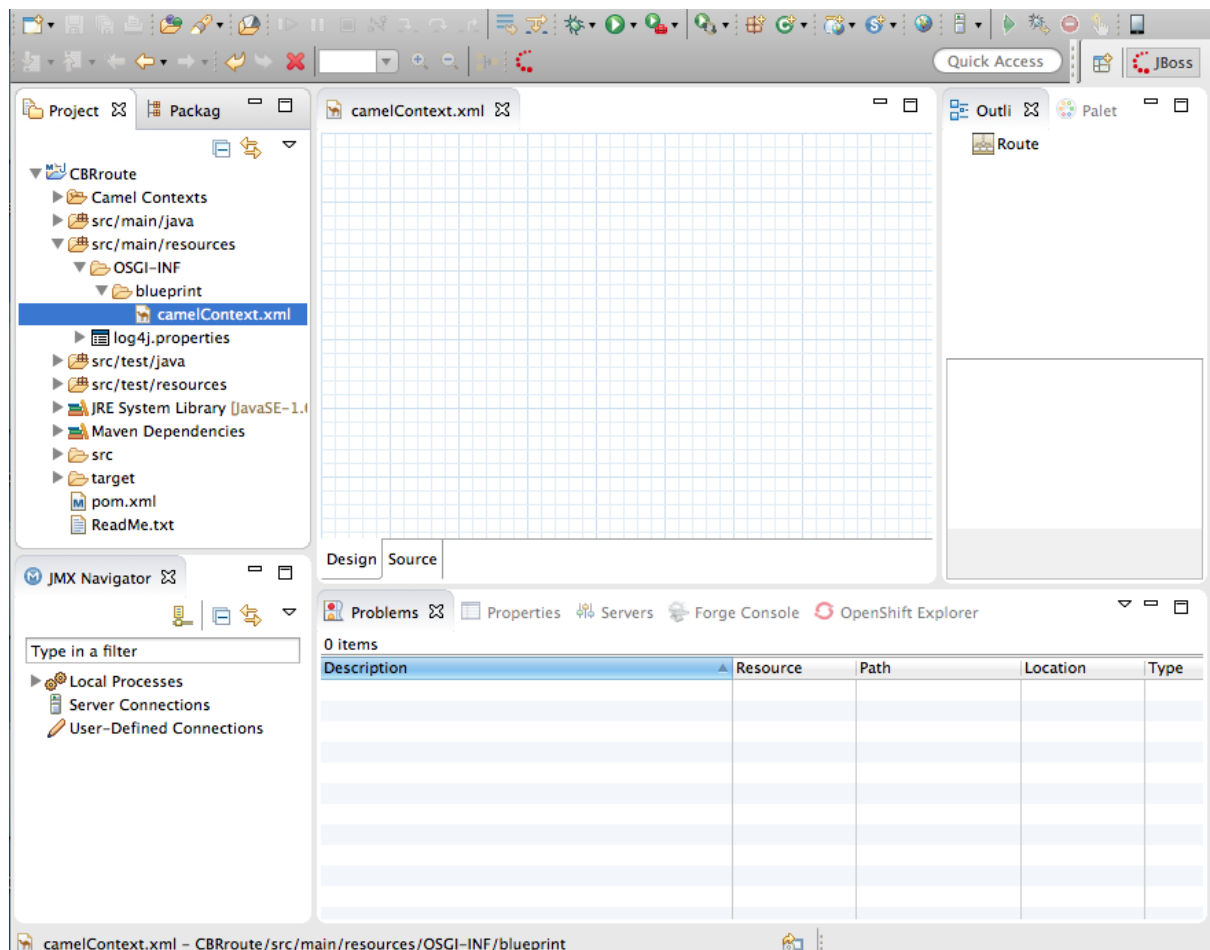
NOTE

The **Browse...** 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 **OSGi Blueprint** appears in the **Framework** field, or select it from the field's drop-down list.
10. Click **Finish**.

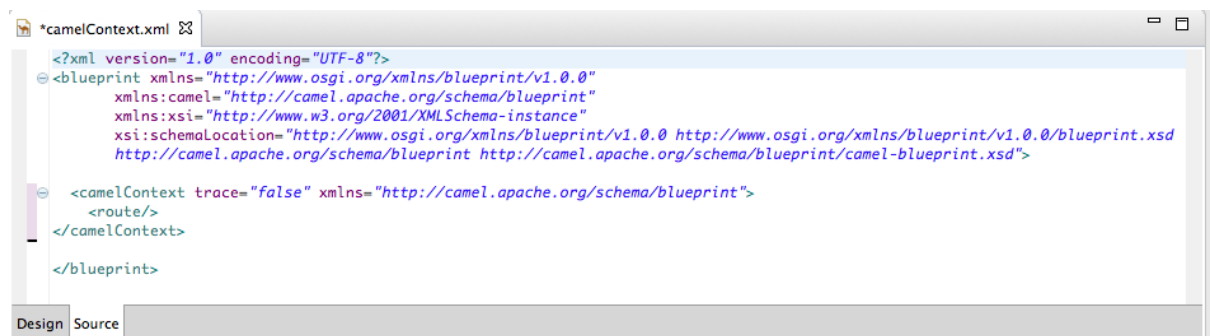
The `camelContext.xml` file opens in the route editor's **Design** view, displayed as an empty canvas, as shown in [Figure 2.7](#).

Figure 2.7. New camelContext .xml file in Design view




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 2.8, "New camelContext file in source view"](#).

Figure 2.8. New camelContext file in source view



CREATING THE ROUTE

To create the route:

1. Click the **Design** tab at the bottom, left of the canvas to return to the route editor's **Design** view.
2. Drag a **File** component () from the **PaLETTE's** Components drawer to the canvas.

**NOTE**

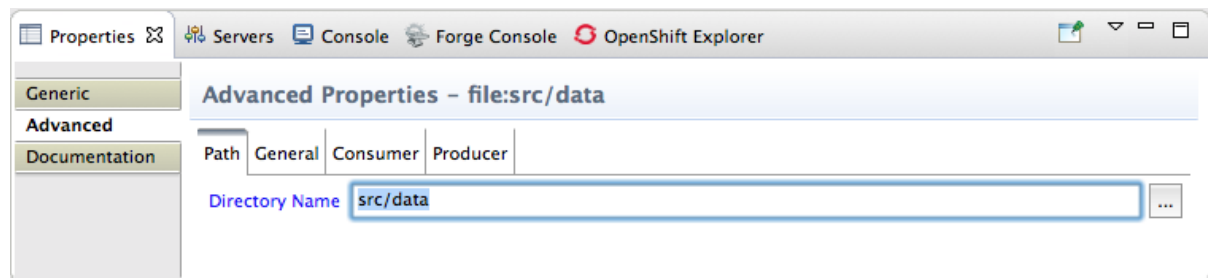
The **File** component changes to a `file:directoryName...` node on the canvas.

3. Drag another **File** component from the **Palette's Components** drawer to the canvas.
4. Select the first `file:directoryName` node you dragged onto the canvas.

The **Properties** editor, located below the canvas, displays the node's property fields for editing.

5. Select the **Advanced** tab, as shown in [Figure 2.9](#).

Figure 2.9. File source property editor



6. On the **Path** tab, click the button next to the **Directory Name** field, to browse to the `src/data` folder you previously created in your CBRroute project.
7. Click **Open**.

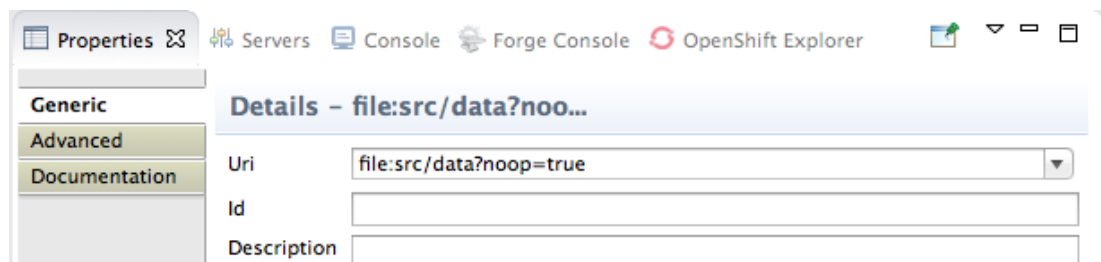
The full path appears in the **Directory Name** field.

8. Delete everything in the path string, except `src/data`.
9. Click the **Consumer** tab, and enable the **Noop** option by clicking its check box.

The **Noop** option prevents the `message#.xml` files being deleted from the `src/data` folder, and it enables idempotency to ensure that each `message#.xml` file is consumed only once.

10. Click the **Generic** tab to open the file node's **Details** page

Figure 2.10. File Details page

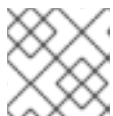
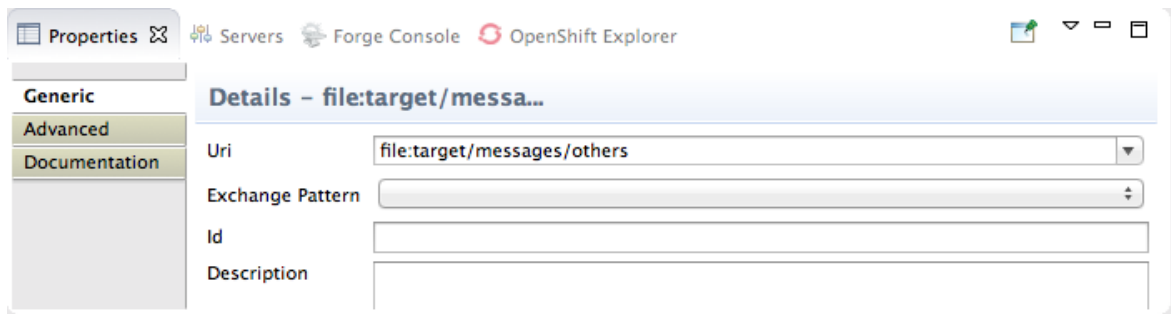


The tooling automatically populates the **Uri** field with the **Directory name** and **Noop** properties you configured on the **Advanced** tab.

11. Select the second `file:directoryName` node you dragged onto the canvas.


- In the **Generic** tab's **Uri** field, replace *directoryName* with **target/messages/others**. Leave the other fields blank.

Figure 2.11. File destination property editor



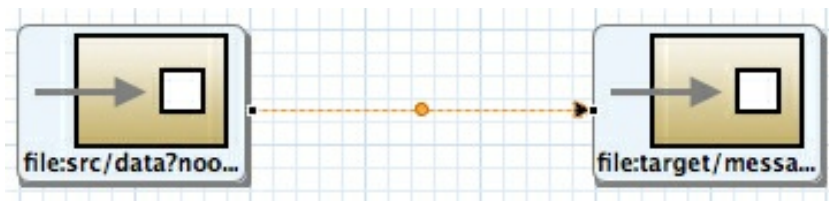
NOTE

The **target/messages/others** folder will be created at runtime.

- On the canvas, select the first file: node (**file:src/data?noop=true**), and drag its connector arrow () to the second file node (**file:target/messages/others**), then release it.

A segmented line connects the two endpoints, as shown in [Figure 2.12](#).

Figure 2.12. 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.

- To quickly align the connected endpoints, right-click the canvas to open the context menu, and then select **Layout Diagram**.
- Select **File** → **Save** to save the route.
- 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 2.1](#).

Example 2.1. XML for CBRroute

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
```

```

xmlns:camel="http://camel.apache.org/schema/blueprint"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.osgi.org/xmlns/blueprint/v1.0.0
http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd
http://camel.apache.org/schema/blueprint
http://camel.apache.org/schema/blueprint/camel-
blueprint.xsd">

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

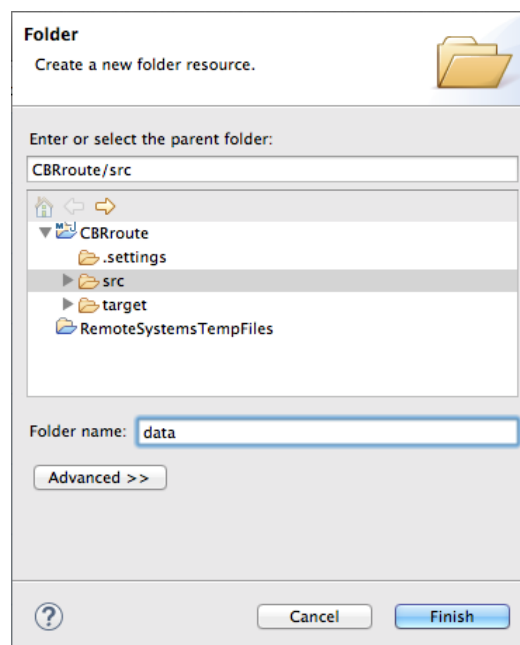
</blueprint>

```

CREATING TEST MESSAGES

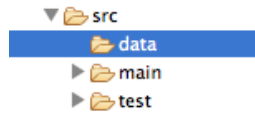
Before you can run your route, you need to create test messages to send through it.

1. In **Project Explorer**, right-click **CBRRoute/src** to open the context menu.
2. Select **New** → **Folder** to open the **New Folder** wizard:

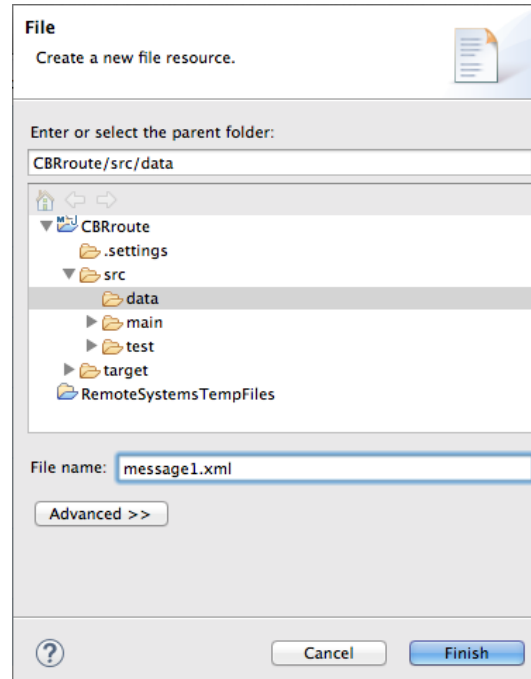


3. Check that **CBRRoute/src** appears in the **Enter or select the parent folder:** field. Otherwise enter it manually, or select it from the graphical representation of the project's hierarchy
4. In the **Folder name:** field, enter **data**, and then click **Finish**.

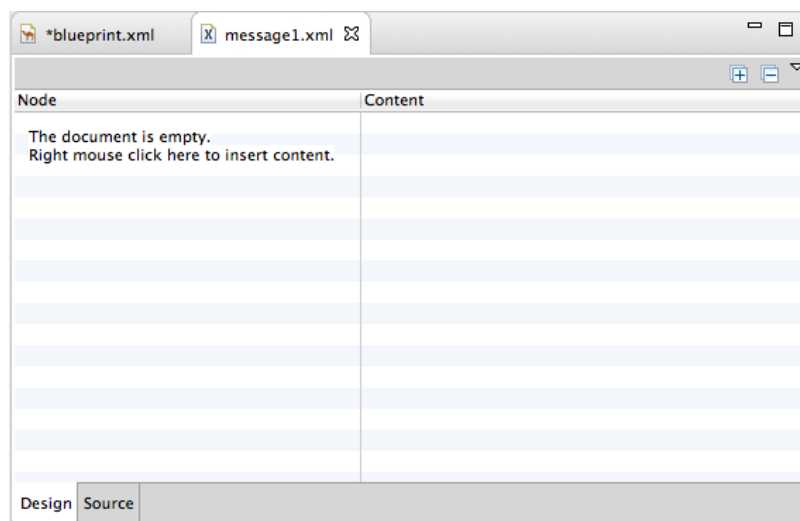
The new **data** folder appears in **Project Explorer**, under the **src** folder:



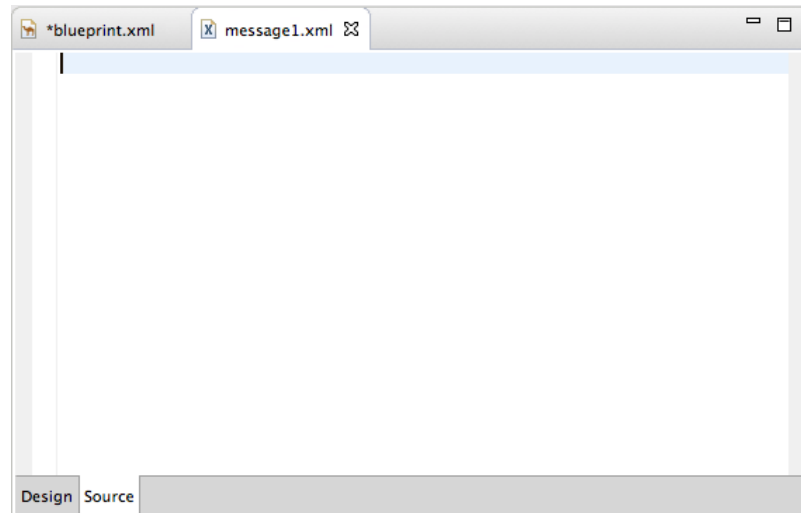
5. In **Project Explorer**, right-click **CBRroute** to open the context menu.
6. Click **New** → **Fuse Message** to open the **Fuse Message File** wizard:



7. Check that **CBRroute/src/data** appears in the **Enter or select the parent folder** field. Otherwise enter it manually, or select it from the graphical representation of the project's hierarchy.
8. In **File Name:**, enter **message1.xml**.
9. Click **Finish** to open the test message, **message1.xml**, in **Design View**:



10. Click the **Source** tab at the bottom, right of the canvas to switch to **Source view**:



11. In **Source** view, enter this text:

```
<?xml version="1.0" encoding="UTF-8"?>
<order>
  <customer>
    <name>Brooklyn Zoo</name>
    <city>Brooklyn</city>
    <country>USA</country>
  </customer>
  <orderline>
    <animal>wombat</animal>
    <quantity>15</quantity>
    <maxAllowed>25</maxAllowed>
  </orderline>
</order>
```

12. Save the file.

13. Click [Messages.zip](#) to download the five remaining preconstructed test message files (`message2.xml` through `message6.xml`), and then unpack them into the `CBRroute/src/data` folder. You will use all six test messages in the remaining Fuse Tooling tutorials.

[Table 2.1](#) shows the contents of each preconstructed message file.

Table 2.1. Preconstructed test messages

msg#	<name>	<city>	<country>	<animal>	<quantity>	<maxAllowed>
2	San Diego Zoo	San Diego	USA	giraffe	3	2
3	London Zoo	London	Great Britain	penguin	12	20

msg#	<name>	<city>	<country>	<animal>	<quantity>	<maxAllowed>
4	Bristol Zoo	Bristol	Great Britain	emu	5	4
5	Paris Zoo	Paris	France	giraffe	2	2
6	Hellabrunn Gardens	Munich	Germany	penguin	18	20

NEXT STEPS

After you have created and designed your route, you can run it by deploying it into your local Apache Camel runtime, as described in [Chapter 3, 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/documentation/en-US/Red_Hat_JBoss_Fuse/6.2/html/Tooling_User_Guide/RiderEditRoute.html
- Apache Camel endpoints, see [Red Hat JBoss Fuse: Component Reference](#).

CHAPTER 3. 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 (without tests)
- send messages through the route
- examine the messages received by the endpoints

PREREQUISITES

To complete this tutorial you will need the CBRroute project created in [Chapter 2, To Create a New Route](#).

RUNNING THE ROUTE

To run the route:

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



NOTE

If you select **Local Camel Context** instead, the tooling automatically runs the routing context against the supplied JUnit test, and it will fail. In the [Chapter 8, To Test a Route with JUnit](#) tutorial, you will replace the supplied JUnit test with one you create for the this project.

The **Console** panel opens to display log messages that reflect the progress of the project's execution. At the beginning, Maven downloads the resources necessary to update the local Maven repository, which may take a few minutes.

Messages similar to the following indicate that the route executed successfully.

