



# Red Hat JBoss Fuse 6.3

## Tooling Tutorials

Tooling Tutorials



# Red Hat JBoss Fuse 6.3 Tooling Tutorials

---

Tooling Tutorials

## Legal Notice

Copyright © 2018 Red Hat, Inc.

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

<b>CHAPTER 1. USING THE FUSE TOOLING RESOURCE FILES</b> .....	<b>4</b>
PREREQUISITES	4
DOWNLOADING THE RESOURCE FILES	4
INSTALLING THE PREFABRICATED ROUTING CONTEXT FILES	4
<b>CHAPTER 2. TO CREATE A NEW ROUTE</b> .....	<b>6</b>
GOALS	6
PREREQUISITES	6
CREATING THE FUSE INTEGRATION PROJECT	7
CREATING THE ROUTE	12
CREATING TEST MESSAGES	16
NEXT STEPS	19
FURTHER READING	19
<b>CHAPTER 3. TO RUN A ROUTE</b> .....	<b>20</b>
GOALS	20
PREREQUISITES	20
RUNNING THE ROUTE	20
VERIFYING THE ROUTE	21
FURTHER READING	22
<b>CHAPTER 4. TO ADD A CONTENT-BASED ROUTER</b> .....	<b>23</b>
GOALS	23
PREREQUISITES	23
ADDING AND CONFIGURING A CONTENT-BASED ROUTER	23
ADDING AND CONFIGURING LOGGING	25
ADDING AND CONFIGURING MESSAGE HEADERS	27
ADDING AND CONFIGURING AN OTHERWISE BRANCH	30
NEXT STEPS	36
FURTHER READING	37
<b>CHAPTER 5. TO ADD ANOTHER ROUTE TO THE CBR ROUTING CONTEXT</b> .....	<b>38</b>
GOALS	38
PREREQUISITES	38
RECONFIGURING THE EXISTING ROUTE FOR DIRECT CONNECTION	38
ADDING THE SECOND ROUTE	39
BUILDING AND CONFIGURING THE USA BRANCH OF THE SECOND ROUTE	40
BUILDING AND CONFIGURING THE GREAT BRITAIN BRANCH OF THE SECOND ROUTE	47
BUILDING AND CONFIGURING THE GERMANY BRANCH OF THE SECOND ROUTE	51
BUILDING AND CONFIGURING THE FRANCE BRANCH OF THE SECOND ROUTE	55
FINISHING UP	58
NEXT STEPS	60
FURTHER READING	61
<b>CHAPTER 6. TO DEBUG A ROUTING CONTEXT</b> .....	<b>62</b>
GOALS	62
PREREQUISITES	62
SETTING BREAKPOINTS	62
STEPPING THROUGH THE CBRROUTE ROUTING CONTEXT	64
CHANGING THE VALUE OF A VARIABLE	69
NEXT STEPS	76
<b>CHAPTER 7. TO TRACE A MESSAGE THROUGH A ROUTE</b> .....	<b>77</b>

GOALS	77
PREREQUISITES	77
ACCESSING FUSE INTEGRATION PERSPECTIVE	77
STARTING MESSAGE TRACING	80
DROPPING MESSAGES ON THE RUNNING CBRROUTE PROJECT	82
INITIALIZING AND CONFIGURING MESSAGES VIEW	83
ARRANGING DIAGRAM VIEW	85
STEPPING THROUGH MESSAGE TRACES	85
FINISHING UP	86
NEXT STEPS	87
<b>CHAPTER 8. TO TEST A ROUTE WITH JUNIT</b> .....	<b>89</b>
OVERVIEW	89
GOALS	89
PREREQUISITES	89
CREATING THE SRC/TEST FOLDER	90
CREATING THE JUNIT TEST CASE	92
MODIFYING THE BLUEPRINTXMLTEST FILE	95
MODIFYING THE POM.XML FILE	99
RUNNING THE JUNIT TEST	103
FURTHER READING	104
<b>CHAPTER 9. TO PUBLISH A FUSE PROJECT TO JBOSS FUSE</b> .....	<b>105</b>
GOALS	105
PREREQUISITES	105
DEFINING A RED HAT JBOSS FUSE SERVER	105
CONFIGURING THE PUBLISHING OPTIONS	109
STARTING THE RED HAT JBOSS FUSE SERVER	110
CONNECTING TO THE JBOSS FUSE 6.3 RUNTIME SERVER	114
UNINSTALLING THE CBRROUTE PROJECT	115



## CHAPTER 1. USING THE FUSE TOOLING RESOURCE FILES

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 routing context files are specific to particular tutorials. With the exception of [Chapter 2, To Create a New Route](#), using these prefabricated resource files enables you to complete the remaining 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

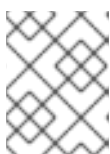
You must complete [Chapter 2, To Create a New Route](#), to create the project, the new routing context, and the folder that will hold the test messages. The code generated by this tutorial is used by [Chapter 3, To Run a Route](#) and by [Chapter 4, To Add a Content-Based Router](#).

### DOWNLOADING THE RESOURCE FILES

Click [here](#) to download the `jbds-10.1.zip` file. Move it to a convenient location external to the CBRroute project's workspace, and unzip it. It contains two folders:

- **Messages**  
This folder contains six prefabricated message files named `message1.xml`, `message2.xml`, ..., `message6.xml` used in all of the tutorials. In [Chapter 2, To Create a New Route](#), you will create the directory in which to store these message files, and also learn how to create them.
- **blueprintContexts**  
This folder contains two prefabricated routing context files named `blueprint5.xml`, and `blueprint6.xml`, which can be used in one or more of the tutorials:

Use prefabricated routing context file:	To complete tutorials:
<code>blueprint5.xml</code>	To Add Another Route to the CBR Routing Context
<code>blueprint6.xml</code>	To Debug a Routing Context To Trace a Message Through a Route To Test a Route with JUnit To Publish a Fuse Project to Red Hat JBoss Fuse



#### NOTE

`blueprint5.xml` is the routing context file generated by completing [Chapter 4, To Add a Content-Based Router](#).

### INSTALLING THE PREFABRICATED ROUTING CONTEXT FILES

To install the `blueprint#.xml` files:



1. Delete the existing `blueprint.xml` file from the `CBRroute/src/main/resources/OSGI-INF/blueprint/` folder.
2. Copy the `blueprint#.xml` file that corresponds to the tutorial that you want to complete to the vacated `CBRroute/src/main/resources/OSGI-INF/blueprint/` folder.
3. Rename the `blueprint#.xml` file `blueprint.xml`.
4. Follow the instructions for completing the target tutorial.

## CHAPTER 2. TO CREATE A NEW ROUTE

This tutorial walks you through the process of creating a Fuse Integration 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

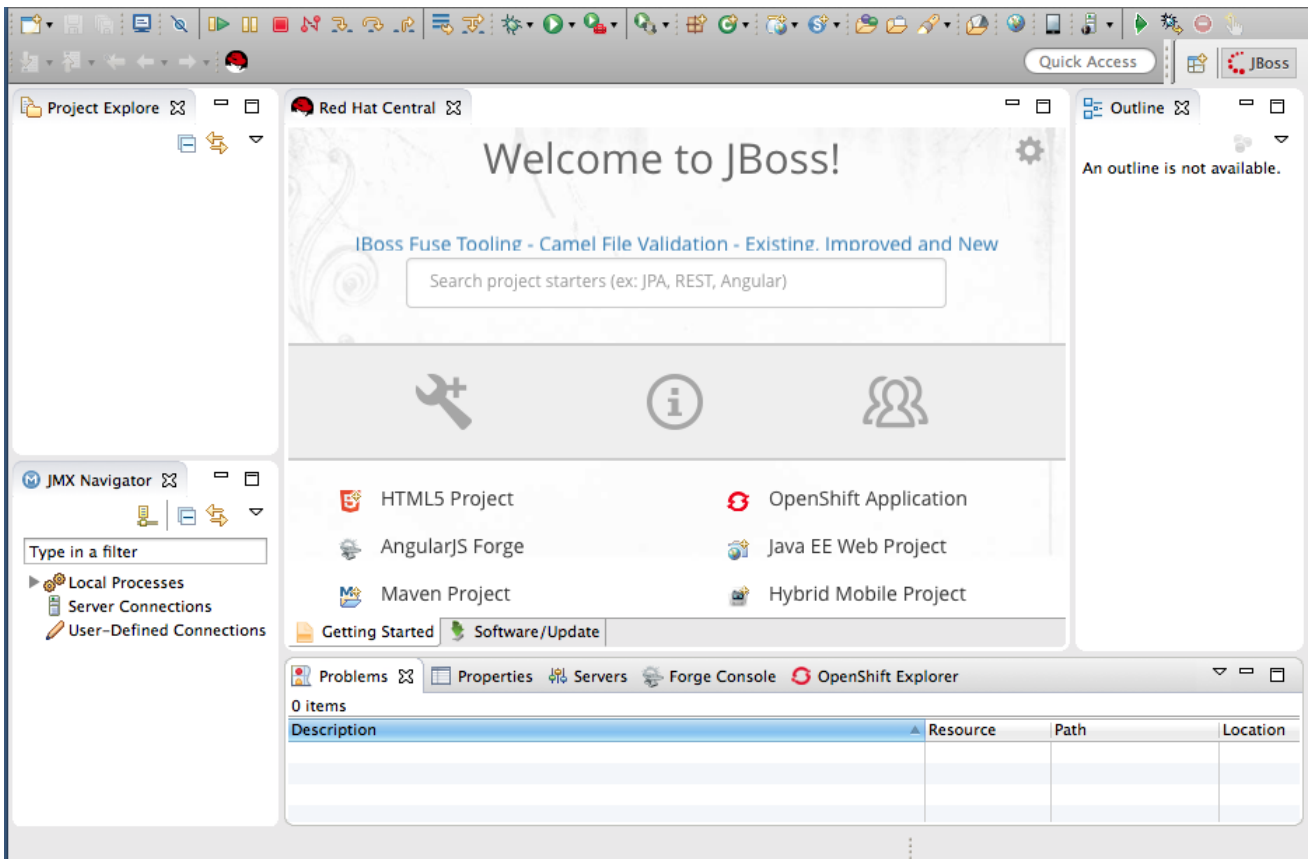
In this tutorial you will:

- Create a Fuse Integration project
- Create a new routing context
- Create a route
  - Add file 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

- JBoss Developer Studio 11.2 installed
- Red Hat JBoss Fuse Tooling 10.2 installed in JBoss Developer Studio 11.2
- In Developer Studio, select menu:Window[ > > Preferences > > Fuse Tooling > > Editor > ] and confirm selection of this option: **If enabled the ID values will be used for labels if existing** . This ensures that the label of the patterns and components that you place on the canvas will be the same as the labels shown in these tutorials.

When you start Developer Studio for the first time, it opens in the **JBoss** perspective:



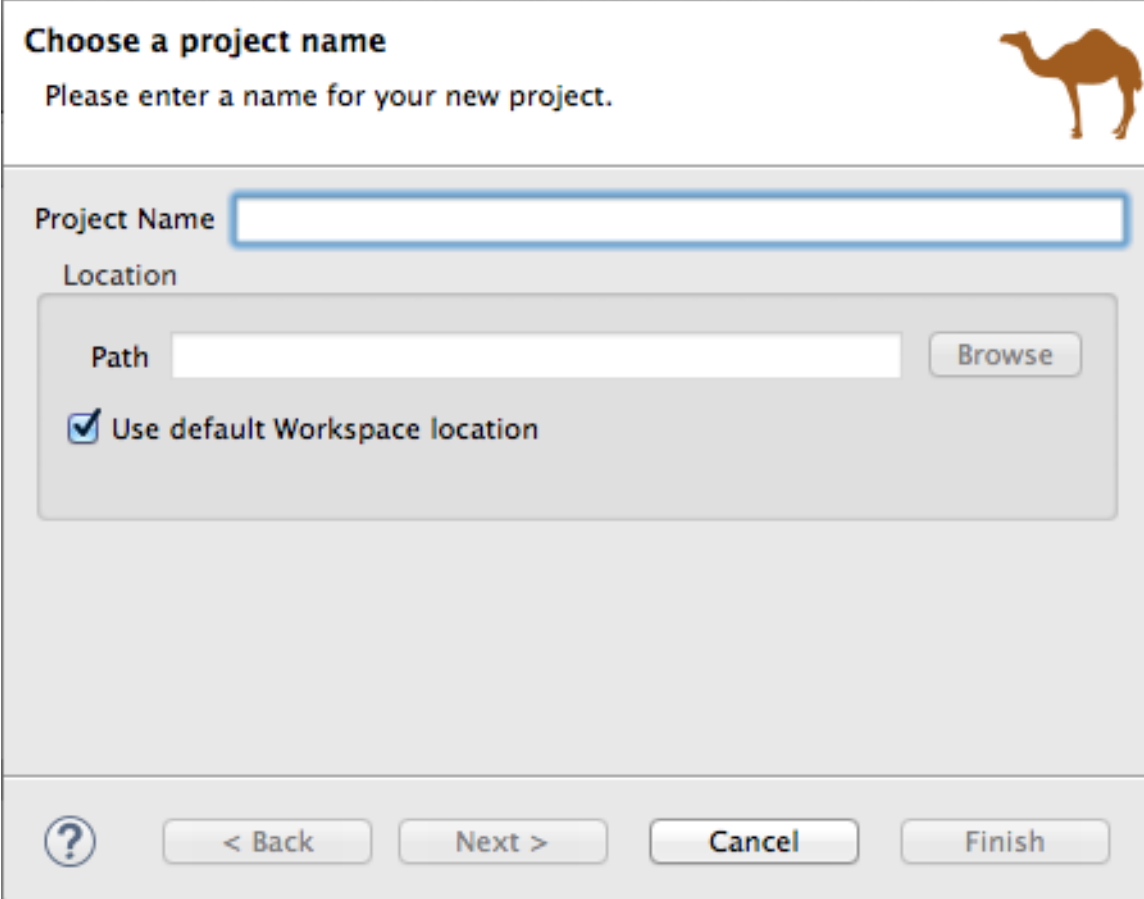
## NOTE

You can start this tutorial in the **JBoss** perspective or in the **Fuse Integration** perspective. If you start it in the **JBoss** perspective, the tooling will ask to switch you to the **Fuse Integration** perspective at the appropriate point in the tutorial.

## CREATING THE FUSE INTEGRATION PROJECT

To create a Fuse Integration project:

1. From the menu bar, select menu:File[ > > New > > Other > > JBoss Fuse > > Fuse Integration Project > ] and click **Next** to open the **New Fuse Integration Project** wizard:



**Choose a project name**

Please enter a name for your new project.

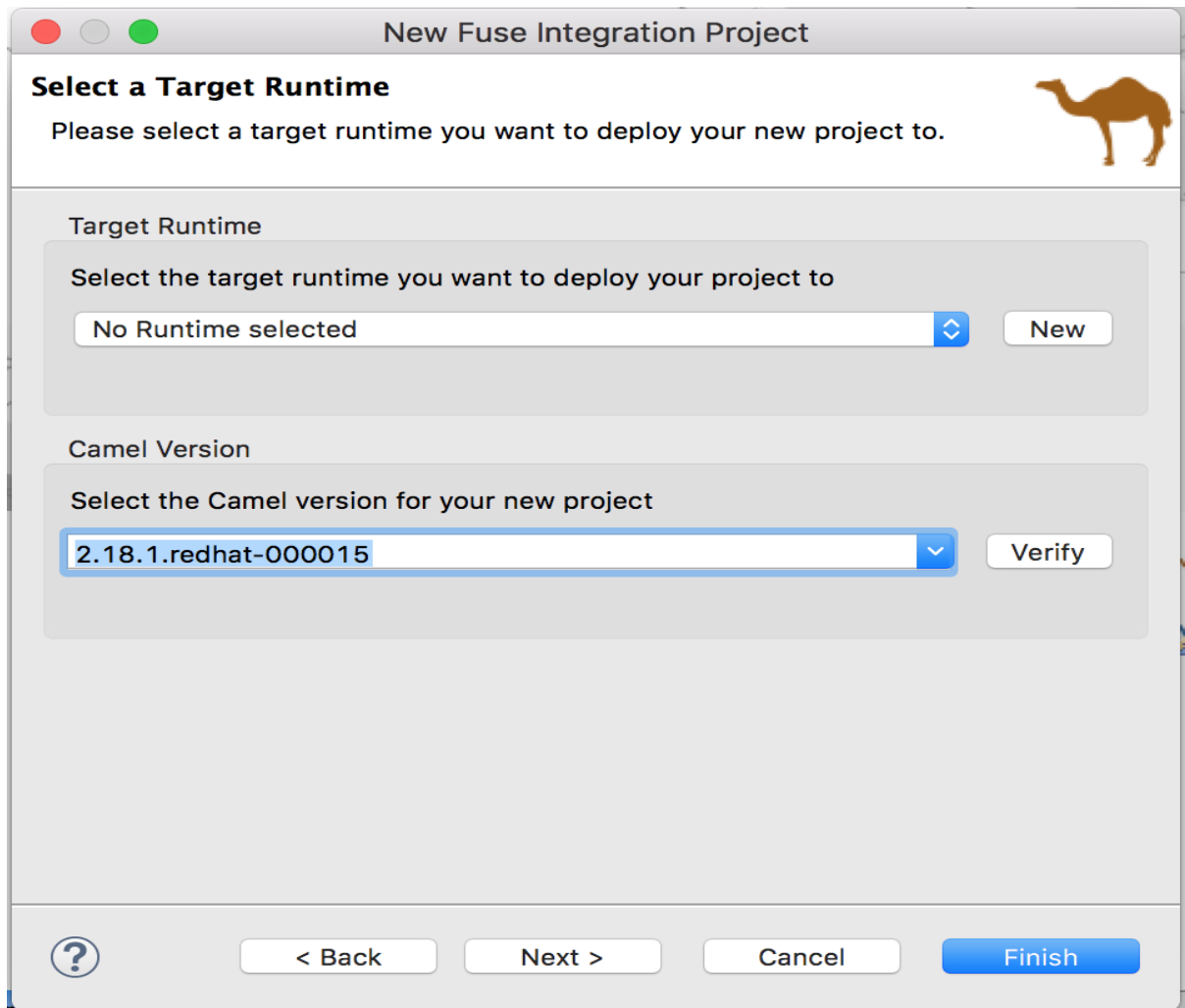
Project Name

Location

Path

Use default Workspace location

2. In the **Project Name** field, enter **CBRroute**.
3. Leave the **Use default workspace location** option as is.
4. Click **Next** to open the **Select a Target Runtime** page:



**Select a Target Runtime**

Please select a target runtime you want to deploy your new project to.

Target Runtime

Select the target runtime you want to deploy your project to

No Runtime selected

Camel Version

Select the Camel version for your new project

2.18.1.redhat-000015

5. Accept **No Runtime selected** for **Target Runtime**, and **2.18.1.redhat-000015** for **Camel Version**.



#### NOTE

You will add the runtime later in the tutorial [Chapter 9, To Publish a Fuse Project to JBoss Fuse](#).

6. Click **Next** to open the **Advanced Project Setup** page:

**Advanced Project Setup**

Please select how you would like to setup your project.



What would you like to do?

Start with an empty project

Use a predefined template

type filter text (filters on name, descrip

- ▶ JBoss Fuse
- ▶ Fuse on EAP
- ▶ Fuse on OpenShift

Select the project type

Blueprint DSL

Spring DSL

Java DSL

7. Leave the **Start with an empty project** and **Blueprint DSL** options selected.

8. Click **Finish**.

Fuse Tooling starts downloading from the Maven repository all of the files it needs to build the project, and then adds the new project to **Project Explorer**.

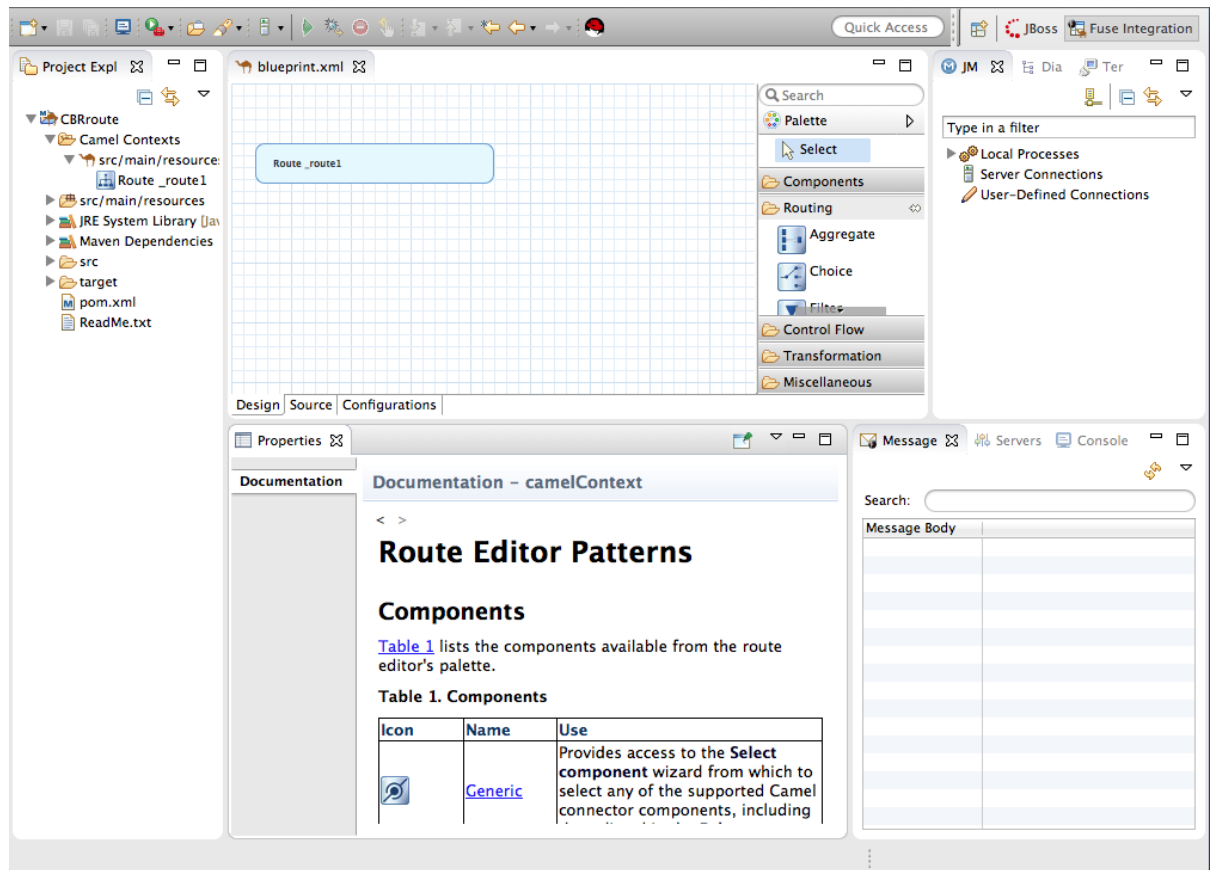
If you are not in the **Fuse Integration** perspective, the tooling asks whether you want to switch to it now:

**Open Associated Perspective?**

This kind of project is associated with the **Fuse Integration** perspective. Do you want to open this perspective now?

Remember my decision

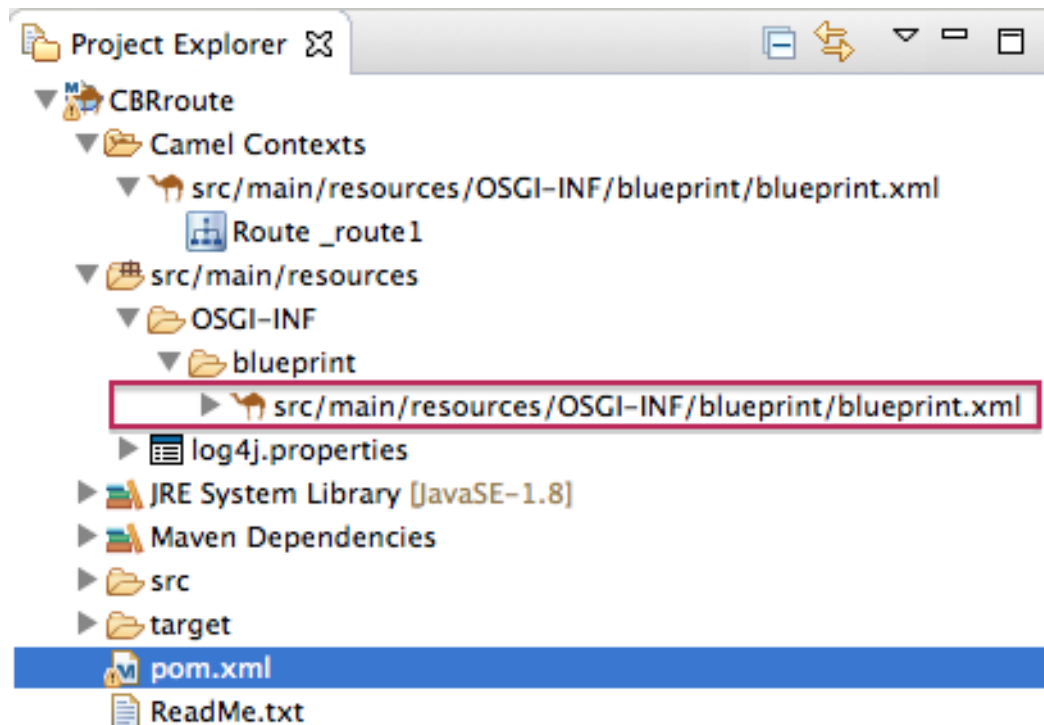
9. Click **Yes** to open the new **CBRRoute** project in the **Fuse Integration** perspective:



The new CBRroute project contains everything needed to create and run routes. As shown in Figure 2.1, “Generated project files”, the files generated for CBRroute include:

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

Figure 2.1. Generated project files






## NOTE

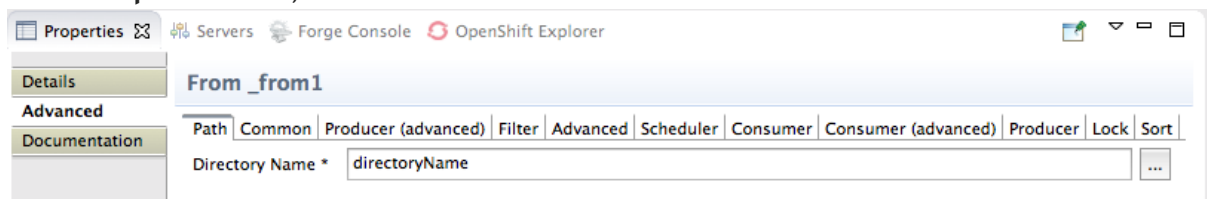
Notice that the `pom.xml` entry in **Project Explorer** is decorated with a warning symbol.

You can safely ignore this warning or eliminate it by opening the `pom.xml` file in the tooling's XML editor, and delete the `<version>` element from each dependency: `camel-core`, `camel-blueprint`, and `camel-test-blueprint`. Save the `pom.xml` file.

## CREATING THE ROUTE

To create the route:

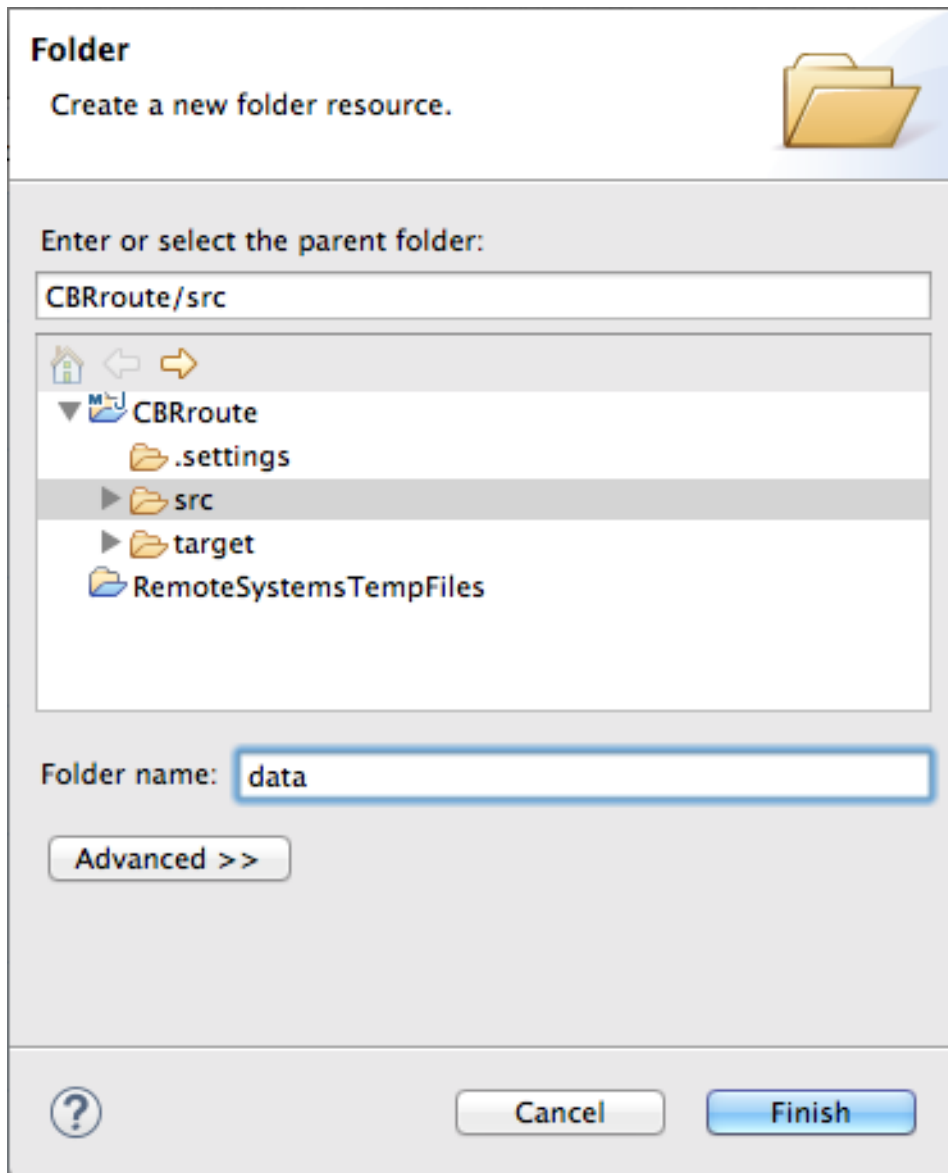
1. Click the **Design** tab at the bottom, left of the canvas to return to the graphic display of the route.
2. Drag a **File** component (  ) from the **Palette's Components** drawer to the canvas, and drop it in the `Route_route1` container node.  
The **File** component changes to a **From \_from1** node inside the `Route_route1` container node.
3. On the canvas, select the **From \_from1** node.  
The **Properties** view, located below the canvas, displays the node's property fields for editing.
4. In the **Properties** view, click the **Advanced** tab:



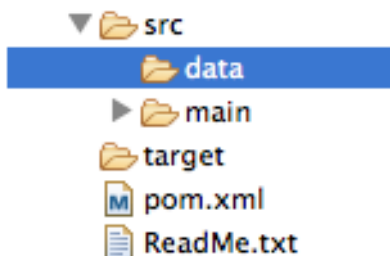
You need to create a folder for the project's source data and enter that folder's name in the **Directory Name** field.

- a. In **Project Explorer**, right-click `CBRRoute/src/` to open the context menu.
- b. Select menu: [ > New > > Folder > ] to open the **New Folder** wizard:

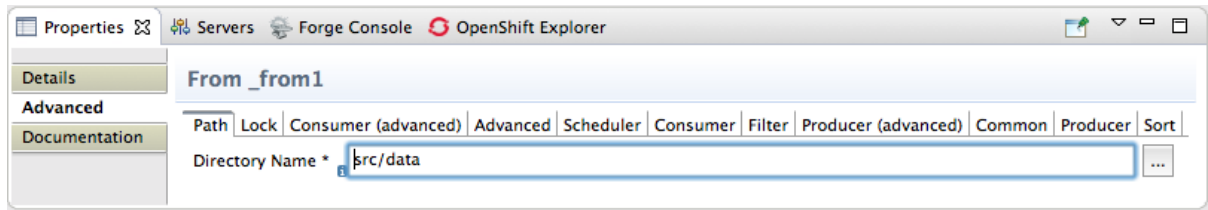




- c. Check that **CBRroute/src** appears in the **Enter or select the parent folder** field. Otherwise enter it manually, or select it from the graphical representation of the project's hierarchy.
- d. In the **Folder name** field, enter **data**, and then click **Finish**.  
The new **data** folder appears in **Project Explorer**, under the **src** folder:

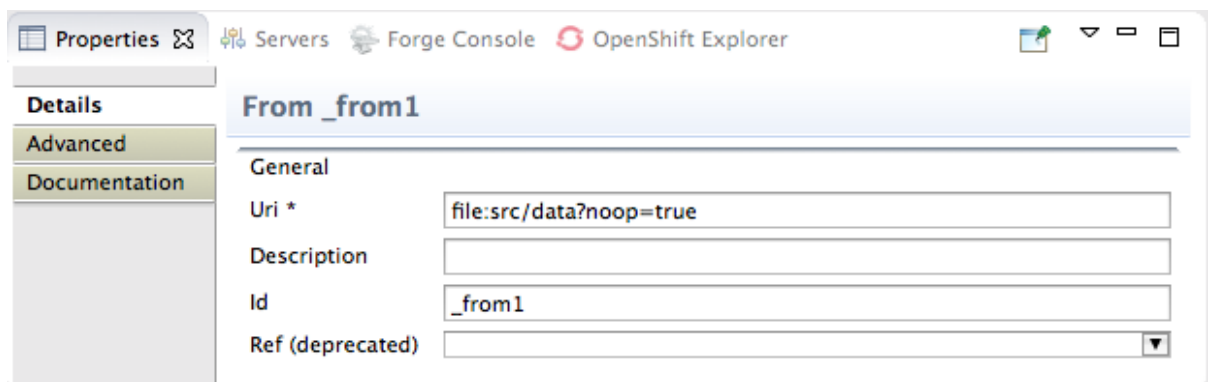


5. In the **Properties** view, return to the **From \_from1** node's **Advanced** tab.
6. In the **Directory Name** field, enter **src/data**:

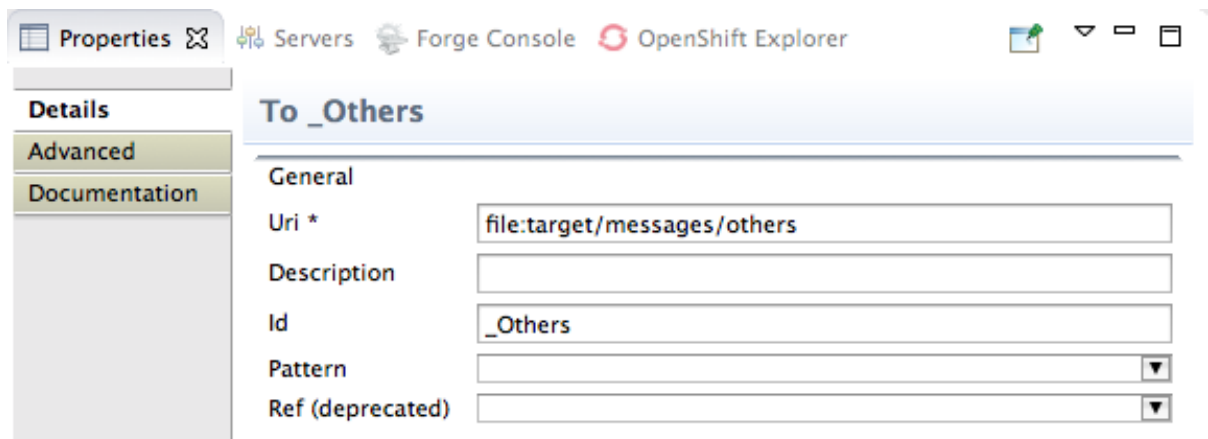


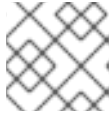
The path `src/data` is relative to the project's directory.

7. On the **Consumer** tab, 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.
8. Select the **Details** tab to open the file node's **Details** page.  
The tooling automatically populates the **Uri** field with the **Directory Name** and **Noop** properties you configured on the **Advanced** tab. It also populates the **Id** field with an autogenerated ID (`_from1`):



9. Leave the autogenerated **Id** as is.
10. Drag another **File** component from the **Palette's Components** drawer and drop it in the **Route\_route1** container node.  
The **File** component changes to a **To\_to1** node inside the **Route\_route1** container node.
11. On the canvas, select the **To\_to1** node.  
The **Properties** view, located below the canvas, displays the node's property fields for editing.
12. On the **Details** tab, enter `file:target/messages/others` in the **Uri** field, and `_Others` in the **Id** field:



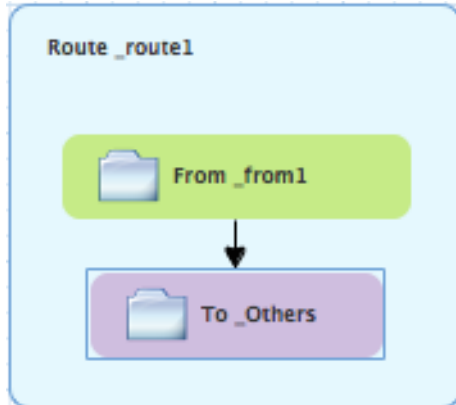
**NOTE**

The tooling will create the `target/messages/others` folder at runtime.

13. In the `Route_route1` container, select the `From_from1` node and drag its connector arrow (



) over the `To_Others` node, then release it:

**NOTE**

The two file nodes are connected and aligned on the canvas according to the route editor's layout direction preference setting. The choices are **Right** and **Down** (default).

**NOTE**

If you do not connect the nodes before you close the project, the tooling automatically connects them when you reopen it.

14. Select **File** → **Save** to save the route.

15. Click the **Source** tab at bottom, left of the canvas to display the XML for the route. The `camelContext` element will look like [Example 2.1, “XML for CBRroute”](#):

**Example 2.1. XML for CBRroute**

```
<?xml version="1.0" encoding="UTF-8"?>

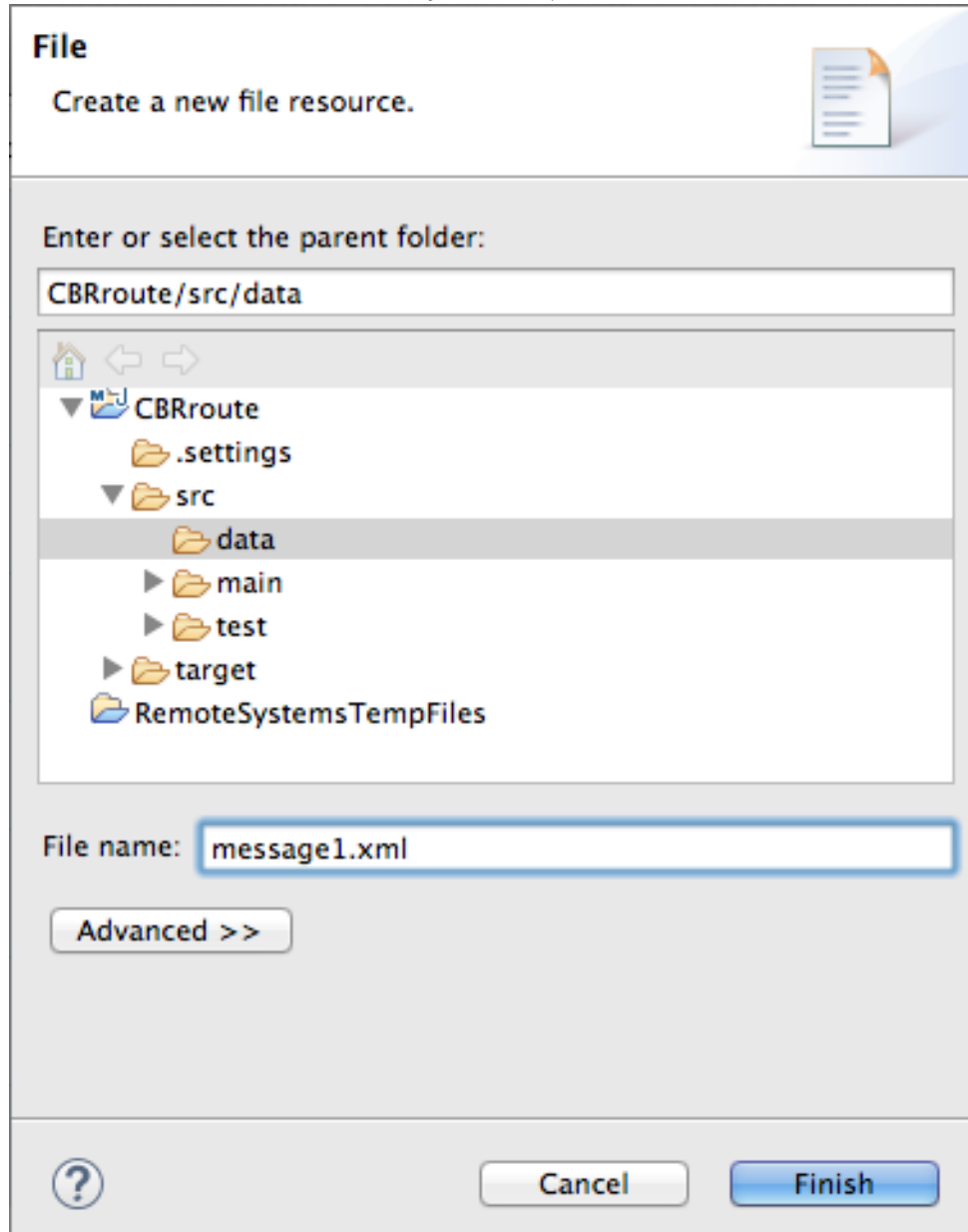
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.osgi.org/xmlns/blueprint/v1.0.0
    https://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd
    http://camel.apache.org/schema/blueprint
    http://camel.apache.org/schema/blueprint/camel-blueprint.xsd">

  <camelContext id="_context1"
    xmlns="http://camel.apache.org/schema/blueprint">
    <route id="_route1">
      <from id="_from1" uri="file:src/data?noop=true"/>
      <to id="_Others" uri="file:target/messages/others"/>
    </route>
  </camelContext>
</blueprint>
```

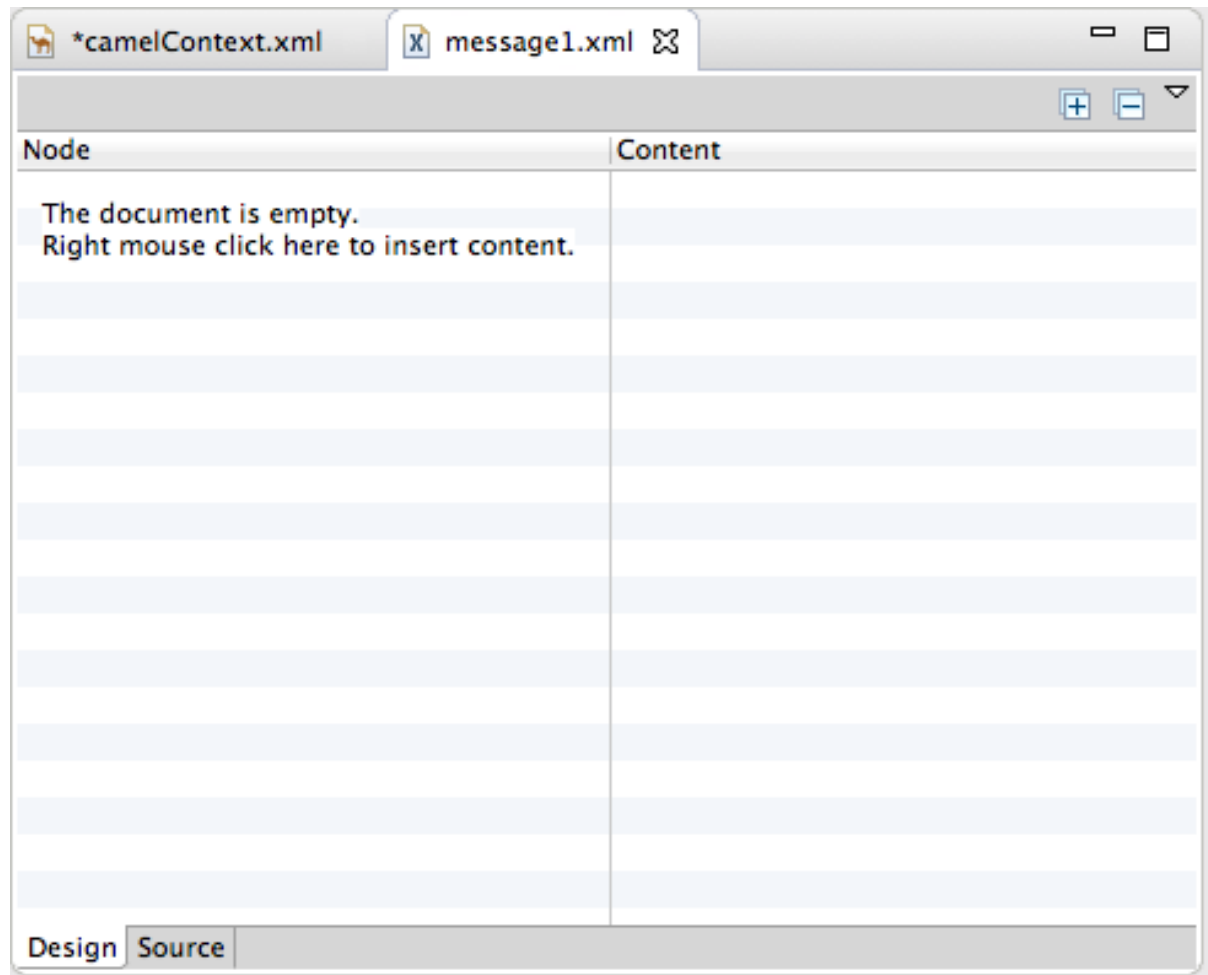
## CREATING TEST MESSAGES

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

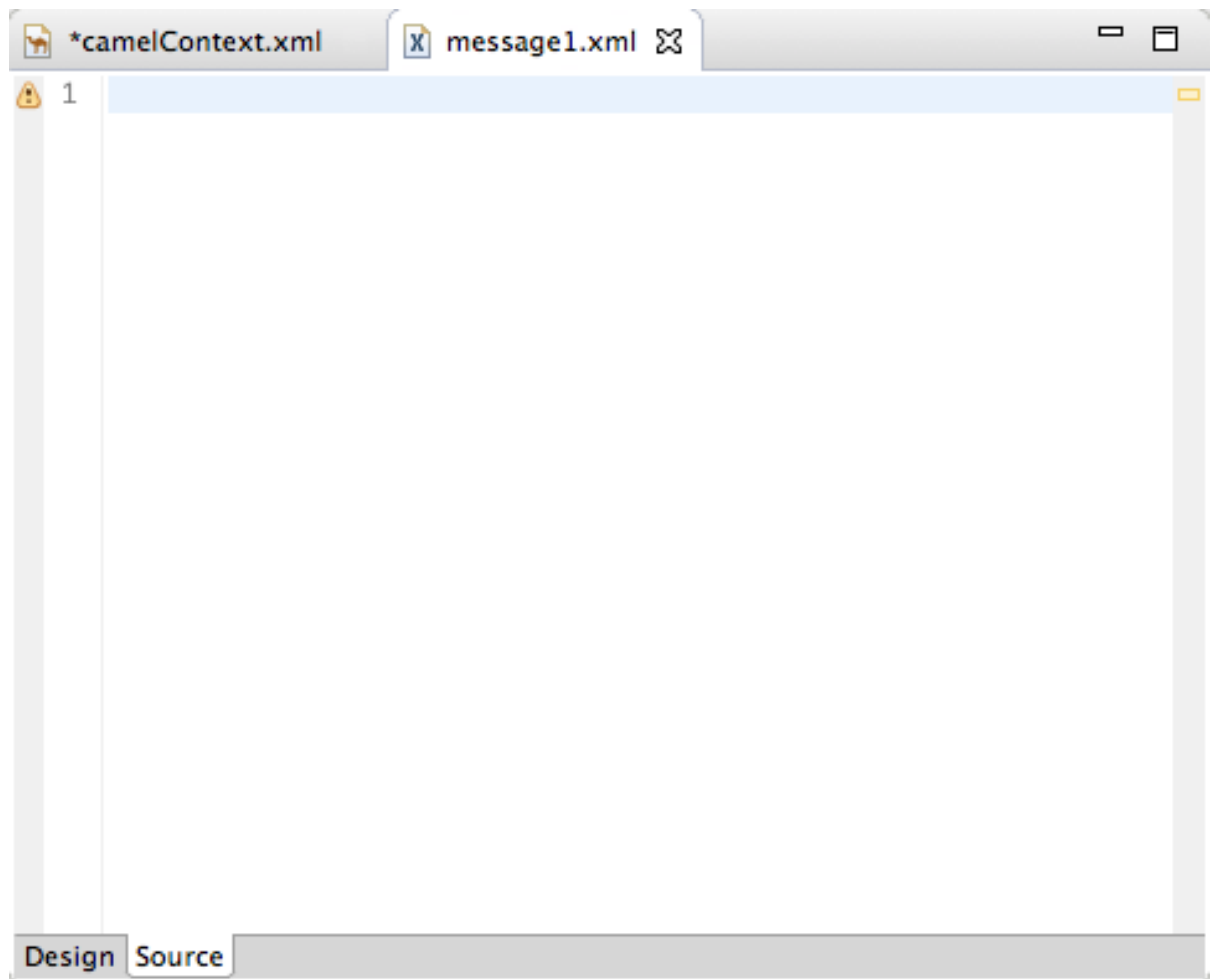
1. In **Project Explorer**, right-click **CBRRoute** to open the context menu.
2. Click menu:New[ > > Fuse Message > ] to open the **New File** wizard:



3. 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.
4. In the **File Name:** field, enter **message1.xml**.
5. Click **Finish** to open the test message, **message1.xml** in the **Design** tab:



6. Click the **Source** tab at the bottom of the canvas:




7. In the Source tab, 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>
```



#### NOTE

You can safely ignore the  on the last line of the newly created `message1.xml` file, which advises you that there are no grammar constraints (DTD or XML Schema) referenced by the document.

8. Save the file, and close it.
9. If you haven't already done so, download the prefabricated test message files (see [Chapter 1](#),

Using the [Fuse Tooling Resource Files](#) for instructions). Copy `message2.xml` through `message6.xml` into the newly created `CBRroute/src/data` folder. You will use all six test messages in the remaining Fuse Tooling tutorials.

Table 2.1, “Preconstructed test messages” shows the contents of each remaining prefabricated 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
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 the [Editing a routing context in the route editor](#) section in “Tooling User Guide”.

## CHAPTER 3. TO RUN A ROUTE

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

### GOALS

In this tutorial you will:

- Run a route as a Local Camel Context (without tests)
- Send messages through the route
- Examine the messages received by the endpoints

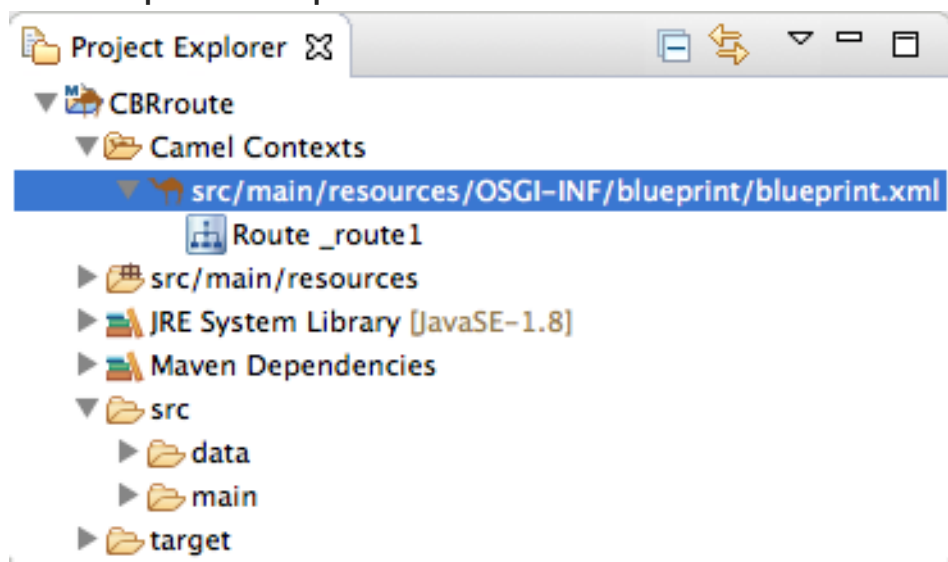
### PREREQUISITES

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 Integration project”](#).
2. In **Project Explorer**, select **CBRRoute/Camel Contexts/src/main/resources/OSGi-INF/blueprint/blueprint.xml**:



3. Right-click it to open the context menu, then select `menu:Run As[ > > Local Camel Context (without tests) > ]`.



**NOTE**

If you select **Local Camel Context** instead, the tooling automatically tries to run the routing context against a supplied JUnit test. Because one does not exist, the tooling reverts to running the routing context without tests. In the [Chapter 8, To Test a Route with JUnit](#) tutorial, you will create a JUnit test case and modify it specifically for testing the **CBRroute** 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 at the end of the output indicate that the route executed successfully:

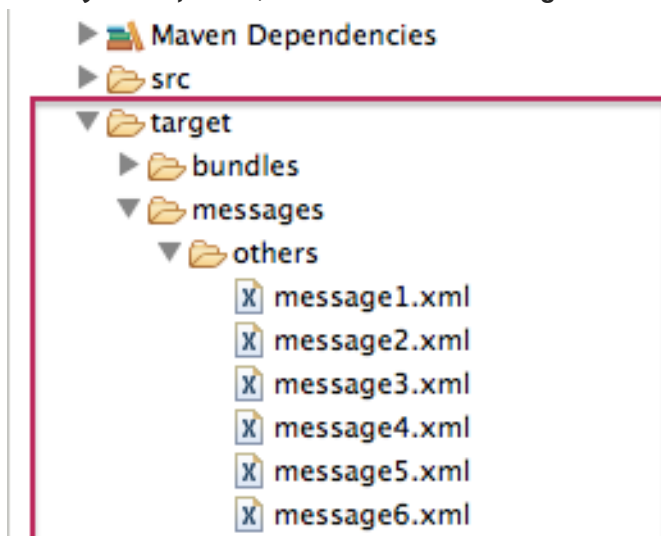
```
...
[Blueprint Extender: 3] BlueprintCamelContext INFO  Route: _route1
started and consuming from:Endpoint[file://src/data?noop=true]
[Blueprint Extender: 3] BlueprintCamelContext INFO  Total 1 routes,
of which 1 are started.
[Blueprint Extender: 1] BlueprintCamelContext INFO  Apache Camel
2.18.0.redhat-000015 (CamelContext: ...) started in 0.163 seconds
[Blueprint Extender: 3] BlueprintCamelContext INFO  Apache Camel
2.18.0.redhat-000015 (CamelContext: ...) started in 0.918 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 **CBRroute**.
2. Right-click it to open the context menu, then select **Refresh**.
3. In **Project Explorer**, locate the folder **target/messages/** and expand it:



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 route editor's **Design** tab, then select the **Source** tab at the bottom, left of the canvas to see the XML code. Its contents should match that shown in [Example 3.1, "Contents of message1.xml"](#).

**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 see the [Editing a routing context in the route editor](#) section in "Tooling User Guide".
- Deploying Apache Camel applications see [Developing and Deploying Applications](#).

## CHAPTER 4. TO ADD A CONTENT-BASED ROUTER

This tutorial shows how to add 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 Set Header EIP after each log endpoint
  - Add an Otherwise branch to the content-based router

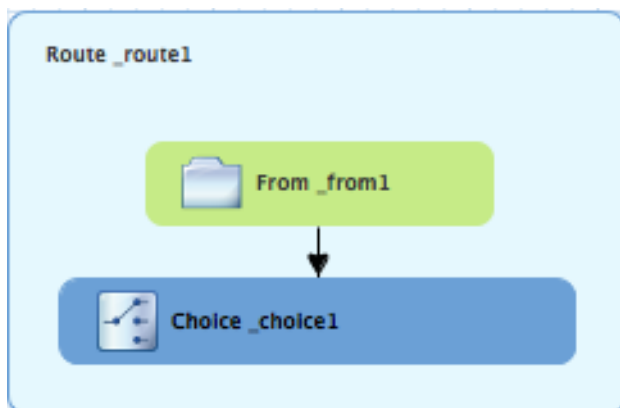
### PREREQUISITES

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

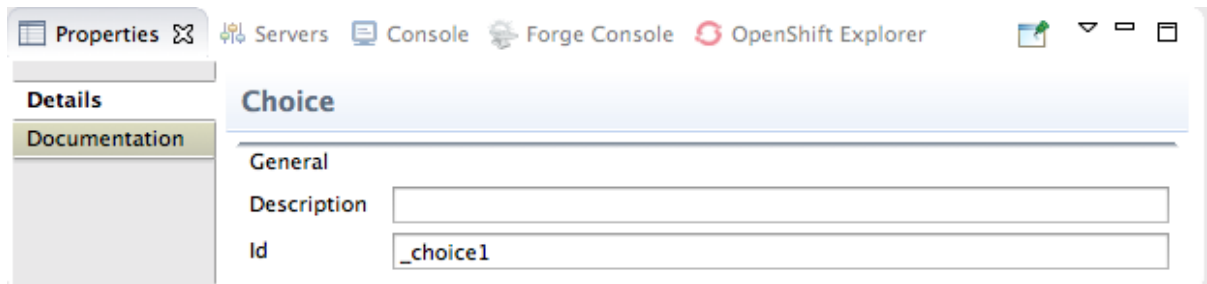
### ADDING AND CONFIGURING A CONTENT-BASED ROUTER

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

1. In **Project Explorer**, double-click **CBRroute/src/main/resources/OSGI-INF/blueprint/blueprint.xml** to open your **CBRroute** project.
2. On the canvas, select the **To\_Others** node and then select the trash can above and to the right to delete it.
3. In the **Palette**, open the **Routing** drawer and drag a **Choice** (  ) pattern to the canvas and drop it in the **Route\_route1** container. The **Route\_route1** container expands to accommodate the **Choice\_choice1** node.
4. In the **Route\_route1** container, select the **From\_from1** node and drag its connector arrow over the **Choice\_choice1** node, then release it:

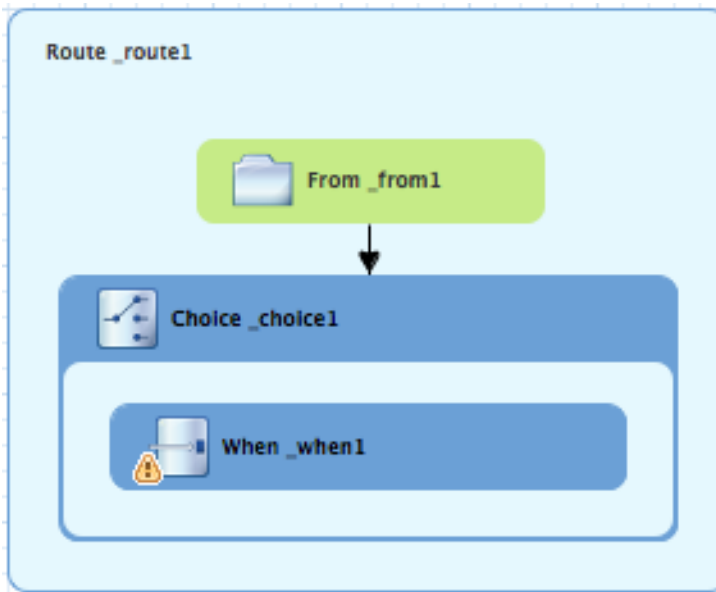



5. In the **Properties** view, **\_choice1** appears in the **Id** field:



Leave the Id field as is.

6. From the **Routing** drawer, drag a **When** (  ) pattern to the canvas and drop it on the **Choice\_choice1** node:



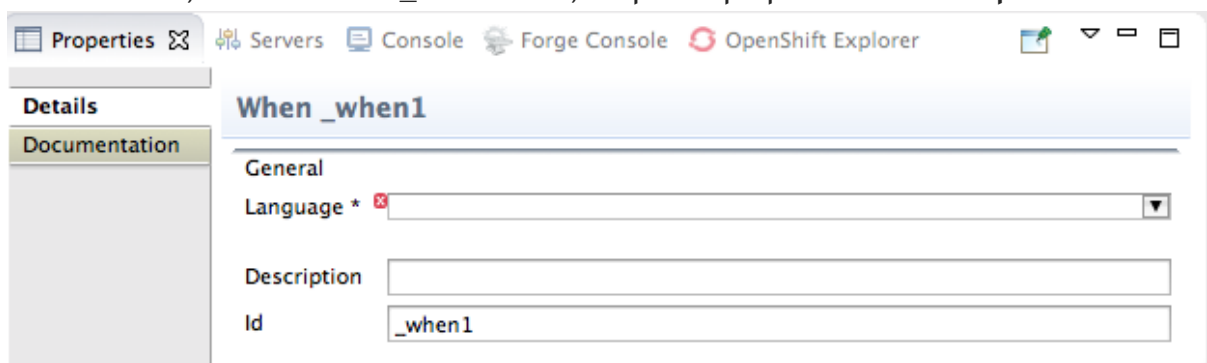
The **Choice\_choice1** container expands to accommodate the **When\_when1** node. The  decorating the **When\_when1** node indicates that one or more required property values must be set.




### NOTE

The tooling prevents you from dropping a pattern at an invalid drop point in a Route container.

7. On the canvas, select the **When\_when1** node, to open its properties in the **Properties** view:



8. Click the  button in the **Language** field to open the list of available languages, and select **xpath**:

The screenshot shows the 'When \_when1' configuration window in OpenShift Explorer. The 'Language' dropdown is set to 'xpath'. The 'Expression' field contains the XPath expression: `/order/orderline/quantity/text() > /order/orderline/maxAllowed/text()`. Below the expression field, there are several empty input fields for 'Document Type', 'Factory Ref', 'Header Name', 'Id', 'Object Model', and 'Result Type'. The 'Log Namespaces' and 'Saxon' checkboxes are unchecked, while the 'Trim' checkbox is checked. The 'Description' field is empty, and the 'Id' field at the bottom is set to `_when1`.



## NOTE

Once you select the expression **Language**, the **Properties** view displays its properties in an indented list directly below the **Language** field. The **Id** property in this list sets the ID of the expression. The **Id** property following the **Description** field sets the ID of the **When** node.

9. 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.

10. Leave each of the remaining properties as is.  
Enabling **Trim** removes any leading or trailing white spaces and line breaks from the message.
11. On the menu bar, click **File** → **Save** to save the routing context file.
12. Click the **Source** tab to view the XML for the route:


```

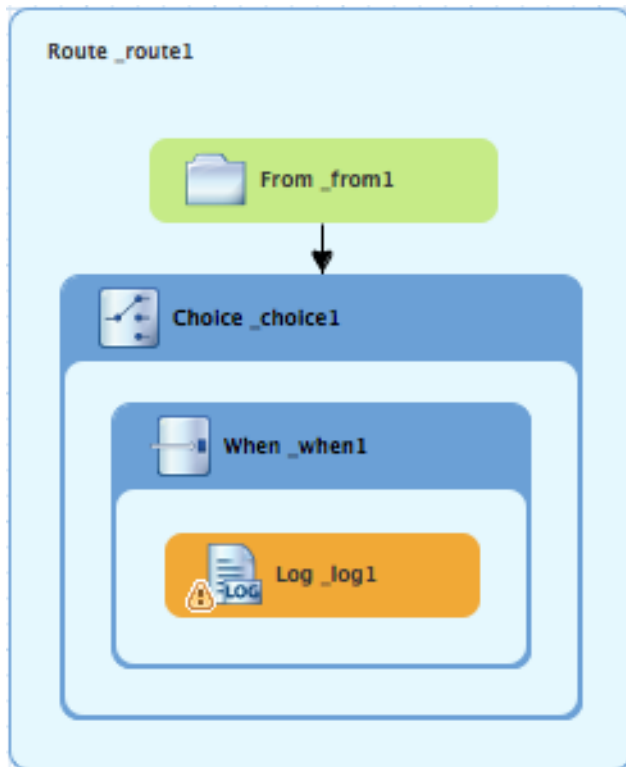
1 <?xml version="1.0" encoding="UTF-8"?>
2 <blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
3   xmlns:camel="http://camel.apache.org/schema/blueprint"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.osgi.org/xmlns/blue
5   <camelContext id="_context1" xmlns="http://camel.apache.org/schema/blueprint">
6     <route id="_route1">
7       <from id="_from1" uri="file:src/data?noop=true"/>
8       <choice id="_choice1">
9         <when id="_when1">
10          <xpath>/order/orderline/quantity/text() &gt; /order/orderline/maxAllowed/text()</xpath>
11        </when>
12      </choice>
13    </route>
14  </camelContext>
15 </blueprint>

```

## ADDING AND CONFIGURING LOGGING

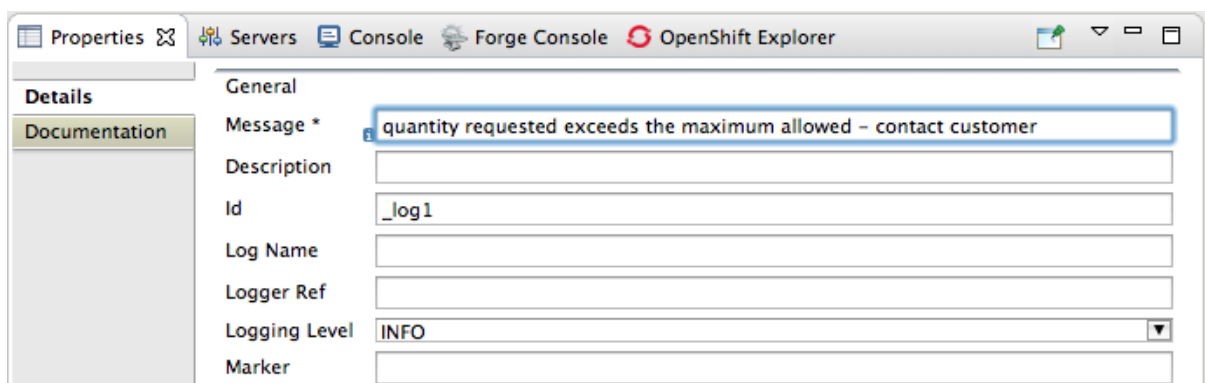
To add logging to your route:

1. In the **Palette**, open the **Components** drawer and select a **Log** (  ) component.
2. Drag the **Log** component to the canvas and drop it on the **When\_when1** node:



The **When\_when1** container expands to accommodate the **Log\_log1** node.