```
[INFO] Starting Camel ... [mel.test.blueprint.Main.main()]
MainSupport INFO Apache Camel 2.13.2 starting
[mel.test.blueprint.Main.main()] Activator INFO Camel activator
starting [mel.test.blueprint.Main.main()] Activator INFO Camel
activator started [mel.test.blueprint.Main.main()] BlueprintExtender
INFO No quiesce support is available, so blueprint components will
```

```

not participate in quiesce operations [ Blueprint Extender: 1]
BlueprintContainerImpl INFO Bundle cbr-route is waiting for
namespace handlers [http://camel.apache.org/schema/blueprint] [
Blueprint Extender: 1] BlueprintCamelContext INFO Apache Camel
2.13.2 (CamelContext: blueprintContext) is starting [ Blueprint
Extender: 1] ManagedManagementStrategy INFO JMX is enabled
[heysmbp.home:1099/jmxrmi/camel] DefaultManagementAgent INFO JMX
Connector thread started and listening at:
service:jmx:rmi:///jndi/rmi://janemurpheysmbp.home:1099/jmxrmi/camel
[ Blueprint Extender: 1] BlueprintCamelContext INFO
AllowUseOriginalMessage is enabled. If access to the original
message is not needed, then its recommended to turn this option off
as it may improve performance. [ Blueprint Extender: 1]
BlueprintCamelContext 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 [ Blueprint
Extender: 1] FileEndpoint INFO Endpoint is configured with noop=true
so forcing endpoint to be idempotent as well [ Blueprint Extender:
1] FileEndpoint INFO Using default memory based idempotent
repository with cache max size: 1000 [ Blueprint Extender: 1]
XPathBuilder INFO Created default XPathFactory
com.sun.org.apache.xpath.internal.jaxp.XPathFactoryImpl@46e48d4 [
Blueprint Extender: 1] BlueprintCamelContext INFO Route: Route1
started and consuming from: Endpoint[file://src/data?noop=true] [
Blueprint Extender: 1] BlueprintCamelContext INFO Total 1 routes, of
which 1 is started. [ Blueprint Extender: 1] BlueprintCamelContext
INFO Apache Camel 2.13.2 (CamelContext: blueprintContext) started in
1.073 seconds

```

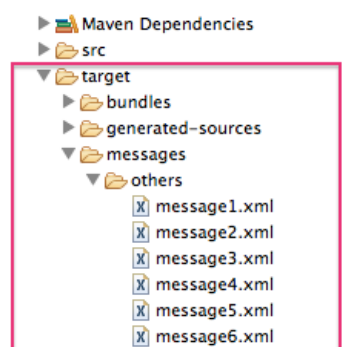
4. To shutdown the route, click  located at the top, right of the **Console** panel.

VERIFYING THE ROUTE

To verify that the route executed properly:

1. In **Project Explorer**, select **CBRoute**.
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 3.1](#).

Figure 3.1. Target message destination in Project Explorer tree



4. Verify that the `target/messages/others` folder contains the six message files, `message1.xml` through `message6.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 3.1](#).

Example 3.1. Contents of `message1.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<order>
  <customer>
    <name>Brooklyn Zoo</name>
    <city>Brooklyn</city>
    <country>USA</country>
  </customer>
  <orderline>
    <animal>wombat</animal>
    <quantity>15</quantity>
    <maxAllowed>25</maxAllowed>
  </orderline>
</order>
```

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/documentation/en-US/Red_Hat_JBoss_Fuse/6.2/html/Tooling_User_Guide/RiderEditRunProfile.html.
- deploying Apache Camel applications see [Red Hat JBoss Fuse: Deploying into the Container](#) .

CHAPTER 4. 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 your route
- configure the content-based router
 - add a log endpoint to each output branch of the content-based router
 - add a SetHeader EIP after each log endpoint
 - add an Otherwise branch to the content-based router

PREREQUISITES

To complete this tutorial you will need the **CBRoute** project you created in [Chapter 2, To Create a New Route](#).

ADDING AND CONFIGURING A CONTENT-BASED ROUTER

To add and configure a content-based router for your route:

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




NOTE

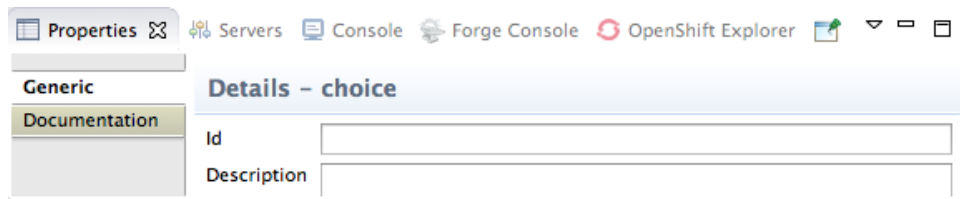
Alternatively, you can delete the connector by selecting it, then selecting **Delete** from the toolbar's **Edit** menu.

4. On the canvas, select the terminal **file:** node, **file:target/messages/others**, and in the **Properties** editor, change the **Uri** and **Id** properties to:
 - **Uri:** `-file:target/messages/validOrders`
 - **Id:** `-toValid`

Then drag the node out of the way. You will connect it to another node later in this tutorial.

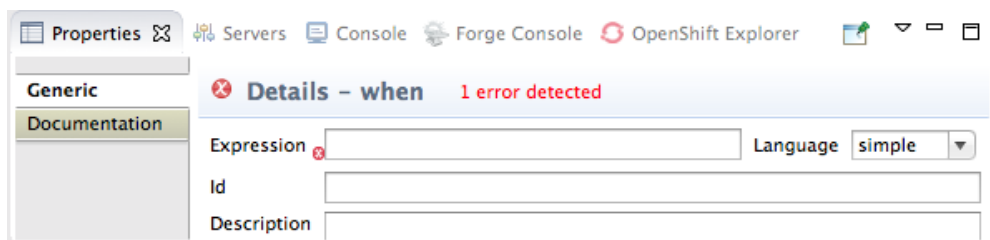
- On the canvas, select the starting `file: node`, `file:src/data?noop=true`, and right-click it to open the context menu.
- Select **Add** → **Routing** → **Choice**.

A **choice** node () appears on the canvas connected to the starting `file: node`.




- In the **Properties** editor, enter `choice1` in the **Id** field.
- On the canvas, select the **choice** node, then right-click it to open the context menu.
- 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:



NOTE

When a required property is blank, the **Properties** editor marks it with . The number of properties that require configuring is displayed in the title bar. Error icons and message disappear when you configure the required properties.

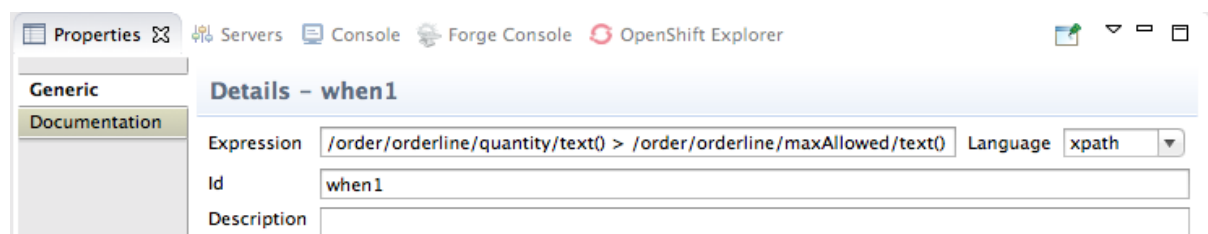
- In the **Expression** field, enter `/order/orderline/quantity/text() > /order/orderline/maxAllowed/text()`.

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

- From the **Language** drop-down menu, select `xpath`.
- In the **Id** field, enter `when1`.

[Figure 4.1](#) shows the `when1` node configured.

Figure 4.1. when1 configuration



ADDING AND CONFIGURING LOGGING

To add logging to your route:

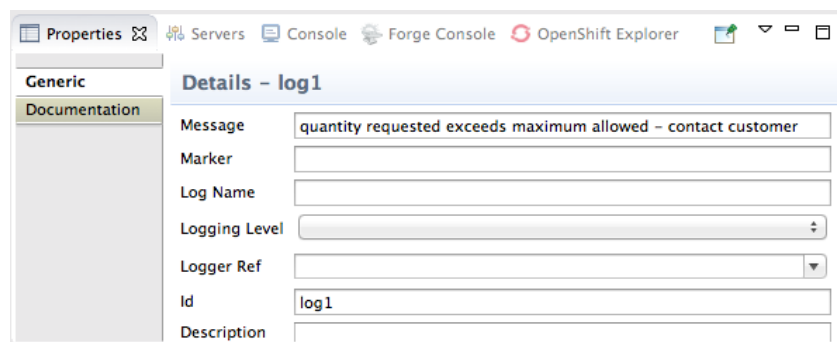
1. On the canvas, select the **when1** node, and then right-click it to open the context menu.
2. Select **Add** → **Components** → **Log**.

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.

3. In the **Message** field, enter **quantity requested exceeds the maximum allowed - contact customer**.
4. In the **Id** field, enter **log1**.

Figure 4.2 shows the **log1** node configured.

Figure 4.2. Log1 configuration




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 7.11, “Fuse Integration perspective's message tracing components”](#)). In the **Console**, it adds the contents of the log node's **Message** field to the log data whenever the route runs.

ADDING AND CONFIGURING MESSAGE HEADERS

To add and configure message headers:

1. On the canvas, select the **log1** node, and then right-click it to open the context menu.
2. Select **Add** → **Transformation** → **SetHeader**.

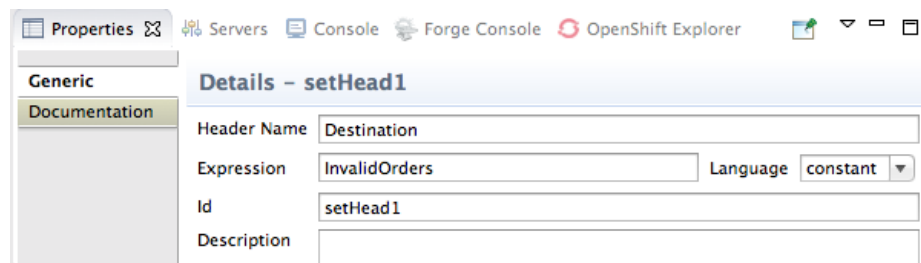
A **setHeader** node () appears on the canvas, connected to the **log1** node. The **Properties** editor opens, displaying the **setHeader** node's property fields for you to edit, as shown:



3. In the **Header Name** field, enter **Destination**.
4. In the **Expression** field, enter **InvalidOrders**.
5. Select **constant** from the **Language** drop-down menu.
6. In the **Id** field, enter **setHead1**.

Figure 4.3 shows the **setHead1** node configured.

Figure 4.3. setHead1 configuration



7. On the canvas, select the **setHead1** node, and then right-click it to open the context menu.
8. Select **Add** → **Components** → **File**.

A **file:directoryName** node appears on the canvas, connected to the **setHead1** node. The **Properties** editor opens, displaying the **file:directoryName** node's property fields for you to edit.

9. On the **Generic** tab, replace *directoryName* with **target/messages/invalidOrders** in the **Uri** field, and enter **toInvalid** in the **Id** field.

ADDING AND CONFIGURING AN OTHERWISE BRANCH

To add and configure the otherwise branch to your route:

1. On the canvas, reselect the **Choice** node, then right-click it to open the context menu.
2. Select **Add** → **Routing** → **Otherwise**.

An **otherwise** node appears on the canvas, connected to the **choice** node. The **Properties** editor opens, displaying the **otherwise** node's property fields for you to edit.

3. In the **Id** field, enter **else2**.

The **else2** node will eventually route to the terminal **file:** node (**file:target/messages/validOrders**) any message that does not match the XPath expression set for the **when1** node.

4. On the canvas, select the `else2` node, and then right-click it to open the context menu.

5. Select **Add** → **Components** → **Log**.

A `log` node appears on the canvas, connected to the `else2` node. The **Properties** editor opens, displaying the `log` node's property fields for you to edit.

6. In the **Message** field, enter `valid order - process`, and in the **Id** field, enter `log2`.

7. On the canvas, select the `log2` node, and then right-click it to open the context menu.

8. Select **Add** → **Transformation** → **SetHeader**.

A `setHeader` node () appears on the canvas, connected to the `log2` node. The **Properties** editor opens, displaying the `setHeader` node's property fields for you to edit.

9. In the **Header Name** field, enter `Destination`.

10. In the **Expression** field, enter `Dispatcher`.

11. Select `constant` from the **Language** drop-down menu.

12. In the **Id** field, enter `setHead2`.

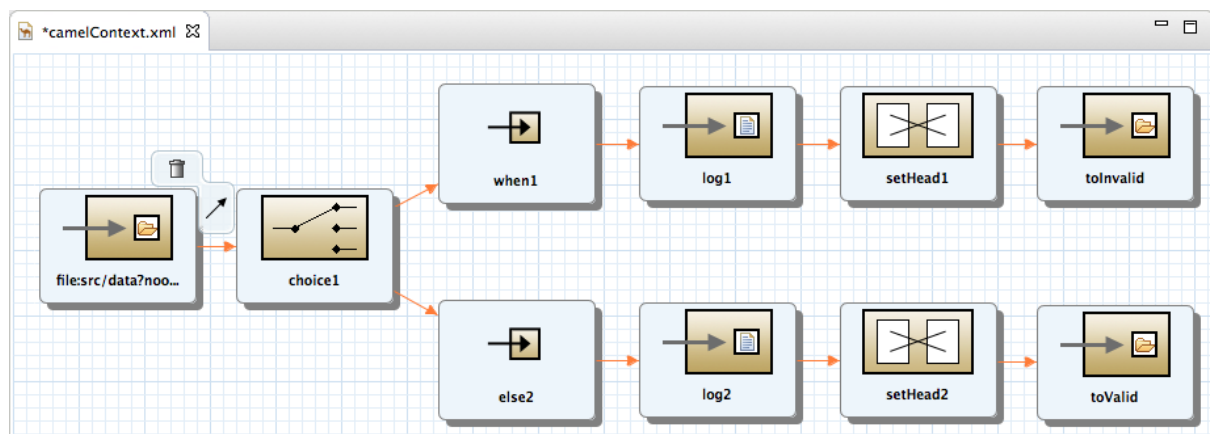
13. On the canvas, drag the terminal `file:` node, `file:target/messages/validOrders`, close to the `setHead2` node.

14. Select the `setHead2` node, and then drag its connector arrow () to the terminal `file:` node and release it.

15. 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 4.4](#).

Figure 4.4. Completed content-based router with logs and message headers



16. On the toolbar, select **File** → **Save** to save the completed route.

17. 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 4.1](#).

■

Example 4.1. XML for content-based router

```

<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
  xmlns:camel="http://camel.apache.org/schema/blueprint"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

  xsi:schemaLocation="http://www.osgi.org/xmlns/blueprint/v1.0.0

  http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd
    http://camel.apache.org/schema/blueprint
      http://camel.apache.org/schema/blueprint/camel-
blueprint.xsd">

  <camelContext trace="false"
  xmlns="http://camel.apache.org/schema/blueprint">
    <route>
      <from uri="file:src/data?noop=true"/>
      <choice id="choice1">
        <when id="when1">
          <xpath>/order/orderline/quantity/text() >
/order/orderline/maxAllowed/text()</xpath>
          <log message="quantity requested exceeds maximum allowed
- contact customer" id="log1"/>
          <setHeader headerName="Destination" id="setHead1">
            <constant>InvalidOrders</constant>
          </setHeader>
          <to uri="file:target/messages/invalidOrders"
id="toInvalid"/>
        </when>
        <otherwise id="else2">
          <log message="valid order - process" id="log2"/>
          <setHeader headerName="Destination" id="setHead2">
            <constant>Dispatcher</constant>
          </setHeader>
          <to uri="file:target/messages/validOrders" id="toValid"/>
        </otherwise>
      </choice>
    </route>
  </camelContext>
</blueprint>

```

NEXT STEPS

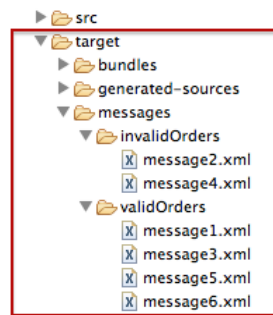
You can run the new route as described in [the section called “Running the route”](#).

After you run it, you can easily verify whether the route executed properly by checking the target destinations in **Project Explorer**:

1. Select **CBRroute**.
2. Right-click it to open the context menu, then select **Refresh**.

- Under the project root node (**CBRRoute**), locate the folder **target/messages/** and expand it, as shown in [Figure 4.5](#).

Figure 4.5. Target message destinations in Project Explorer



- Check that the **target/messages/invalidOrders** folder contains **message2.xml** and **message4.xml**.

In these messages, the value of the quantity element should exceed the value of the `maxAllowed` element.

- Check that the **target/messages/validOrders** folder contains the four message files that contain valid orders: **message1.xml**, **message3.xml**, **message5.xml** and **message6.xml**.

In these messages, the value of the quantity element should be less than or equal to the value of the `maxAllowed` element.



NOTE

To view message content, double-click each message to open it in the route editor's xml editor.

FURTHER READING

To learn more about message enrichment see:

- the [Red Hat JBoss Fuse: Apache Camel Development Guide](#)
- the [Red Hat JBoss Fuse 6.x documentation](#)

CHAPTER 5. TO ADD ANOTHER ROUTE TO THE CBR ROUTING CONTEXT

Abstract

This tutorial walks you through adding a second route to the `camelContext.xml` file in the `CBRroute` project. The second route:

- takes messages directly from the terminal end of the first route's otherwise branch
- sorts the messages according to customers' country
- sends each message to the corresponding `CBRroute/target/messages/<country>` directory

GOALS

In this tutorial you will:

- reconfigure the existing route for direct connection to a second route
- add a second route to your `camelContext`
- configure the new route to take messages directly from the otherwise branch of the first route
- add a content-based router to the new route
- add and configure a message header, logging, and target destination to each output branch of the new route's content-based router

PREREQUISITES

To complete this tutorial you will need the `CBRroute` project you modified in [Chapter 4, To Add a Content-Based Router](#).



NOTE

If you skipped any tutorial after [Chapter 2, To Create a New Route](#), you can use the prefabricated `camelContext5.xml` file to work through this tutorial (for details, see [Chapter 1, Using the Fuse Tooling Resource Files](#)).

RECONFIGURING THE EXISTING ROUTE FOR DIRECT CONNECTION

To configure the existing route for direct connection with the new route:

1. Open your `CBRroute/src/main/resources/OSGI-INF/blueprint/camelContext.xml` in the route editor.
2. Click the canvas to display the existing route's properties in the **Properties** editor.
3. Enter `Route1` in the **Id** field.

4. Select the terminal file: `node file:target/messages/toValid` to display its properties in the **Properties** editor.
5. In the **Uri** field, delete the existing text, and then enter `direct:OrderFulfillment`.
6. In the **Id** field, enter `toFulfill`.

ADDING THE SECOND ROUTE



NOTE

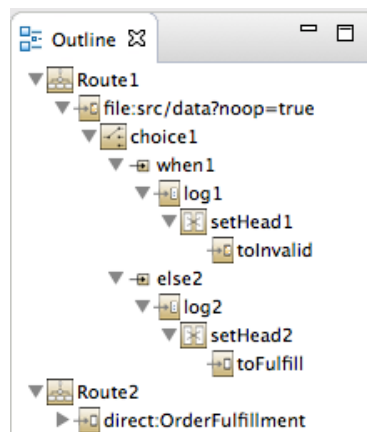
The route editor displays each route in a multiroute routing context on its own slice of canvas.