3. On the canvas, select the **Log\_log1** node to open its properties in the **Properties** view.
4. In the **Message** field, enter **quantity requested exceeds the maximum allowed - contact customer**:



5. Leave each of the remaining properties as is.




## NOTE

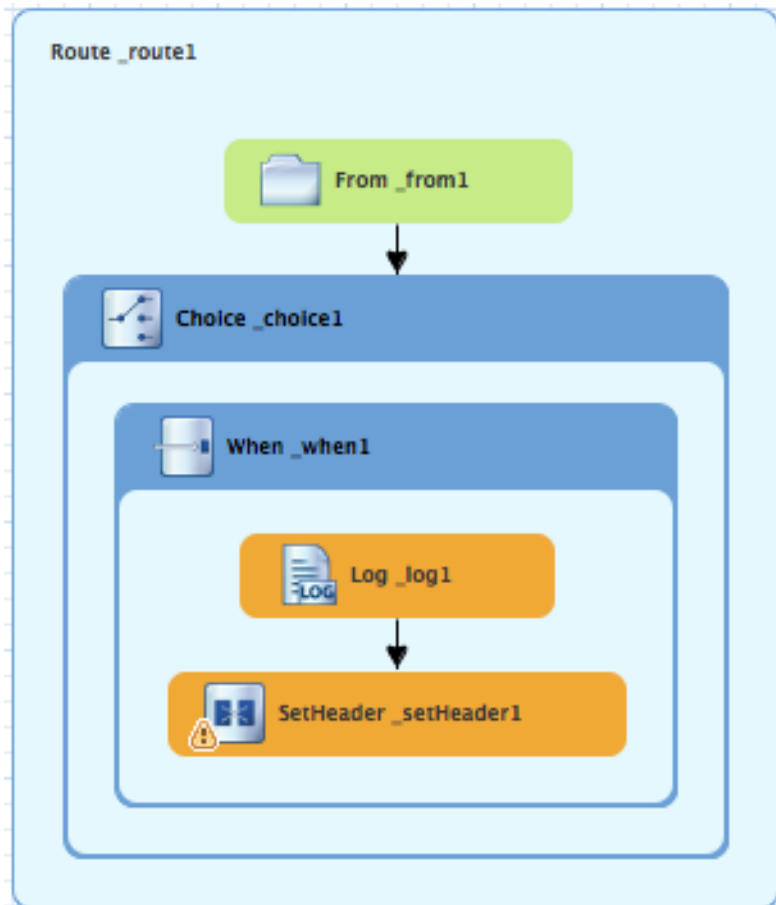
In the **Fuse Integration** perspective's **Messages** view, the tooling inserts the contents of the log node's **Id** field in the **Trace Node Id** column for message instances, when tracing is enabled on the route (see [Chapter 7, To Trace a Message Through a Route](#)). In the **Console**, it adds the contents of the log node's **Message** field to the log data whenever the route runs.

6. On the menu bar, click **File** → **Save** to save the routing context file.

## ADDING AND CONFIGURING MESSAGE HEADERS


To add and configure message headers:

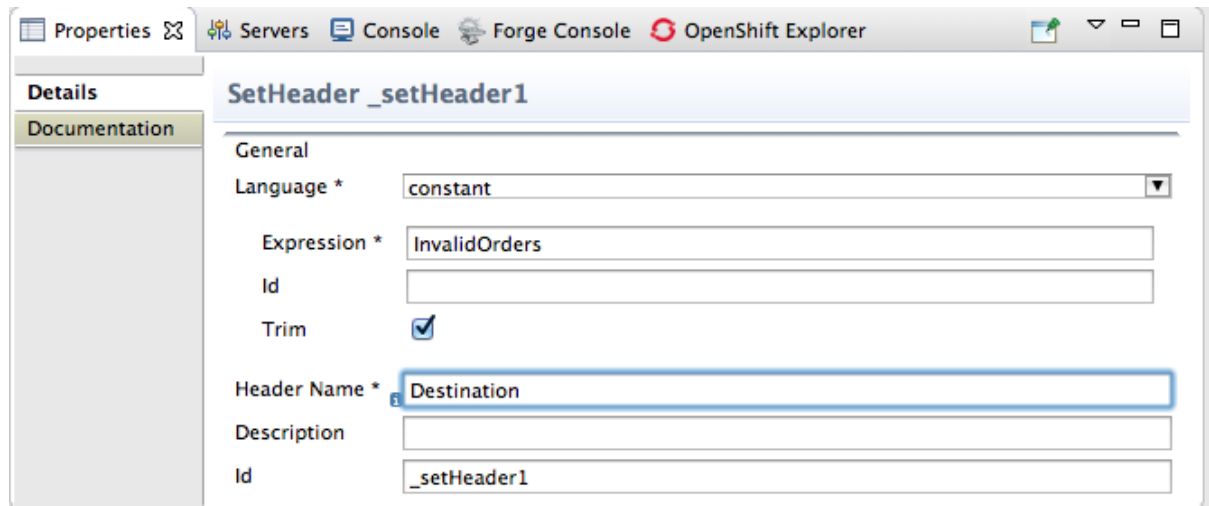
1. In the **Palette**, open the **Transformation** drawer and select a **Set Header** (  ) pattern.
2. Drag the **Set Header** pattern to the canvas and drop it in the **When\_when1** container. The **When\_when1** container expands to accommodate the **SetHeader\_setHeader1** node.
3. On the canvas, select the **Log\_log1** node and drag its connector arrow over the **SetHeader\_setHeader1** node, and then release it:




4. On the canvas, select the **SetHeader\_setHeader1** node to open its properties in the **Properties** view:

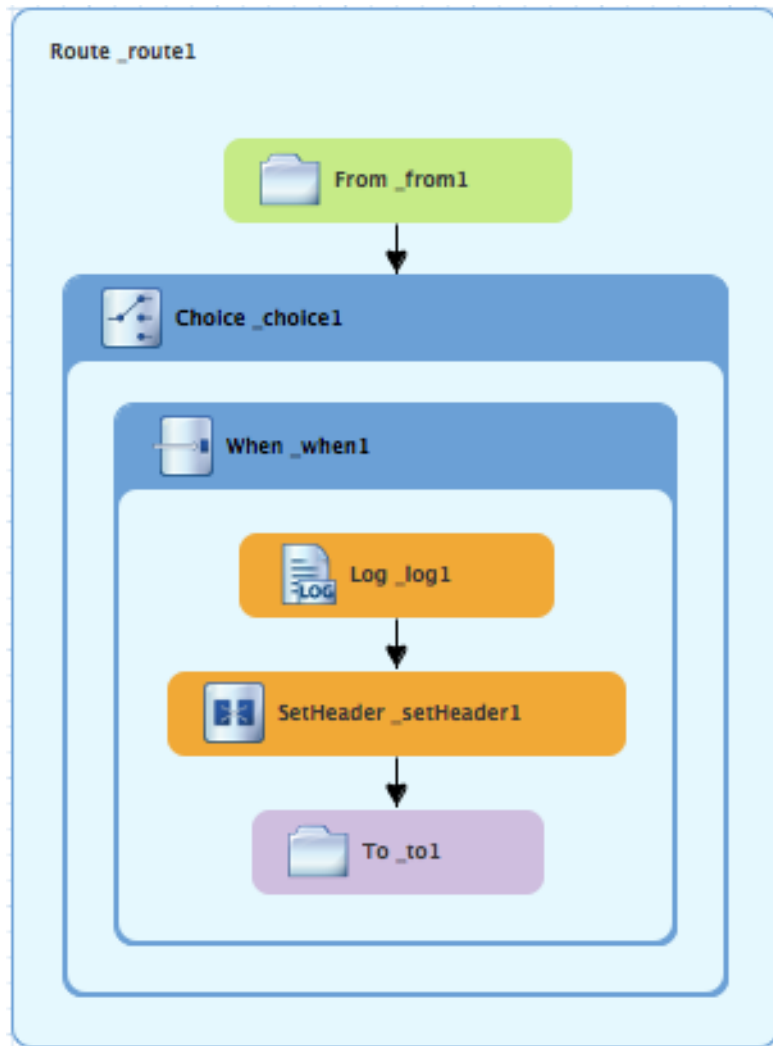
SetHeader_setHeader1	
General	
Language *	<input type="text"/>
Header Name *	<input type="text"/>
Description	<input type="text"/>
Id	<input type="text" value="_setHeader1"/>

5. Click the  button in the **Language** field to open the list of available languages, and select **constant**:



6. In the **Expression** field, enter **InvalidOrders**.
7. In the **Header Name** field, enter **Destination**.
8. Leave each of the remaining properties as is.
9. In the **Palette**, open the **Components** drawer and select the **File** (  ) component.
10. Drag the **File** component to the canvas and drop it in the **When\_when1** container.  
The **When\_when1** container expands to accommodate the **To\_to1** node.
11. On the canvas, select the **SetHeader\_setHeader1** node and drag its connector arrow over the **To\_to1** node, and then release it:





12. On the canvas, select the `To_to1` node to open its properties in the **Properties** view:

To_to1	
General	
Uri *	file:directoryName
Description	
Id	_to1
Pattern	
Ref (deprecated)	

13. On the **Details** tab, replace `directoryName` with `target/messages/invalidOrders` in the **Uri** field, and enter `_Invalid` in the **Id** field:

To_Invalid	
General	
Uri *	file:target/messages/invalidOrders
Description	
Id	_Invalid
Pattern	
Ref (deprecated)	

14. On the menu bar, click **File** → **Save** to save the routing context file.

15. Click the **Source** tab to view the XML for the route:


```

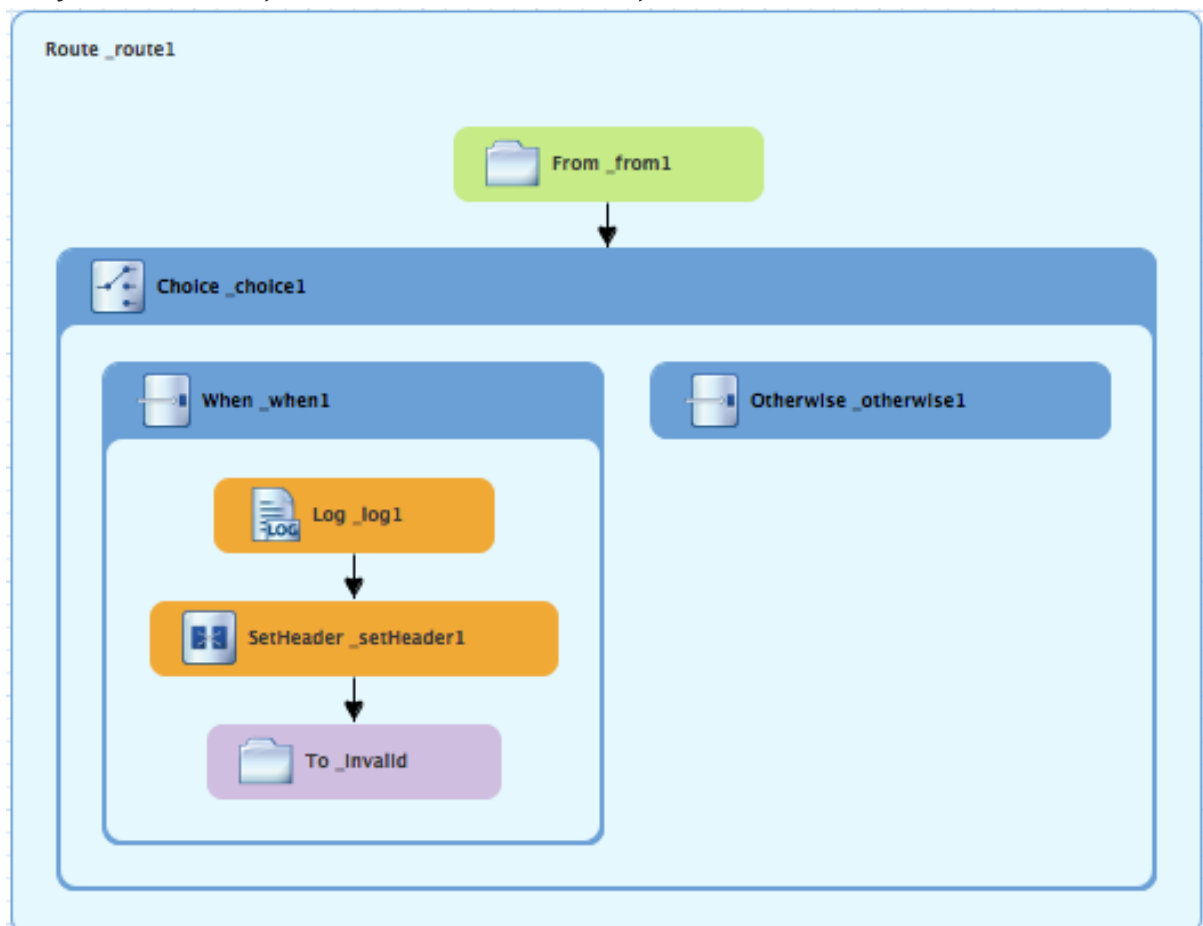
*blueprint.xml
25 <blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
26   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
27   xsi:schemaLocation="http://www.osgi.org/xmlns/blueprint/v1.0.0
28     https://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd
29     http://camel.apache.org/schema/blueprint
30     http://camel.apache.org/schema/blueprint/camel-blueprint.xsd">
31   The namespace for the camelContext element in Blueprint is 'https://camel.apache.org/schema/blueprint'.
32   <camelContext id="_context1" xmlns="http://camel.apache.org/schema/blueprint">
33     <route id="_route1">
34       <from id="_from1" uri="file:src/data?noop=true"/>
35       <choice id="_choice1">
36         <when id="_when1">
37           <xpath>order/orderline/quantity/text() > /order/orderline/maxAllowed/text()</xpath>
38           <log id="_log1" message="quantity requested exceeds the maximum allowed - contact customer"
39             <setHeader headerName="Destination" id="_setHeader1">
40               <constant>InvalidOrders</constant>
41             </setHeader>
42             <to id="_Invalid" uri="file:target/messages/invalidOrders"/>
43           </when>
44         </choice>
45       </route>
46     </camelContext>
47   </blueprint>
Design Source Configurations

```

## ADDING AND CONFIGURING AN OTHERWISE BRANCH

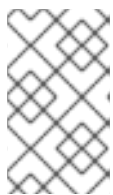
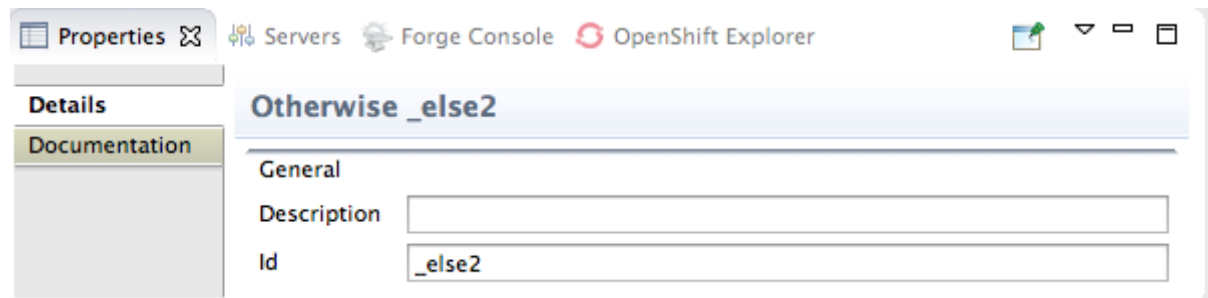
To add and configure the otherwise branch of your route:

1. In the **Palette**, open the **Routing** drawer and select the **Otherwise** (  ) pattern.
2. Drag the **Otherwise** pattern to the canvas and drop it into the **Choice\_choice1** container:




The `Choice_choice1` container expands to accommodate the `Otherwise_otherwise1` node.

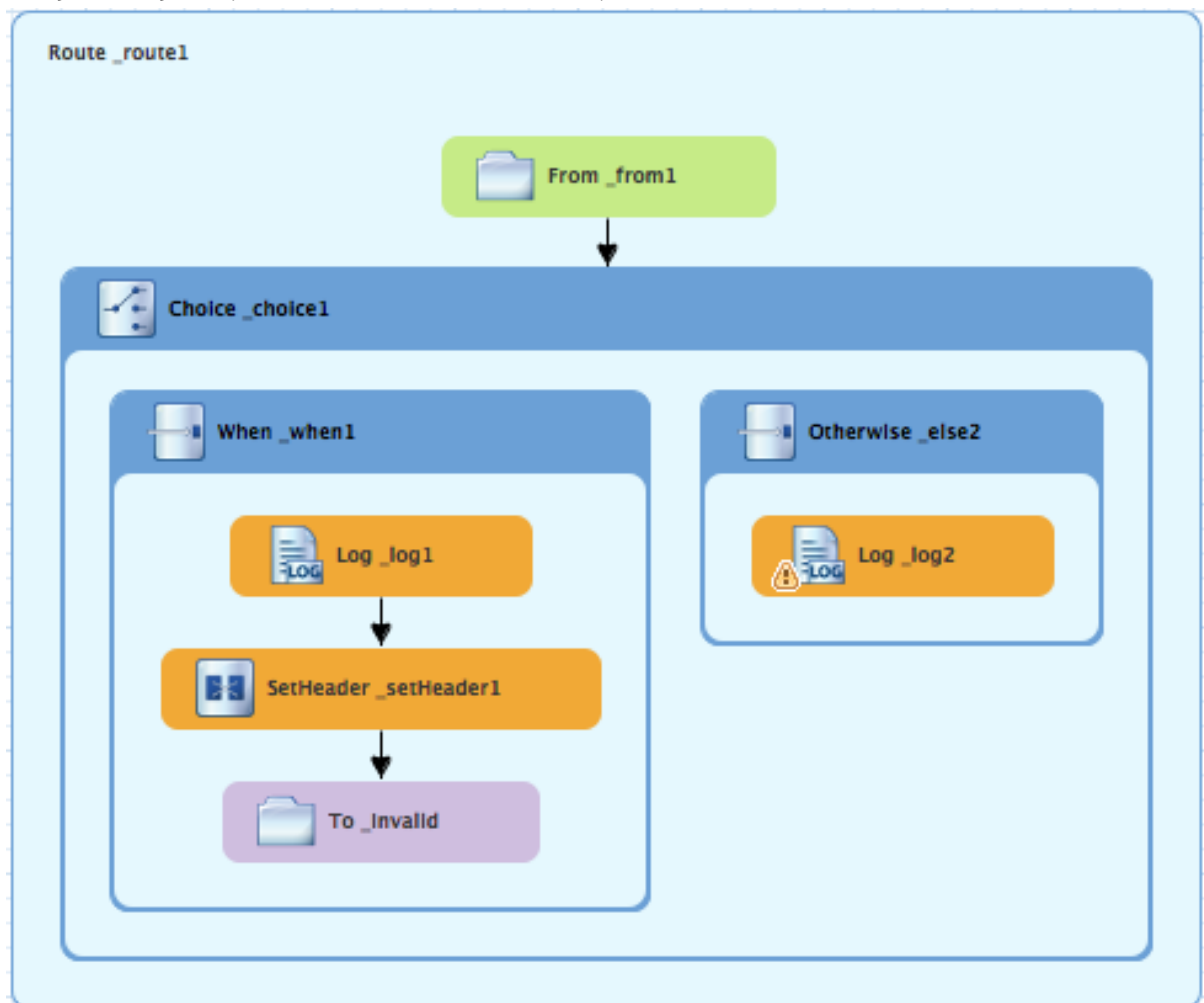
- On the canvas, select the `Otherwise_otherwise1` node to open its properties in the **Properties**.
- In the `Id` field, enter `_else2`:



#### NOTE

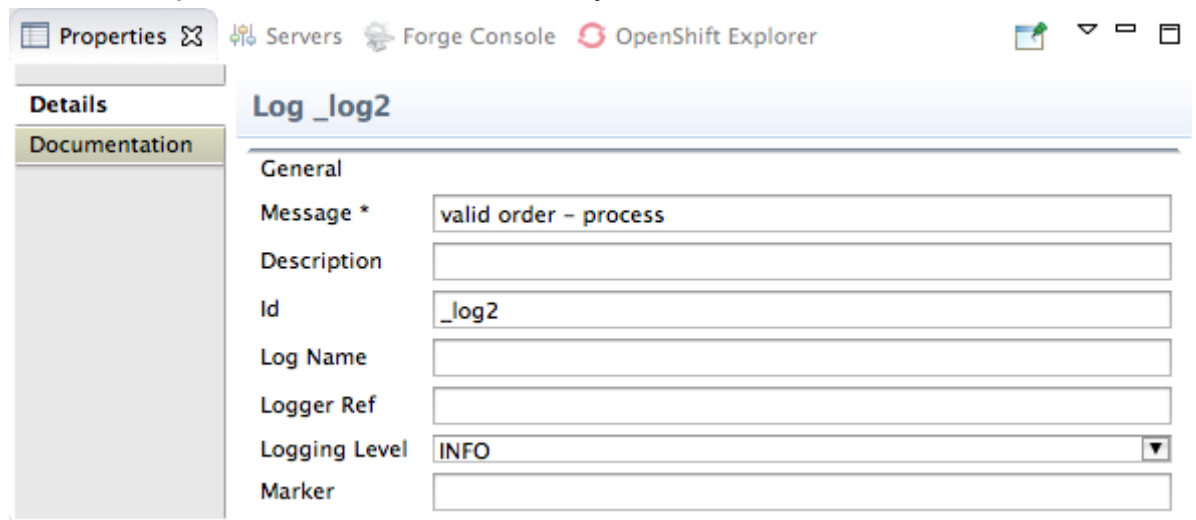
The `else2` node will eventually route to the terminal `file:` node (`file:target/messages/validOrders`) any message that does not match the XPath expression set for the `When_when1` node.

- In the **Palette**, open the **Components** drawer and select the **Log** (  ) component.
- Drag the **Log** component to the canvas and drop it on the `Otherwise_else2` node:



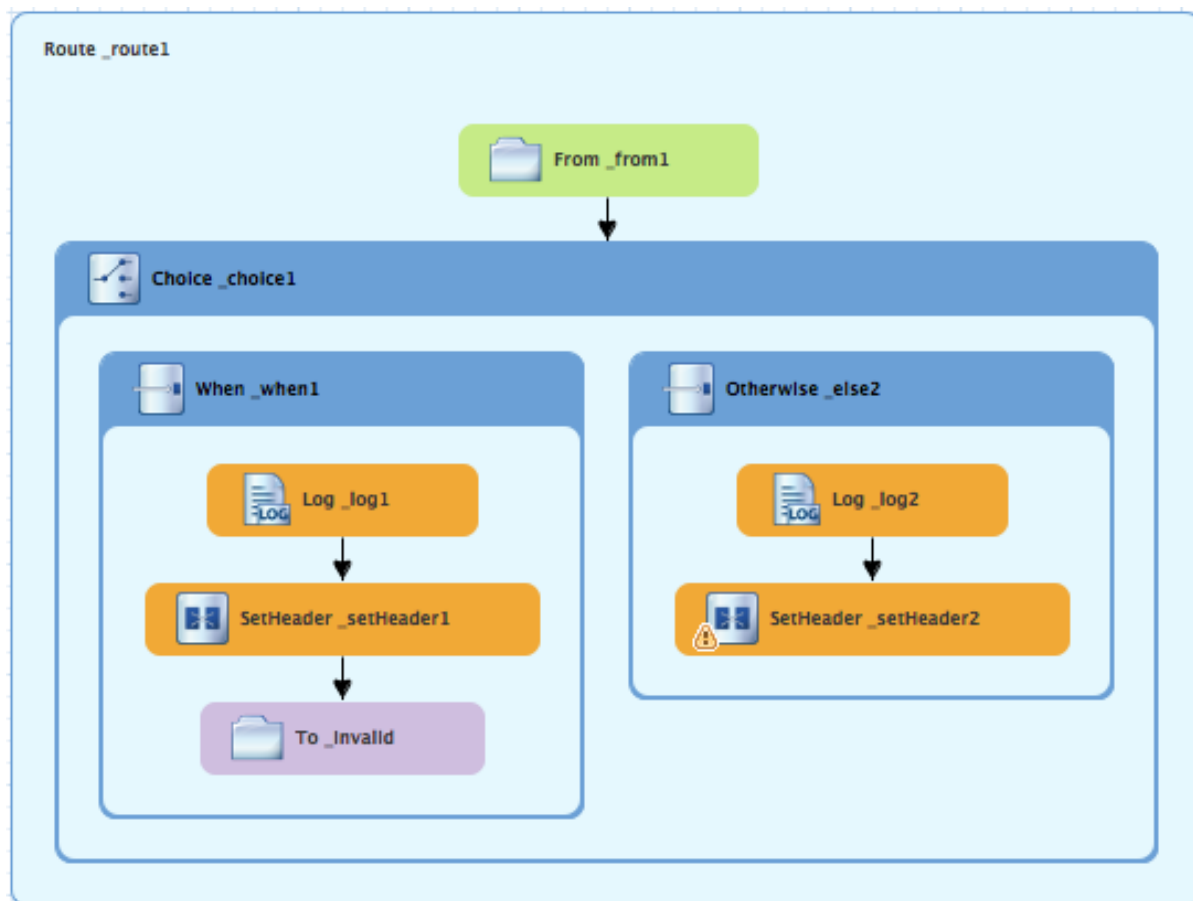
The **Otherwise\_else2** container expands to accommodate the **Log\_log2** node.

- On the canvas, select the **Log\_log2** node to open its properties in the **Properties** view.
- In the **Message** field, enter **valid order - process**:




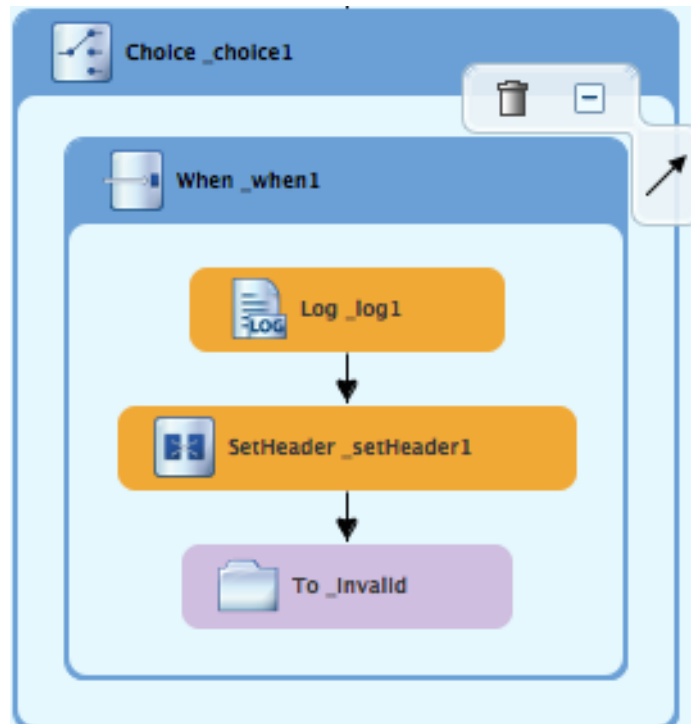
Leave each of the remaining properties as is.

- In the **Palette**, open the **Transformation** drawer and select the **Set Header** pattern.
- Drag the **Set Header** pattern to the canvas and drop it into the **Otherwise\_else2** container. The **Otherwise\_else2** container expands to accommodate the **SetHeader\_setHeader2** node.
- On the canvas, select the **Log\_log2** node and drag its connector arrow over the **SetHeader\_setHeader2** node, and then release it:

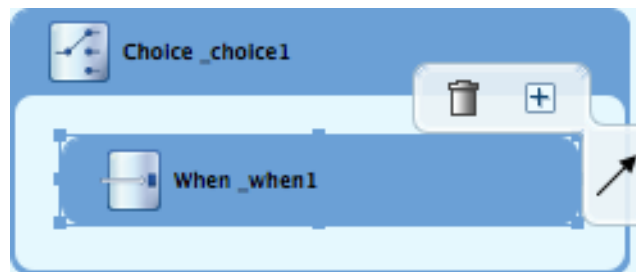


**NOTE**


You can collapse containers to free up space when the diagram becomes congested. To do so, select the container you want to collapse, and then click its  button:

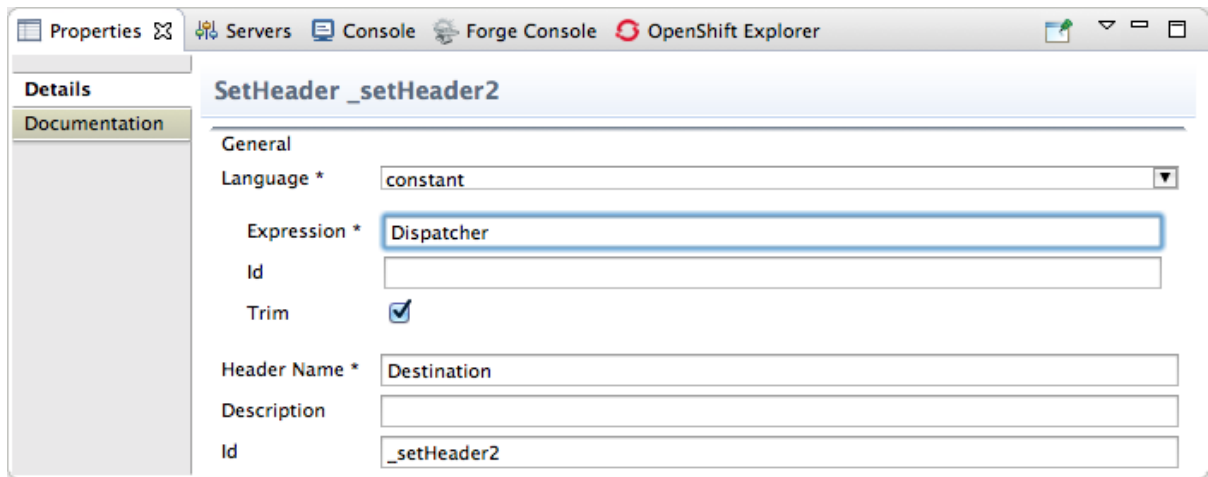


To reopen the container, select it and then click its  button:



Collapsing and expanding containers in the **Design** tab does not affect the routing context file. It remains unchanged.


12. On the canvas, select the `SetHeader_setHeader2` node to open its properties in the **Properties** view.
13. Click the  button in the **Language** field to open the list of available languages, and select **constant**:



14. In the **Expression** field, enter **Dispatcher**.

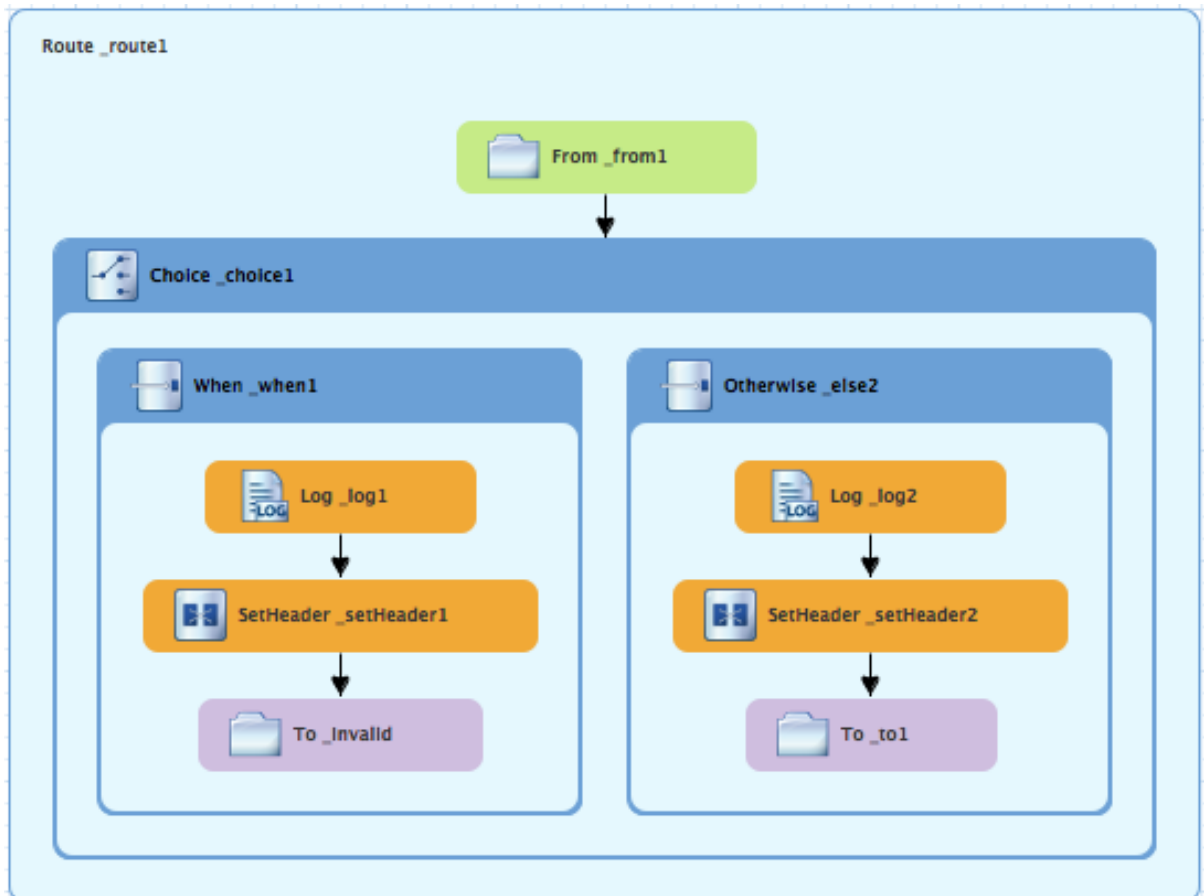
15. In the **Header Name** field, enter **Destination**.

16. Leave each of the remaining properties as is.

17. In the **Palette**, open the **Components** drawer and select the **File** (  ) component.

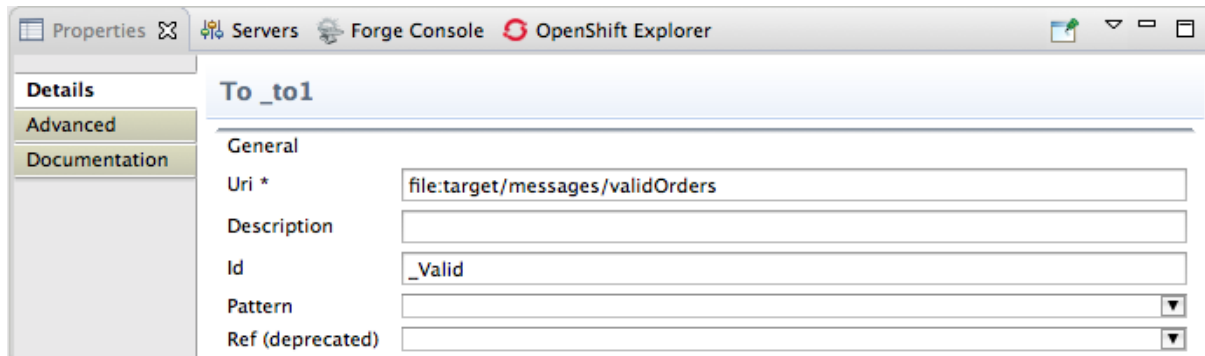
18. Drag the **File** component to the canvas and drop it into the **Otherwise\_else2** container. The **Otherwise\_else2** container expands to accommodate the **To\_to1** node.

19. On the canvas, select the **SetHeader\_setHeader2** node, and drag its connector arrow over the **To\_to1** node and then release it:

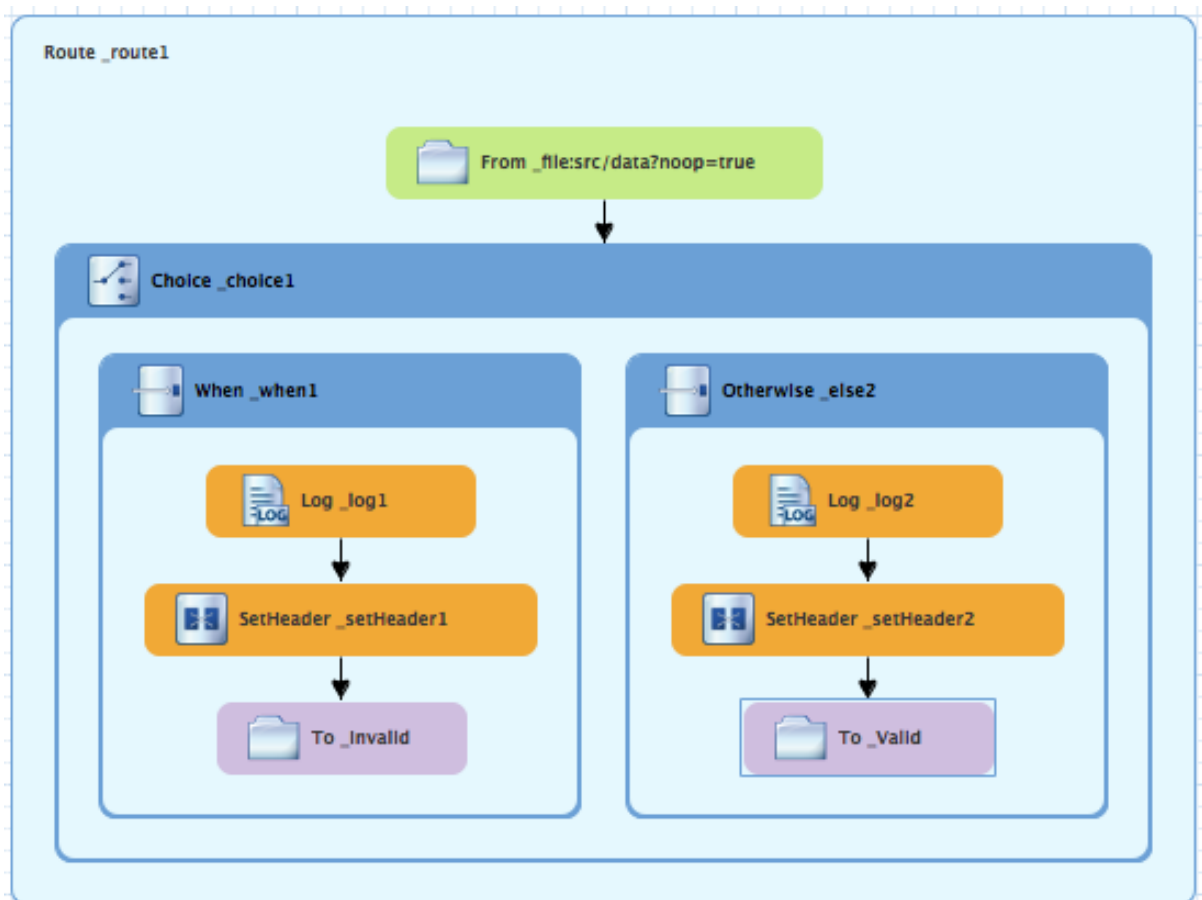


20. On the canvas, select the **To\_to1** node to open its properties in the **Properties** view.

21. In the **URI** field, replace *directoryName* with **target/messages/validOrders**, and in the **Id** field, enter **\_Valid**:



22. On the menu bar, click **File** → **Save** to save the routing context file.  
This is the completed content-based router with logs and message headers:



23. 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, “XML for content-based router”](#).