To add a route to the routing context:

1. Select **Routes** → **Add Route**.

The tooling adds another route to your camelContext, and the route editor opens a clean canvas for you to construct the second route.

Outline view, shown here, displays both routes and their components. Clicking on a route in **Outline** view displays it on the route editor's canvas.



Or you can switch between routes by selecting **Routes** → **Route:RouteName** on the menu bar, where *RouteName* is the string you entered in the route's **Id** field in the **Properties** editor.

2. Click the canvas to display the new route's properties in the **Properties** editor.
3. Enter `Route2` in the **Id** field.

BUILDING AND CONFIGURING THE USA BRANCH OF THE SECOND ROUTE

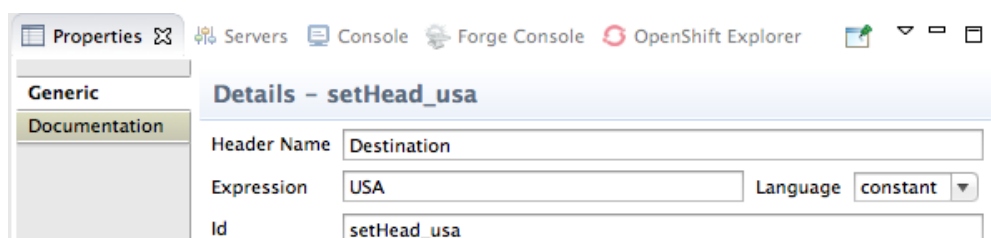
With `Route2` displayed on the route editor's canvas:

1. Drag an **Generic** element () from the **Palette's Components** drawer onto the canvas.
2. In the **Properties** editor, enter `direct:OrderFulfillment` in the **Uri** field.

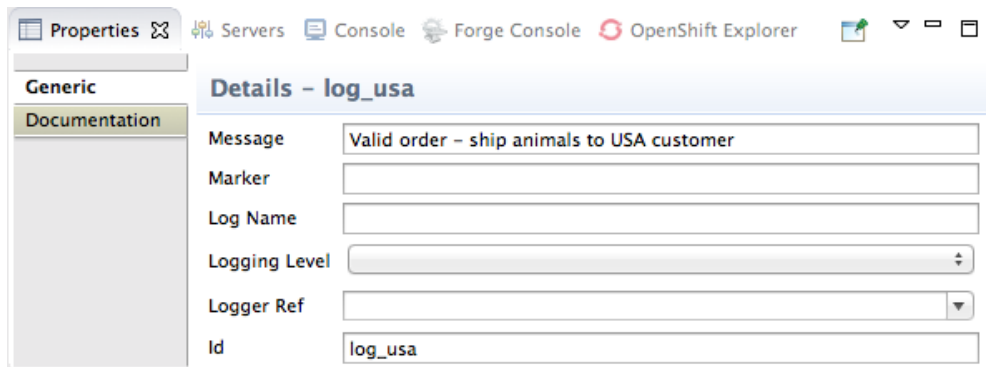
3. Right-click the `direct:OrderFulfi...` node to open the context menu, and select **Add** → **Routing** → **Choice**.
4. In the **Properties** editor, enter `choice2` in the **Id** field.
5. Right-click the `choice2` node to open the context menu, and select **Add** → **Routing** → **When**.
6. In the **Properties** editor:



- Enter `/order/customer/country = 'USA'` in the **Expression** field.
 - Select `xpath` from the **Language** drop-down menu.
 - Enter `when/usa` in the **Id** field.
7. Right-click the `when/usa` node to open the context menu, and select **Add** → **Transformation** → **SetHeader**.
 8. In the **Properties** editor:



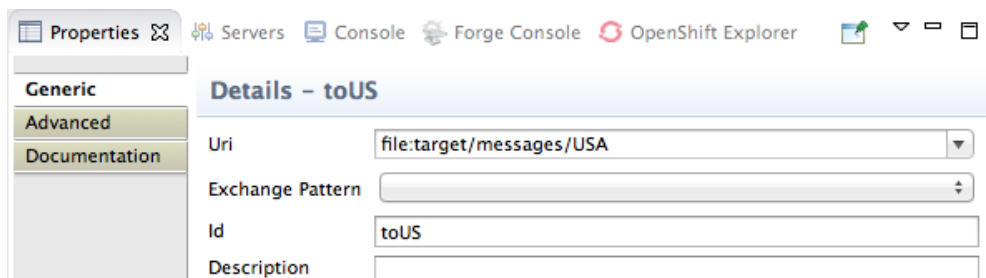
- Enter `Destination` in the **Header Name** field.
 - Enter `USA` in the **Expression** field.
 - Select `constant` from the **Language** drop-down menu.
 - Enter `setHead_usa` in the **Id** field,
9. Right-click the `setHead_usa` node to open the context menu, and select **Add** → **Components** → **Log**.
 10. In the **Properties** editor:



- Enter **Valid order - ship animals to USA customer** in the **Message** field.
- Enter **log_usa** in the **Id** field.

11. Right-click the **log_usa** node to open the context menu, and select **Add → Components → File**.

12. In the **Properties** editor:



- Replace *directoryName* with **target/messages/USA** in the **Uri** field.
- Enter **toUS** in the **Id** field.

The USA branch of **Route2** should look like this:



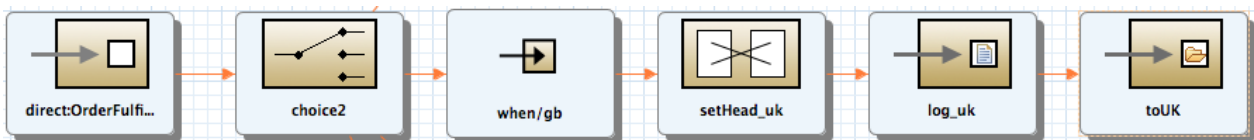
BUILDING AND CONFIGURING THE GREAT BRITAIN BRANCH OF THE SECOND ROUTE

With **Route2** displayed on the canvas:

1. Right-click the **choice2** node again to open the context menu, and select **Add → Routing → When**.
2. In the **Properties** editor:
 - Enter **/order/customer/country = 'Great Britain'** in the **Expression** field.
 - Select **xpath** from the **Language** drop-down menu.
 - Enter **when/gb** in the **Id** field.

3. Right-click the `when/gb` node to open the context menu, and select **Add** → **Transformation** → **SetHeader**.
4. In the **Properties** editor:
 - Enter **Destination** in the **Header Name** field.
 - Enter **UK** in the **Expression** field.
 - Select **constant** from the **Language** drop-down menu.
 - Enter `setHead_uk` in the **Id** field,
5. Right-click the `setHead_uk` node to open the context menu, and select **Add** → **Components** → **Log**.
6. In the **Properties** editor:
 - Enter **Valid order - ship animals to UK customer** in the **Message** field.
 - Enter `log_uk` in the **Id** field.
7. Right-click the `log_uk` node to open the context menu, and select **Add** → **Components** → **File**.
8. In the **Properties** editor:
 - Replace *directoryName* with `target/messages/GreatBritain` in the **Uri** field.
 - Enter `toUK` in the **Id** field.

The Great Britain branch of `Route2` should look like this:



BUILDING AND CONFIGURING THE GERMANY BRANCH OF THE SECOND ROUTE

With `Route2` displayed on the canvas:

1. Right-click the `choice2` node again to open the context menu, and select **Add** → **Routing** → **When**.
2. In the **Properties** editor:
 - Enter `/order/customer/country = 'Germany'` in the **Expression** field.
 - Select **xpath** from the **Language** drop-down menu.
 - Enter `when/ger` in the **Id** field.
3. Right-click the `when/ger` node to open the context menu, and select **Add** → **Transformation** → **SetHeader**.

4. In the **Properties** editor:

- Enter **Destination** in the **Header Name** field.
- Enter **Germany** in the **Expression** field.
- Select **constant** from the **Language** drop-down menu.
- Enter **setHead_ger** in the **Id** field,

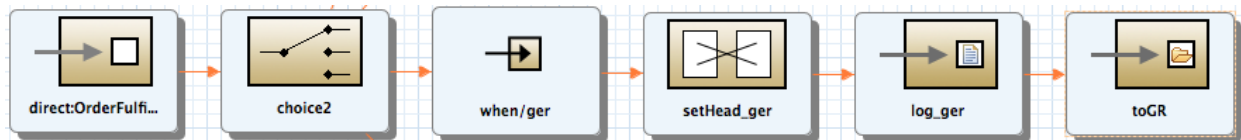
5. Right-click the **setHead_ger** node to open the context menu, and select **Add** → **Components** → **Log**.6. In the **Properties** editor:

- Enter **Valid order - ship animals to Germany customer** in the **Message** field.
- Enter **log_ger** in the **Id** field.

7. Right-click the **log_ger** node to open the context menu, and select **Add** → **Components** → **File**.8. In the **Properties** editor:

- Replace *directoryName* with **target/messages/Germany** in the **Uri** field.
- Enter **toGR** in the **Id** field.

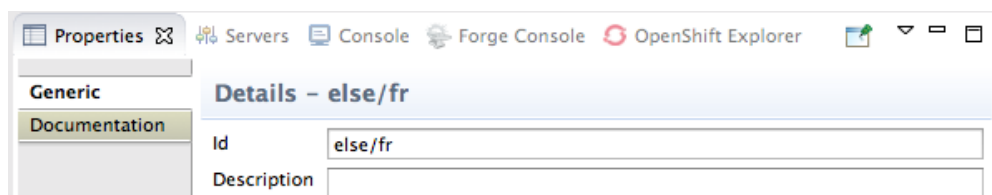
The Germany branch of **Route2** should look like this:



BUILDING AND CONFIGURING THE FRANCE BRANCH OF THE SECOND ROUTE

With **Route2** displayed on the canvas:

1. Right-click the **choice2** node again to open the context menu, and select **Add** → **Routing** → **Otherwise**.
2. In the **Properties** editor:



Enter **else/fr** in the **Id** field.

3. Right-click the **else/fr** node to open the context menu, and select **Add** → **Transformation** → **SetHeader**.

4. In the **Properties** editor:

- Enter **Destination** in the **Header Name** field.
- Enter **France** in the **Expression** field.
- Select **constant** from the **Language** drop-down menu.
- Enter **setHead_fr** in the **Id** field,

5. Right-click the **setHead_fr** node to open the context menu, and select **Add** → **Components** → **Log**.

6. In the **Properties** editor:

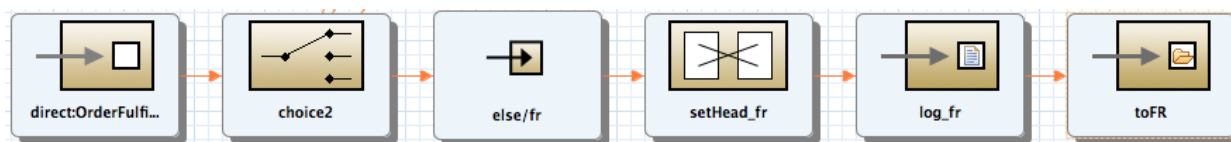
- Enter **Valid order - ship animals to France customer** in the **Message** field.
- Enter **log_fr** in the **Id** field.

7. Right-click the **log_fr** node to open the context menu, and select **Add** → **Components** → **File**.

8. In the **Properties** editor:

- Replace *directoryName* with **target/messages/France** in the **Uri** field.
- Enter **toFR** in the **Id** field.

The France branch of **Route2** should look like this:



SAVING THE NEW ROUTING CONTEXT

1. On the toolbar, select **File** → **Save** to save the routing context.

The routes on the canvas should look like this:

Figure 5.1. Completed route1

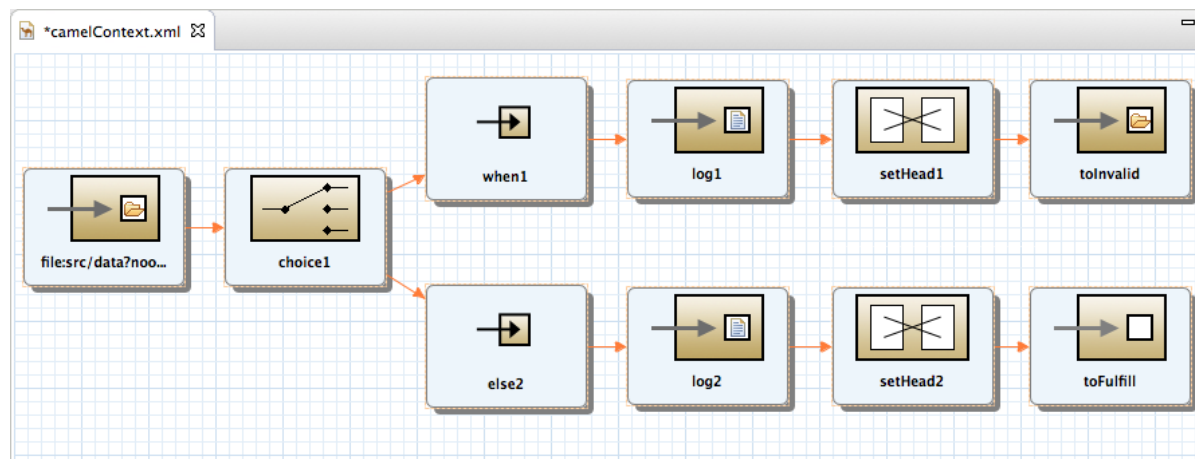
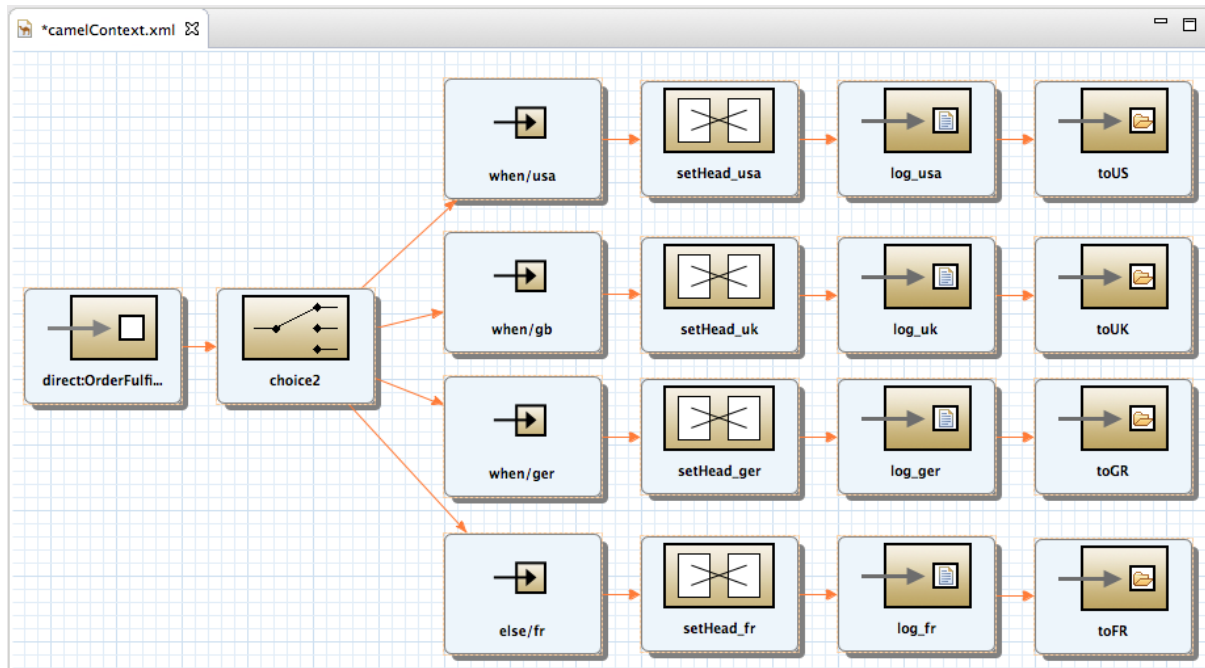


Figure 5.2. Completed route2



2. Click the **Source** tab at the bottom, left of the canvas to display the XML for the route.

The `camelContext` element should look like that shown in [Example 5.1](#).

Example 5.1. XML for dual-route content-based router

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
  xmlns:camel="http://camel.apache.org/schema/blueprint"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

  xsi:schemaLocation="http://www.osgi.org/xmlns/blueprint/v1.0.0
    http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd
    http://camel.apache.org/schema/blueprint
    http://camel.apache.org/schema/blueprint/camel-
    blueprint.xsd">

  <camelContext trace="false"
    xmlns="http://camel.apache.org/schema/blueprint">
    <route id="Route1">
      <from uri="file:src/data?noop=true"/>
      <choice id="choice1">
        <when id="when1">
          <xpath>/order/orderline/quantity/text() >
/ order/orderline/maxAllowed/text()</xpath>
          <log message="quantity requested exceeds maximum allowed
- contact customer" id="log1"/>
          <setHeader headerName="Destination" id="setHead1">
            <constant>InvalidOrders</constant>
          </setHeader>
          <to uri="file:target/messages/invalidOrders"
id="toInvalid"/>
        </when>
        <otherwise id="else2">
          <log message="valid order - process" id="log2"/>

```



```

        <setHeader headerName="Destination" id="setHead2">
            <constant>Dispatcher</constant>
        </setHeader>
        <to uri="direct:OrderFulfillment" id="toFulfill"/>
    </otherwise>
</choice>
</route>
<route id="Route2">
    <from uri="direct:OrderFulfillment"/>
    <choice id="choice2">
        <when id="when/usa">
            <xpath>/order/customer/country = 'USA'</xpath>
            <setHeader headerName="Destination" id="setHead_usa">
                <constant>USA</constant>
            </setHeader>
            <log message="Valid order - ship animals to USA customer"
id="log_usa"/>
            <to uri="file:target/messages/USA" id="toUS"/>
        </when>
        <when id="when/gb">
            <xpath>/order/customer/country = 'Great Britain'</xpath>
            <setHeader headerName="Destination" id="setHead_uk">
                <constant>UK</constant>
            </setHeader>
            <log message="Valid order - ship animals to UK customer"
id="log_uk"/>
            <to uri="file:target/messages/GreatBritain" id="toUK"/>
        </when>
        <when id="when/ger">
            <xpath>/order/customer/country = 'Germany'</xpath>
            <setHeader headerName="Destination" id="setHead_ger">
                <constant>Germany</constant>
            </setHeader>
            <log message="Valid order - ship animals to Germany
customer" id="log_ger"/>
            <to uri="file:target/messages/Germany" id="toGR"/>
        </when>
        <otherwise id="else/fr">
            <setHeader headerName="Destination" id="setHead_fr">
                <constant>France</constant>
            </setHeader>
            <log message="Valid order - ship animals to France
customer" id="log_fr"/>
            <to uri="file:target/messages/France" id="toFR"/>
        </otherwise>
    </choice>
</route>
</camelContext>
</blueprint>

```

NEXT STEPS

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

Check the end of the Console's output. You should see these lines:

```

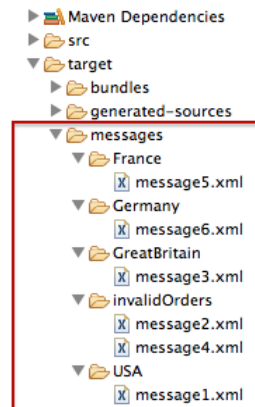
/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (Jan 14, 2015, 7:39:49 PM)
[ Blueprint Extender: 1] BlueprintCamelContext INFO Total 2 routes, of which 2 is started.
[ Blueprint Extender: 1] BlueprintCamelContext INFO Apache Camel 2.13.2 (CamelContext: blueprintContext) started in :
[t] thread #2 - file://src/data] Route1 INFO valid order - process
[t] thread #2 - file://src/data] Route2 INFO Valid order - ship animals to USA customer
[t] thread #2 - file://src/data] Route1 INFO quantity requested exceeds maximum allowed - contact customer
[t] thread #2 - file://src/data] Route1 INFO valid order - process
[t] thread #2 - file://src/data] Route2 INFO Valid order - ship animals to UK customer
[t] thread #2 - file://src/data] Route1 INFO quantity requested exceeds maximum allowed - contact customer
[t] thread #2 - file://src/data] Route2 INFO valid order - process
[t] thread #2 - file://src/data] Route2 INFO Valid order - ship animals to France customer
[t] thread #2 - file://src/data] Route1 INFO valid order - process
[t] thread #2 - file://src/data] Route2 INFO Valid order - ship animals to Germany customer

```

Check the target destinations in **Project Explorer** to verify that the routes executed properly:

1. Select **CBRRoute**.
2. Right-click it to open the context menu, then select **Refresh**.
3. Expand the folder **target/messages/** as shown in [Figure 5.3](#). The message* .xml files should be dispersed in your target destinations like this:

Figure 5.3. Target message destinations in Project Explorer



NOTE

To view message content, double-click a message to open it in the route editor's xml editor.

FURTHER READING

To learn more about the direct component see the *Red Hat JBoss Fuse: Apache Camel Component Reference* at [Red Hat JBoss Fuse 6.x documentation](#)

CHAPTER 6. TO DEBUG A ROUTING CONTEXT

Abstract

The Camel debugger works only on the routing context. The routing context and each node with a breakpoint set must have a unique ID. You can set them manually or let the tooling set them automatically.

GOALS

In this tutorial you will:

- In **Source** view, enter a unique ID for the `camelContext` element
- In **Design** view, set breakpoints on the nodes of interest in `Route1`
- Switch to `Route2`, and set breakpoints on the nodes of interest
- Invoke the Camel debugger
- Step through the route, examining route and message variables as they change
- Step through the route again, changing the value of message variables and observing the effects

PREREQUISITES

To complete this tutorial you will need the **CBRRoute** project you updated in [Chapter 5, To Add Another Route to the CBR Routing Context](#).



NOTE

If you skipped any tutorial after [Chapter 2, To Create a New Route](#) you can use the prefabricated `camelContext6.xml` file to work through this tutorial (for details, see [Chapter 1, Using the Fuse Tooling Resource Files](#)).

SETTING BREAKPOINTS


You can set both conditional and unconditional breakpoints, but in this tutorial, you will set unconditional breakpoints only.

1. If necessary, open your `CBRRoute/src/main/resources/OSGI-INF/blueprint/camelContext.xml` in the route editor.

By default, the route editor displays `Route1` on the canvas.



2. Click the **Source** tab at the bottom of the route editor to switch to **Source** view.
3. In the `camelContext` element, add `id="blueprintContext"` like this:

```
<camelContext trace="false" id="blueprintContext"
  xmlns="http://camel.apache.org/schema/blueprint">
```

4. Click **File** → **Save** to save your routing context file.
5. Click the **Design** tab at the bottom of the route editor to switch to Design view.
6. Select the **choice1** node, and then click its  icon to set an unconditional breakpoint.



NOTE

In the route editor, you can disable or delete a specific breakpoint by clicking the node's  icon or its  icon, respectively. You can delete all set breakpoints by right-clicking the canvas and selecting **Delete all breakpoints**.



NOTE

If you had not already set unique ID on the `camelContext` element and the nodes in your routing context, the **Please Confirm** dialog would have appeared now prompting you to let the tooling do it for you.

7. Repeat [Step 6](#) to set an unconditional breakpoint on the following `Route1` nodes:
 - `log1`
 - `setHead1`
 - `toInvalid`
 - `log2`
 - `setHead2`
 - `toFulfill`
8. Click **File** → **Save** to save your routing context file.
9. Click **Routes** → **Route2** to open `Route2` on the canvas.
10. Repeat [Step 6](#) to set an unconditional breakpoint on the following `Route2` nodes:
 - `choice2`
 - `setHead_usa`
 - `log_usa`
 - `toUSA`
 - `setHead_uk`

- `log_uk`
- `toUK`
- `setHead_ger`
- `log_ger`
- `toGR`
- `setHead_fr`
- `log_fr`
- `toFR`

11. Click **File** → **Save** to save your routing context file.





NOTE

Each time you modify your routing context, be sure to save it before you invoke the Camel debugger, otherwise the changes you made will not be included in the debugging session.


STEPPING THROUGH THE CBRROUTE ROUTING CONTEXT

You can step through the routing context in two ways:

- **Step over** ()—Jumps to the next node of execution in the routing context, regardless of breakpoints.
- **Resume** ()—Jumps to the next active breakpoint in the routing context.



NOTE

You can temporarily narrow then later re-expand the debugger's focus by disabling and re-enabling the breakpoints you set in the routing context. This enables you, for example, to focus on problematic nodes in your routing context. To do so, open the **Breakpoints** tab and clear the check box of each breakpoint you want to temporarily disable. Then use  to step through the route. It will skip over the disabled breakpoints.

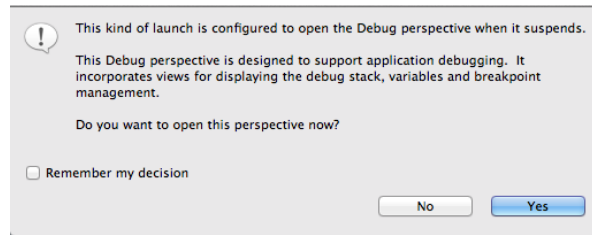
1. In **Project Explorer**, expand the root node **CBRroute** to expose the `camelContext.xml` file in the **Camel Contexts** folder.
2. Right-click the `camelContext.xml` file to open its context menu, and then click **Debug As...** → **Local Camel Context (without tests)**.



NOTE

If you select **Local Camel Context**, it will fail to run because you have not yet created a JUnit test for the CBRroute project. You will do that later in [Chapter 8, To Test a Route with JUnit](#)

The Camel debugger suspends execution at the first breakpoint it encounters and asks whether you want to open **Debug** perspective now.



3. Click Yes.



NOTE

If you click **No**, the confirmation pane appears several more times. After the third refusal, it disappears, and the Camel debugger resumes execution. To interact with the debugger at this point, you need to open the **Debug** perspective by clicking **Window** → **Open Perspective** → **Debug**.

Debug perspective opens with the routing context suspended at **choice1** in **Route1 [camelContext]** as shown in **Debug** view.

The screenshot displays the JBoss Fuse IDE interface. The top toolbar includes icons for running, debugging, and other actions. The 'Servers' view on the left shows a list of running threads and the Camel Context. The 'Variables' view on the right shows the current state of the Camel message, including the exchange and message headers. The 'Design' view in the center shows a Camel route diagram with a 'choice1' breakpoint highlighted in a red dashed box. The 'Console' view at the bottom shows log output from the BacklogDebugger.



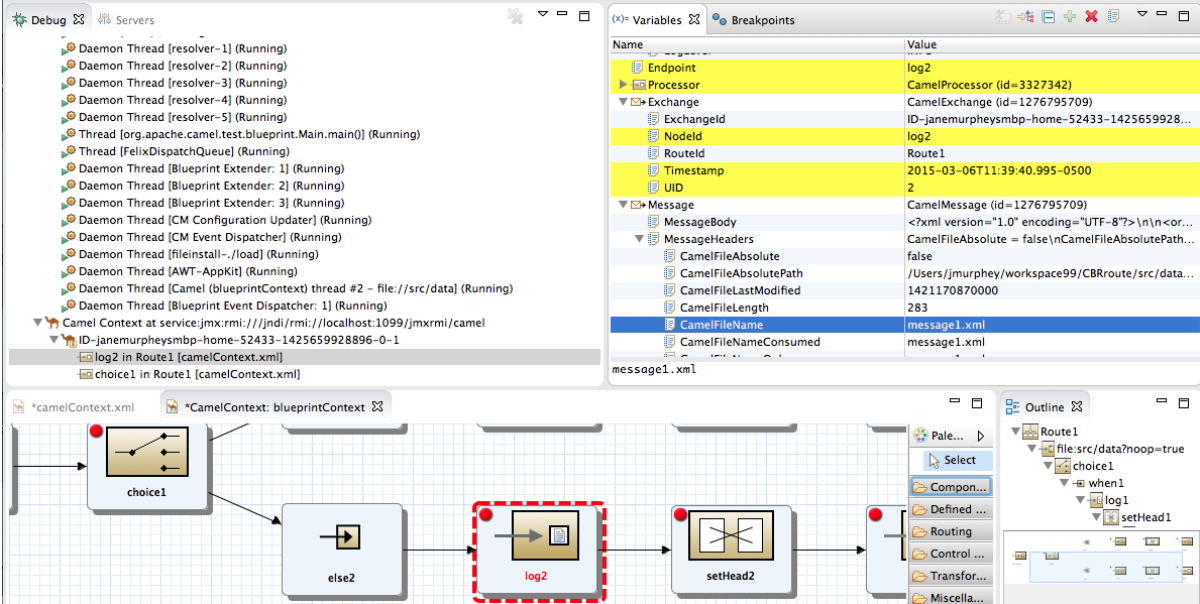
NOTE

Breakpoints are held for a maximum of five minutes before the debugger automatically resumes, moving on to the next breakpoint or to the end of the routing context, whichever comes next.

- In **Variables** view, expand the nodes to expose the variables and values available for each node.

As you step through the routing context, the variables whose values have changed since the last breakpoint are highlighted in yellow. You may need to expand the nodes at each breakpoint to reveal variables that have changed.

- Click  to step to the next breakpoint, **log2** in **Route1 [camelContext]**.



The screenshot displays the Camel IDE interface. On the left, the **Debug** view shows a list of threads and the current message flow. The **Variables** view on the right shows the state of the message at the **log2** breakpoint. The **Routing Context** diagram at the bottom shows the message flow through **choice1**, **else2**, **log2**, and **setHead2**. The **log2** node is highlighted with a red dashed box, indicating it is the current breakpoint.

Name	Value
Endpoint	log2
Processor	CamelProcessor (id=3327342)
Exchange	CamelExchange (id=1276795709)
ExchangeId	ID-janemurpheysmbp-home-52433-1425659928...
NodeId	log2
RouteId	Route1
Timestamp	2015-03-06T11:39:40.995-0500
UID	2
Message	CamelMessage (id=1276795709)
MessageBody	<?xml version="1.0" encoding="UTF-8"?>\n\n<cor...
MessageHeaders	CamelFileAbsolute = false\nCamelFileAbsolutePath...
CamelFileAbsolute	/Users/jmurphey/workspace99/CBRroute/src/data...
CamelFileAbsolutePath	1421170870000
CamelFileLastModified	283
CamelFileLength	message1.xml
CamelFileName	message1.xml
CamelFileNameConsumed	message1.xml

- Expand the nodes in **Variables** view to examine the variables that have changed since the last breakpoint at **choice1** in **Route1 [camelContext]**.

- Click  to step to the next breakpoint, **setHead2** in **Route1 [camelContext]**.

Examine the variables that changed since the breakpoint at **log2** in **Route1 [camelContext]**.

- In **Debug** view, click **log2** in **Route1 [camelContext]** to populate **Variables** view with the variable values from the breakpoint **log2** in **Route1 [camelContext]** for a quick comparison.

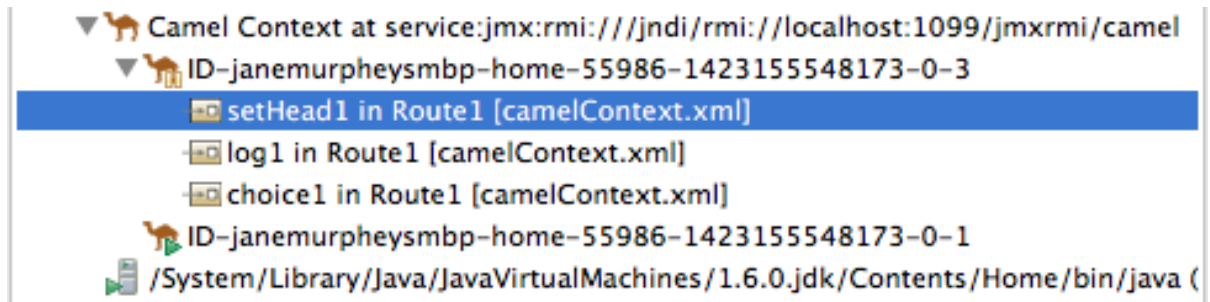
In **Debug** view, you can switch between breakpoints within the same message flow to quickly compare and monitor changing variable values in **Variables** view.



NOTE

Message flows can vary in length. For messages that transit the **invalidOrders** branch of **Route1**, the message flow is short. For messages that transit the **validOrders** branch of **Route1**, which continues on to **Route2**, the message flow is longer.

- Continue stepping through the routing context. When one message completes the routing context and the next message enters it, the new message flow appears in **Debug** view, tagged with a new breadcrumb ID.



In this case, `ID-janemurpheysmbp-home-55986-1423155548173-0-3` identifies the second message flow, corresponding to `message2.xml` having entered the routing context. Breadcrumb IDs are incremented by 2.



NOTE

Exchange and Message IDs are identical and remain unchanged throughout a message's passage through the routing context. Their IDs are constructed from the message flow's breadcrumb ID, and incremented by 1. So, in the case of `message2.xml`, its `ExchangeId` and `MessageId` are `ID-janemurpheysmbp-home-55986-1423155548173-0-4`.

- When `message3.xml` enters the breakpoint `choice1 in Route1 [camelContext]`, examine the Processor variables. The values displayed are the metrics accumulated for `message1.xml` and `message2.xml`, which previously transited the routing context.

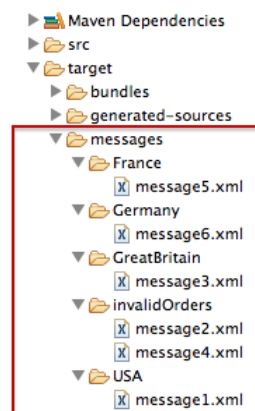
- In **Project Explorer**, open the project's context menu, and select **Refresh** to refresh the display.



NOTE

If you terminated the session prematurely, before all messages transited the routing context, you might see, under the **CBRroute/CamelContexts/** folder, a file that looks like `this:target/.CamelContextInDebug_XXXXXXXXXXXXXXXXXXXX_temp/camelContext.xml`. To remove it, open **Project Explorer**'s context menu and click **Refresh**.

- Expand the **CBRroute/target/messages/*** directories to check that the messages were delivered to their expected destinations:



- Leave the routing context as is, with all previous breakpoints set and enabled.

CHANGING THE VALUE OF A VARIABLE

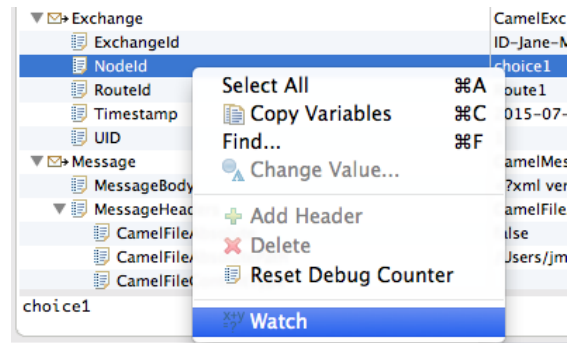
In this session, you will add variables to a watch list to easily check how their values change as messages pass through the routing context. You will also change the value of a variable in the body of two messages and observe how the change affects each message's route through the routing context.

Follow [Step 1](#) through [Step 3](#) in the section called “Stepping through the CBRroute routing context” to rerun the Camel debugger on the CBRroute project.

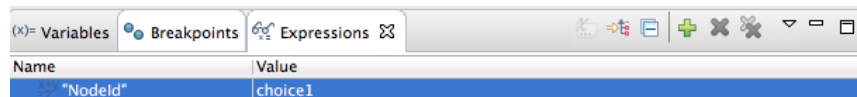
- With `message1` stopped at the first breakpoint, `choice1` in `Route1` [`camelContext.xml`], add the variables `NodeId` and `RouteId` (in the **Exchange** category) and `MessageBody` and `CamelFileName` (in the **Message** category) to the watch list.

For each of the four variables:

- In **Variables** view, expand the appropriate category to expose the target variable:
- Right-click the variable (in this case, `NodeId` in the **Exchange** category) to open the context menu and select **Watch**:

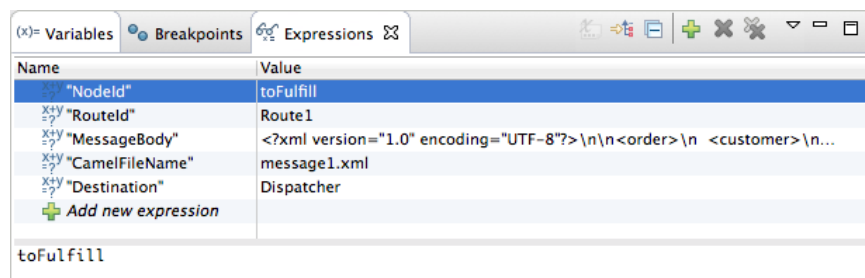


The **Expressions** tab opens, listing the variable you selected to watch:



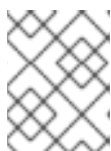
- c. Repeat [Step 1.b](#) for each of the three remaining variables.
 - d. Switch back to **Variables** view.
2. Step message1 through the routing context until it reaches the fourth breakpoint, **toFulfill** in **Route1 [camelContext.xml]**.
 3. In **Variables** view, expand the **Message** category.
 4. Repeat [Step 1.b](#) to add the variable **Destination** to the watch list.

Expressions view should now contain these variables:



NOTE

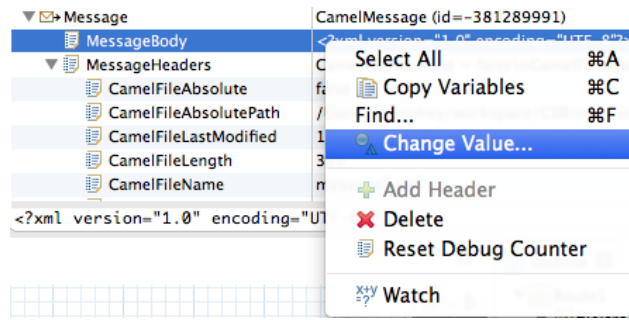
The pane below the list of variables displays the value of the selected variable.



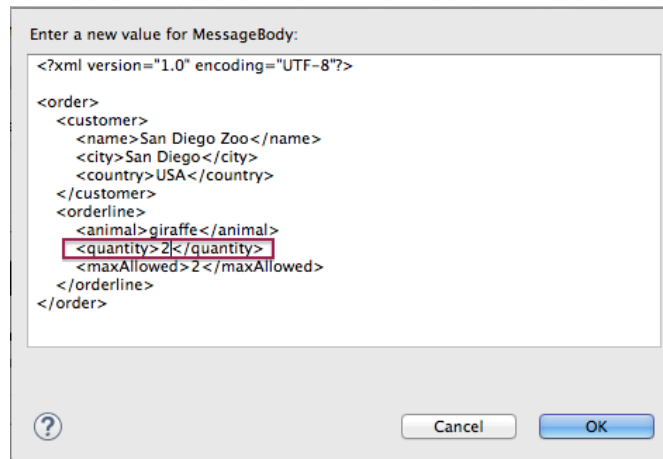
NOTE

Expressions view retains all variables you add to the list until you explicitly remove them.

5. Step message1 through the rest of the routing context.
6. Stop message2 at **choice1** in **Route1 [camelContext.xml]**.
7. In **Variables** view, expand the **Message** category to expose the **MessageBody** variable.
8. Right-click **MessageBody** to open its context menu, and select **Change Value...**



9. Change the value of quantity from 3 to 2.

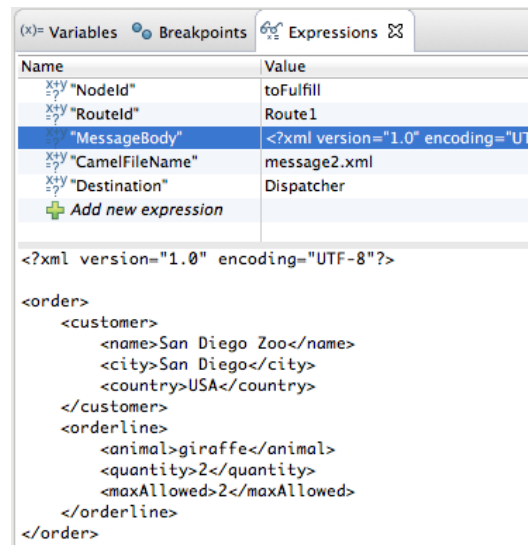


This changes the in-memory value only.

10. Click **OK**.

11. Switch to **Expressions** view, and select the **MessageBody** variable.

The pane below the list of variables displays the entire body of message2, making it easy to check the current value of order items:

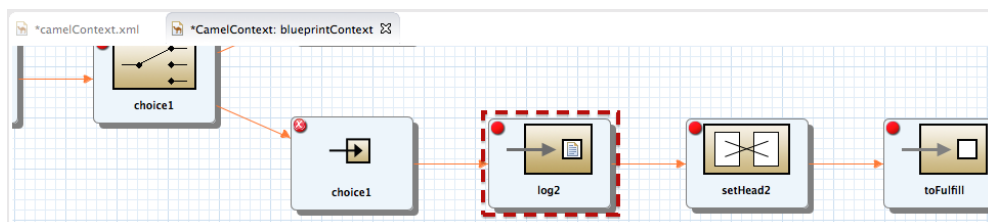


NOTE

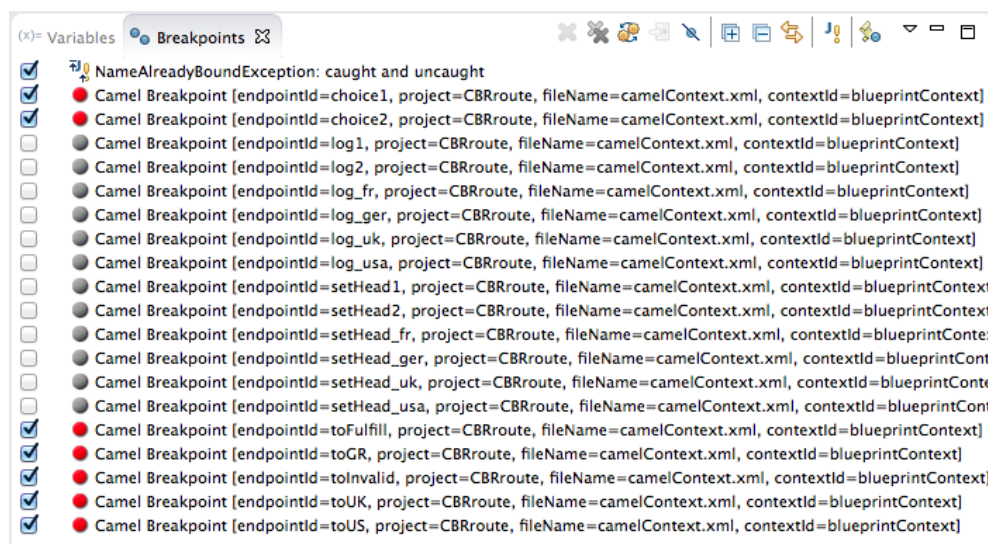
Creating a watch list makes it easy for you to quickly check the current value of multiple variables of interest.

12. Click  to step to the next breakpoint.

Instead of following the branch leading to `InvalidOrders`, message2 now follows the branch leading to `toFulfill`.

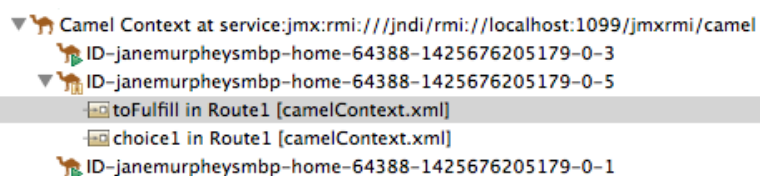


13. Step message2 through the routing context, checking **Debug** view, **Variables** view, and **Console** output at each step.
14. Stop message3 at `choice1` in `Route1 [camelContext.xml]`.
15. Switch to **Breakpoints** view, and disable all breakpoints (12) between `choice1` and `toFulfill`:



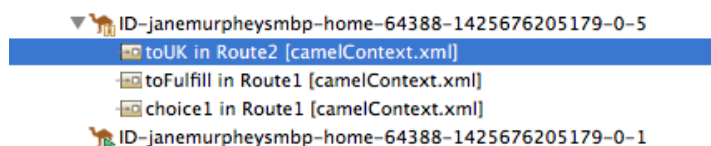
16. Switch back to **Variables** view.

17. Click  to step to the next breakpoint.




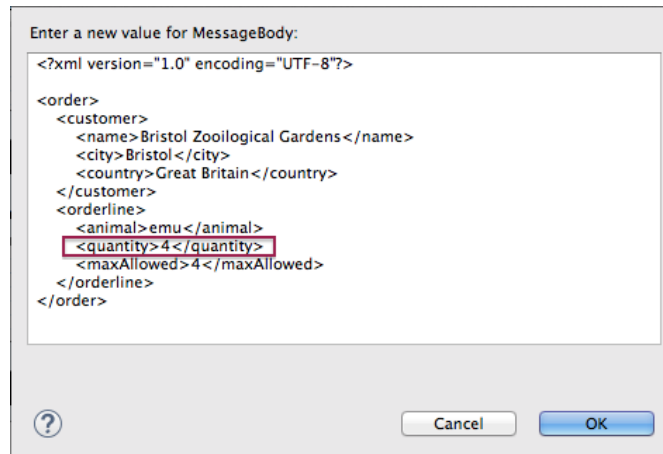
The debugger jumps to `toFulfill` in `Route1 [camelContext.xml]`.



18. Click  again to step to the next breakpoint.

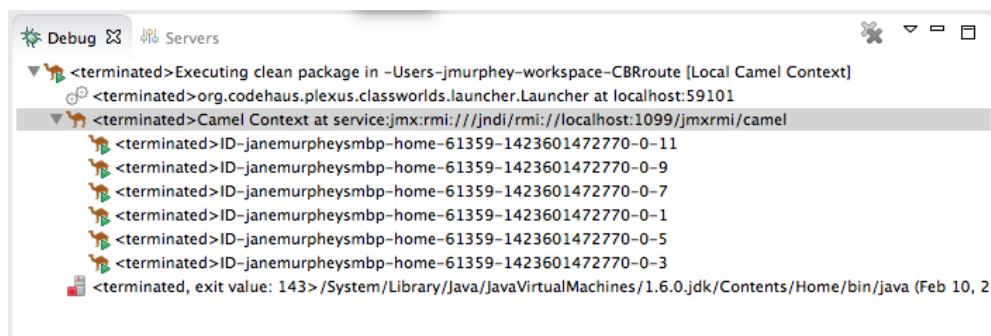



The debugger jumps to `toUK` in `Route2 [camelContext.xml]`.


19. Switch to **Breakpoints** view, and re-enable all disabled breakpoints.
20. Switch back to **Variables** view.
21. Click  to step to the next breakpoint, and stop message4 at `choice1` in `Route1 [camelContext.xml]`.
22. Right-click `MessageBody` to open its context menu, and select **Change Value . . .**
23. Change the value of quantity from **5** to **4**.



24. Click **OK**.
25. Switch to **Expressions** view, and select the `MessageBody` variable to check the value of quantity in the body of message4.
26. Repeat [Step 12](#) and [Step 13](#) to step message4 through the routing context.
27. Click  repeatedly to quickly step message5 and message6 through the routing context.
28. In the Menu bar, click  to terminate the Camel debugger.

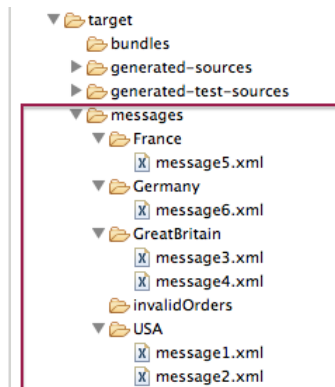


This will also cause the Console to terminate, but you will have to click its  button to clear the output.

29. In the Menu bar, right-click  to open the context menu, and then select **Close** to close **Debug** perspective.

Doing so automatically returns you to **JBoss** perspective.

30. In **Project Explorer**, open the project's context menu, and select **Refresh** to refresh the display.
31. Expand the **CBRroute/target/messages/*** directories to check whether the messages were delivered as expected:



You should see that no messages were sent to the **invalidOrders** folder. Instead, **message2.xml** should appear in the **USA** folder, and **message4.xml** should appear the **GreatBritain** folder.

NEXT STEPS

Next you will trace messages through your routing context to see where you can optimize and fine tune your routing context's performance, as described in [Chapter 7, To Trace a Message Through a Route](#).

CHAPTER 7. 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 the CBRroute in the **Fuse Integration** perspective
- enable tracing on the CBRroute
- drop messages onto the CBRroute and track them through all route nodes

PREREQUISITES

To complete this tutorial you will need the **CBRroute** project you updated in [Chapter 5, To Add Another Route to the CBR Routing Context](#).



NOTE

If you skipped any tutorial after [Chapter 2, To Create a New Route](#), you can use the prefabricated `camelContext6.xml` file to work through this tutorial (for details, see [Chapter 1, Using the Fuse Tooling Resource Files](#)).

ACCESSING FUSE INTEGRATION PERSPECTIVE

To open **Fuse Integration** perspective and optimally arrange its layout:

1. Select **Window** → **Open Perspective** → **Other...** → **Fuse Integration** to open the **Fuse Integration** perspective, as shown in [Figure 7.1](#).



NOTE


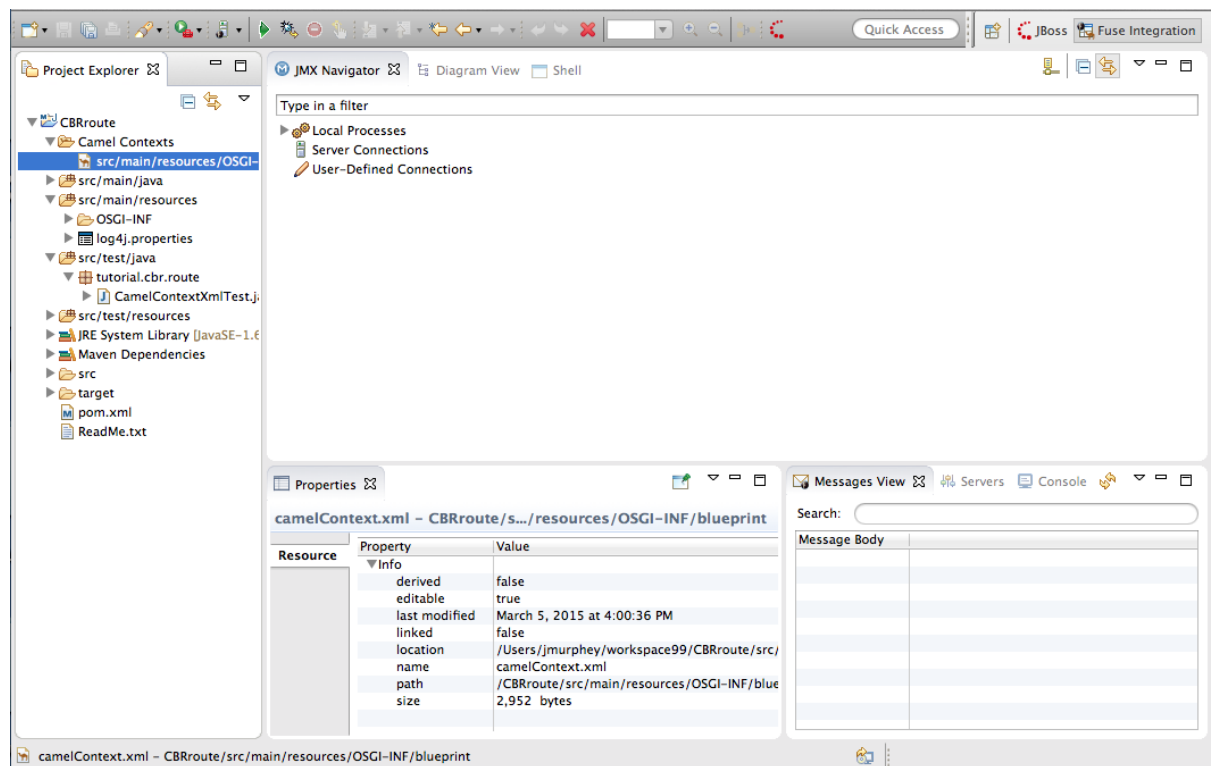
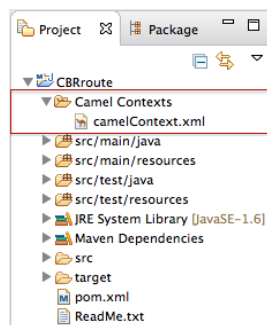
You can use the **Open Perspective** icon () in the perspectives tab to access the list of available perspectives.

Figure 7.1. Fuse Integration perspective



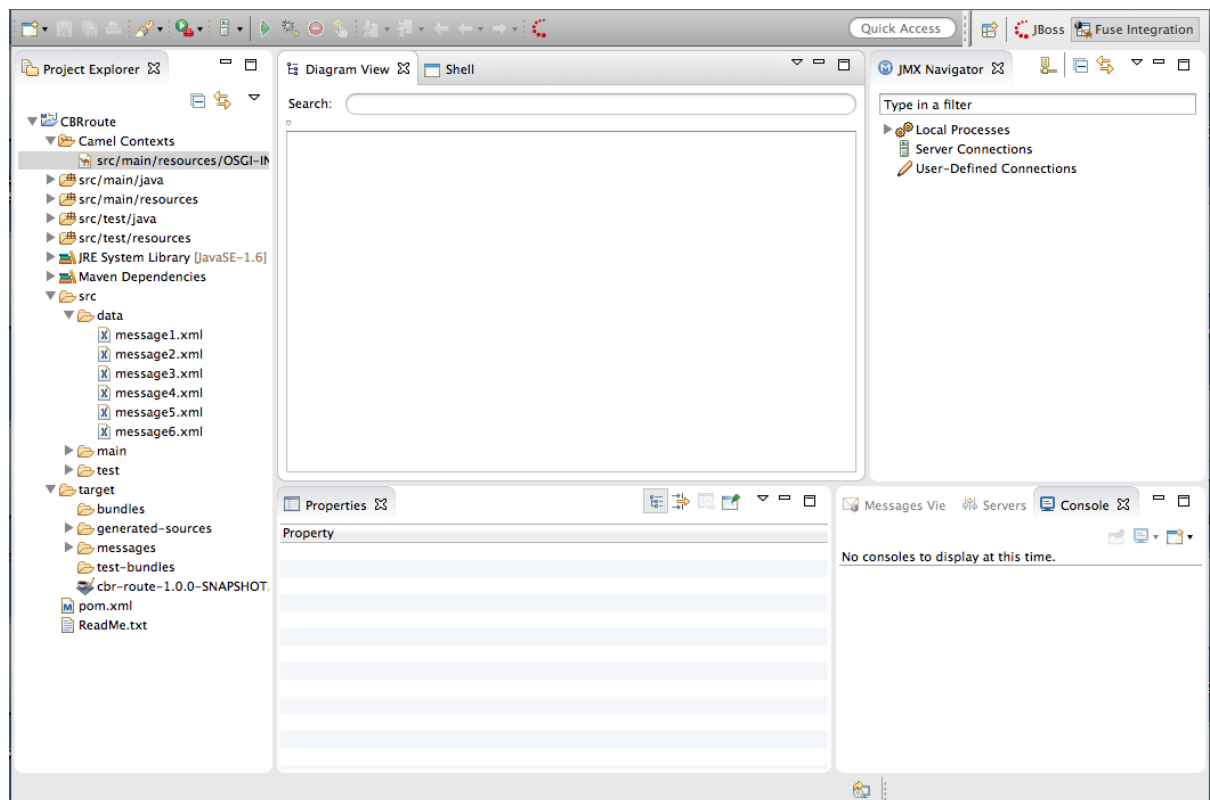
NOTE

To make it easy to access a Camel Context .xml file, especially when a project consists of multiple contexts, the tooling lists them in the **Camel Contexts** folder, beneath the project's root folder in **Project Explorer**; for example:



2. Drag the **Diagram View** tab and the **Shell** tab, located to the left of the **JMX Navigator** tab, and drop them to the right of the **JMX Navigator** tab, as shown in [Figure 7.2](#).

Figure 7.2. Fuse Integration perspective rearranged



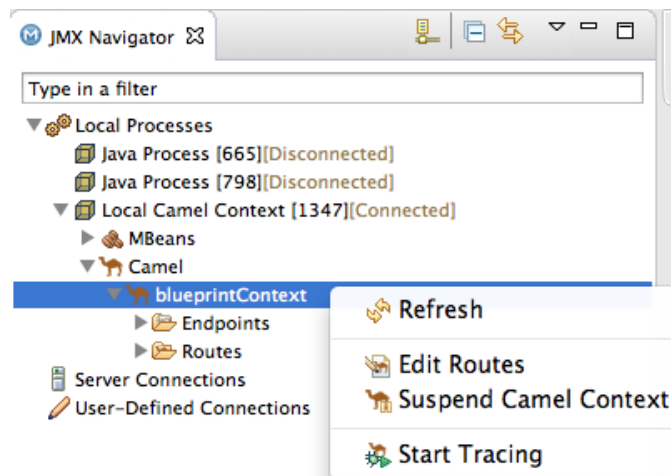
This layout will provide more space for **Diagram View** to display the route's nodes in a graphical representation, enabling you to visually trace the path that messages take in traversing the routing context.

STARTING MESSAGE TRACING

To start message tracing on the CBRroute project:

1. In **Project Explorer**, expand the CBRroute project to expose the `src/main/resources/OSGI-INF/blueprint/camelContext.xml` file.
2. Select **Run As** → **Local Camel Context (without tests)** from the `camelContext.xml` file's context menu.
3. In **JMX Navigator**, expand **Local Processes**.
4. Double click **Local Camel Context [Id] [Disconnected]** to connect to it and expand the elements of your route, as shown in [Figure 7.3, "Route elements in JMX Navigator"](#).

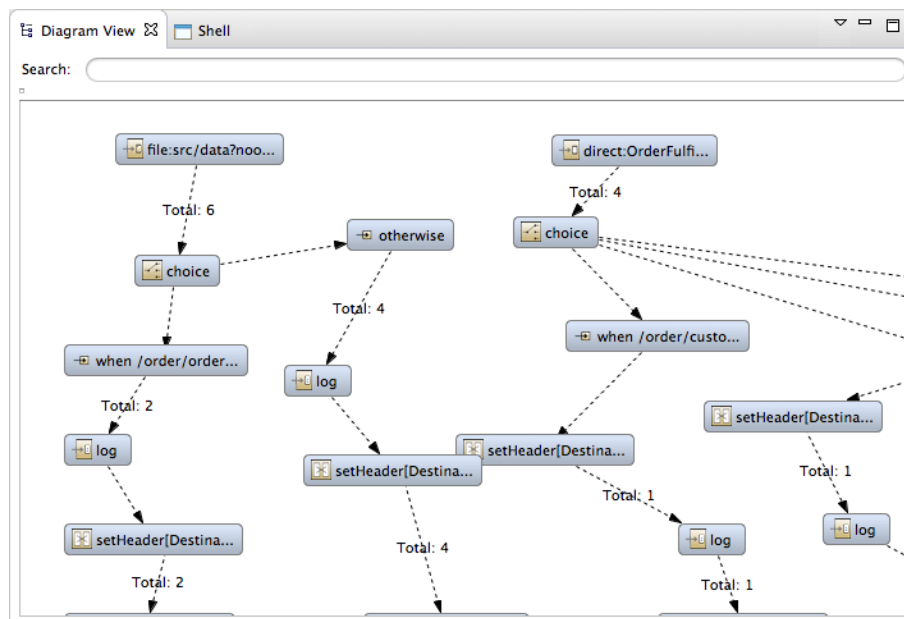
Figure 7.3. Route elements in JMX Navigator



5. Right-click the **blueprintContext** node to open the context menu, and select **Start Tracing**.

The tooling displays a graphical representation of your route in **Diagram View**, as shown in [Figure 7.4](#).

Figure 7.4. Routes' graphical representation

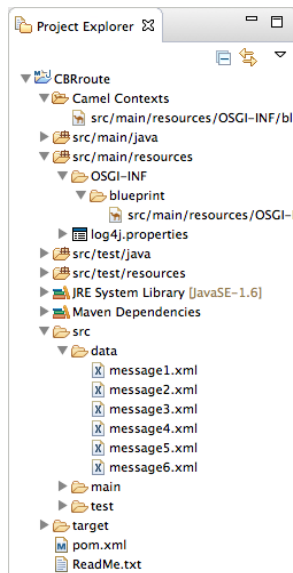


DROPPING MESSAGES ON THE RUNNING CBRROUTE PROJECT

To drop messages on the running CBRroute project:

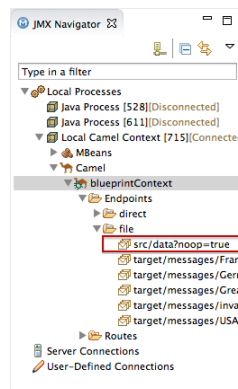
1. In **Project Explorer**, expand **CBRroute/src/data**, so you can access the message files (message1.xml through message6.xml), as shown in [Figure 7.5](#), “Message files in CBRroute project”.

Figure 7.5. Message files in CBRroute project



2. Drag `message1.xml` and drop it on the `blueprintContext>Endpoints>file>src/data?noop=true` node in **JMX Navigator**, as shown in [Figure 7.6](#).

Figure 7.6. Local Camel Context tree expanded to input source node



As the message traverses the route, the tooling traces and records its passage at each step. To update **Diagram View** with the new message count, you need to click the `blueprintContext` node in **JMX Navigator**.



NOTE

The **Local Camel Context [xxx]** tree collapses to the `blueprintContext` node after you drop the next message on the input `src` node. You need not re-expand it. When dragging the other messages, hover over each node in the tree to expose the next node, until you reach the `src/data?noop=true` node. Then drop the message on it. This method prevents the tooling from redrawing the graphical representation in **Diagram View**.

INITIALIZING AND CONFIGURING MESSAGES VIEW

You need to initialize **Messages View** before it will display message traces. You also need to configure the columns in **Messages View** to persist across all message traces.

1. Switch from **Console** to **Messages View**.

- Click the `blueprintContext` node in **JMX Navigator** to initialize **Messages View** with `message1.xml`'s details.



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. For your arrangement to persist, you must use the **View Menu** → **Configure Columns...** method instead.


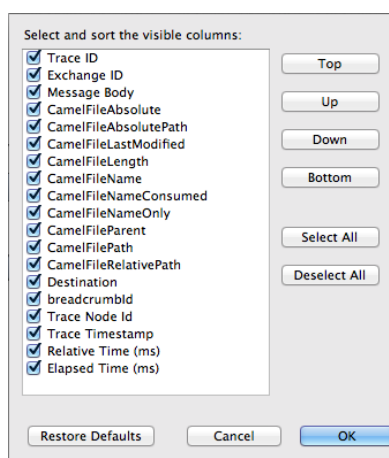
- In **Messages View**, click the  icon on the panel's menu bar, and select **Configure Columns...** to open the **Configure Columns** wizard, as shown in [Figure 7.7](#).

Figure 7.7. Configure Columns defaults



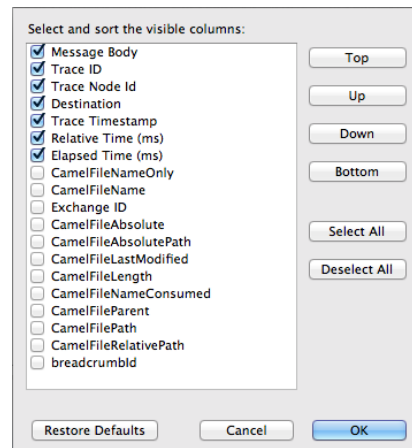
NOTE

Notice that the message header, **Destination**, which you set for the messages in your routing context, appears in the list.

You can include or exclude items from **Messages View** by selecting or deselecting them. You can rearrange the columnar order in which items appear in **Messages View** by highlighting individual, selected items and moving them up or down in the list.

- In the **Configure Columns** wizard, select and order the items as shown in [Figure 7.8](#).

Figure 7.8. Configure Columns set



These items and their order will persist in **Messages View** until you change them again.

ARRANGING DIAGRAM VIEW

To see all message flow paths clearly, you'll probably need to rearrange the nodes by dragging them to fit neatly in **Diagram View**. You may also need to adjust the size of the other views and tabs in Red Hat JBoss Developer Studio to allow **Diagram View** to expand.

STEPPING THROUGH MESSAGE TRACES

To step through the message traces:

1. In **Messages View**, click the 🔄 (Refresh button) on top, right of the panel's menu bar to populate the view with `message1.xml`'s message traces.

Each time you drop a message on the input `src` node in **JMX Navigator**, you need to refresh **Messages View** to populate it with the message traces.

2. Click one of the message traces to see more details about it in **Properties view**, as shown in [Figure 7.9](#).

Figure 7.9. Message trace selected

Name	Value
CamelFileAbsolute	false
CamelFileAbsolutePath	/Users/jmurphey/workspace99/CBRroute/src/dat...
CamelFileLastModified	1426271998000
CamelFileLength	283
CamelFileName	ID: jmurphey@redhat.com_40505_1426270274

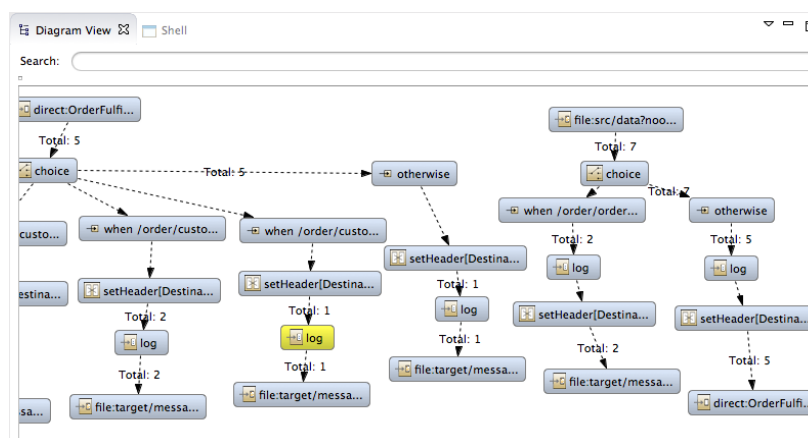
Message Body	Trace ID	Trace Node Id	Destination	Trace Timestamp	Relative T	Elapsed Ti
<?xml version=...	1			Fri Mar 13 14:3...	0	
<?xml version=...	2	choice1		Fri Mar 13 14:3...	0	0
<?xml version=...	3	log2		Fri Mar 13 14:3...	7	7
<?xml version=...	4	setHead2		Fri Mar 13 14:3...	9	2
<?xml version=...	5	toFulfill	Dispatcher	Fri Mar 13 14:3...	10	1
<?xml version=...	6		Dispatcher	Fri Mar 13 14:3...	0	-10
<?xml version=...	7	choice2	Dispatcher	Fri Mar 13 14:3...	12	12
<?xml version=...	8	setHead_usa	Dispatcher	Fri Mar 13 14:3...	15	3
<?xml version=...	9	log_usa	USA	Fri Mar 13 14:3...	16	1
<?xml version=...	10	toUS	USA	Fri Mar 13 14:3...	18	2

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

You can step through the message instances by highlighting each one to see how a particular message traversed the route and whether it was processed as expected at each step in the route.

In **Diagram View**, the associated step in the route is highlighted, as shown in [Figure 7.10](#).

Figure 7.10. Diagram View: message trace node



FINISHING UP

1. Drag `message2.xml` and drop it on the `blueprintContext>Endpoints>file>src/data?noop=true` node in **JMX Navigator**.

Hover over each node in the tree until you expose the `src/data?noop=true` node, then drop `message2.xml` on it.

2. Switch from **Console** to **Messages View**.
3. In **Messages View**, click the 🔄 (Refresh button) on top, right of the panel's menu bar to populate the view with `message2.xml`'s message traces.



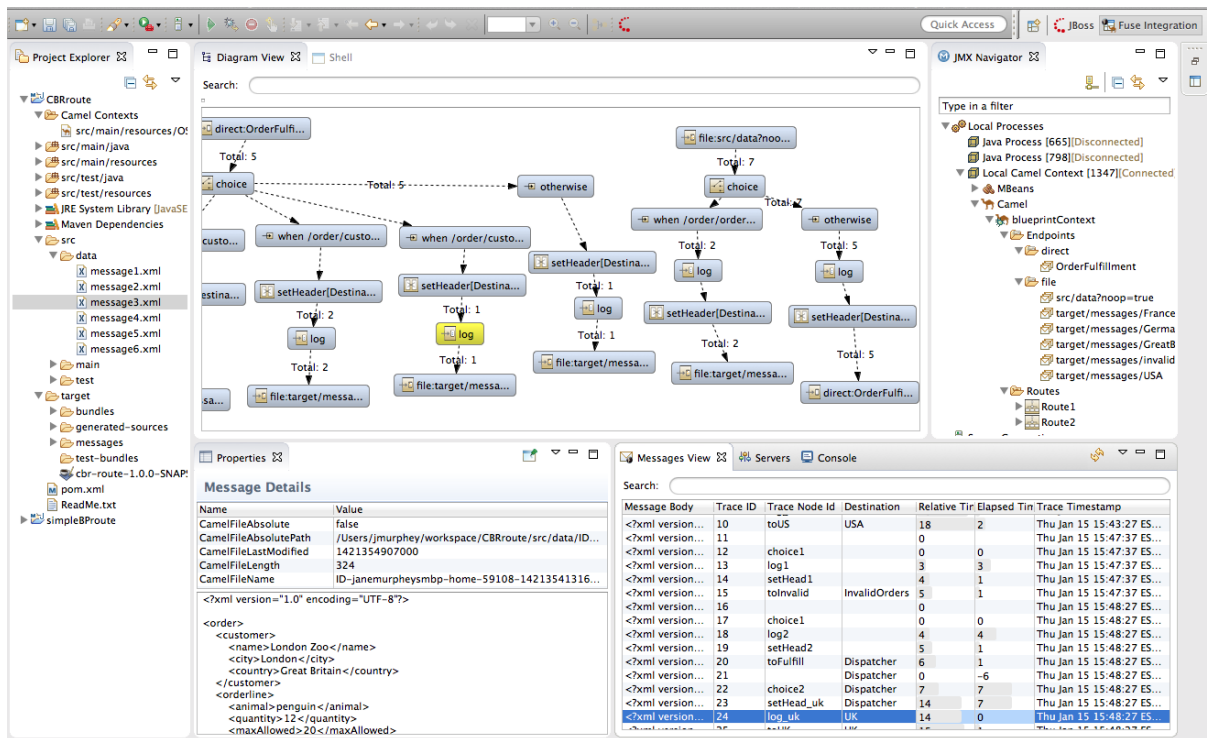
NOTE

You can repeat [Step 2](#) through [Step 2](#) for the remaining messages in `CBRroute/src/data/` at any time, as long as tracing remains enabled.

On each subsequent drop, remember to click the 🔄 (Refresh button) on the panel's menu bar to populate **Messages View** with the new message traces.

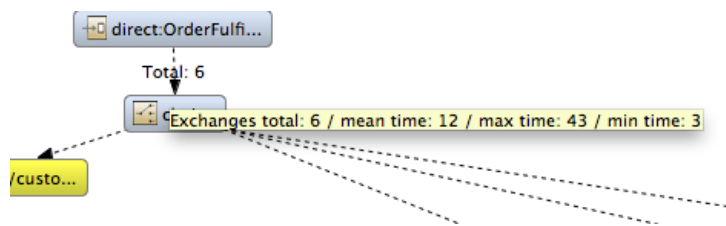
As shown in [Figure 7.11](#), the tooling draws the route in **Diagram View**, tagging paths exiting a processing step with timing and performance metrics (in milliseconds). Only the metric **Total exchanges** is displayed in the diagram.

Figure 7.11. Fuse Integration perspective's message tracing components





Hovering over the displayed metrics reveals additional metrics about message flow, as shown in Figure 7.12.

Figure 7.12. Additional message metrics



- 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

4. When done:

- In JMX Navigator, right-click blueprintContext and select Stop Tracing Context from the context menu.
- Open the Console and click the  button in the upper right of the panel to stop the Console. Then click the  button to clear console output.

NEXT STEPS

After you create a JUnit test case for your project, you can run your project with it, as described in [Chapter 8, To Test a Route with JUnit](#)

CHAPTER 8. 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 new route.

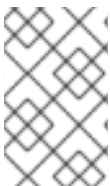
GOALS

In this tutorial you will:

- delete the existing JUnit test case
- generate a new JUnit test case for the CBRroute project
- modify the newly generated JUnit test case
- modify the CBRroute project's `pom.xml` file
- run the CBRroute with the New JUnit test case
- observe the output

PREREQUISITES

To complete this tutorial you will need the **CBRroute** project you used in [Chapter 7, To Trace a Message Through a Route](#)

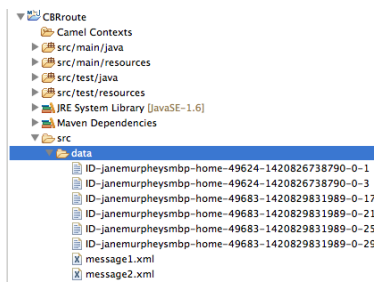


NOTE

If you skipped any tutorial after [Chapter 2, To Create a New Route](#), you can use the prefabricated `camelContext6.xml` file to work through this tutorial (for details, see [Chapter 1, Using the Fuse Tooling Resource Files](#)).

Delete any trace-generated messages from the **CBRroute** project's `/src/data/` directory and `/target/messages/` subdirectories in **Project Explorer**. Trace-generated messages begin with the **ID-** prefix. For example, [Figure 8.1, “Trace-generated messages”](#) shows six trace-generated messages:

Figure 8.1. Trace-generated messages



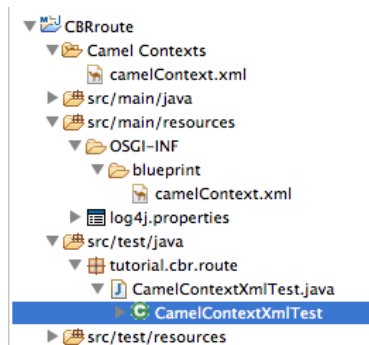
Select all trace-generated messages in batch, right-click to open the context menu, and select **Delete**.

DELETING THE EXISTING TEST CASE

To delete the existing Apache Camel test case:

1. In **Project Explorer**, expand `src/test/java` to expose the `CamelContextXmlTest` file, as shown in [Figure 8.2](#).

Figure 8.2. CamelContextXmlTest.java file location



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

A dialog box opens asking you to confirm deletion of the test case file.

3. Click **OK**.
4. Verify that the existing test case has been deleted by right-clicking `CBRroute` and selecting **Refresh**.

The `src/test/java/tutorial.cbr.route` directory should be empty.

CREATING THE NEW TEST CASE


To create a new Apache Camel test case:

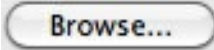
1. In **Project Explorer**, select `src/test/java`.
2. Right-click it to open the context menu, and then select **New** → **Camel Test Case** to open the **New Camel JUnit Test Case** wizard, as shown in [Figure 8.3](#).

Figure 8.3. New Camel JUnit Test Case wizard

3. Make sure the **Source folder** field contains `CBRroute/src/test/java`.

**NOTE**

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

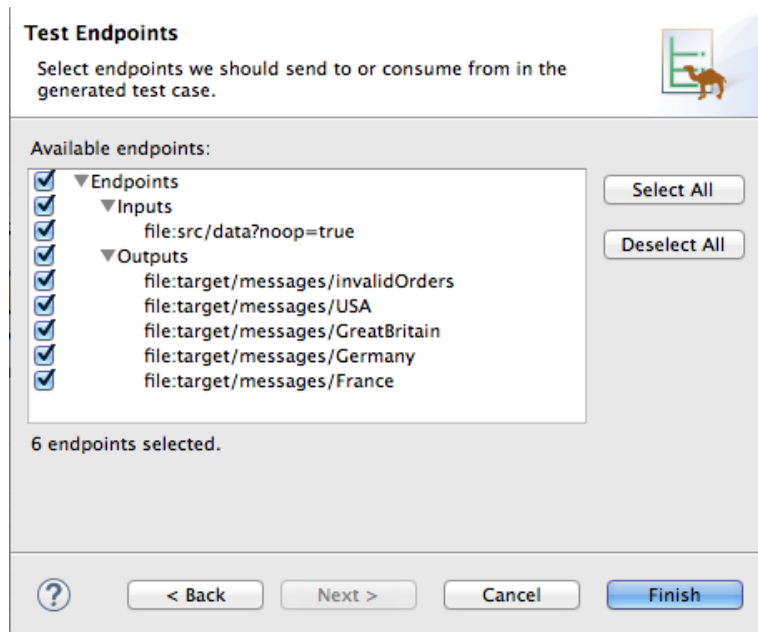
4. The **Package** field defaults to `tutorial.cbr.route`. To include the test case in a different package, enter the name of the package.
5. In the **Camel XML file under test** field, enter `src/main/resources/OSGI-INF/blueprint/camelContext.xml`, or use  to open a file explorer, configured to screen 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 8.4](#).

Figure 8.4. Test Endpoints page



- By default, all endpoints are selected and will be included in the test case. Leave them selected for this tutorial.

**NOTE**

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.

- Click **Finish**.

**NOTE**

If prompted, add JUnit to the build path.

The artifacts for the test are added to your project and appear in **Project Explorer** under `src/test/java`. The class implementing the test case opens in the tooling's Java editor:

```
package tutorial.cbr.route;

import org.apache.camel.EndpointInject;
import org.apache.camel.Produce;
import org.apache.camel.ProducerTemplate;
import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.component.mock.MockEndpoint;
import org.apache.camel.test.blueprint.CamelBlueprintTestSupport;
import org.junit.Test;

public class CamelContextXmlTest extends CamelBlueprintTestSupport {

    // TODO Create test message bodies that work for the route(s) being
    // tested
    // Expected message bodies
    protected Object[] expectedBodies = {
```

```

    "<something id='1'>expectedBody1</something>",
    "<something id='2'>expectedBody2</something>";
// Templates to send to input endpoints
@Produce(uri = "file:src/data?noop=true")
protected ProducerTemplate inputEndpoint;
// Mock endpoints used to consume messages from the output endpoints and
then perform assertions
@EndpointInject(uri = "mock:output")
protected MockEndpoint outputEndpoint;
@EndpointInject(uri = "mock:output2")
protected MockEndpoint output2Endpoint;
@EndpointInject(uri = "mock:output3")
protected MockEndpoint output3Endpoint;
@EndpointInject(uri = "mock:output4")
protected MockEndpoint output4Endpoint;
@EndpointInject(uri = "mock:output5")
protected MockEndpoint output5Endpoint;