#### 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:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.osgi.org/xmlns/blueprint/v1.0.0
    https://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 id="_context1"
xmlns="http://camel.apache.org/schema/blueprint">
  <route id="_route1">
    <from id="_from1" uri="file:src/data?noop=true"/>
    <choice id="_choice1">
      <when id="_when1">
        <xpath>order/orderline/quantity/text() >
/ order/orderline/maxAllowed/text()</xpath>
        <log id="_log1" message="quantity requested exceeds
the maximum allowed - contact customer"/>
        <setHeader headerName="Destination"
id="_setHeader1">
          <constant>InvalidOrders</constant>
        </setHeader>
        <to id="_Invalid"
uri="file:target/messages/invalidOrders"/>
      </when>
      <otherwise id="_else2">
        <log id="_log2" message="valid order - process"/>
        <setHeader headerName="Destination"
id="_setHeader2">
          <constant>Dispatcher</constant>
        </setHeader>
        <to id="_Valid"
uri="file:target/messages/validOrders"/>
      </otherwise>
    </choice>
  </route>
</camelContext>
</blueprint>
```

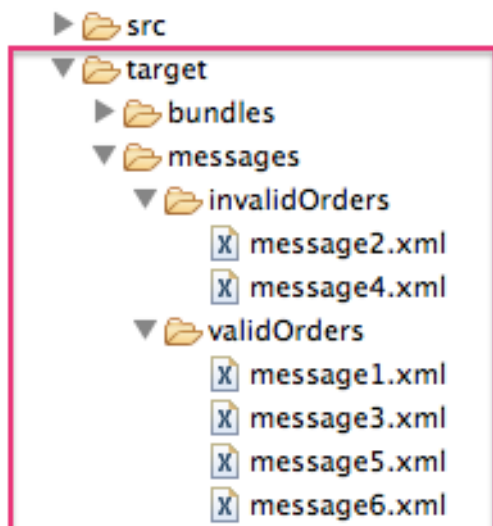
## NEXT STEPS

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**.
3. Under the project root node (**CBRroute**), locate the folder **target/messages/** and expand it.





4. 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.
5. 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:

- [Red Hat JBoss Fuse: Apache Camel Development Guide](#)
- [Red Hat JBoss Fuse 6.3 documentation](#)

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

This tutorial shows you how to add a second route to the `blueprint.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 **country** folder in the `CBRroute/target/messages` folder.

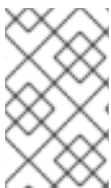
### GOALS

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 `blueprint5.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/blueprint.xml` in the route editor.
2. On the canvas, select the `Route_route1` container to open its properties in the **Properties** view.
3. Scroll down to the **Shutdown Route** property and enter `Default`.

4. On the canvas, select the terminal file node `To_Valid` to display its properties in the **Properties** view.
5. In the **Uri** field, delete the existing text, and then enter `direct:OrderFulfillment`.
6. In the **Id** field, enter `_Fulfill`.




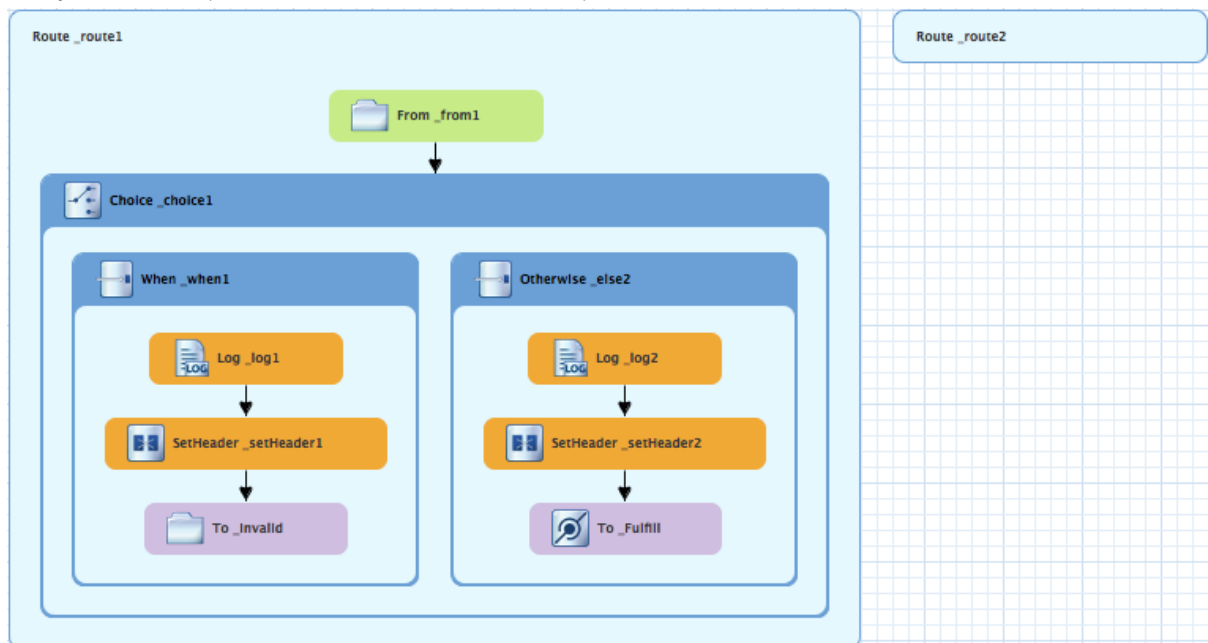
#### NOTE

Instead of repurposing the existing `To_Valid` terminal file node, you could have replaced it with a **Components** → **Direct** component, configuring it with the same property values as the repurposed `To_Valid` node.

## ADDING THE SECOND ROUTE

To add a route to the routing context:

1. In the **Palette**, open the **Routing** drawer and select the **Route** () pattern.
2. Drag the **Route** pattern to the canvas and drop it next to the `Route_route1` container:

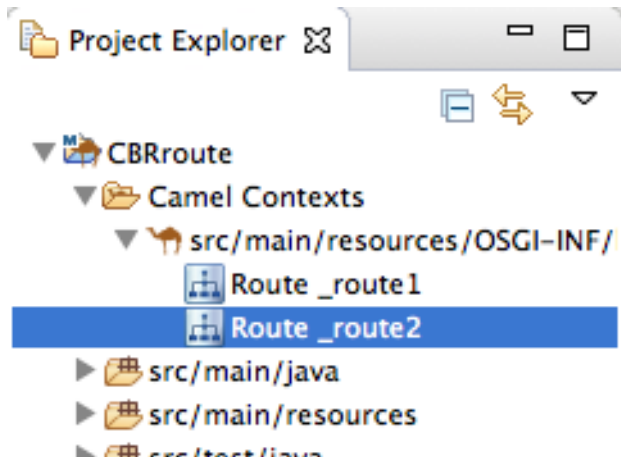


The **Route** pattern becomes the `Route_route2` container node on the canvas.

3. Click the `Route_route2` container node to display its properties in the **Properties** view.
4. Leave each of the properties as is.


**NOTE**

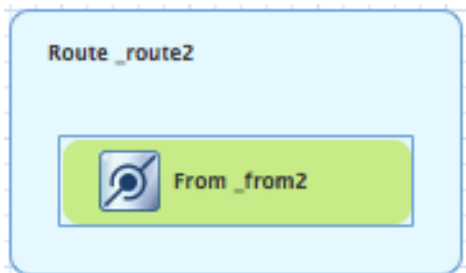
As your multiroute routing context grows in complexity, you may want to focus the route editor on an individual route while you work on it. To do so, in **Project Explorer**, double-click the route you want the route editor to display on the canvas; for example `Route_route2`:



To display all routes in the routing context on the canvas, double-click the project's `.xml` context file entry ( `src/main/resources/OSGI-INF/...` ) at the top of the **Camel Contexts** folder.

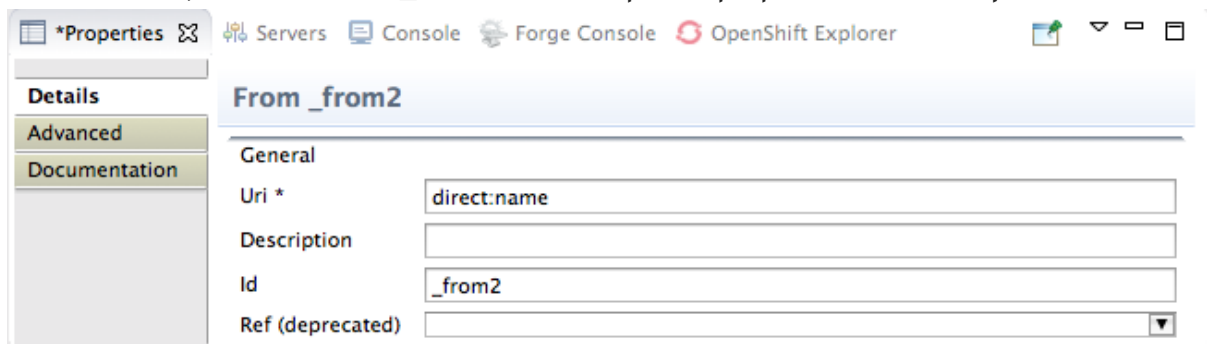
## BUILDING AND CONFIGURING THE USA BRANCH OF THE SECOND ROUTE

1. In the **Palette**, open the **Components** drawer and drag a **Direct component** (  ) to the canvas and drop it in the `Route_route2` container:




The `Route_route2` container expands to accommodate the `From_from2` node.

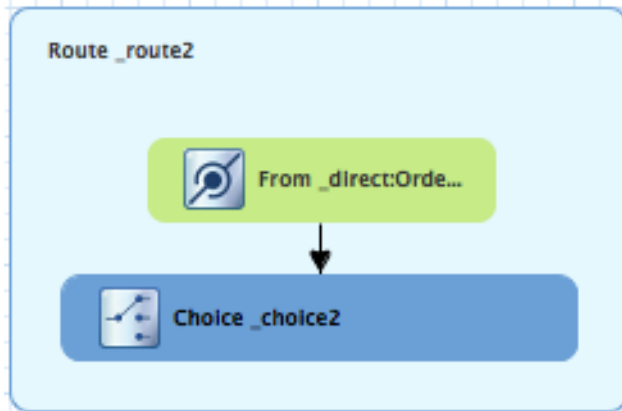
2. On the canvas, select the `From_from2` node to open its properties in the **Properties** view:




3. In the **Uri** field, replace `name` (following `direct:`) with `OrderFulfillment`, and in the **Id** field, enter `_direct:OrderFulfillment`.

- In the **Palette**, open the **Routing** drawer and drag a **Choice** (  ) pattern to the canvas and drop it in the **Route\_route2** container.  
The **Route\_route2** container expands to accommodate the **Choice\_choice2** node.

- In the **Route\_route2** container, select the **direct:OrderFulfillment** node and drag its connector arrow over the **Choice\_choice2** node, then release it:



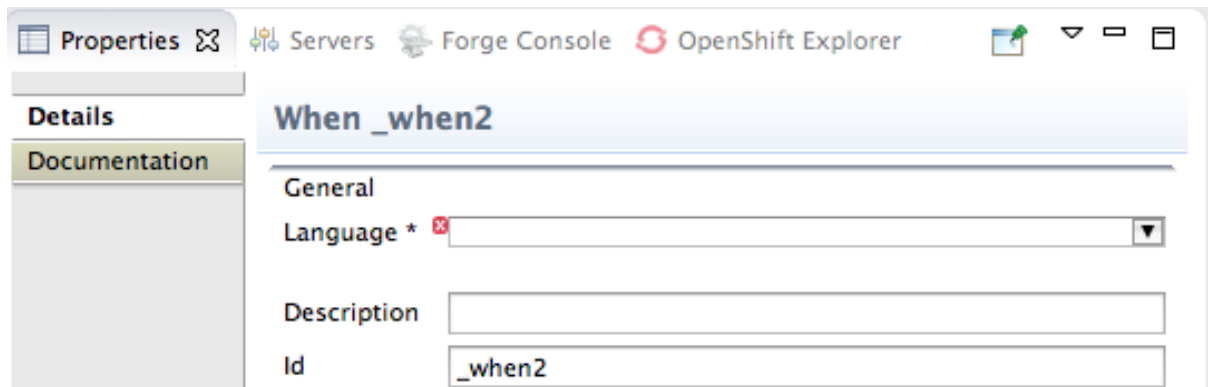
- In the **Properties** view, leave each of the **Choice\_choice2** node's properties as is.

- In the **Palette**, open the **Routing** drawer and drag a **When** (  ) pattern to the canvas and drop it in the **Choice\_choice2** container:

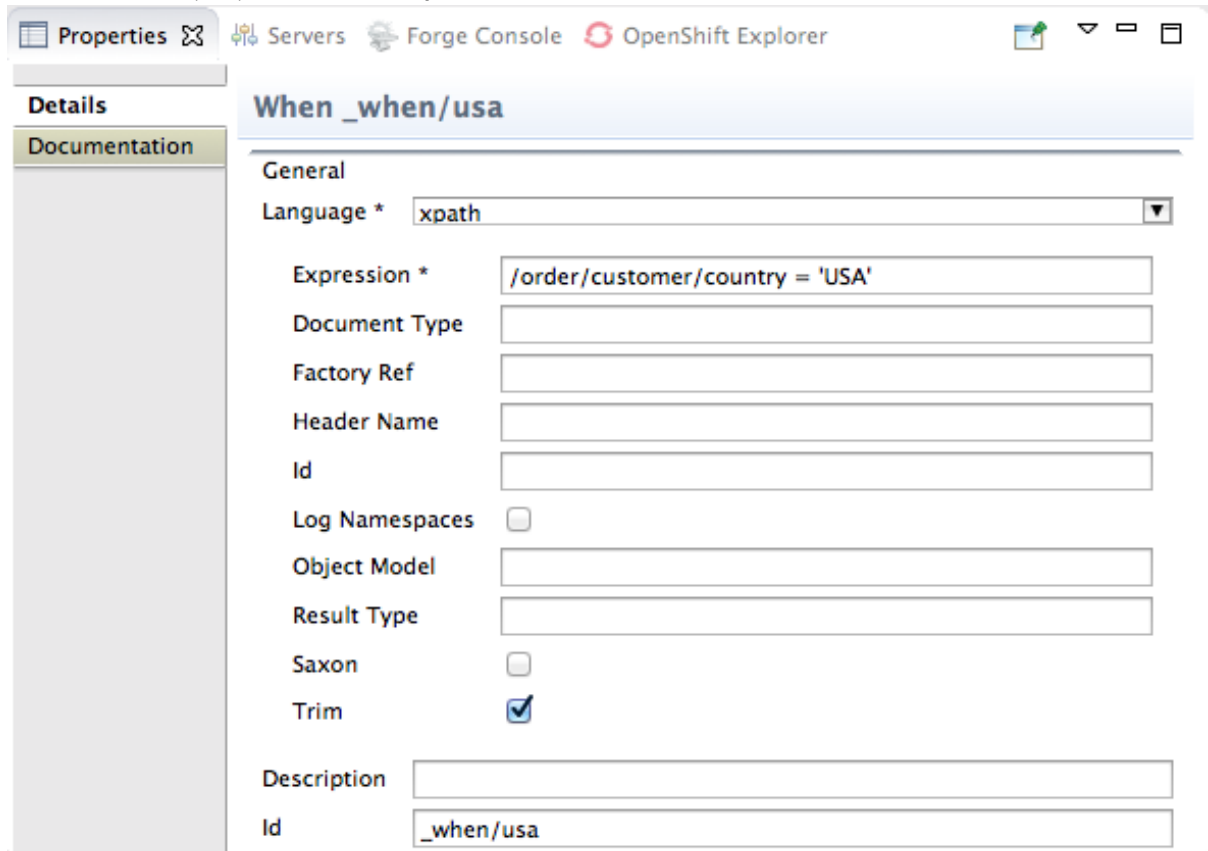


The **Choice\_choice2** container expands to accommodate the **When\_when2** node.

- On the canvas, select the **When\_when2** node to open its properties in the **Properties** view:



9. Set the node's properties this way:



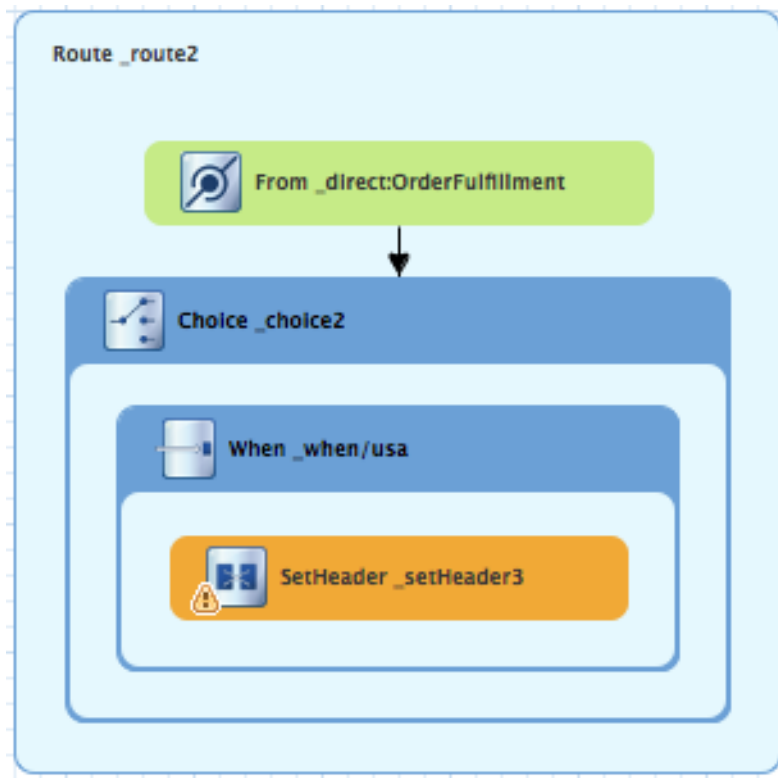
- Select `xpath` from the **Language** drop-down menu.
- Enter `/order/customer/country = 'USA'` in the **Expression** field.
- Leave **Trim** enabled.
- Enter `_when/usa` in the **Id** field.



#### NOTE

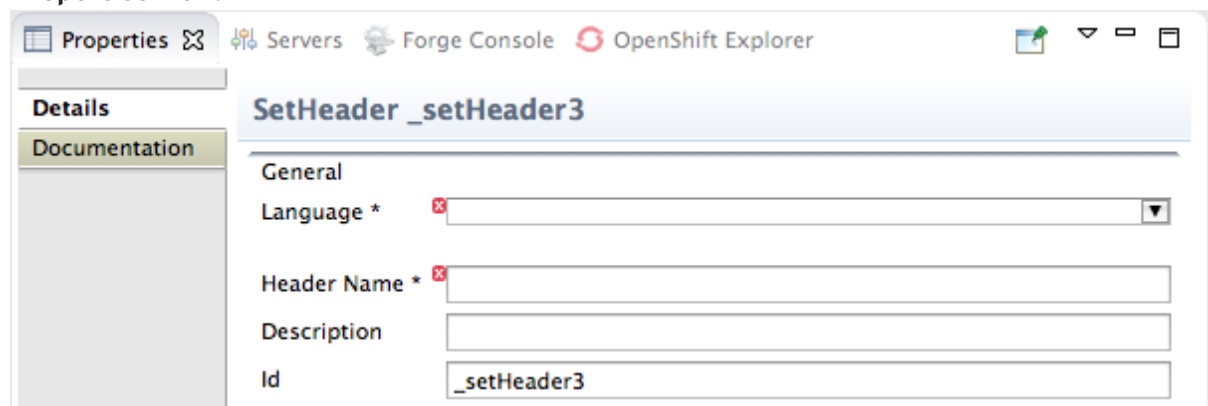
Once you select the expression **Language**, the **Properties** view displays its properties in an indented list directly below the **Language** field. The **Id** property in this list sets the ID of the expression. The **Id** property following the **Description** field sets the ID of the **When** node.

10. In the **Palette**, open the **Transformation** drawer and drag the **Set Header** pattern to the canvas and drop on the `When_when/usa` node:

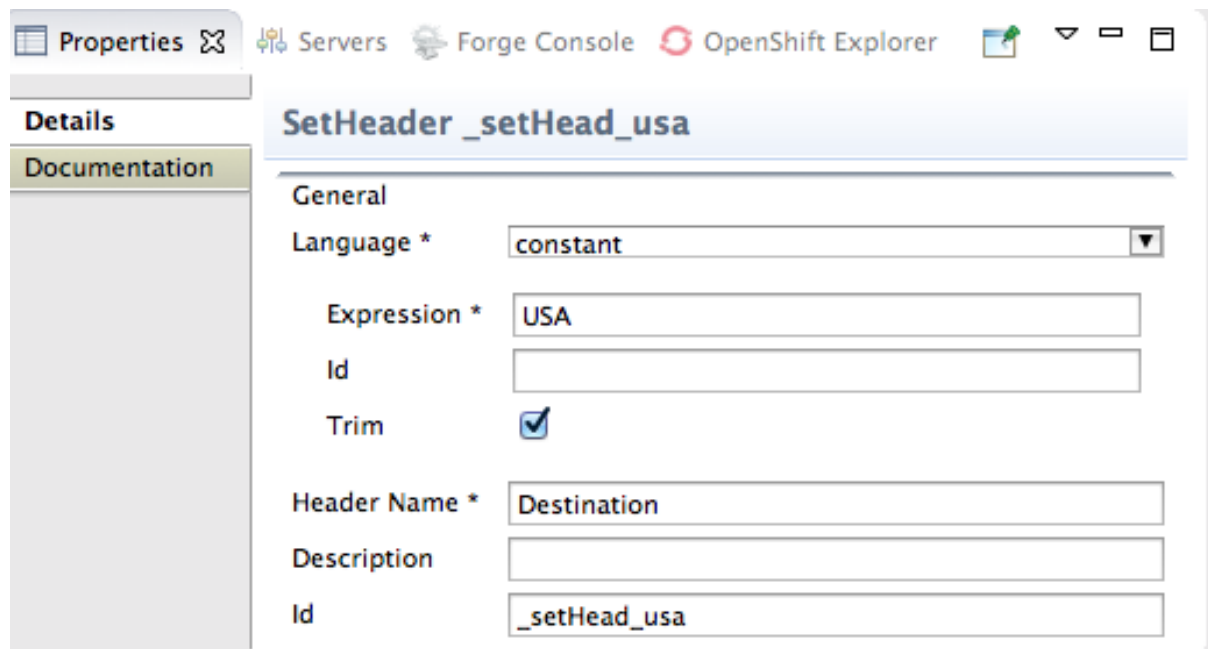



The `When _when/usa` container expands to accommodate the `SetHeader _setHeader3` node.

11. On the canvas, select the `SetHeader _setHeader3` node to open its properties in the **Properties** view:

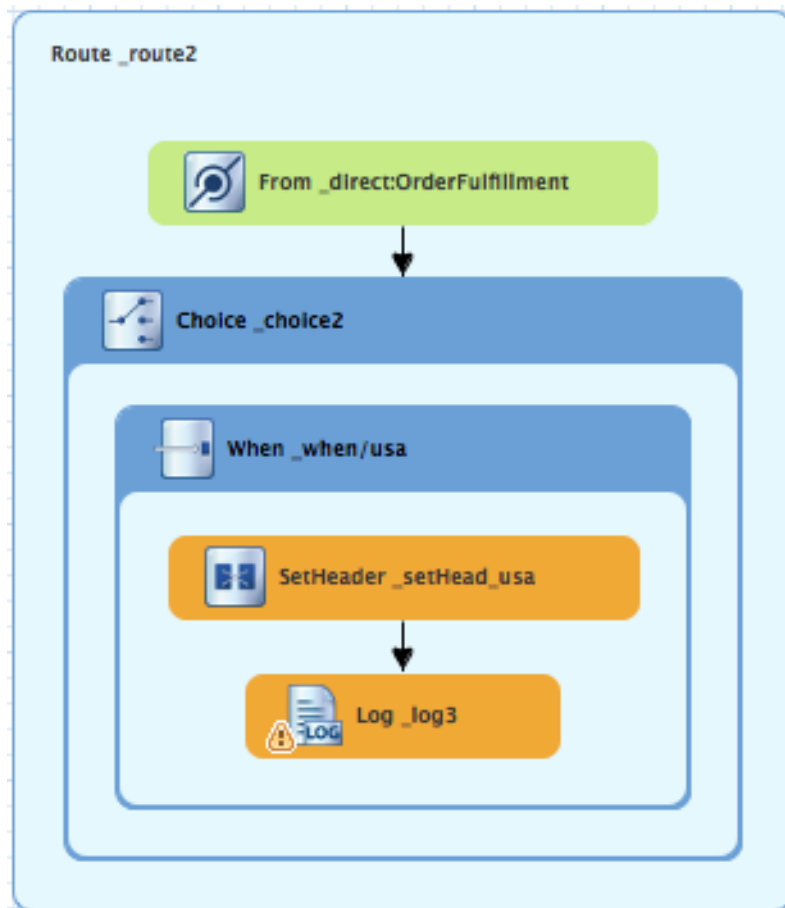


12. Set the node's properties this way:



- Select **constant** from the Language drop-down menu.
  - Enter **USA** in the **Expression** field.
  - Leave **Trim** enabled.
  - Enter **Destination** in the **Header Name** field.
  - Enter **\_setHead\_usa** in the **Id** field.
13. In the **Palette**, open the **Components** drawer and drag a **Log** component (  ) to the canvas and drop it in the **When\_when/usa** container.  
The **When\_when/usa** container expands to accommodate the **Log\_log3** node.
14. On the canvas, select the **SetHeader\_setHead\_usa** node and drag its connector arrow over the **Log\_log3** node, then release it:





15. On the canvas, select the **Log\_log3** node to open its properties in the **Properties** view:

Properties Servers Forge Console OpenShift Explorer

**Log\_log3**

General

Message \*

Description

Id

Log Name

Logger Ref

Logging Level

Marker

16. In the **Properties** view:

\*Properties Servers Console Forge Console OpenShift Explorer

**Log\_usa**

General

Message \*

Description

Id


Log Name

Logger Ref

Logging Level

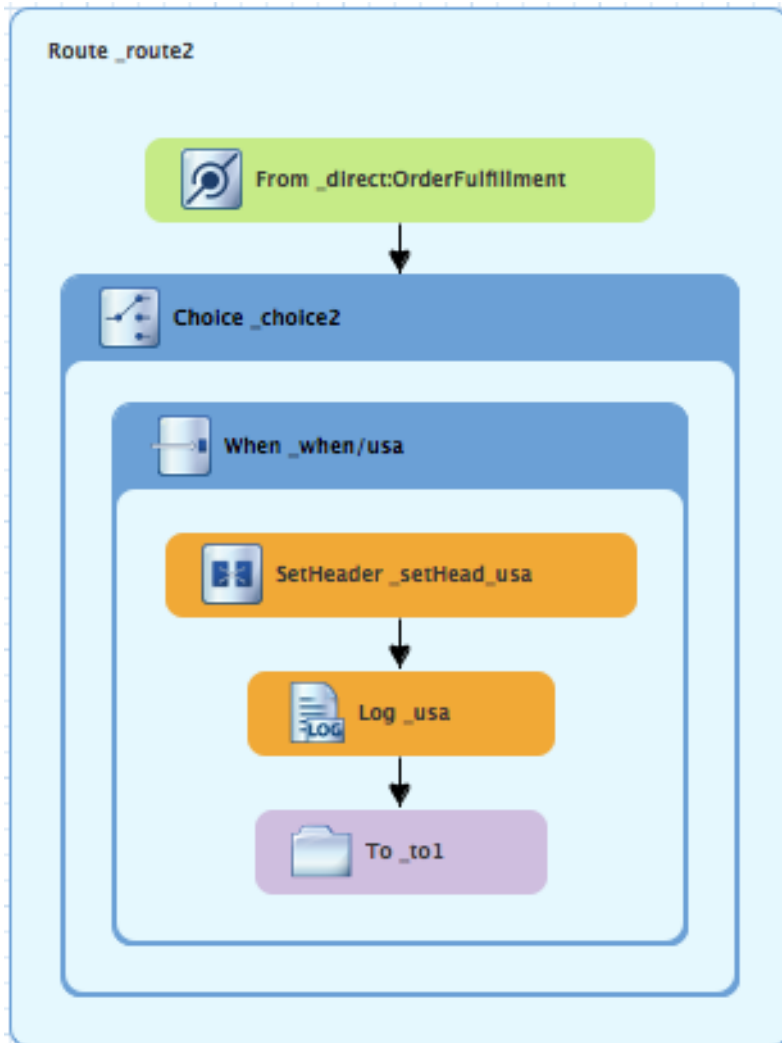
Marker

- Enter **Valid order - ship animals to USA** customer in the **Message** field.
- Enter **\_usa** in the **Id** field.
- Leave **Logging Level** as is.

17. In the **Palette**, open the **Components** drawer and drag a **File** component (  ) to the canvas and drop it in the **When\_when/usa** container.

The **When\_when/usa** container expands to accommodate the **To\_to1** node.

18. On the canvas, select the **Log\_usa** node and drag its connector arrow over the **To\_to1** node, then release it:



19. In the **Properties** view:

Properties Servers Forge Console OpenShift Explorer

**Details**

**To\_US**

General

Uri \*

Description

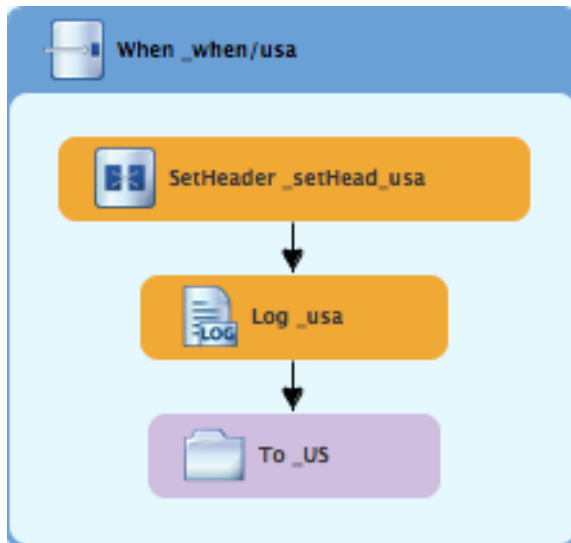
Id

Pattern

Ref (deprecated)


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

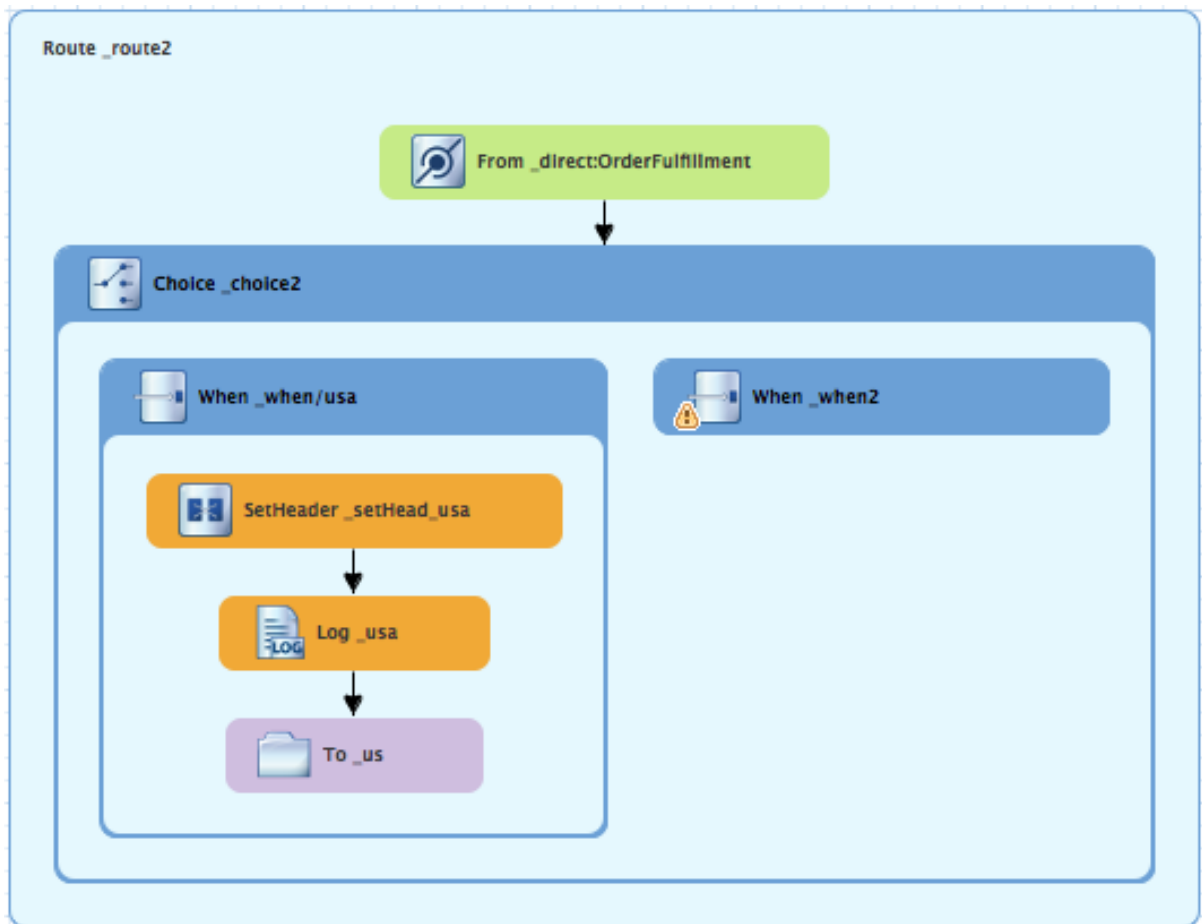
20. On the menu bar, click **File** → **Save** to save the routing context file.  
The USA branch of `Route_route2` should look like this:




## BUILDING AND CONFIGURING THE GREAT BRITAIN BRANCH OF THE SECOND ROUTE

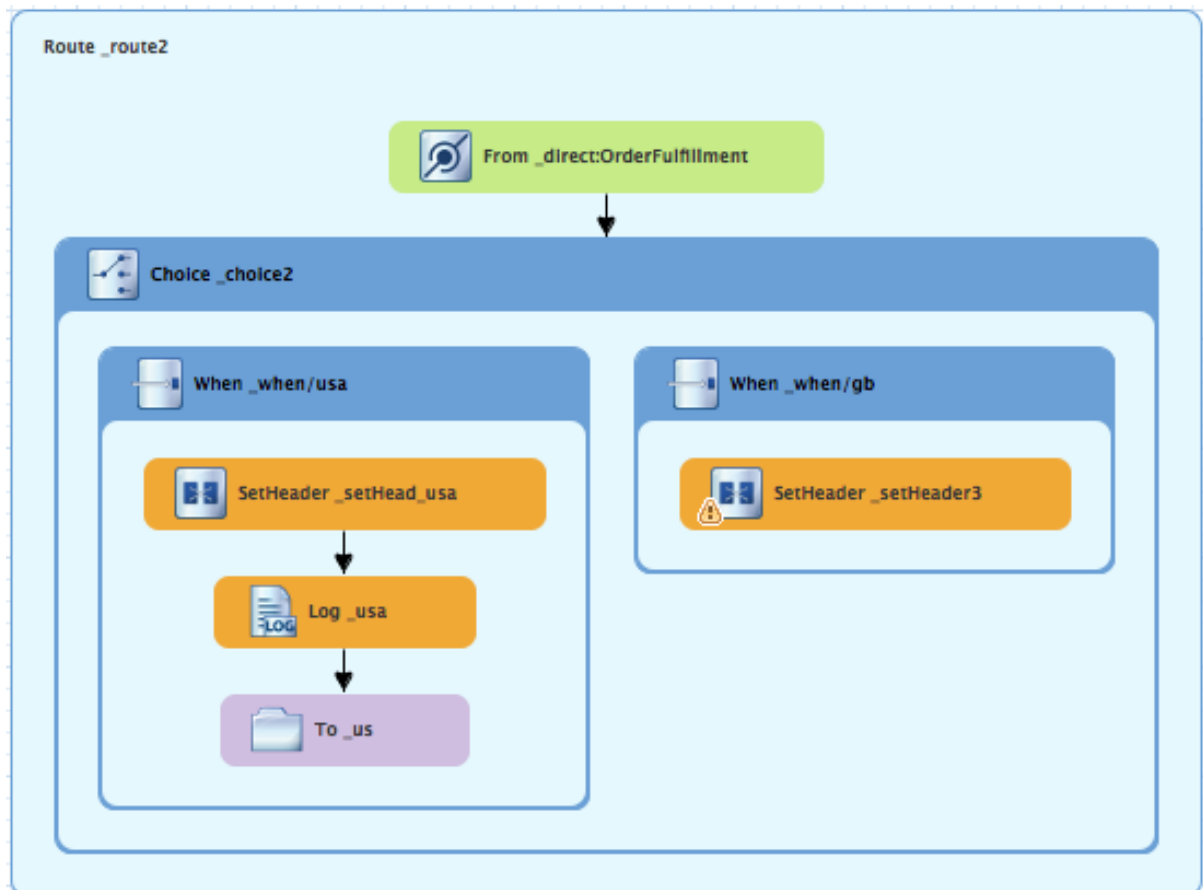
With `Route_route2` displayed on the canvas:

1. In the **Palette**, open the **Routing** drawer and drag a **When** pattern (  ) to the canvas and drop it in the `Choice_choice2` container:




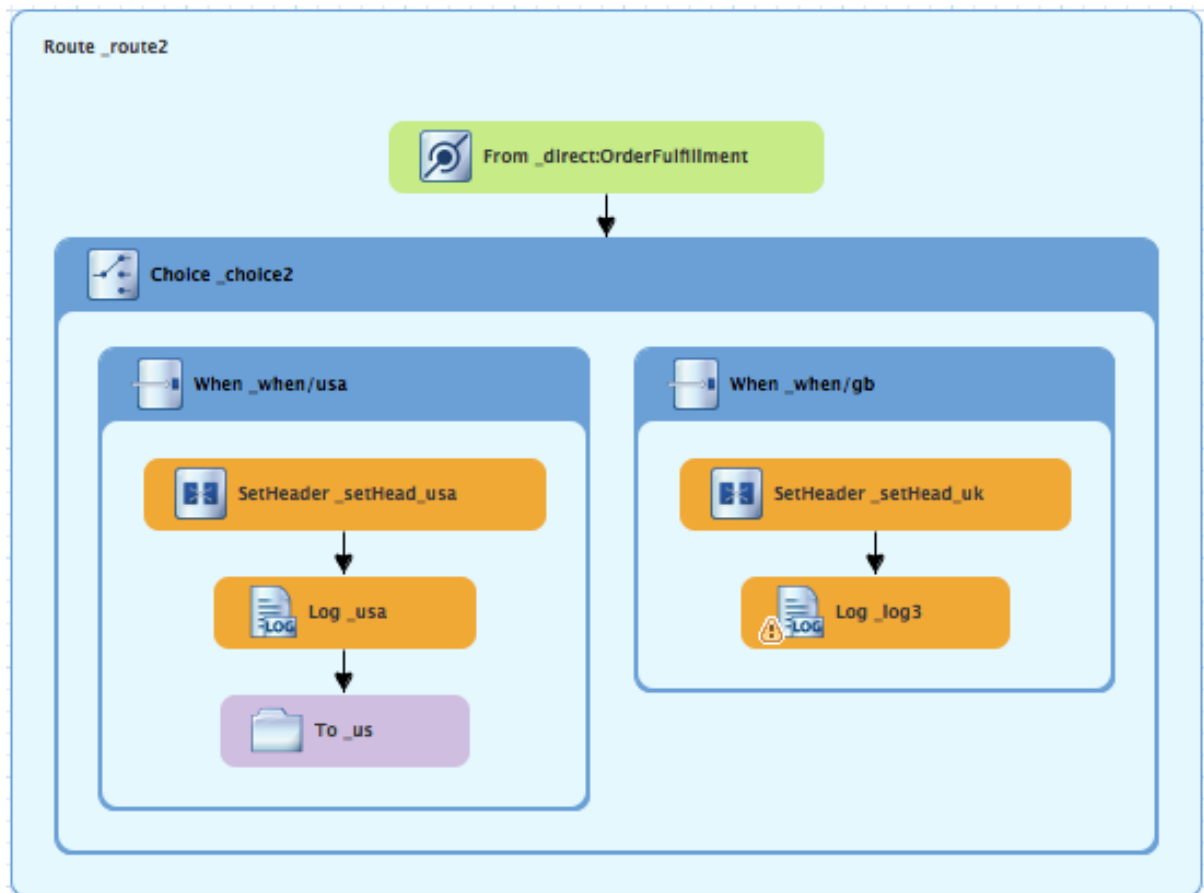
The **Choice\_choice2** container expands to accommodate the **When\_when2** node.

2. On the canvas, select the **When\_when2** node to open its properties in the **Properties** view.
3. In the **Properties** view:
  - Select **xpath** from the **Language** drop-down menu.
  - Enter `/order/customer/country = 'Great Britain'` in the **Expression** field.
  - Leave **Trim** enabled.
  - Enter `_when/gb` in the **Id** field.
4. In the **Palette**, open the **Transformation** drawer and drag a **Set Header** pattern (  ) to the canvas and drop it on the **When\_when/gb** node:



The **When\_when/gb** container expands to accommodate the **SetHeader\_setHeader3** node.

5. On the canvas, select the **SetHeader\_setHeader3** node to open its properties in the **Properties** view.
6. In the **Properties** view:
  - Select **constant** from the **Language** drop-down menu.
  - Enter **UK** in the **Expression** field.
  - Leave **Trim** as is.
  - Enter **Destination** in the **Header Name** field.
  - Enter **\_setHead\_uk** in the **Id** field.
7. In the **Palette**, open the **Components** drawer and drag a **Log** pattern (  ) to the canvas and drop it in the **When\_when/gb** container.  
The **When\_when/gb** container expands to accommodate the **Log\_log3** node.
8. On the canvas, select the **SetHeader\_setHead\_uk** node and drag its connector arrow over the **Log\_log3** node, and then release it:



9. On the canvas, select the **Log\_log3** node to open its properties in the **Properties** view.

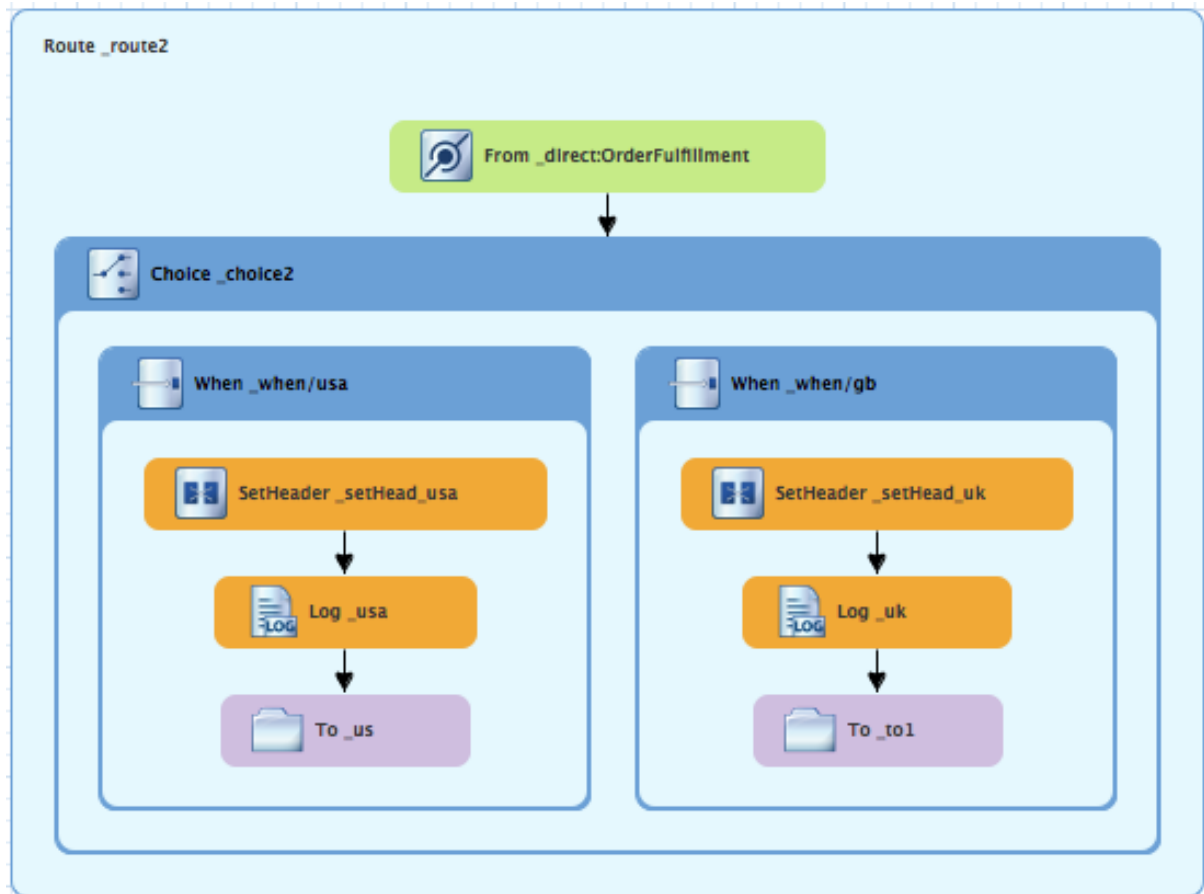
10. In the **Properties** view:

- Enter **Valid order - ship animals to UK customer** in the **Message** field.
- Enter **\_uk** in the **Id** field.
- Leave the **Logging Level** as is.

11. From the **Components** drawer, drag a **File pattern** (  ) to the canvas and drop it in the **When\_when/gb** container.

The **When\_when/gb** container expands to accommodate the **To\_to1** node.

12. On the canvas, select the **Log\_uk** node and drag its connector arrow over the **To\_to1** node, and then release it:



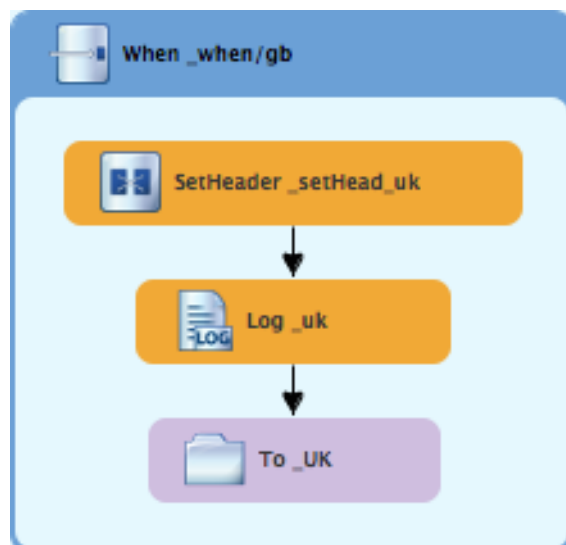
13. On the canvas, select the `To_to1` node to open its properties in the **Properties** view.

14. In the **Properties** view:

- Replace `directoryName` with `target/messages/GreatBritain` in the **Uri** field.
- Enter `_UK` in the **Id** field.


15. On the menu bar, click **File** → **Save** to save the routing context file.

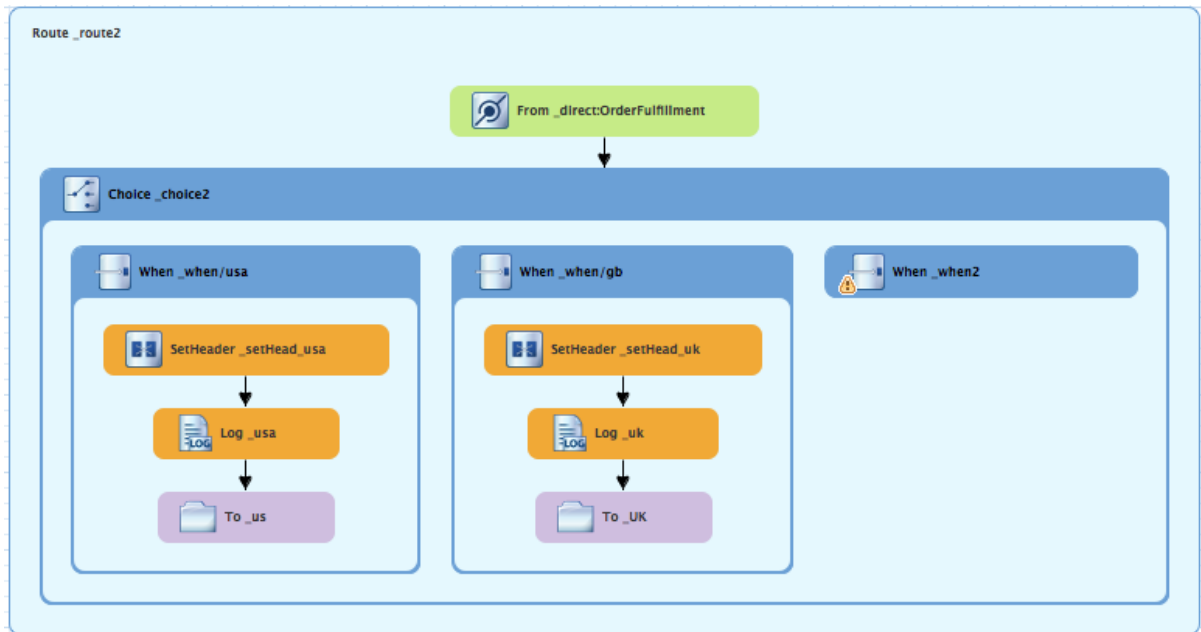
The Great Britain branch of `Route_route2` should look like this:




## BUILDING AND CONFIGURING THE GERMANY BRANCH OF THE SECOND ROUTE

With `Route_route2` displayed on the canvas:

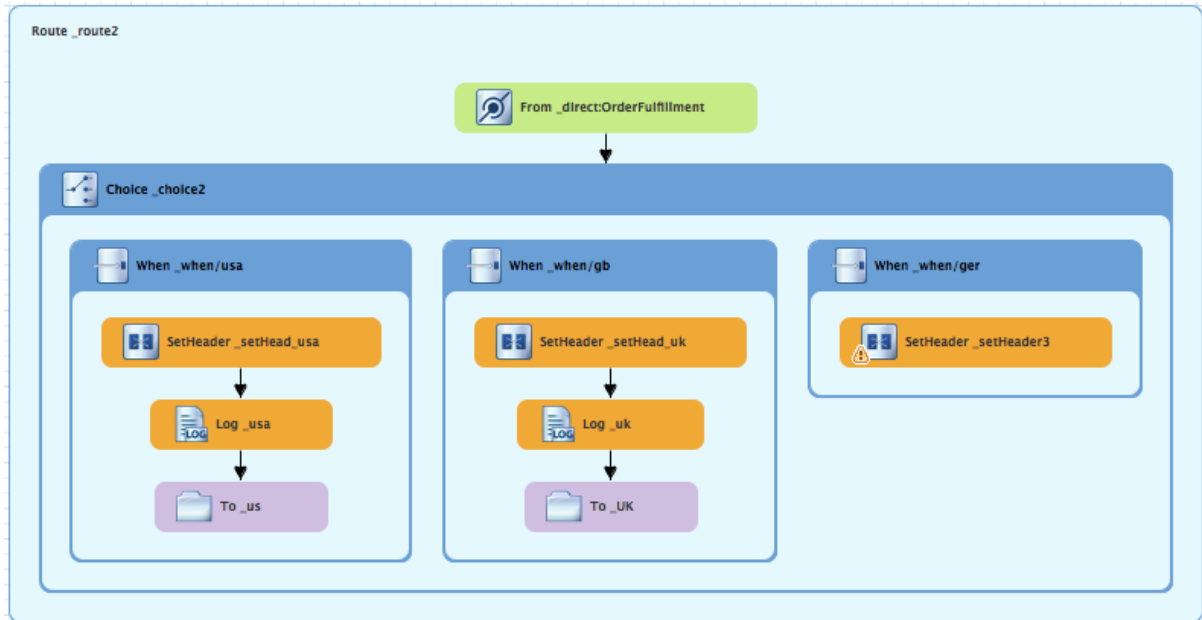
1. In the **Palette**, open the **Routing** drawer and drag a **When** pattern (  ) to the canvas and drop it in the `Choice_choice2` container:




The `Choice_choice2` container expands to accommodate the `When_when2` node.

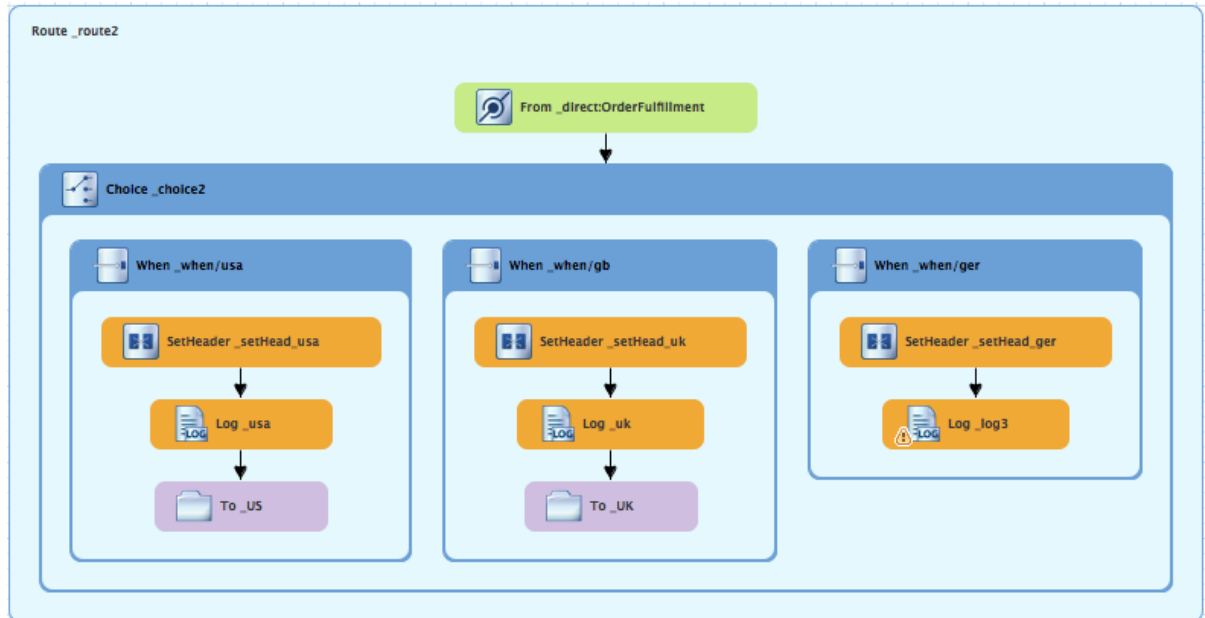
2. On the canvas, select the `When_when2` node to open its properties in the **Properties** view.
3. In the **Properties** view:
  - Select `xpath` from the **Language** drop-down menu.
  - Enter `/order/customer/country = 'Germany'` in the **Expression** field.
  - Leave **Trim** enabled.
  - Enter `_when/ger` in the **Id** field.
4. In the **Palette**, open the **Transformation** drawer and drag a **Set Header** pattern (  ) to the canvas and drop it on the `When_when/ger` node:





The **When\_when/ger** container expands to accommodate the **SetHeader\_setHeader3** node.

5. On the canvas, select the **SetHeader\_setHeader3** node to open its properties in the **Properties** view.
6. In the **Properties** view:
  - Select **constant** from the **Language** drop-down menu.
  - Enter **Germany** in the **Expression** field.
  - Leave **Trim** as is.
  - Enter **Destination** in the **Header Name** field.
  - Enter **\_setHead\_ger** in the **Id** field.
7. In the **Palette**, open the **Components** drawer and drag a **Log** pattern (  ) to the canvas and drop it in the **When\_when/ger** container.  
The **When\_when/ger** container expands to accommodate the **Log\_log3** node.
8. On the canvas, select the **SetHeader\_setHead\_ger** node and drag its connector arrow over the **Log\_log3** node, and then release it:



9. On the canvas, select the **Log\_log3** node to open its properties in the **Properties** view.

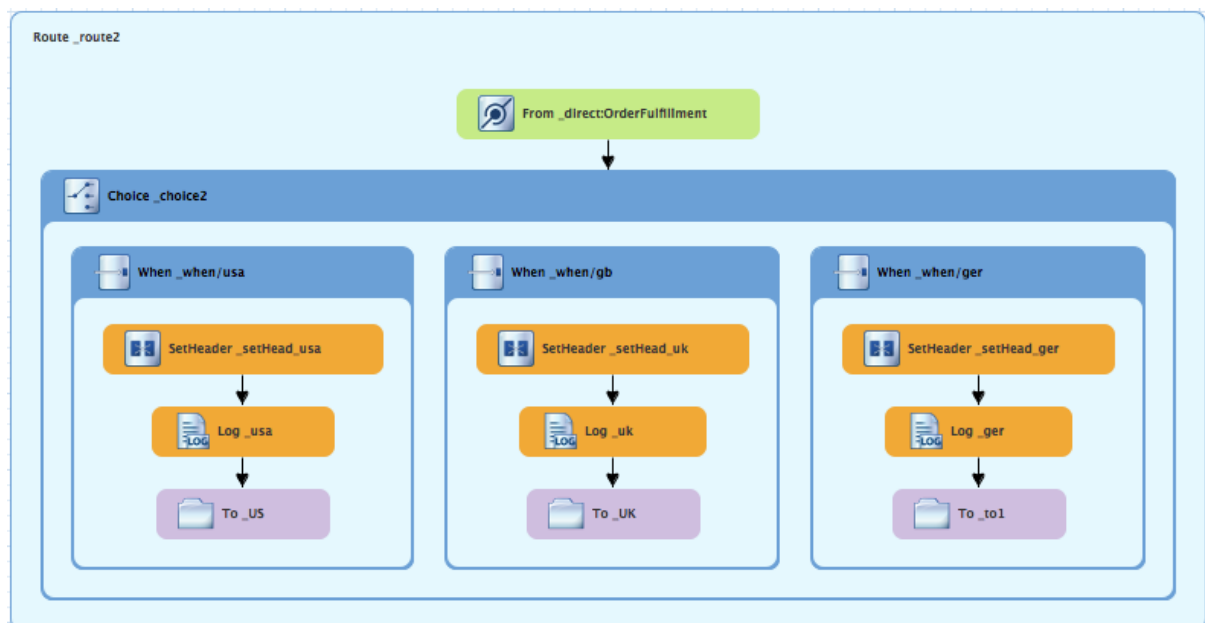
10. In the **Properties** view:

- Enter **Valid order - ship animals to Germany customer** in the **Message** field.
- Enter **\_ger** in the **Id** field.
- Leave the **Logging Level** as is.

11. From the **Components** drawer, drag a **File** pattern (  ) to the canvas and drop it in the **When\_when/ger** container.

The **When\_when/ger** container expands to accommodate the **To\_to1** node.

12. On the canvas, select the **Log\_ger** node and drag its connector arrow over the **To\_to1** node, and then release it:



13. On the canvas, select the **To\_to1** node to open its properties in the **Properties** view.

14. In the **Properties** view:


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

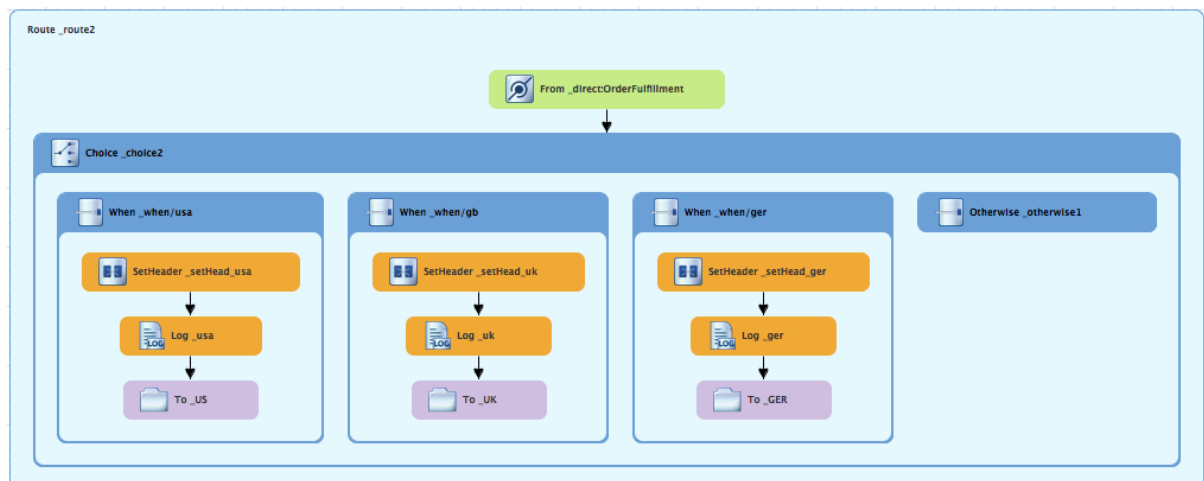
15. On the menu bar, click **File** → **Save** to save the routing context file.  
The Germany branch of **Route\_route2** should look like this:



## BUILDING AND CONFIGURING THE FRANCE BRANCH OF THE SECOND ROUTE


With **Route\_route2** displayed on the canvas:

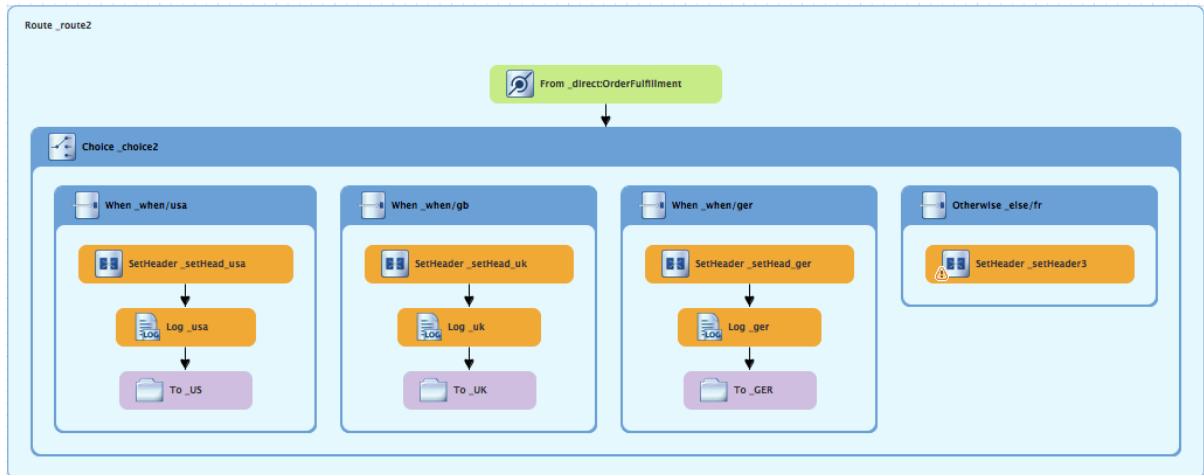
1. In the **Palette**, open the **\*Routing** drawer and drag an **Otherwise** pattern (  ) to the canvas and drop it in the **Choice\_choice2** container:



The **Choice\_choice2** container expands to accommodate the **Otherwise\_otherwise1** node.

2. On the canvas, select the **Otherwise\_otherwise1** node to open its properties in the **Properties** view.
3. In the **Properties** view, enter **\_else/fr** in the **Id** field.

4. In the **Palette**, open the **Transformation** drawer and drag a **Set Header** pattern (  ) to the canvas and drop it on the **Otherwise\_else/fr** node:




The **Otherwise\_else/fr** container expands to accommodate the **SetHeader\_setHeader3** node.

5. On the canvas, select the **SetHeader\_setHeader3** node to open its properties in the **Properties** view.

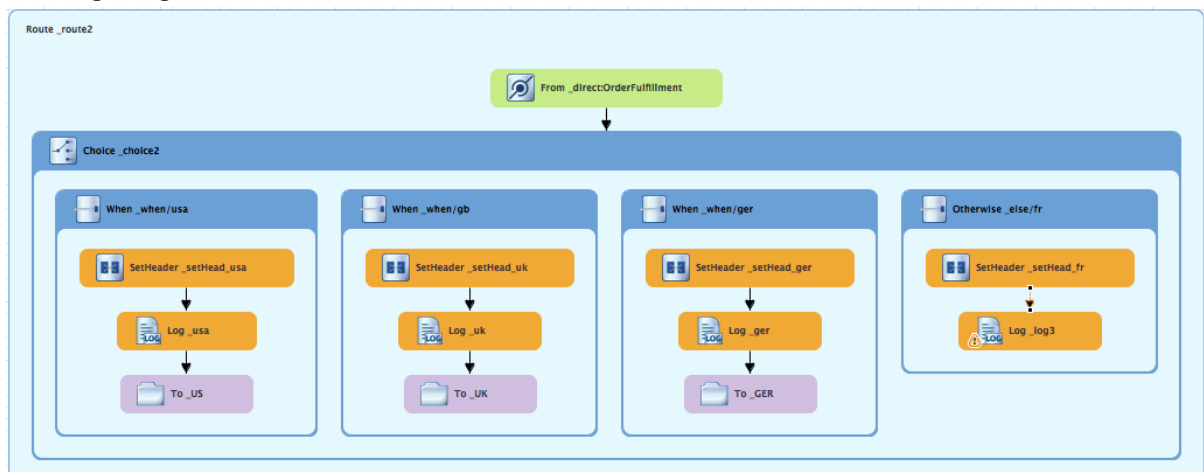
6. In the **Properties** view:

- Select **constant** from the **Language** drop-down menu.
- Enter **France** in the **Expression** field.
- Leave **Trim** as is.
- Enter **Destination** in the **Header Name** field.
- Enter **\_setHead\_fr** in the **Id** field.

7. In the **Palette**, open the **Components** drawer and drag a **Log** pattern (  ) to the canvas and drop it in the **Otherwise\_else/fr** container.

The **Otherwise\_else/fr** container expands to accommodate the **Log\_log3** node.

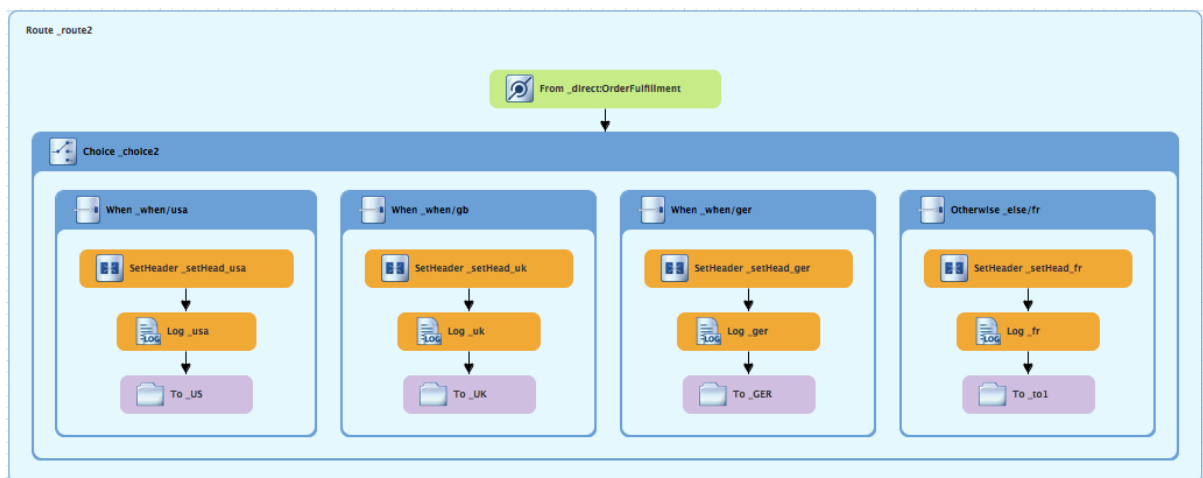
8. On the canvas, select the **SetHeader\_setHead\_fr** node and drag its connector arrow over the **Log\_log3** node, and then release it:



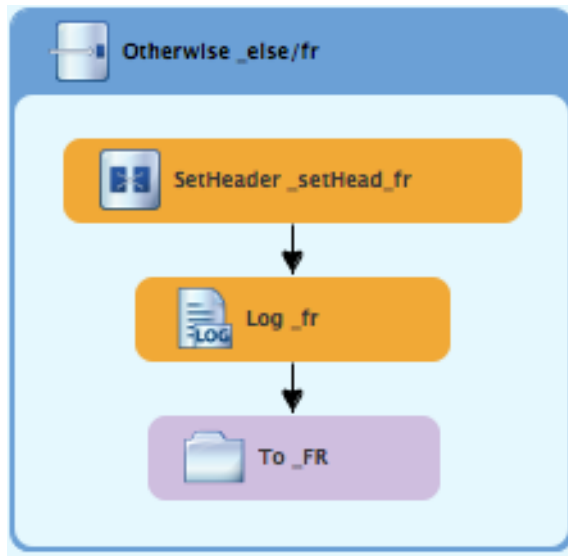
9. On the canvas, select the **Log\_log3** node to open its properties in the **Properties** view.
10. In the **Properties** view:
  - Enter **Valid order - ship animals to France customer** in the **Message** field.
  - Enter **\_fr** in the **Id** field.
  - Leave the **Logging Level** as is.

11. From the **Components** drawer, drag a **File pattern** (  ) to the canvas and drop it in the **Otherwise\_else/fr** container.  
The **Otherwise\_else/fr** container expands to accommodate the **To\_to1** node.

12. On the canvas, select the **Log\_fr** node and drag its connector arrow over the **To\_to1** node, and then release it:



13. On the canvas, select the **To\_to1** node to open its properties in the **Properties** view.
14. In the **Properties** view:
  - Replace *directoryName* with **target/messages/France** in the **Uri** field.
  - Enter **\_FR** in the **Id** field.
15. On the menu bar, click **File** → **Save** to save the routing context file.  
The France branch of **Route\_route2** should look like this:



## FINISHING UP

1. If needed, on the menu bar, select **File** → **Save** to save the routing context.  
The routes on the canvas should look like this:

image:///images/tutCBRRte1Completed.png[Completed first route in the CBRroute routing context]

image:///images/tutCBRRte2Completed.png[Completed second route in the CBRroute routing context]

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, “XML for dual-route content-based router”](#):

### 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:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.osgi.org/xmlns/blueprint/v1.0.0
    https://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 id="_context1"
    xmlns="http://camel.apache.org/schema/blueprint">
    <route id="_route1" shutdownRoute="Default">
      <from id="_from1" uri="file:src/data?noop=true"/>
      <choice id="_choice1">
        <when id="_when1">
          <xpath>/order/orderline/quantity/text() >
            /order/orderline/maxAllowed/text()</xpath>
          <log id="_log1" message="quantity exceeds the
            maximum allowed - contact customer"/>
          <setHeader headerName="Destination"
            id="_setHeader1">
            <constant>InvalidOrders</constant>
```

```

        </setHeader>
        <to id="_Invalid"
uri="file:target/messages/invalidOrders"/>
        </when>
        <otherwise id="_else2">
        <log id="_log2" message="valid order -
process"/>
        <setHeader headerName="Destination"
id="_setHeader2">
            <constant>Dispatcher</constant>
        </setHeader>
        <to id="_Fulfill"
uri="direct:OrderFulfillment"/>
        </otherwise>
    </choice>
</route>
<route id="_route2">
    <from id="_direct:OrderFulfillment"
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 id="_usa" message="Valid order - ship to
USA customer"/>
            <to id="_US" uri="file:target/messages/USA"/>
            </when>
            <when id="_when/gb">
                <xpath>/order/customer/country = 'Great
Britain'</xpath>
                <setHeader headerName="Destination"
id="_setHead_uk">
                    <constant>UK</constant>
                </setHeader>
                <log id="_uk" message="Valid order - ship
animals to UK customer"/>
                <to id="_UK"
uri="file:target/messages/GreatBritain"/>
                </when>
                <when id="_when/ger">
                    <xpath>/order/customer/country =
'Germany'</xpath>
                    <setHeader headerName="Destination"
id="_setHead_ger">
                        <constant>Germany</constant>
                    </setHeader>
                    <log id="_ger" message="Valid order - ship to
Germany customer"/>
                    <to id="_GER"
uri="file:target/messages/Germany"/>
                    </when>
                    <otherwise id="_else/fr">
                        <setHeader headerName="Destination"

```

```

id="_setHead_fr">
    <constant>France</constant>
    </setHeader>
    <log id="_fr" message="Valid order - ship
animals to France customer"/>
    <to id="_FR"
uri="file:target/messages/France"/>
    </otherwise>
    </choice>
</route>
</camelContext>
</blueprint>

```



## IMPORTANT

If the tooling added the attribute `shutdownRoute=""` to the second route element (`<route id="route2">`), delete that attribute. Otherwise, the CBRroute project might fail to run.

## 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:

```

[      Blueprint Extender: 3] BlueprintCamelContext      INFO Total 2 routes, of which 2 are started.
[      Blueprint Extender: 3] BlueprintCamelContext      INFO Apache Camel 2.17.0.redhat-630187 (CamelContext: _context1) started in 0.813
[1] thread #2 - file://src/data] Route1                 INFO valid order - process
[1] thread #2 - file://src/data] Route2                 INFO Valid order - ship animals to USA customer
[1] thread #2 - file://src/data] Route1                 INFO quantity requested exceeds maximum allowed - contact customer
[1] thread #2 - file://src/data] Route2                 INFO valid order - process
[1] thread #2 - file://src/data] Route1                 INFO Valid order - ship animals to UK customer
[1] thread #2 - file://src/data] Route2                 INFO quantity requested exceeds maximum allowed - contact customer
[1] thread #2 - file://src/data] Route1                 INFO valid order - process
[1] thread #2 - file://src/data] Route2                 INFO Valid order - ship animals to France customer
[1] thread #2 - file://src/data] Route1                 INFO valid order - process
[1] thread #2 - file://src/data] Route2                 INFO Valid order - ship animals to Germany customer

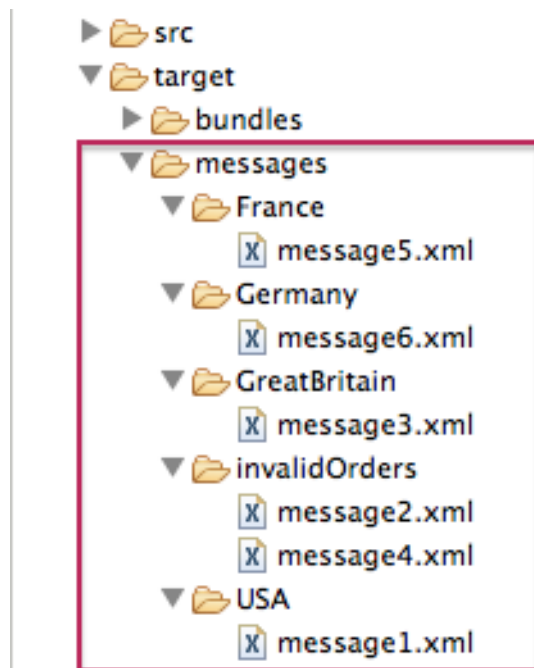
```

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 5.1, “Target message destinations in Project Explorer”](#). The `message*.xml` files should be dispersed in your target destinations like this:



Figure 5.1. Target message destinations in Project Explorer

**NOTE**

To view message content, double-click a message to open it in the route view'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 6.3 documentation](#)

## CHAPTER 6. TO DEBUG A ROUTING CONTEXT

This tutorial shows how to use the Camel debugger for only a locally running routing context. The routing context and each node with a breakpoint set must have a unique ID. The tooling automatically assigns a unique ID to the camelContext element and to components and patterns dropped on the canvas, but you can change these IDs to customize your project.

### GOALS

In this tutorial you will:

- In the **Design** tab, 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](#).




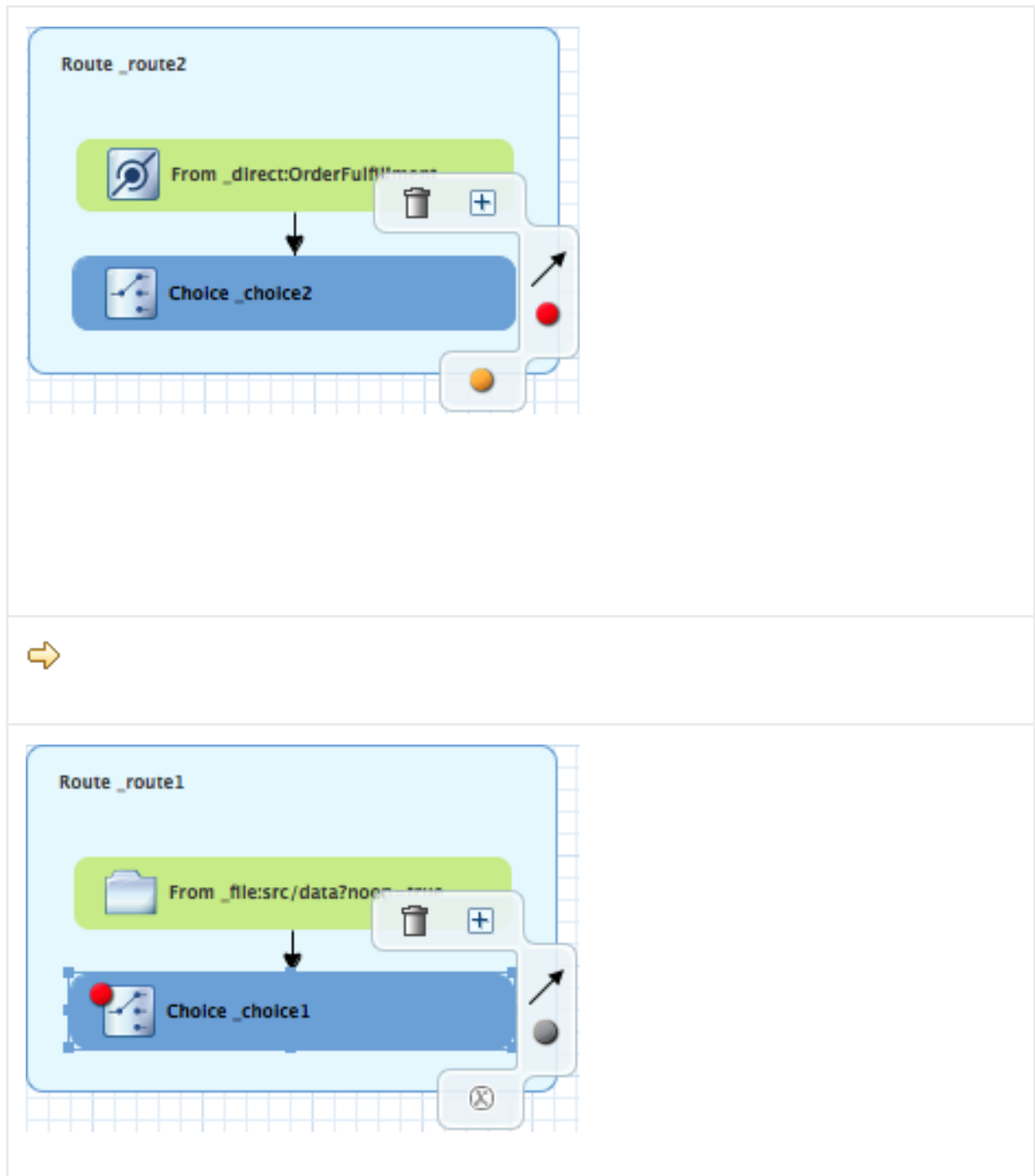
#### NOTE

If you skipped any tutorial after [Chapter 2, To Create a New Route](#) you can use the prefabricated `blueprintContext6.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/blueprint.xml` in the route editor.
2. In **Project Explorer**, expand `Camel Contexts/src/main/resources/OSGI-INF/blueprint` to expose both route entries.
3. Double-click the `Route_route1` entry to switch focus to `Route_route1` in the **Design** tab.
4. On the canvas, select the `Choice_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 , respectively. You can delete all set breakpoints by right-clicking the canvas and selecting **Delete all breakpoints**.



5. Repeat [\[setBPstep\]](#) to set an unconditional breakpoint on the following **Route\_Route1** nodes:

- **Log\_log1**
- **SetHeader\_setHeader1**
- **To\_Invalid**
- **Log\_log2**

- `SetHeader_setHeader2`
  - `To_Fulfill`
6. In **Project Explorer**, double-click `Route_route2` under `src/main/resources/OSGI-INF/blueprint` to open `Route_route2` on the canvas.
7. Repeat [\[setBPstep\]](#) to set an unconditional breakpoint on the following `Route_Route2` nodes:
- `Choice_choice2`
  - `SetHeader_setHead_usa`
  - `Log_usa`
  - `To_US`
  - `SetHeader_setHead_uk`
  - `Log_uk`
  - `To_UK`
  - `SetHeader_setHead_ger`
  - `Log_ger`
  - `To_GER`
  - `SetHeader_setHead_fr`
  - `Log_fr`
  - `To_FR`


## STEPPING THROUGH THE CBRROUTE ROUTING CONTEXT

You can step through the routing context in two ways:

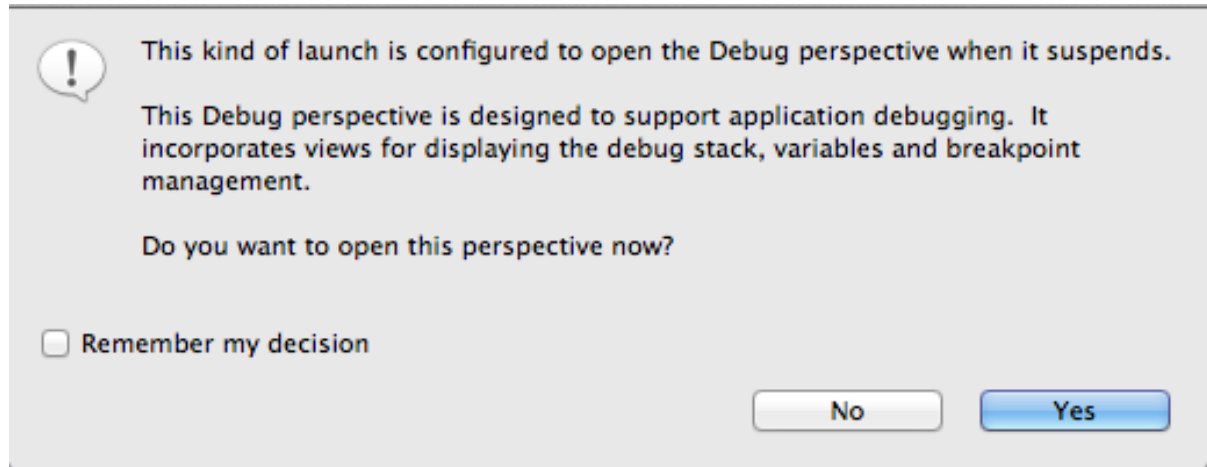
- Step over (  ) - Jumps to the next node of execution in the routing context, regardless of breakpoints.
- Resume (  ) - Jumps to the next active breakpoint in the routing context.



### 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. The debugger will skip over the disabled breakpoints.

1. In **Project Explorer**, expand the root node **CBRRoute** to expose the **blueprint.xml** file in the **Camel Contexts** folder.
2. Right-click the **blueprint.xml** file to open its context menu, and then click menu:Debug As...[ > > Local Camel Context > ].  
The Camel debugger suspends execution at the first breakpoint it encounters and asks whether you want to open **Debug** perspective now:



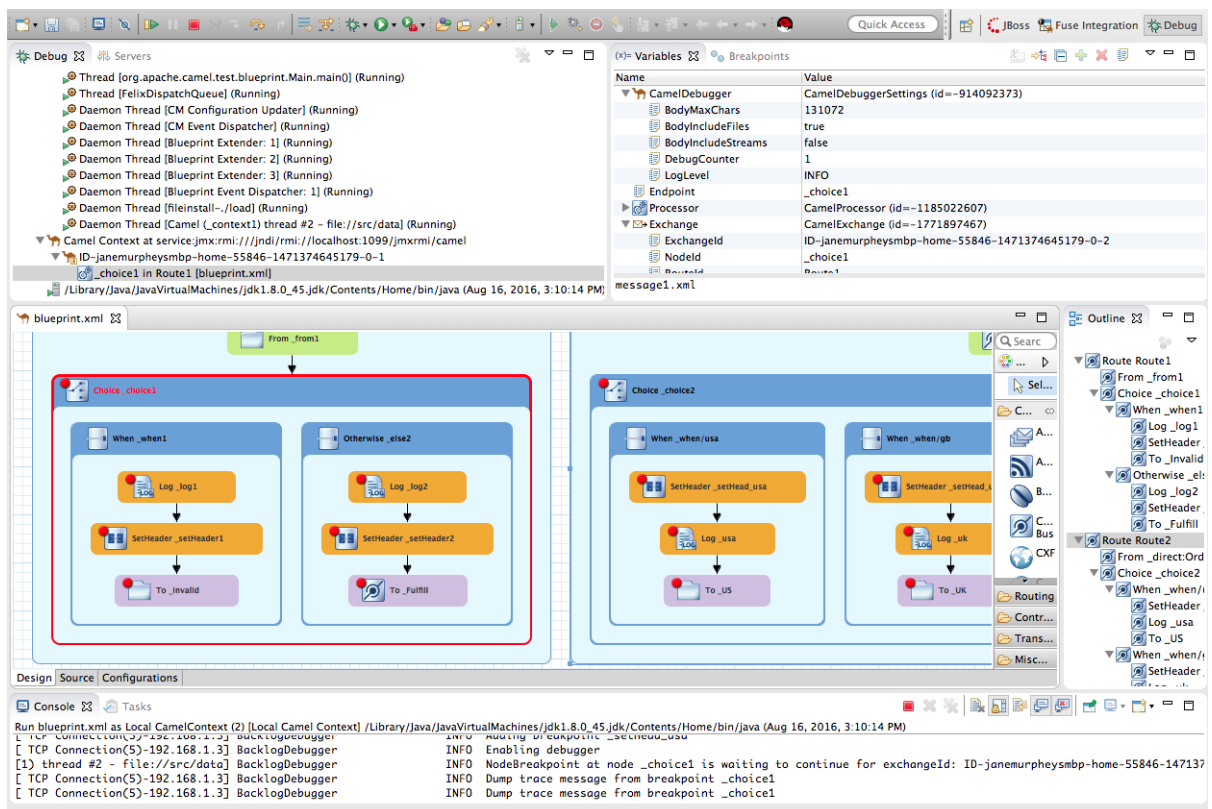
3. Click **Yes**.



#### 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 menu:Window[ > > Open Perspective > > Debug > ].

**Debug** perspective opens with the routing context suspended at **\_choice1** in **Route1** [**blueprint.xml**] as shown in the **Debug** view:

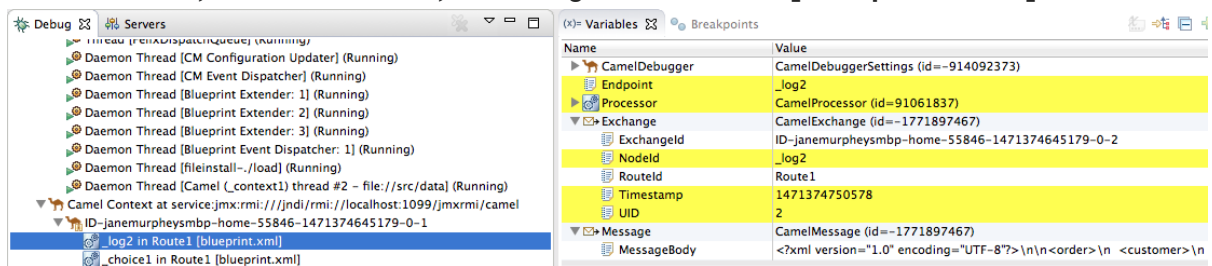


**NOTE**

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.

4. In the **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.

5. Click to step to the next breakpoint, **\_log2 in Route1 [blueprint.xml]**:



6. Expand the nodes in the **Variables** view to examine the variables that have changed since the last breakpoint at **\_choice1 in Route1 [blueprintxt.xml]**.
7. Click to step to the next breakpoint, **\_setHeader2 in Route1 [blueprint.xml]**. Examine the variables that changed since the breakpoint at **\_log2 in Route1 [blueprint.xml]**.

- In the **Debug** view, click **\_log2 in Route1 [blueprint.xml]** to populate the **Variables** view with the variable values from the breakpoint **\_log2 in Route1 [blueprint.xml]** for a quick comparison.

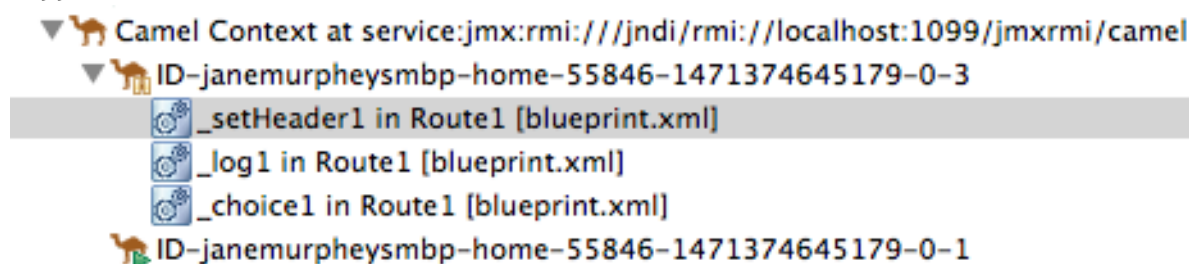
In the **Debug** view, you can switch between breakpoints within the same message flow to quickly compare and monitor changing variable values in the **Variables** view.



#### NOTE

Message flows can vary in length. For messages that transit the **InvalidOrders** branch of **Route\_route1**, the message flow is short. For messages that transit the **ValidOrders** branch of **Route\_route1**, which continues on to **Route\_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 the **Debug** view, tagged with a new breadcrumb ID:





























In this case, **ID-janemurpheysmbp-home-55846-1471374645179-0-3** 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-55846-1471374645179-0-4**.

- When **message3.xml** enters the breakpoint **\_choice1 in Route\_route1 [blueprint.xml]**, examine the **Processor** variables. The values displayed are the metrics accumulated for **message1.xml** and **message2.xml**, which previously transited the routing context:


(x)= Variables  Breakpoints  Expressions 		
Name	Value	
▼  CamelDebugger	CamelDebuggerSettings (id=-914092373)	
 BodyMaxChars	131072	
 BodyIncludeFiles	true	
 BodyIncludeStreams	false	
 DebugCounter	13	
 LogLevel	INFO	
 Endpoint	_choice1	
▼  Processor	CamelProcessor (id=-1185022607)	
 ProcessorId	_choice1	
 RouteId	Route1	
 CamelId	_context1	
 CompletedExchanges	2	
 FailedExchanges	0	
 TotalExchanges	2	
 Redeliveries	0	
 ExternalRedeliveries	0	
 HandledFailures	0	
 LastProcessingTime	84876	
 MinProcessingTime	84876	
 AverageProcessingTime	122602	
 MaxProcessingTime	160329	
 TotalProcessingTime	245205	
▼  Exchange	CamelExchange (id=-1771897463)	

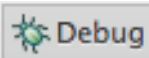
Timing metrics are in milliseconds.

- Continue stepping each message through the routing context, examining variables and console output at each processing step. When `message6.xml` enters the breakpoint `To_GER` in `Route2 [blueprint.xml]`, the debugger begins shutting down the breadcrumb threads.
- In the Menu bar, click  to terminate the Camel debugger. This will cause the Console to terminate, but you will have to manually clear the output.



#### NOTE

With a thread or endpoint selected under the Camel Context node in the **Debug** view, you need to click  twice - first to terminate the thread or endpoint and second to terminate the Camel Context, thus the session.

- 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 perspective from which you launched the Camel debugger.



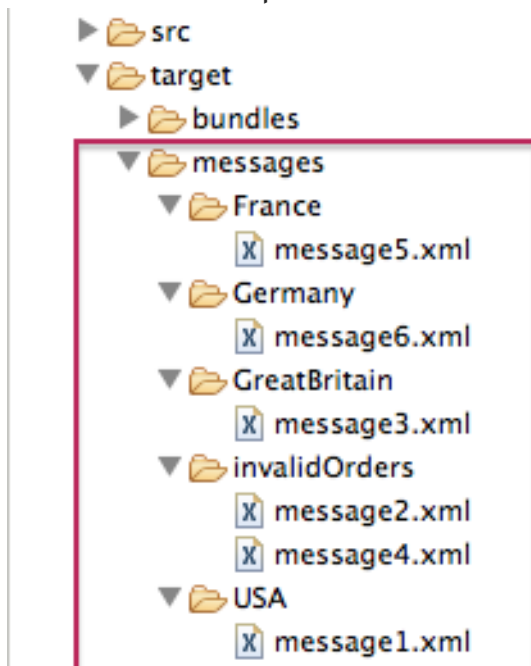
- In **Project Explorer**, open the project's context menu, and select **Refresh** to refresh the display.



### NOTE

If you terminated the session prematurely, before all messages transited the routing context, you might see, under the **CBRroute/src/data** folder, a message like this: `message3.xml.camelLock`. You need to remove it before you run the debugger on the project again. To do so, double-click the `.camelLock` message to open its context menu, and then select **Delete**. When asked, click **OK** to confirm deletion.

- 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.

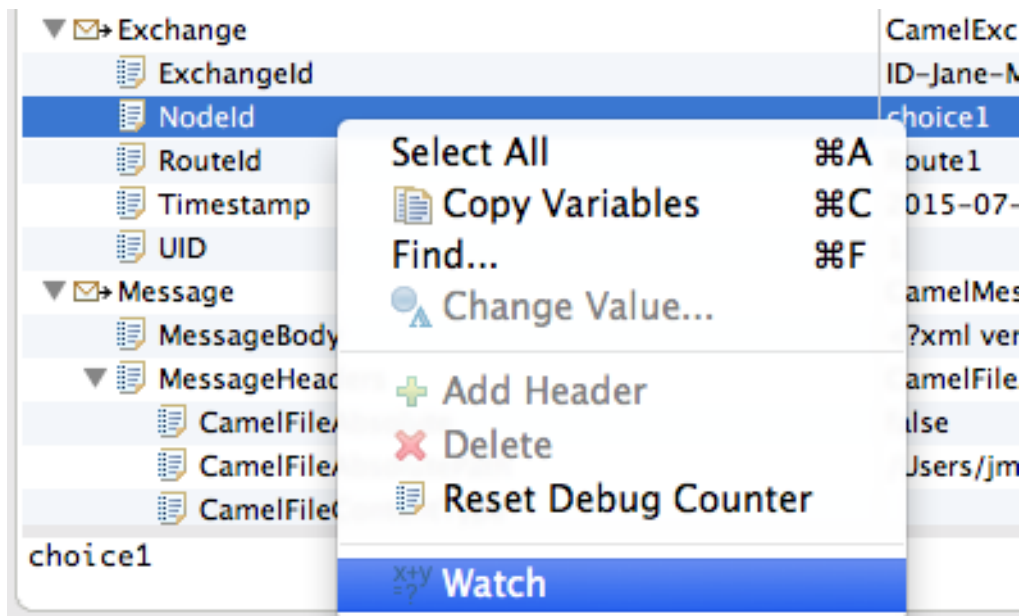
## CHANGING THE VALUE OF A VARIABLE

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.