@Test
public void testCamelRoute() throws Exception {
    // Create routes from the output endpoints to our mock endpoints so we
can assert expectations
    context.addRoutes(new RouteBuilder() {
        @Override
        public void configure() throws Exception {
            from("file:target/messages/Germany").to(output4Endpoint);
            from("file:target/messages/GreatBritain").to(output3Endpoint);
            from("file:target/messages/USA").to(output2Endpoint);
            from("file:target/messages/France").to(output5Endpoint);
            from("file:target/messages/invalidOrders").to(outputEndpoint);
        }
    });

    // Define some expectations

    // TODO Ensure expectations make sense for the route(s) we're testing
output4Endpoint.expectedBodiesReceivedInAnyOrder(expectedBodies);

    // Send some messages to input endpoints
    for (Object expectedBody : expectedBodies) {
        inputEndpoint.sendBody(expectedBody);
    }

    // Validate our expectations
    assertMockEndpointsSatisfied();
}

@Override
protected String getBlueprintDescriptor() {
    return "OSGI-INF/blueprint/camelContext.xml";
}
}

```

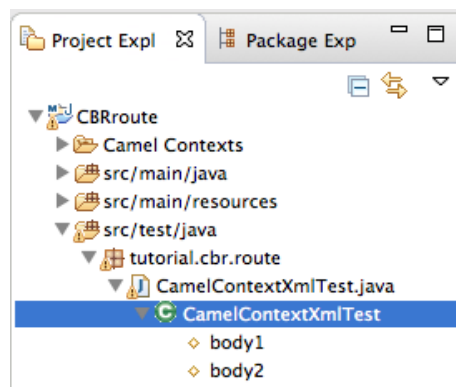
This generated JUnit test case is insufficient for the CBRroute project, and it will fail to run successfully. You need to modify it and the project's `pom.xml`, as described in [the section called “Modifying the CamelContextXmlTest file”](#) and [the section called “Modifying the pom.xml file”](#).

MODIFYING THE CAMELCONTEXTXMLTEST FILE

You need to modify the `CamelContextXmlTest.java` file to:

- import several classes that support required file functions
- create variables for holding the content of the various source `.xml` files
- read the content of the source `.xml` files
- define appropriate expectations

1. In **Project Explorer**, expand the CBRroute project to expose the `CamelContextXmlTest` item.



2. Double-click `CamelContextXmlTest` to open the file in the route editor.
3. In the route editor, click the expand button next to `import org.apache.camel.EndpointInject`; to expand the list.
4. Add the three lines shown here:

```

package tutorial.cbr.route;

import org.apache.camel.EndpointInject;
import org.apache.camel.Producer;
import org.apache.camel.ProducerTemplate;
import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.component.mock.MockEndpoint;
import org.apache.camel.test.blueprint.CamelBlueprintTestSupport;
import org.apache.commons.io.FileUtils;
import org.junit.Test;

import java.io.File;
import java.util.Scanner;

```

5. Scroll down to the lines that follow directly after `// Expected message bodies`.
6. Replace those lines—`protected Object[] expectedBodies={ expectedBody2</something>"};`—with the `protected String body#;` lines shown here:

```

public class CamelContextXmlTest extends CamelBlueprintTestSupport {

    // TODO Create test message bodies that work for the route(s) being tested
    // Expected message bodies

    // To assert that everything works as it should you must read the content of created xml files.

    protected String body1;
    protected String body2;
    protected String body3;
    protected String body4;
    protected String body5;
    protected String body6;

```

7. Scroll down to the line `public void testcamelRoute() throws Exception {`, and insert directly after it the lines `body# = FileUtils.readFileToString(new File("src/data/message#.xml"));` shown here:

```

@Test
public void testCamelRoute() throws Exception {
    // Easy way of reading content of xml files to a String object. But you must
    body1 = FileUtils.readFileToString(new File("src/data/message1.xml"));
    body3 = FileUtils.readFileToString(new File("src/data/message3.xml"));
    body5 = FileUtils.readFileToString(new File("src/data/message5.xml"));
    body6 = FileUtils.readFileToString(new File("src/data/message6.xml"));

    // Invalid orders.
    body2 = FileUtils.readFileToString(new File("src/data/message2.xml"));
    body4 = FileUtils.readFileToString(new File("src/data/message4.xml"));

```

8. Scroll down to the lines that follow directly after `// TODO Ensure expectations make sense for the route(s) we're testing`.
9. Replace the block of code that begins with `output4Endpoint.expectedBodiesReceivedInAnyOrder(expectedBodies);` and ends with `...inputEndpoint.sendBody(expectedBody); }` with the lines shown here:

```

// TODO Ensure expectations make sense for the route(s) we're testing

// Invalid orders
outputEndpoint.expectedBodiesReceived(body2, body4);

// For each country one order
output2Endpoint.expectedBodiesReceived(body1);
output3Endpoint.expectedBodiesReceived(body3);
output4Endpoint.expectedBodiesReceived(body6);
output5Endpoint.expectedBodiesReceived(body5);

```

Leave the remaining code as is.

10. Save the file.
11. Check that your updated `CamelContextXmlTest.java` file has the required modifications. It should look something like this:

```

package tutorial.cbr.route;

import org.apache.camel.EndpointInject;
import org.apache.camel.Produce;
import org.apache.camel.ProducerTemplate;
import org.apache.camel.builder.RouteBuilder;

```

```
import org.apache.camel.component.mock.MockEndpoint;
import org.apache.camel.test.blueprint.CamelBlueprintTestSupport;
import org.apache.commons.io.FileUtils;
import org.junit.Test;

import java.io.File;
import java.util.Scanner;

public class CamelContextXmlTest extends CamelBlueprintTestSupport {

    // TODO Create test message bodies that work for the route(s) being
    // tested
    // Expected message bodies

    // To assert that everything works as it should, read the content
    // of created xml files.
    protected String body1;
    protected String body2;
    protected String body3;
    protected String body4;
    protected String body5;
    protected String body6;

    // Templates to send to input endpoints
    @Produce(uri = "file:src/data?noop=true")
    protected ProducerTemplate inputEndpoint;
    // Mock endpoints used to consume messages from the output
    // endpoints and then perform assertions
    @EndpointInject(uri = "mock:output")
    protected MockEndpoint outputEndpoint;
    @EndpointInject(uri = "mock:output2")
    protected MockEndpoint output2Endpoint;
    @EndpointInject(uri = "mock:output3")
    protected MockEndpoint output3Endpoint;
    @EndpointInject(uri = "mock:output4")
    protected MockEndpoint output4Endpoint;
    @EndpointInject(uri = "mock:output5")
    protected MockEndpoint output5Endpoint;

    @Test
    public void testCamelRoute() throws Exception {
        // Easy way of reading content of xml files to a String object.
        // But you must add dependency on commons-io project to pom.xml
        body1 = FileUtils.readFileToString(new
File("src/data/message1.xml"));
        body3 = FileUtils.readFileToString(new
File("src/data/message3.xml"));
        body5 = FileUtils.readFileToString(new
File("src/data/message5.xml"));
        body6 = FileUtils.readFileToString(new
File("src/data/message6.xml"));

        // Invalid orders.
        body2 = FileUtils.readFileToString(new
File("src/data/message2.xml"));
    }
}
```



```

    body4 = FileUtils.readFileToString(new
File("src/data/message4.xml"));

    // Create routes from the output endpoints to our mock endpoints
    // so we can assert expectations
    context.addRoutes(new RouteBuilder() {
        @Override
        public void configure() throws Exception {
            from("file:target/messages/Germany").to(output4Endpoint);
            from("file:target/messages/GreatBritain").to(output3Endpoint);
            from("file:target/messages/USA").to(output2Endpoint);
            from("file:target/messages/France").to(output5Endpoint);
            from("file:target/messages/invalidOrders").to(outputEndpoint);
        }
    });

    // Define some expectations

    // TODO Ensure expectations make sense for the route(s) we're
    testing

    // Invalid orders
    outputEndpoint.expectedBodiesReceived(body2, body4);

    // For each country one order
    output2Endpoint.expectedBodiesReceived(body1);
    output3Endpoint.expectedBodiesReceived(body3);
    output4Endpoint.expectedBodiesReceived(body6);
    output5Endpoint.expectedBodiesReceived(body5);

    // Validate our expectations
    assertMockEndpointsSatisfied();
}

@Override
protected String getBlueprintDescriptor() {
    return "OSGI-INF/blueprint/camelContext.xml";
}
}

```

MODIFYING THE POM.XML FILE

You need to add a dependency on the commons-io project to the CBRroute project's `pom.xml` file.

1. In **Project Explorer**, double-click `pom.xml`, located below the `target` folder, to open the file in the route editor.
2. Click the `pom.xml` tab at the bottom of the page to open the file for editing.
3. Add these lines to the end of the `<dependencies>` section:

```

<dependency>
    <groupId>commons-io</groupId>

```

```

    <artifactId>commons-io</artifactId>
    <version>2.4</version>
    <scope>test</scope>
</dependency>

```

4. Save the file.

The contents of the entire `pom.xml` file should look like this:

```

<?xml version="1.0" encoding="UTF-8"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd"
  xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <modelVersion>4.0.0</modelVersion>
  <groupId>tutorial</groupId>
  <artifactId>cbr-route</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <packaging>bundle</packaging>
  <name>A Camel Blueprint Route</name>
  <url>http://www.myorganization.org</url>
  <properties>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEnc
oding>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.apache.camel</groupId>
      <artifactId>camel-core</artifactId>
      <version>2.15.0</version>
    </dependency>
    <dependency>
      <groupId>org.apache.camel</groupId>
      <artifactId>camel-blueprint</artifactId>
      <version>2.15.0</version>
    </dependency>
    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-api</artifactId>
      <version>1.7.7</version>
    </dependency>
    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-log4j12</artifactId>
      <version>1.7.7</version>
    </dependency>
    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>jcl-over-slf4j</artifactId>
      <version>1.7.7</version>
    </dependency>
    <dependency>
      <groupId>log4j</groupId>

```

```

    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-test-blueprint</artifactId>
    <version>2.15.0</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-infinispan</artifactId>
    <version>2.15.0</version>
</dependency>
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-jgroups</artifactId>
    <version>2.15.0</version>
</dependency>
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-test</artifactId>
    <version>2.15.0</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.4</version>
    <scope>test</scope>
</dependency>
</dependencies>
<build>
    <defaultGoal>install</defaultGoal>
    <plugins>
        <plugin>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>2.5.1</version>
            <configuration>
                <source>1.6</source>
                <target>1.6</target>
            </configuration>
        </plugin>
        <plugin>
            <artifactId>maven-resources-plugin</artifactId>
            <version>2.6</version>
            <configuration>
                <encoding>UTF-8</encoding>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.apache.felix</groupId>
            <artifactId>maven-bundle-plugin</artifactId>
            <version>2.3.7</version>
            <extensions>true</extensions>
            <configuration>

```