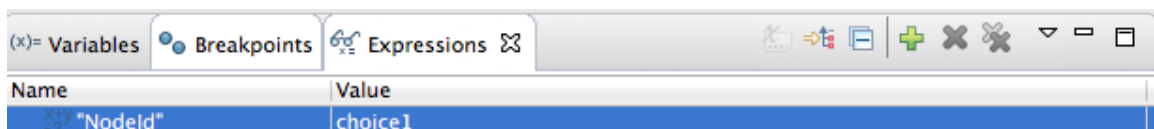
- Follow [\[startDebugger1\]](#) through [\[startDebugger3\]](#) in the section called “Stepping through the CBRroute routing context” to rerun the Camel debugger on the **CBRroute** project.
- With `message1` stopped at the first breakpoint, `_choice1` in `Route1 [blueprint.xml]`, add the variables `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:

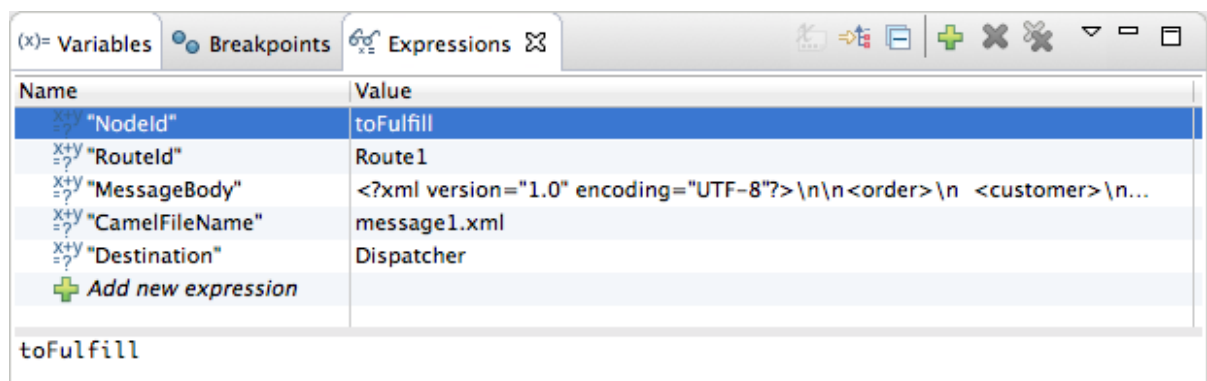
- In the **Variables** view, expand the appropriate category to expose the target variable:
- Right-click the variable (in this case, `NodeId` in the **Exchange** category) to open the context menu and select **Watch**:



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



- c. Repeat [\[selectVariable\]](#) for each of the three remaining variables.
  - d. Switch back to the **Variables** view.
3. Step `message1` through the routing context until it reaches the fourth breakpoint, `_Fulfill` in `Route1` [`blueprint.xml`].
  4. In the **Variables** view, expand the **Message** category.
  5. Repeat [\[selectVariable\]](#) to add the variable `Destination` to the watch list.
- The **Expressions** view should now contain these variables:



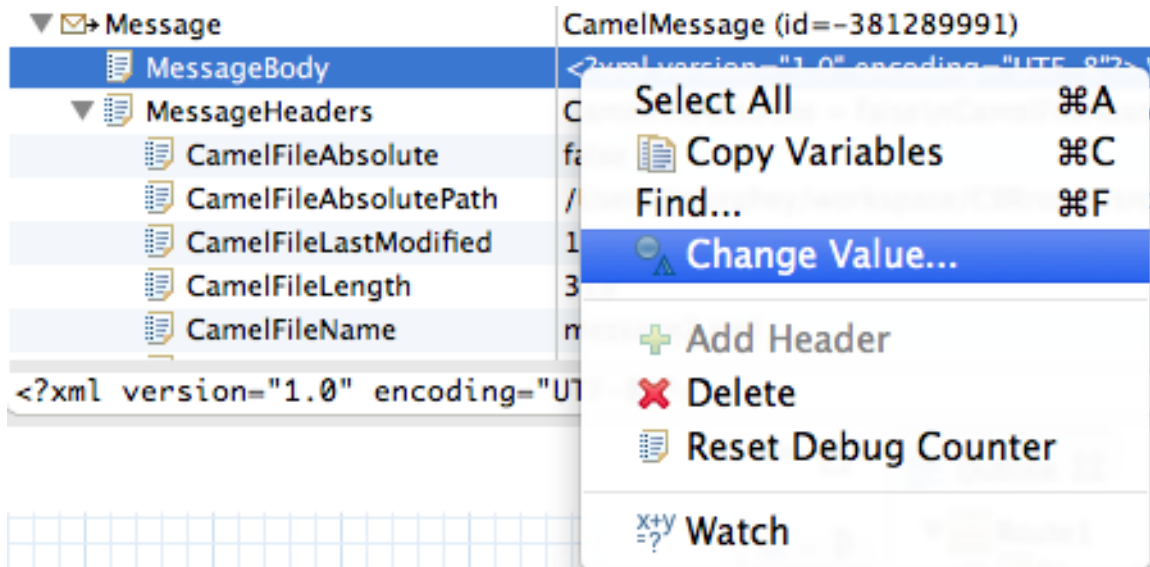
#### NOTE

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

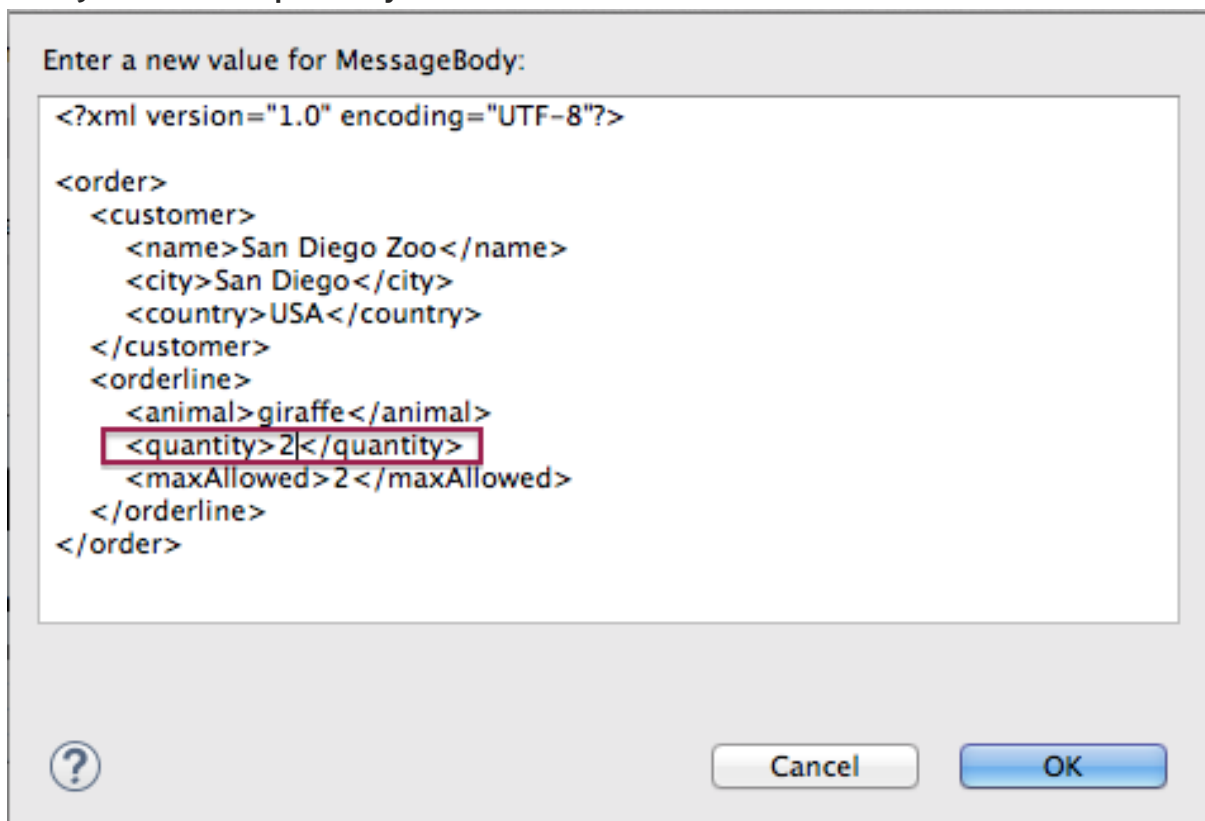
**NOTE**

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

6. Step `message1` through the rest of the routing context.
7. Stop `message2` at `_choice1` in `Route1` [`blueprint.xml`].
8. In the **Variables** view, expand the **Message** category to expose the **MessageBody** variable.
9. Right-click **MessageBody** to open its context menu, and select **Change Value...**:

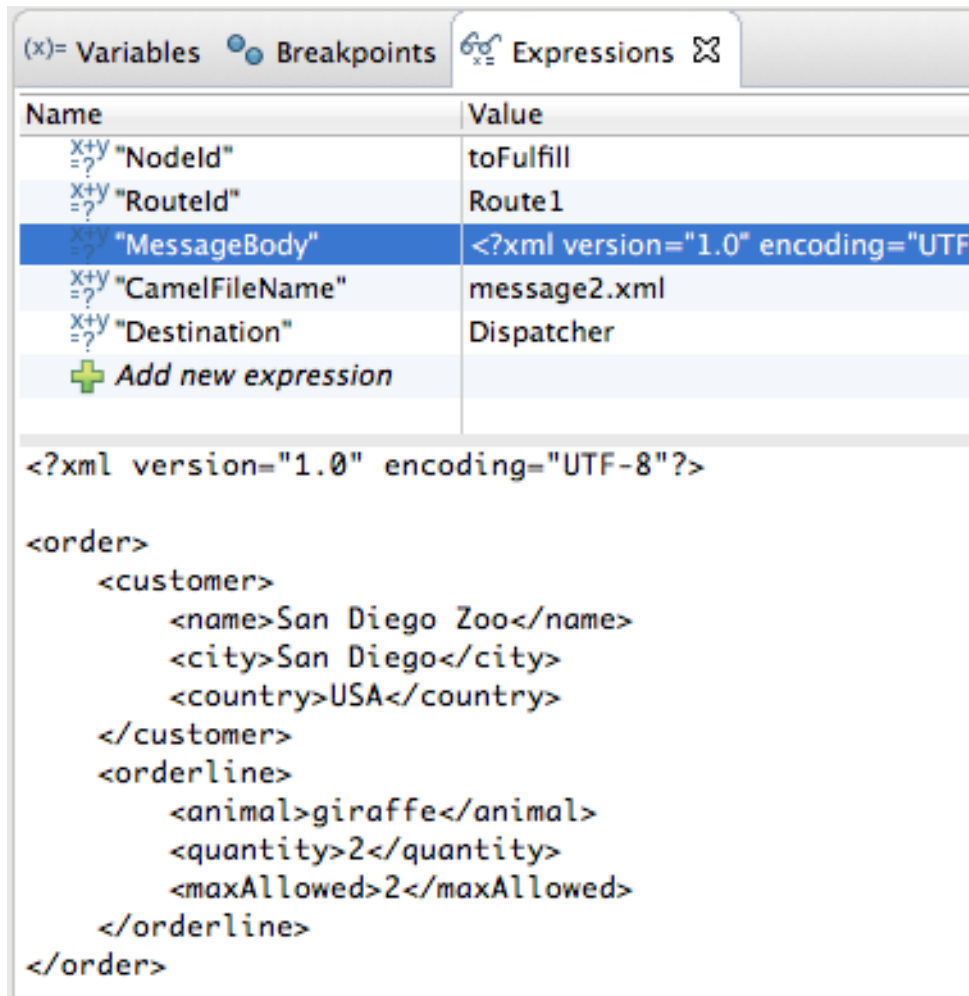


10. Change the value of `quantity` from 3 to 2:



This changes the in-memory value only.

11. Click **OK**.
12. Switch to the **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:

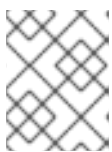


Name	Value
<code>x+y=?</code> "NodId"	toFulfill
<code>x+y=?</code> "RouteId"	Route1
<code>x+y=?</code> "MessageBody"	<?xml version="1.0" encoding="UTF
<code>x+y=?</code> "CamelFileName"	message2.xml
<code>x+y=?</code> "Destination"	Dispatcher
Add new expression	

```


<?xml version="1.0" encoding="UTF-8"?>
<order>
  <customer>
    <name>San Diego Zoo</name>
    <city>San Diego</city>
    <country>USA</country>
  </customer>
  <orderline>
    <animal>giraffe</animal>
    <quantity>2</quantity>
    <maxAllowed>2</maxAllowed>
  </orderline>
</order>

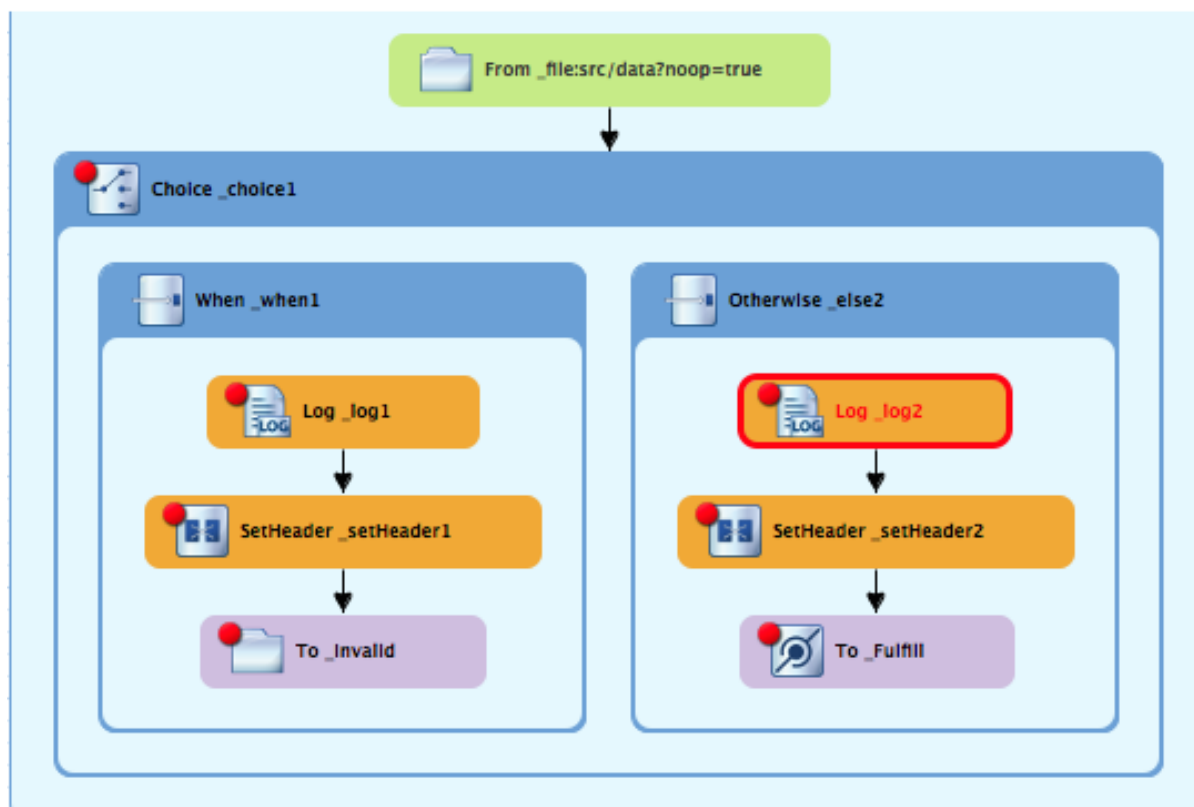
```



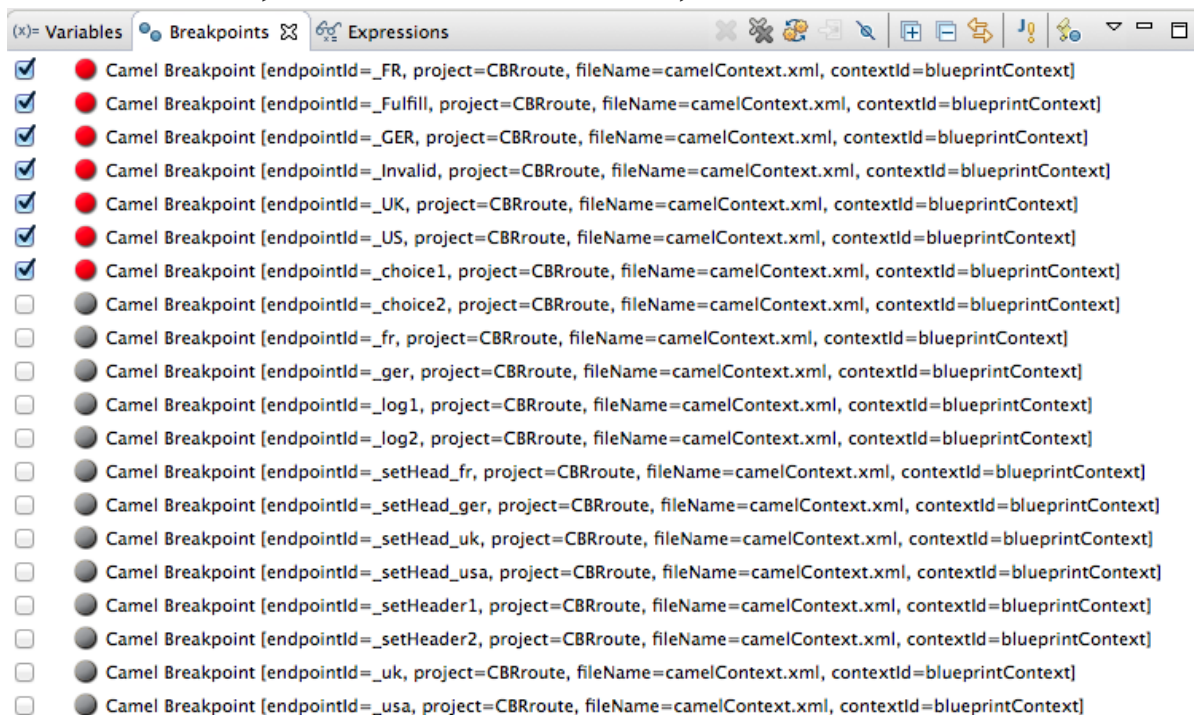
#### NOTE


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

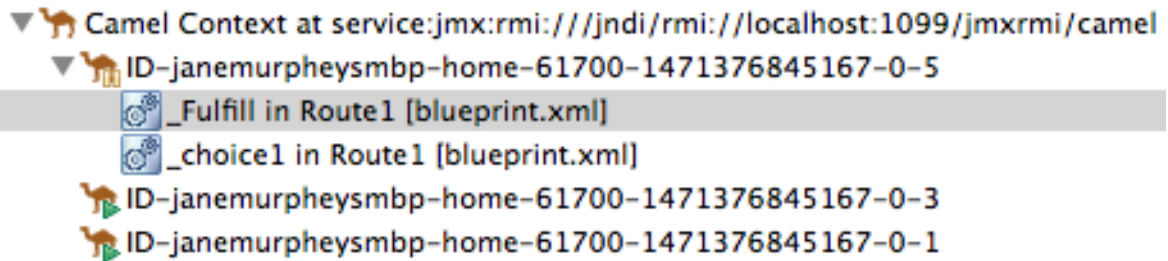
13. Click  to step to the next breakpoint.  
Instead of following the branch leading to `To_Invalid`, `message2` now follows the branch leading to `To_Fulfill` and `Route_route2`:



14. Step message2 through the routing context, checking the **Debug** view, the **Variables** view, and the **Console** output at each step.
15. Stop message3 at `_choice1` in `Route1 [blueprint.xml]`.
16. Switch to the **Breakpoints** view, and disable all breakpoints (13) listed below `_choice1`:

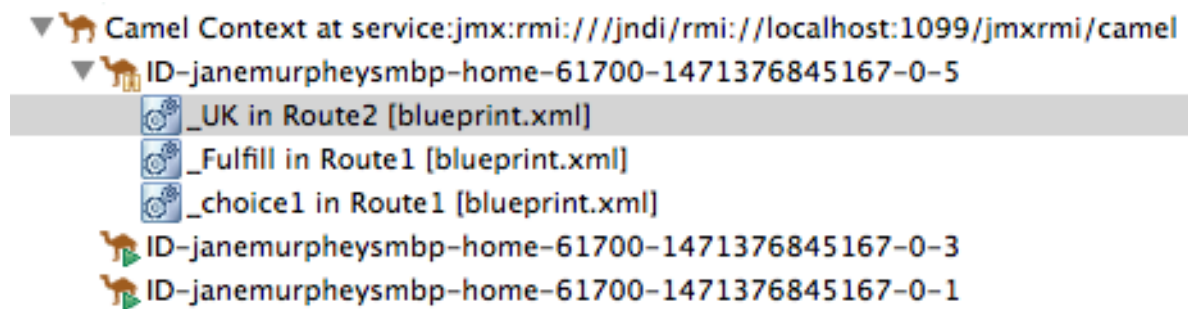


17. Click  to step to the next breakpoint:




The debugger jumps to `_Fulfill in Route1 [blueprint.xml]`.

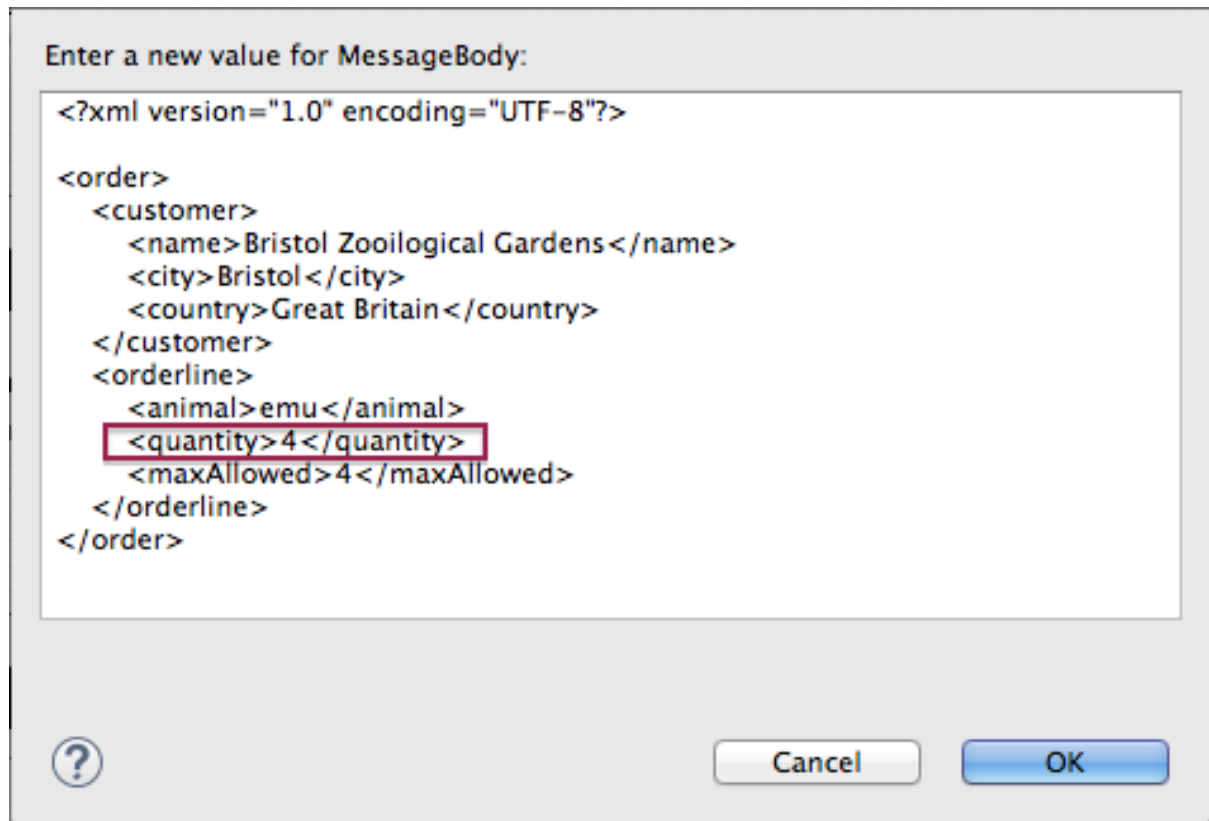
18. Click  again to step to the next breakpoint:





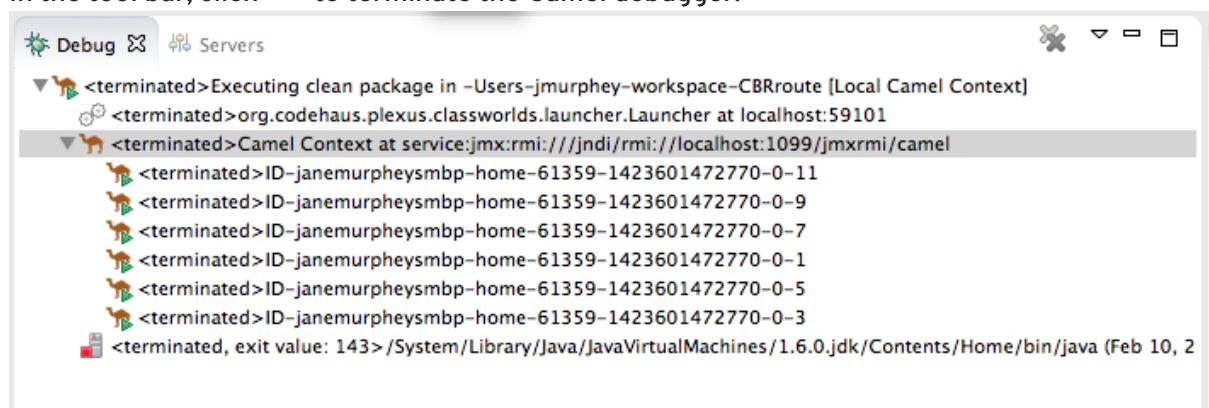
The debugger jumps to `_UK in Route2 [blueprint.xml]`.


19. In the **Breakpoints** view, re-enable all disabled breakpoints.
20. Switch back to the **Variables** view.
21. Click  to step to the next breakpoint, and stop `message4` at `_choice1 in Route1 [blueprint.xml]`.
22. Right-click `MessageBody` to open its context menu, and select **Change Value....**
23. Change the value of `quantity` from 5 to 4:

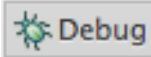




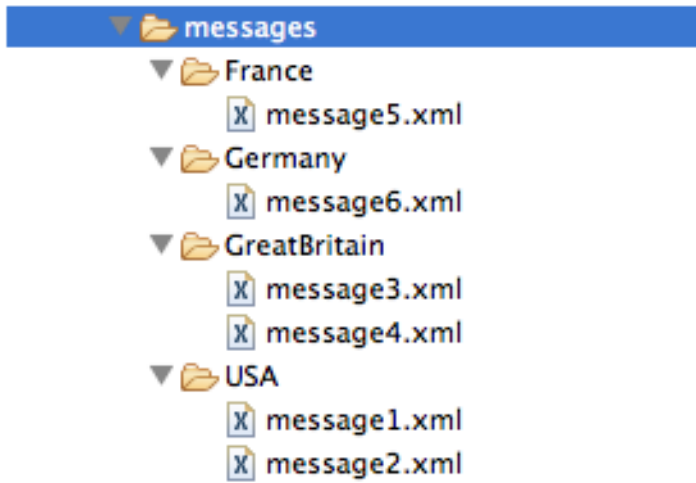
24. Click **OK**.
25. Switch to the **Expressions** view, and select the **MessageBody** variable to check the value of **quantity** in the body of **message4**.
26. Repeat [\[varChgRestart1\]](#) and [\[varChgRestart2\]](#) to step **message4** through the routing context.
27. Click  repeatedly to quickly step **message5** and **message6** through the routing context.
28. In the tool 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 the perspective from which you launched the Camel debugger.

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`. 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

This tutorial shows you how to trace 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](#).




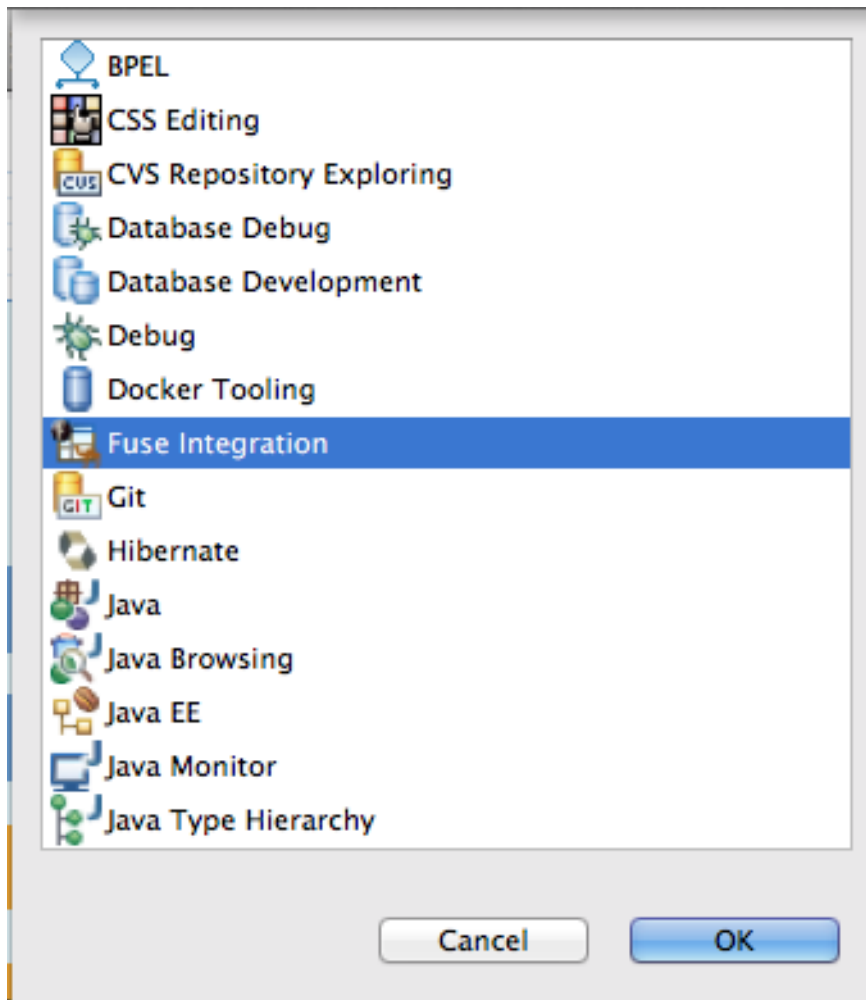
#### NOTE

If you skipped any tutorial after [Chapter 2, To Create a New Route](#), you can use the prefabricated `blueprint6.xml` file to work through this tutorial (for details, see [Chapter 1, Using the Fuse Tooling Resource Files](#)).