```

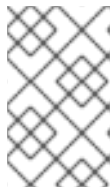
    <instructions>
      <Bundle-SymbolicName>cbr-route</Bundle-SymbolicName>
      <Private-Package>tutorial.cbr.route.*</Private-Package>
      <Import-Package>*</Import-Package>
    </instructions>
  </configuration>
</plugin>
<plugin>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-maven-plugin</artifactId>
  <version>2.15.0</version>
  <configuration>
    <useBlueprint>true</useBlueprint>
  </configuration>
</plugin>
</plugins>
</build>
</project>

```

RUNNING THE JUNIT TEST

To run the test:

1. Switch to **JBoss** perspective to free up more workspace.
2. Select the project root, **CBRroute**, in the **Project Explorer**.
3. Open the context menu.
4. Select **Run As** → **JUnit Test**.

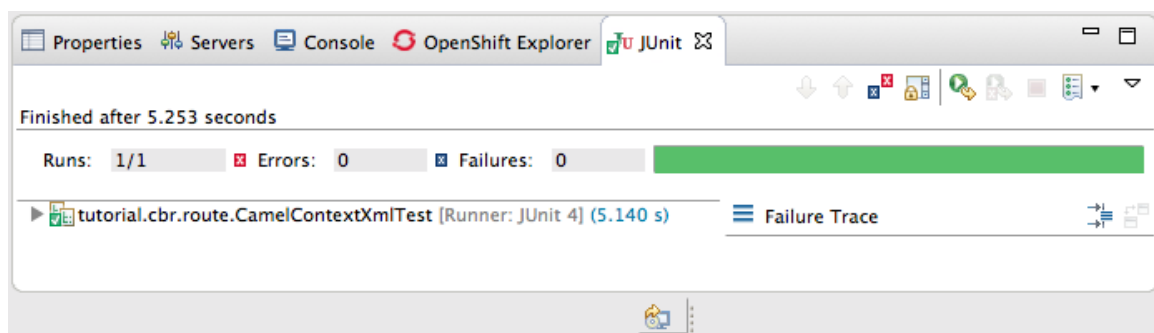


NOTE

By default, **JUnit** view opens in the sidebar. (To provide a better view, drag it to the bottom, right panel that displays the **Console**, **Servers**, and **Properties** tabs.)

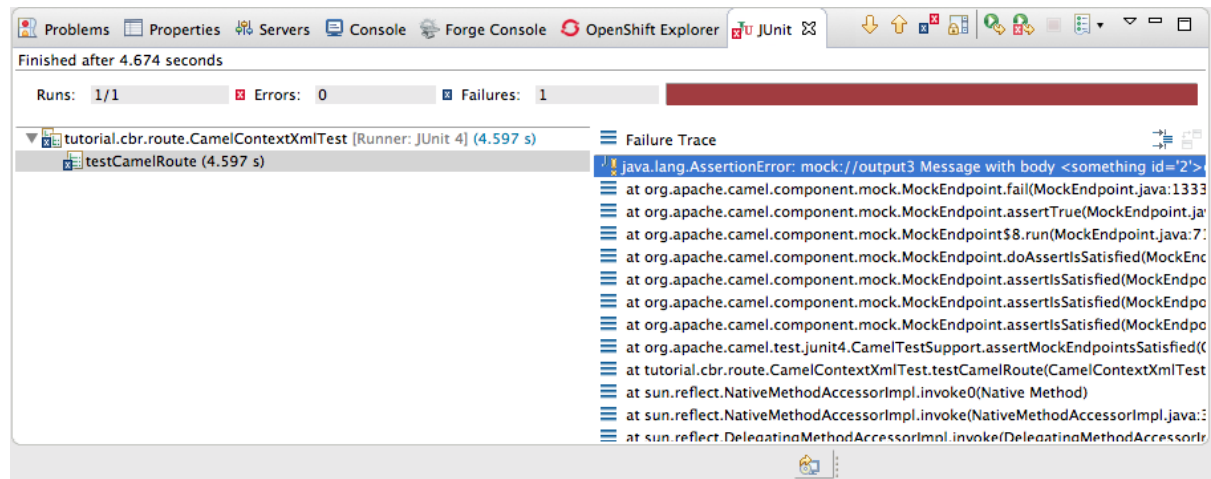
5. If the test runs successfully, you'll see something like this:

Figure 8.5. Successful JUnit run



If the test fails, you'll see something like this:

Figure 8.6. Failed JUnit run



NOTE

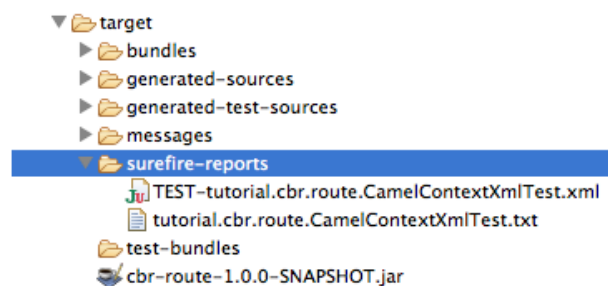
JUnit will fail if your execution environment is not set to Java SE 7 or above. The message bar at the top of the JUnit tab will display an error message indicating that it cannot find the correct SDK.

To resolve the issue, open the project's context menu, and select **Run As** → **Run Configurations** → **JRE tab**. Click the **Environments** button next to the **Execution environment**: field to locate and select a Java SE 7 environment.

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

To see more of the errors displayed in the JUnit panel, click on the panel's menu bar to maximize the view. You can also check the surefire reports in the **surefire-reports** tab in **Project Explorer**, as shown in [Figure 8.7](#).

Figure 8.7. JUnit surefire reports



Before you run the JUnit test case again, delete any JUnit-generated test messages from the CBRroute project's `/src/data` folder in **Project Explorer** (see [Figure 8.1](#), “Trace-generated messages”).

FURTHER READING

To learn more about JUnit testing:

- see [JUnit](#)

CHAPTER 9. TO PUBLISH A FUSE PROJECT TO RED HAT JBOSS FUSE

Abstract

This tutorial walks you through the process of deploying an Apache 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:

- define a Red Hat JBoss Fuse server
- configure the publishing options
- start up the Red Hat JBoss Fuse server and publish the CBRroute project
- connect to the Red Hat JBoss Fuse server
- verify whether the CBRroute project's bundle was successfully built and published
- uninstall the CBRroute project

PREREQUISITES

To complete this tutorial you will need

- access to a Red Hat JBoss Fuse 6.2 instance
- the CBRroute project you updated in [Chapter 8, To Test a Route with JUnit](#)

DEFINING A RED HAT JBOSS FUSE SERVER

To define a server:

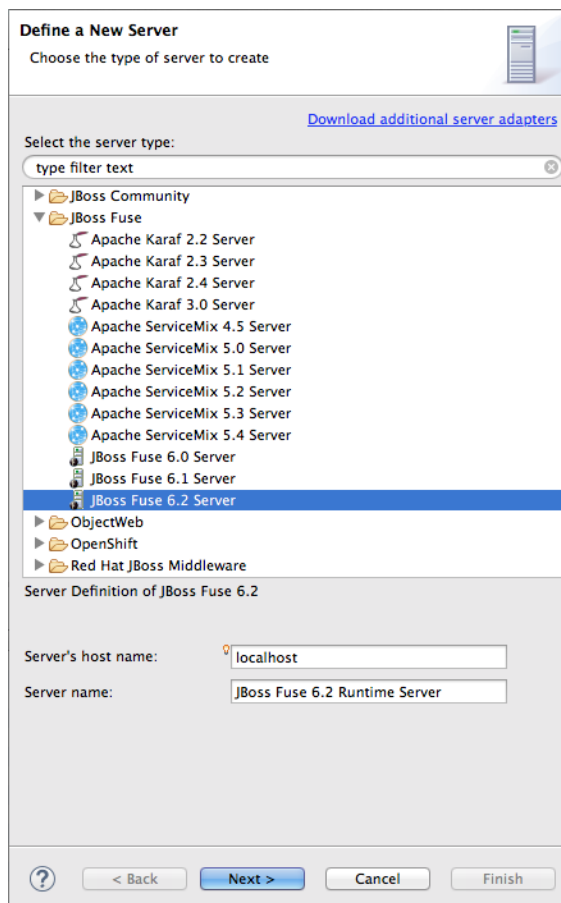
1. Open **Fuse Integration** perspective.
2. Click the **Servers** tab in the lower, right panel to open the **Servers** view.
3. Click the link **No servers are available. Click this link to create a new server...** to open the **Define a New Server** page.



NOTE

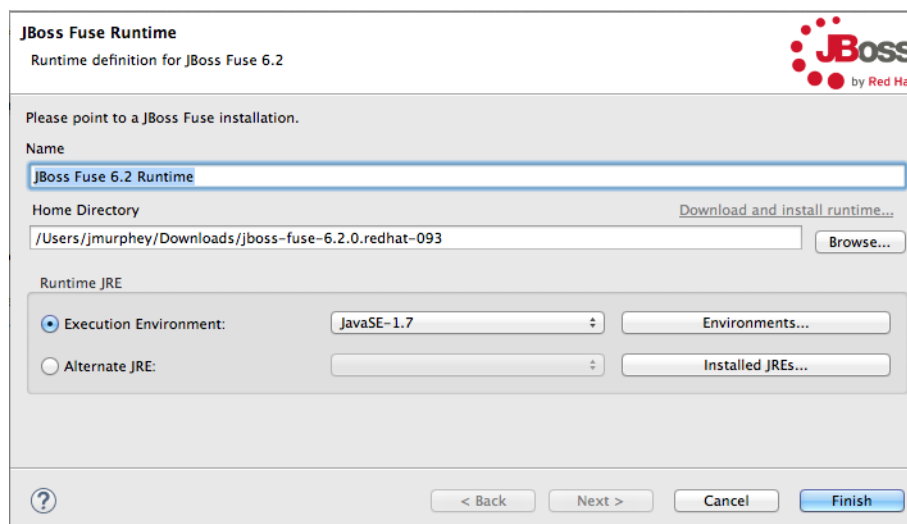
To define a new server when one is already defined, right-click inside **Servers** view to open the context menu, and then select **New** → **Server**.

4. Expand the **JBoss Fuse** node to expose the available server options:



5. Click **JBoss Fuse 6.2 Server**.

6. Accept the defaults for **Server's host name** (*localhost*) and **Server name** (*JBoss Fuse 6.2 Runtime Server*), and then click **Next** to open the **JBoss Fuse Runtime** page:



NOTE

If you do not have JBoss Fuse 6.2 already installed, you can download it now using the **Download and install runtime...** link.

**NOTE**

If you have already defined a JBoss Fuse 6.2 server, the tooling skips this page, and instead displays the configuration details page shown in [Step 11](#).

7. Accept the default for **Name** (*JBoss Fuse 6.2 Runtime*).
8. In **Home Directory**, enter the path where the JBoss Fuse 6.2 installation is located, or click **Browse** to find and select it.
9. Select the runtime JRE from the drop-down menu next to **Execution Environment**.

Select either JavaSE-1.7 or JavaSE-1.8. If neither appears as an option, click the **Environments . . .** button and select either version from the list.

**NOTE**

The JBoss Fuse 6.2 server requires Java 7 or Java 8. To select either version for the **Execution Environment**, you must have previously installed it.

10. Leave the **Alternate JRE** option as is.
11. Click **Next** to save the runtime definition for JBoss Fuse 6.2 Server and open the **JBoss Fuse server configuration details** page:

12. Accept the default for **SSH Port** (*8101*).

The runtime uses the SSH port to connect to the server's Karaf shell. If this default is incorrect, you can discover the correct port number by looking in the Red Hat JBoss Fuse *installDir/etc/org.apache.karaf.shell.cfg* file.

13. In **User Name**, enter the name used to log into the server.

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

**NOTE**

If the default user has been activated (uncommented) in the */etc/users.properties* file, the tooling autofills **User Name** and **Password** with the default user's name and password, as shown in [Step 11](#).

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

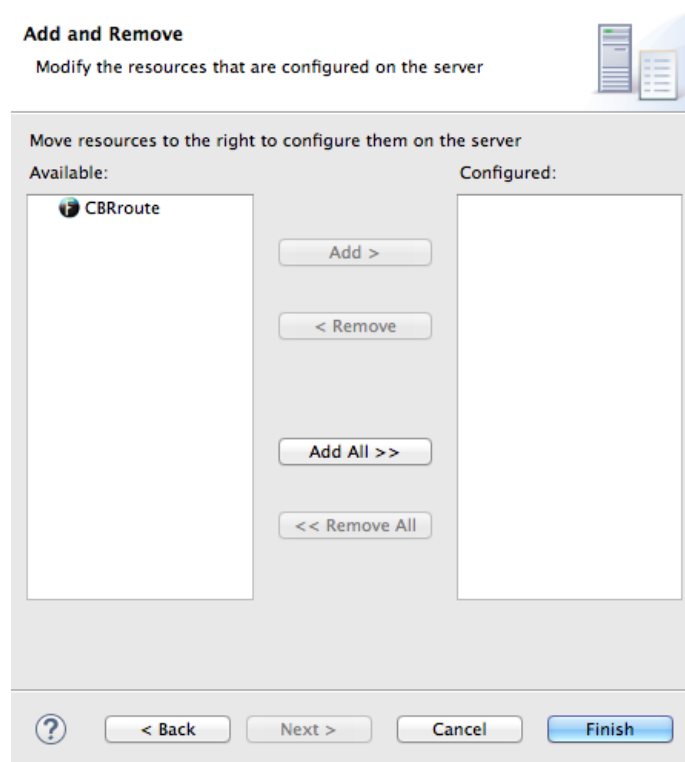
- `jaas:realms`—to list the realms
- `jaas:manage --index 1`—to edit the first (server) realm
- `jaas:useradd <username> <password>`—to add a user and associated password
- `jaas:roleadd <username> Administrator`—to specify the new user's role
- `jaas:update`—to update the realm with the new user information

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`.

14. 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.

15. Click `Next` to open the `Add and Remove` resources page:

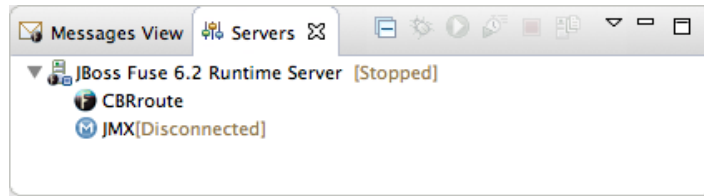


16. Select `CBRroute`, and click `Add>` to assign it to the JBoss Fuse server.

17. Click `Finish`.

`JBoss Fuse 6.2 Runtime Server [stopped]` appears in `Servers` view.

18. In `Servers` view, expand `JBoss Fuse 6.2 Runtime Server [stopped]`:



The **CBRRoute** module and **JMX[Disconnected]** appear as nodes under **JBoss Fuse 6.2 Runtime Server [stopped]** entry.

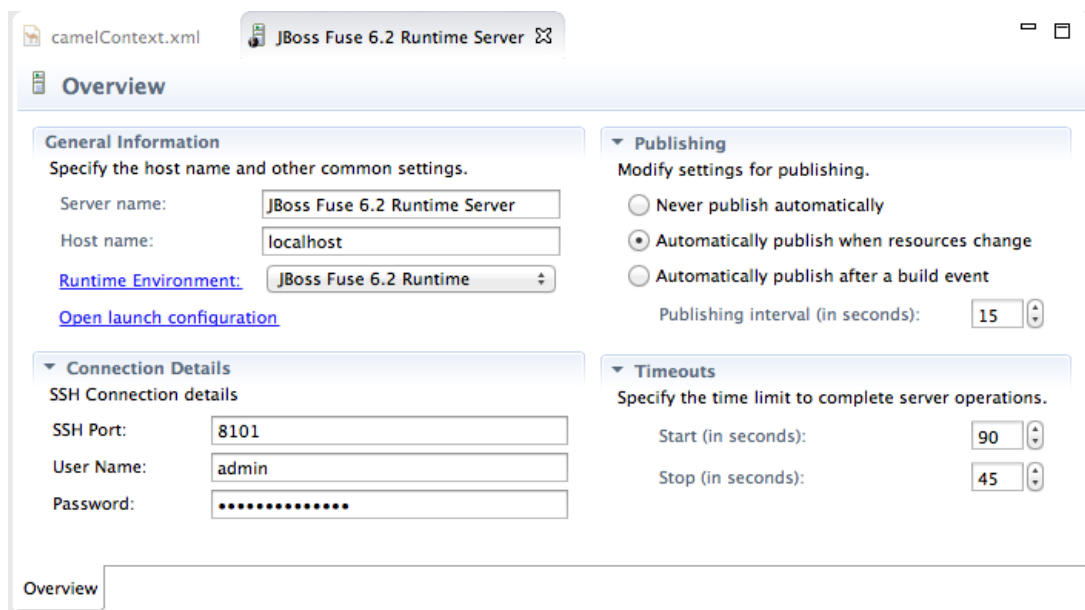
CONFIGURING THE PUBLISHING OPTIONS

Using publishing options, you can configure how and when your **CBRRoute** project is published to a running server:

- Automatically, immediately upon saving changes made to the project
- Automatically, at configured intervals after you have changed and saved the project
- Manually, when you select a publish operation

In this tutorial, you are going to configure immediate publishing upon saving changes to the **CBRRoute** project. To do so:

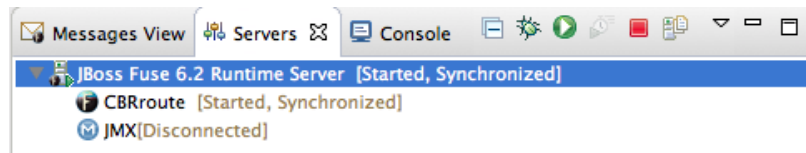
1. In **Servers** view, double-click the **JBoss Fuse 6.2 Runtime Server [stopped]** entry to open the server's editor:



2. On the server editor's **Overview** page, expand the **Publishing** section to expose the options.

Make sure the option **Automatically publish when resources change** is enabled.

Change the value of **Publishing interval** to speed up or delay publishing the project when changes have been made.



- o **JBoss Fuse 6.2 Runtime Server [Started, Synchronized]**



NOTE

For a server, synchronized means that all modules published on the server are identical to their local counterparts.

- o **CBRroute [Started, Synchronized]**

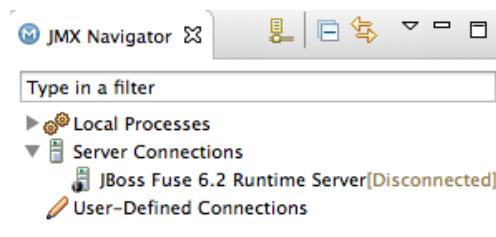


NOTE

For a module, synchronized means that the published module is identical to its local counterpart. Because automatic publishing is enabled, changes made to the CBRroute project are published in seconds (according to the value of the **Publishing interval**).

- o **JMX[Disconnected]**

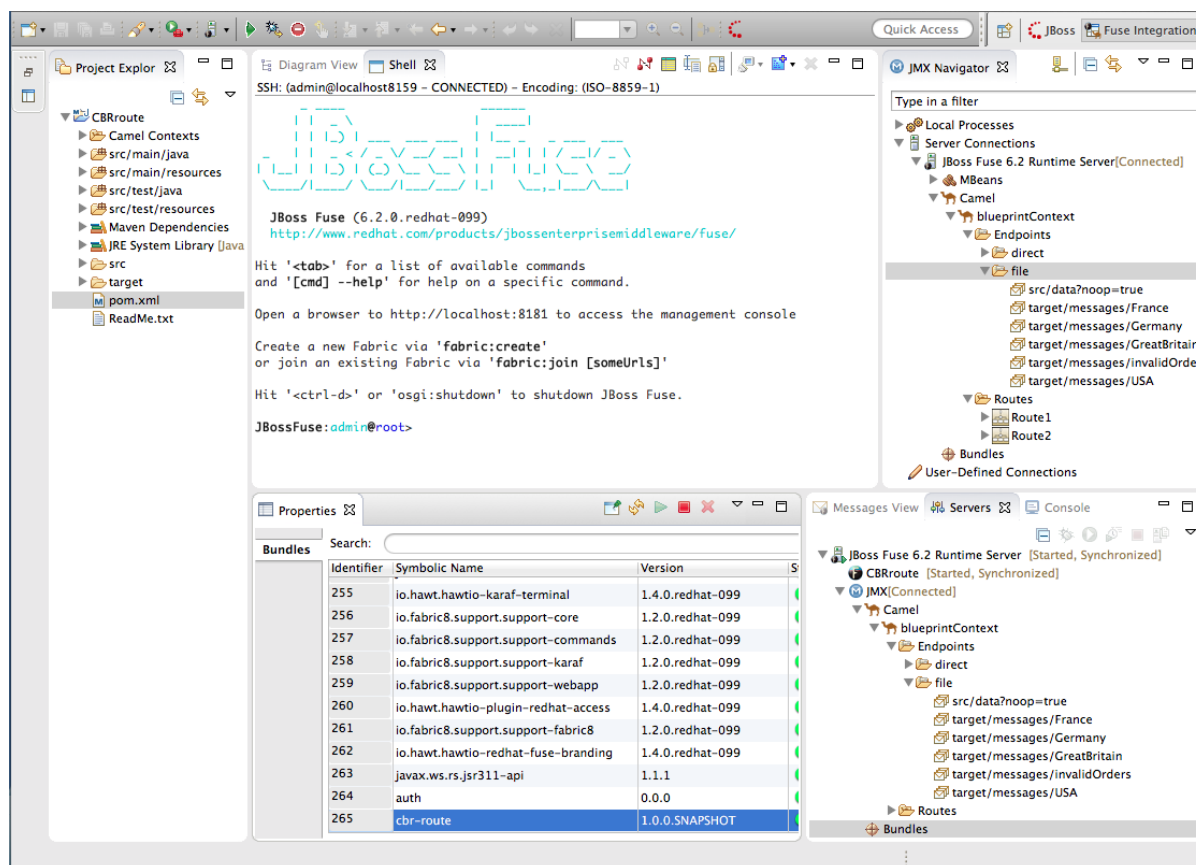
- **JMX Navigator displays JBoss Fuse 6.2 Runtime Server[Disconnected]:**



CONNECTING TO THE JBOSS FUSE 6.2 RUNTIME SERVER

When you connect to the **JBoss Fuse 6.2 Runtime Server**, you can see the published elements of your CBRroute project and interact with them.

1. In **Servers** view, double-click **JMX[Disconnected]** to connect to the runtime server.



IMPORTANT

If the CBRroute project contains a failed JUnit test, the published module will not be started nor its bundle installed. The published module will appear in **Servers view** under **JBoss Fuse 6.2 Runtime Server [Started, Synchronized]** as **CBRoute [Synchronized]**.

You need to correct the JUnit test case (see [the section called “Modifying the CamelContextXmlTest file”](#) for details) so that it runs on the CBRroute without errors, and save the updated test file. Saving the test file will trigger immediate publishing when that option is enabled. The module should then be started and its bundle installed.

2. Expand the **Camel** node in **JMX Navigator** to expose the elements of the CBRroute.

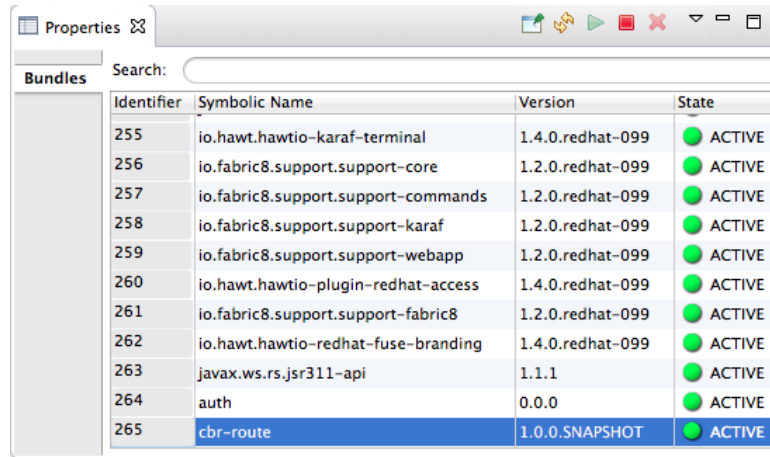
You can interact with the CBRroute routing context using either **Servers view** or **JMX Navigator**, but **JMX Navigator** provides more room to expand the routing context's nodes, making it easier for you to access them.



NOTE

Once the **blueprintCamel** node appears in **JMX Navigator** under **Server Connections** (or in **Servers view** under **JMX [Connected]**), you can start tracing on it, as described in [Chapter 7, To Trace a Message Through a Route](#).

3. Click the **Bundles** node to populate **Properties** view with the list of bundles installed on the **JBoss Fuse 6.2 Runtime Server**.



Identifier	Symbolic Name	Version	State
255	io.hawt.hawtio-karaf-terminal	1.4.0.redhat-099	ACTIVE
256	io.fabric8.support.support-core	1.2.0.redhat-099	ACTIVE
257	io.fabric8.support.support-commands	1.2.0.redhat-099	ACTIVE
258	io.fabric8.support.support-karaf	1.2.0.redhat-099	ACTIVE
259	io.fabric8.support.support-webapp	1.2.0.redhat-099	ACTIVE
260	io.hawt.hawtio-plugin-redhat-access	1.4.0.redhat-099	ACTIVE
261	io.fabric8.support.support-fabric8	1.2.0.redhat-099	ACTIVE
262	io.hawt.hawtio-redhat-fuse-branding	1.4.0.redhat-099	ACTIVE
263	javax.ws.rs.jsr311-api	1.1.1	ACTIVE
264	auth	0.0.0	ACTIVE
265	cbr-route	1.0.0.SNAPSHOT	ACTIVE

Start typing *cbr-route* in **Properties** view's Search tool to quickly determine whether your project's *cbr - route* bundle is included in the list. Note that it is the last bundle in the list, identified by its **Symbolic Name**, *cbr - route*, which is the Artifact Id you gave it in [Step 6](#) when you created the CBRroute project.



NOTE

Alternatively, you can issue the `list` command in **Shell** view to see a generated list of installed bundles.

UNINSTALLING THE CBRROUTE PROJECT

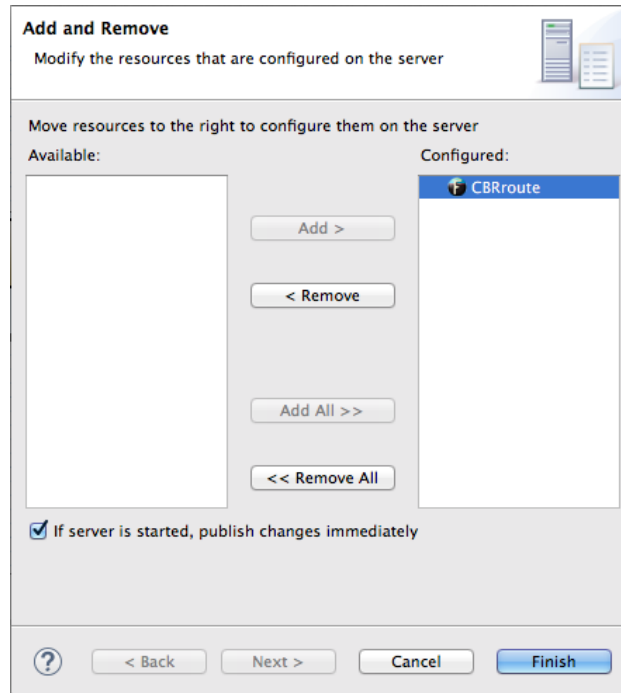


NOTE

You do not need to disconnect the JMX connection or stop the server to uninstall a published resource.

To remove the CBRroute resource from JBoss Fuse 6.2 Runtime Server:

1. In **Servers** view, right-click **JBoss Fuse 6.2 Runtime Server** to open the context menu.
2. Select **Add and Remove**:



3. In the **Configured** column, select **CBRroute**, and then click **Remove** to move the CBRroute resource to the **Available** column.
4. Click **Finish**.
5. In **Servers** view, right-click **JMX[Connected]** to open the context menu, and then click **Refresh**.

The **Camel** tree under **JMX[Connected]** disappears.



NOTE

In **JMX Navigator**, the **Camel** tree under **Server Connections > JBoss Fuse 6.2 Runtime Server[Connected]** also disappears.

6. With the **Bundles** page displayed, start typing *cbr-route* in **Properties** view's Search tool to verify that the bundle has been removed.