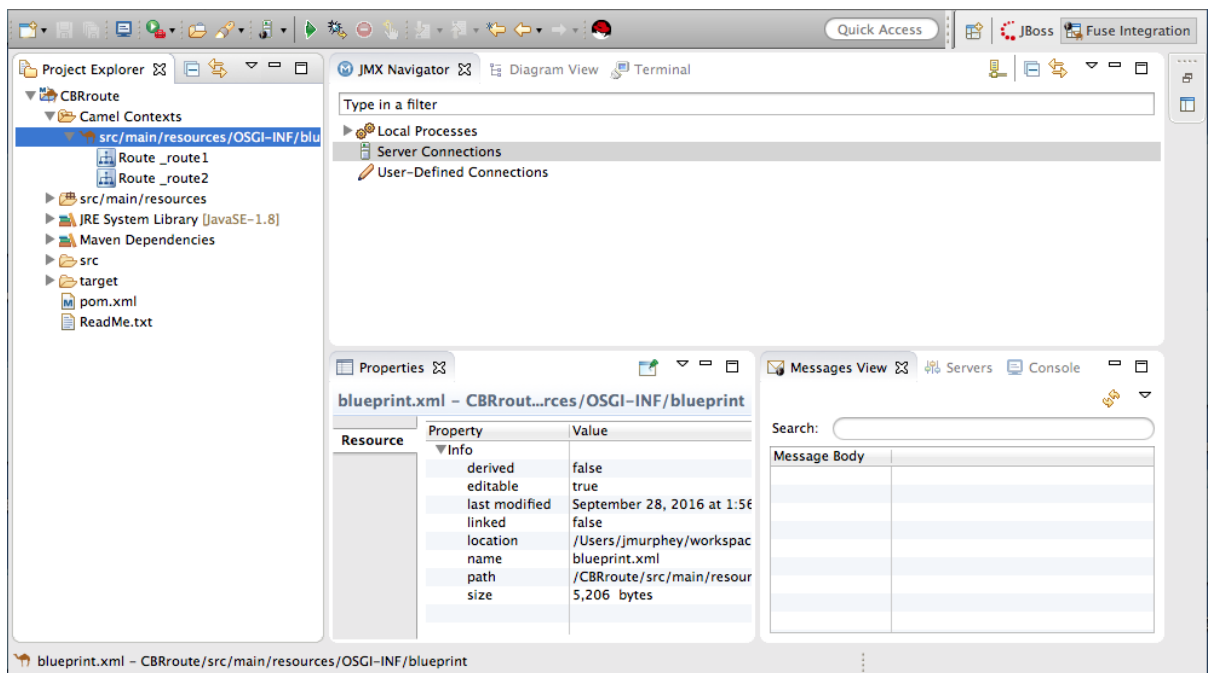
### ACCESSING FUSE INTEGRATION PERSPECTIVE

If you are not already working in **Fuse Integration** perspective:

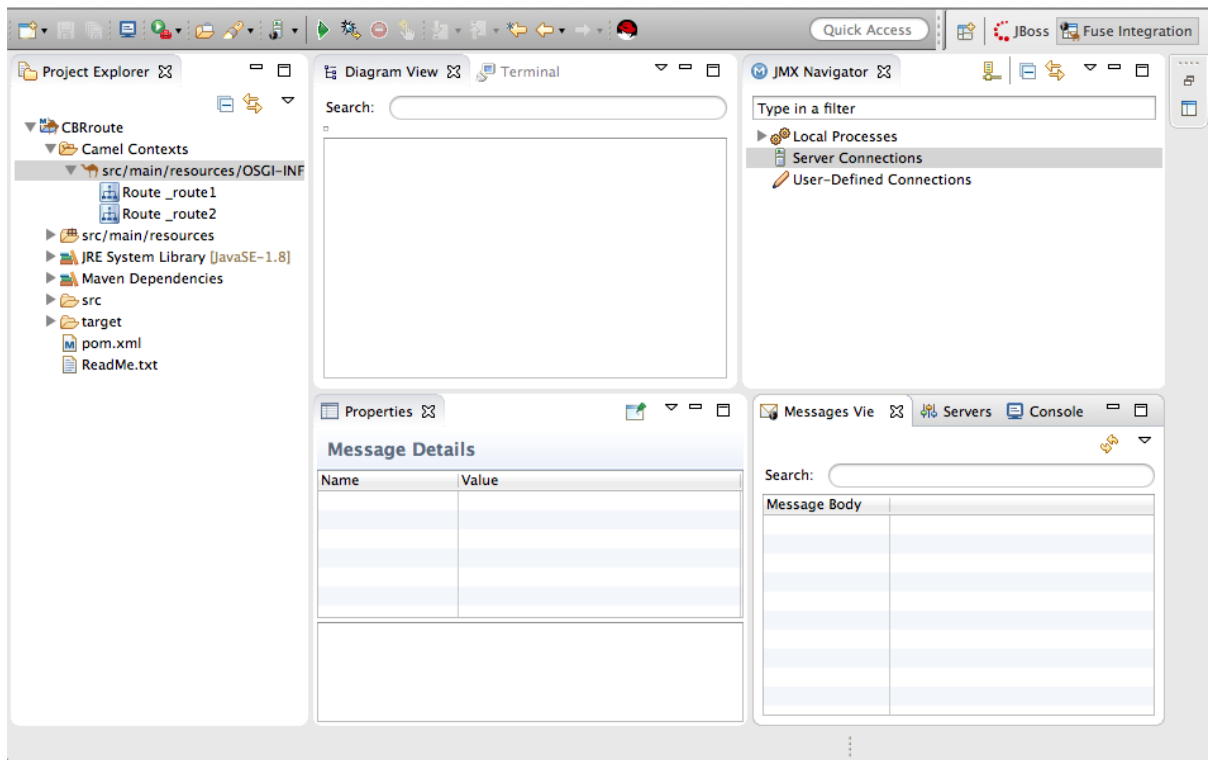
1. Click the  button on the right side of the tool bar, and then select **Fuse Integration** from the list:



Fuse Integration perspective opens in the default layout:



2. Drag the JMX Navigator tab to the far right of the Terminal tab and drop it there:

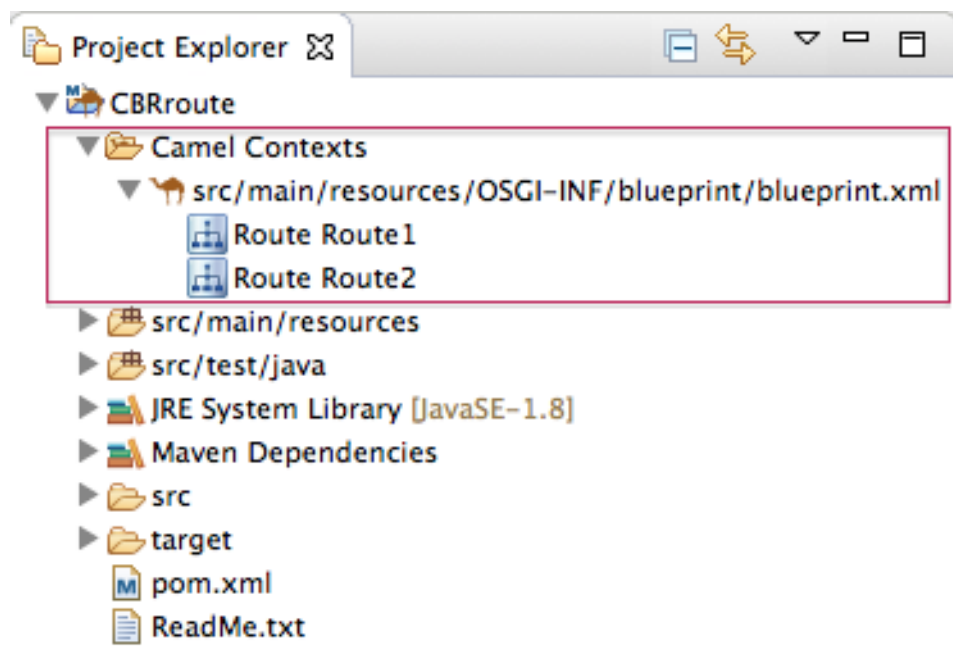


This arrangement provides more space for **Diagram View** to display the routing context's nodes graphically, which makes it easier for you to visually trace the path that messages take in traversing the routing context.

## NOTE

To make it easy to access a routing context `.xml` file, especially when a project consists of multiple contexts, the tooling lists them under the **Camel Contexts** folder in **Project Explorer**.

Additionally, all routes in a routing context are displayed as icons directly under their context file entry. To display a single route in the routing context on the canvas, double-click its icon in **Project Explorer**. To display all routes in the routing context, double-click the context file entry.



## STARTING MESSAGE TRACING

To start message tracing on the `CBRroute` project:

1. In **Project Explorer**, expand the `CBRroute` project to expose `src/main/resources/OSGI-INF/blueprint/blueprint.xml`.
2. Right-click `src/main/resources/OSGI-INF/blueprint/blueprint.xml` to open the context menu.
3. Select menu: `Run As[ > > Local Camel Context (without tests) > ]`.

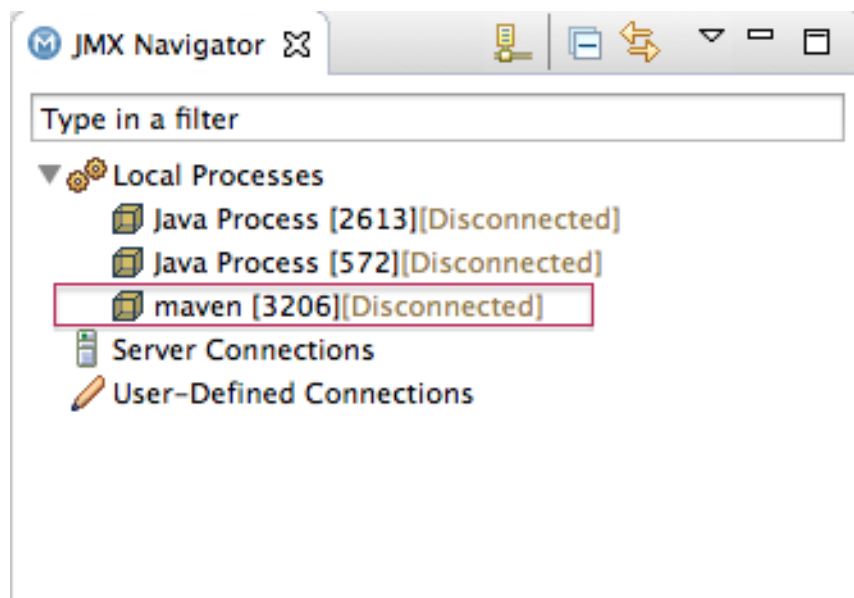


### NOTE

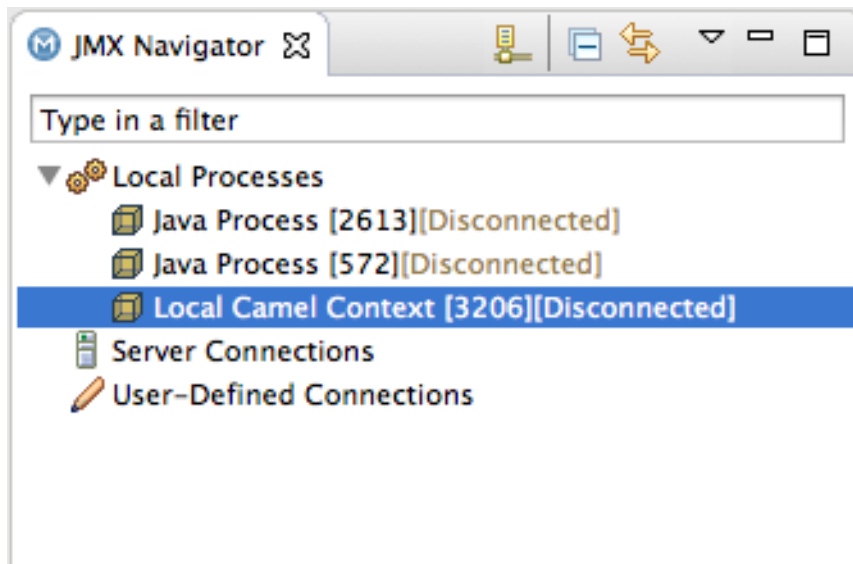
If you select **Local Camel Context**, the tooling reverts to running without tests because you have not yet created a JUnit test for the `CBRroute` project. You will do that later in [Chapter 8, To Test a Route with JUnit](#)

4. In **JMX Navigator**, expand **Local Processes**.

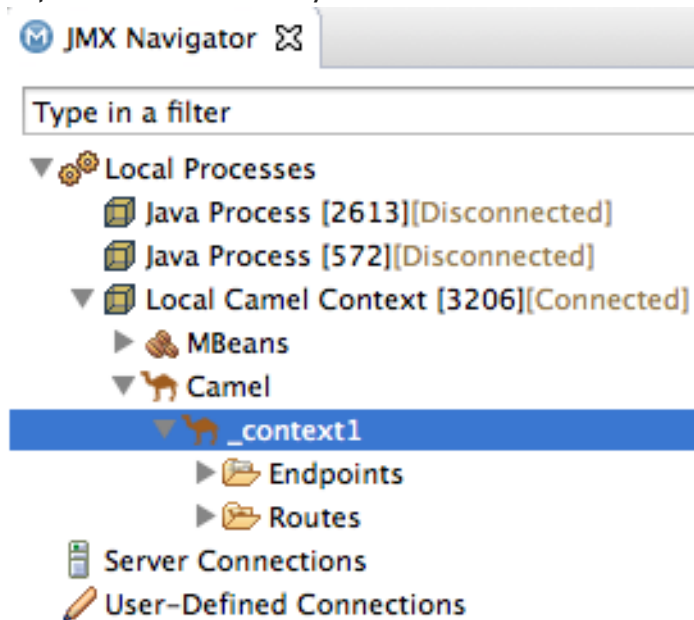
When you first expand **Local Processes**, you see the node `maven[Id][Disconnected]`:



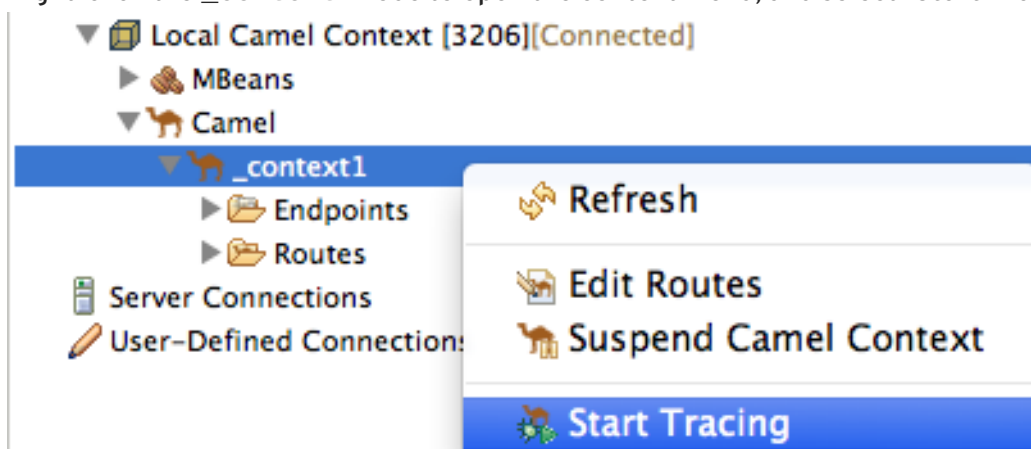
When you click this node, it changes to `Local Camel Context[Id][Disconnected]` (retaining the same `Id` as its predecessor):



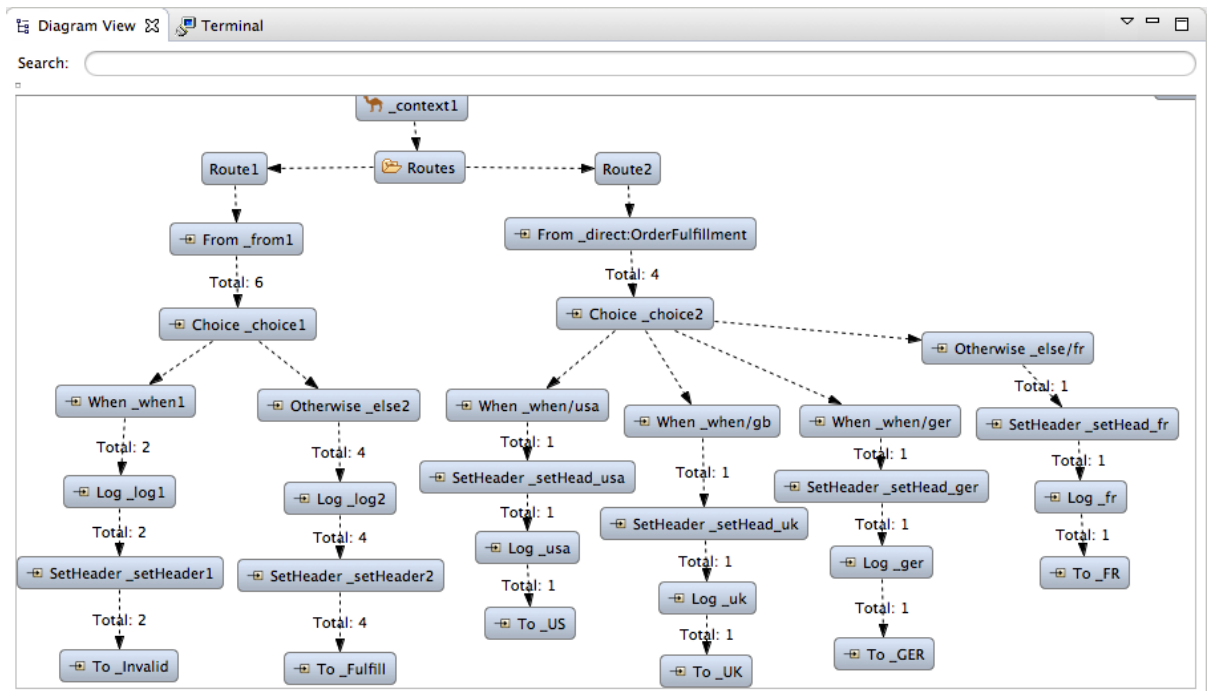
5. Double click **Local Camel Context [Id][Disconnected]** to connect to it, and then expand the elements of your route:



6. Right-click the **\_context1** node to open the context menu, and select **Start Tracing**:



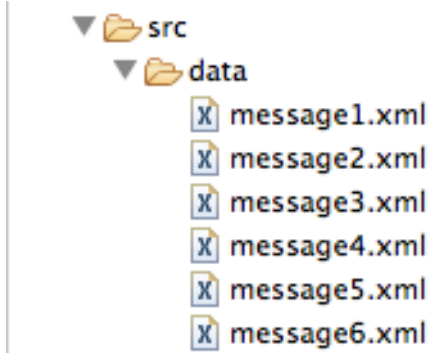
The tooling displays a graphical representation of your routing context in **Diagram View**:



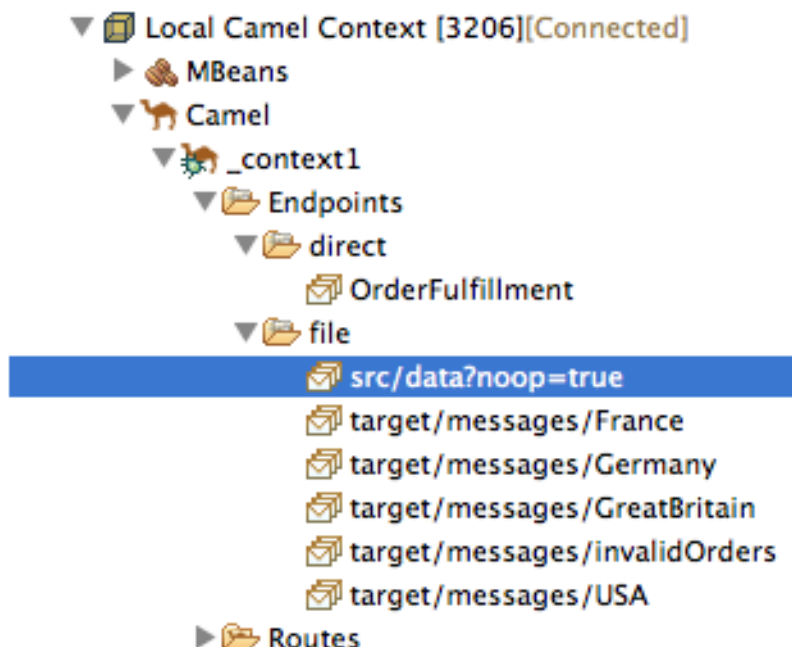
## DROPPING MESSAGES ON THE RUNNING CBRROUTE PROJECT

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):



2. Drag **message1.xml** and drop it on the **\_context1>Endpoints>file>src/data?noop=true** node in **JMX Navigator**:



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 `_context1` node in **JMX Navigator**.



#### NOTE

The `Local Camel Context [xxx]` tree collapses to the `_context1` node after you drop the next message on the input `src/data?noop=true` 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** if you want them to persist across all message traces.

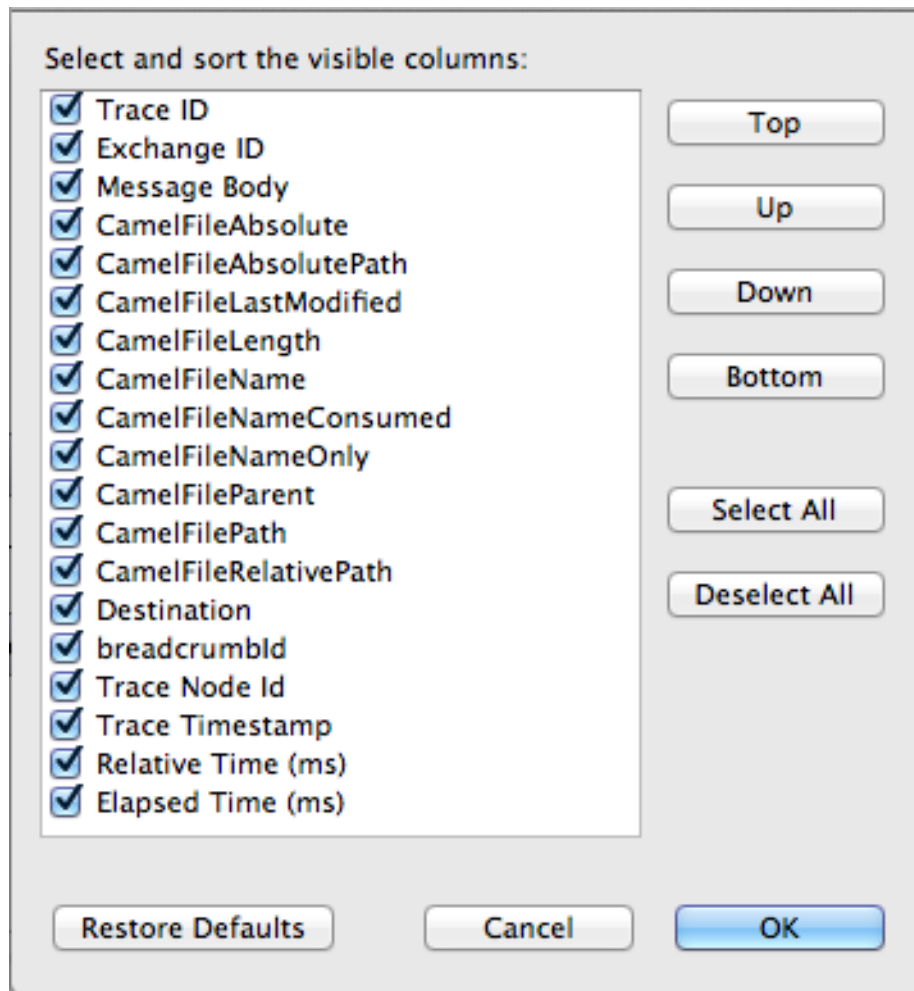
1. Switch from **Console** to **Messages View**.
2. Click the `_context1` node in **JMX Navigator** to initialize **Messages View** with `message1.xml`'s details.



#### 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 `menu:View Menu[ > > Configure Columns... > ]` method instead.

3. In **Messages View**, click the  icon on the panel's menu bar, and select **Configure Columns...** to open the **Configure Columns** wizard:

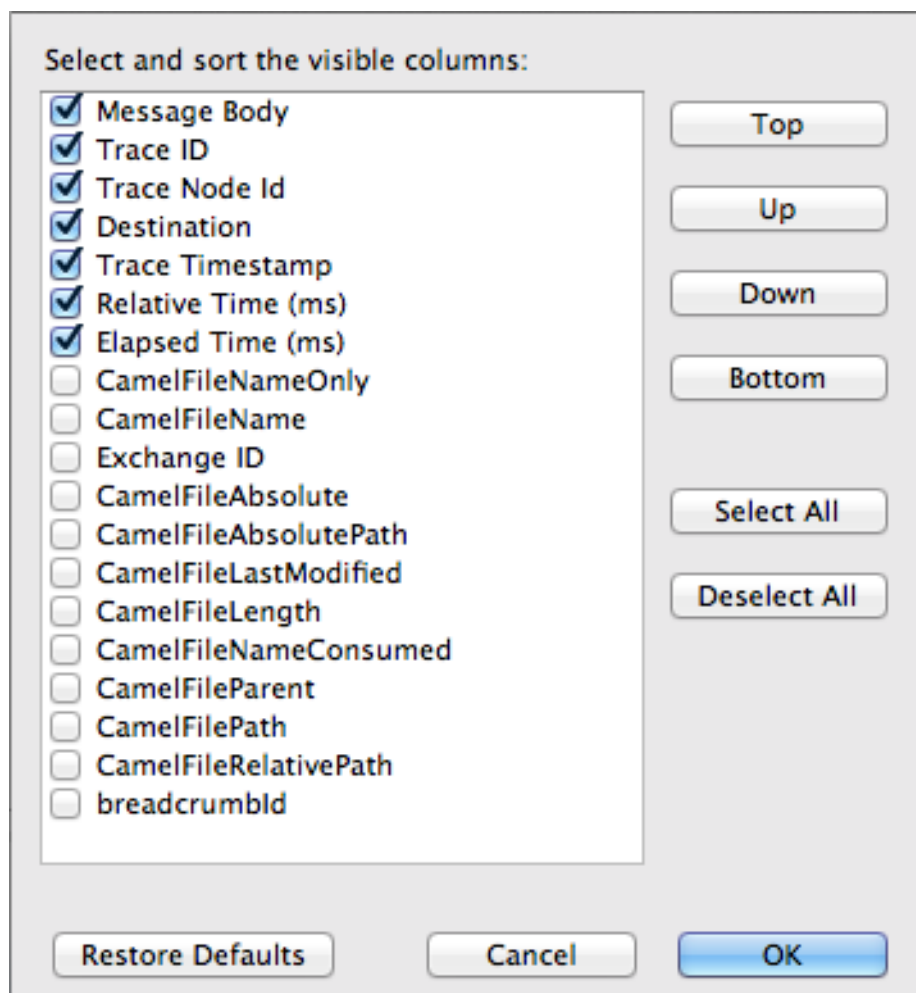
**NOTE**

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.

4. In the **Configure Columns** wizard, select and order the columns this way:






These columns 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:

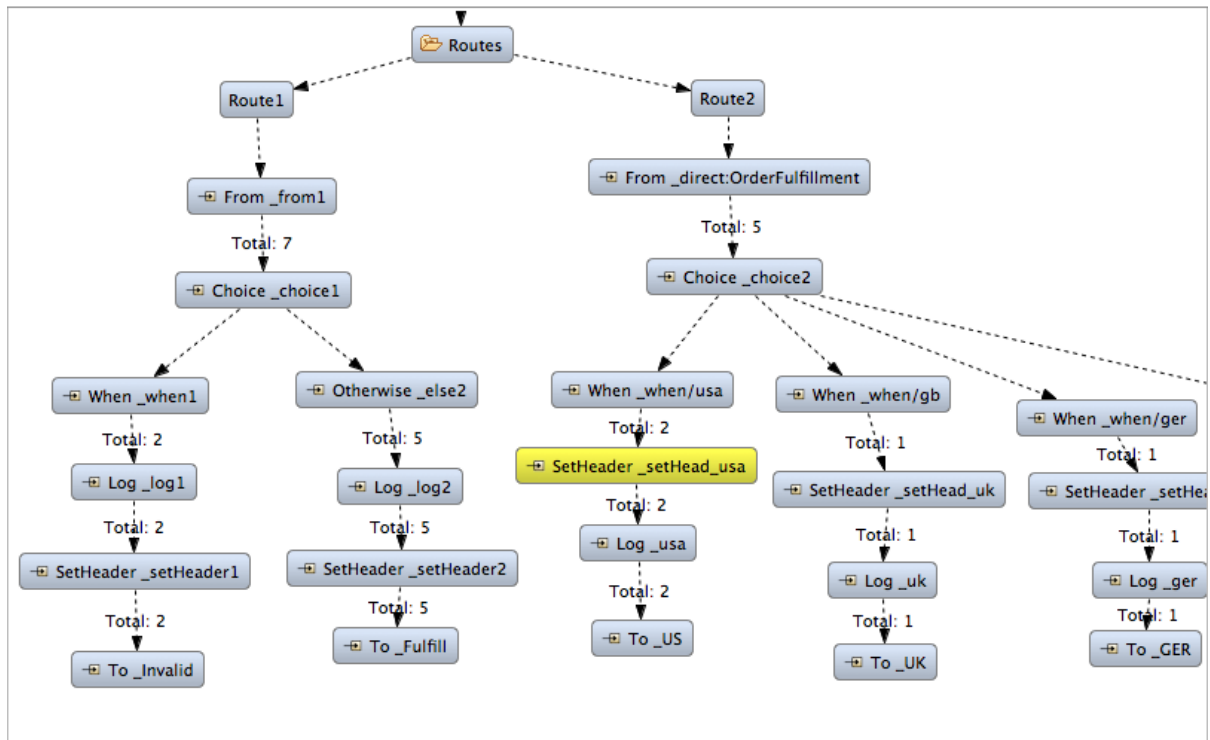
1. In **Messages View**, click the  (Refresh button) on top, right of the panel's menu bar to populate the view with `message1.xml`'s message traces. Each time you drop a message on the input `src` node in **JMX Navigator**, you need to refresh **Messages View** to populate it with the message traces.
2. Click one of the message traces to see more details about it in **Properties view**:

Message Body	Trace ID	Trace Node Id	Destination	Trace Timestamp	Relative T	Elapsed T
<?xml version=...	1	Route1		Wed Aug 10 16:5...	0	
<?xml version=...	2	_choice1		Wed Aug 10 16:5...	0	0
<?xml version=...	3	_log2		Wed Aug 10 16:5...	6	6
<?xml version=...	4	_setHeader2		Wed Aug 10 16:5...	7	1
<?xml version=...	5	_Fulfill	Dispatcher	Wed Aug 10 16:5...	8	1
<?xml version=...	6	Route2	Dispatcher	Wed Aug 10 16:5...	0	-8
<?xml version=...	7	_choice2	Dispatcher	Wed Aug 10 16:5...	9	9
<?xml version=...	8	_setHead_usa	Dispatcher	Wed Aug 10 16:5...	11	2
<?xml version=...	9	_usa	USA	Wed Aug 10 16:5...	12	1
<?xml version=...	10	_US	USA	Wed Aug 10 16:5...	13	1

The tooling displays the details about a message trace (including message headers when they are set) in the top half of the **Properties** view and the contents of the message instance in the bottom half of the **Properties** view. 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:



## FINISHING UP

1. Drag `message2.xml` and drop it on the `_context1>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 [\[msg1drag\]](#) through [\[msgView\]](#) 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.

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:

The screenshot shows the Camel IDE interface. The **Diagram View** displays a route starting with `From_direct:OrderFulfillment` (Total: 6) leading to a `Choice_choice2` node. The choice branches into `When_when/usa` (Total: 2) and `When_when/gb` (Total: 2). The `When_when/usa` branch includes `SetHeader_setHeader_usa` (Total: 2), `Log_usa` (Total: 2), and `To_US` (Total: 2). The `When_when/gb` branch includes `SetHeader_setHeader_uk` (Total: 2), `Log_uk` (Total: 2), and `To_UK` (Total: 2). There is also an `Other` branch leading to `When_when` (Total: 1) and `SetHeader_setHeader` (Total: 1). The `Choice_choice1` node branches into `When_when1` (Total: 3) and `Otherwise_else2` (Total: 6). The `When_when1` branch includes `Log_log1` (Total: 3) and `SetHeader_setHeader1` (Total: 3). The `Otherwise_else2` branch includes `Log_log2` (Total: 6) and `SetHeader_setHeader2` (Total: 6). The route ends with `To_Invalid` (Total: 3) and `To_Fulfill` (Total: 6).

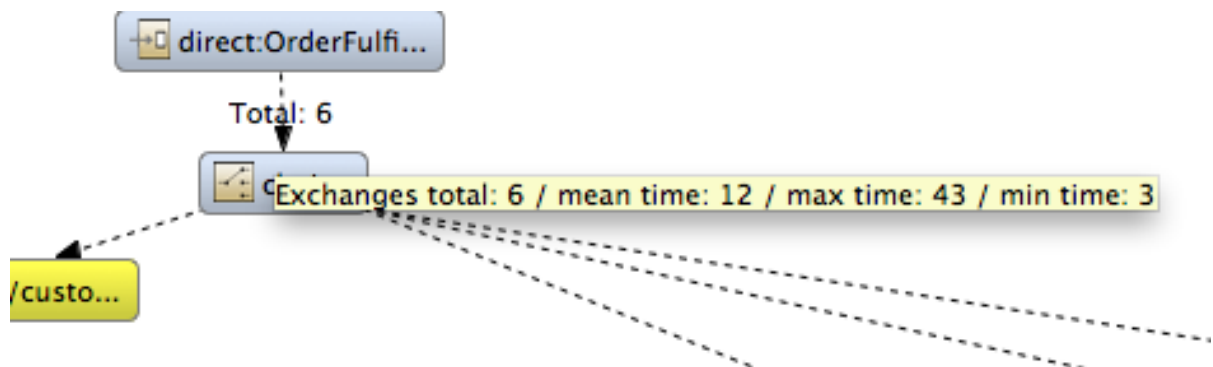
The **Message Details** panel shows the following properties:

Name	Value
CamelFileAbsolutePath	false
CamelFileAbsolutePath	/Users/jmurphy/workspace0804/CBRoute/src/data/ID...
CamelFileLastModified	1470870946000
CamelFileLength	319

The **Messages View** panel shows a table of message traces:



Message Body	Trace ID	Trace Node Id	Destination	Trace Timestamp	Relative Time (t Elapsed)
<?xml version=...	10	_US	USA	Wed Aug 10 19:...	16
<?xml version=...	11	Route1		Wed Aug 10 19:...	0
<?xml version=...	12	_choice1		Wed Aug 10 19:...	0
<?xml version=...	13	_log1		Wed Aug 10 19:...	3
<?xml version=...	14	_setHeader1		Wed Aug 10 19:...	4
<?xml version=...	15	_Invalid	invalidOrders	Wed Aug 10 19:...	4
<?xml version=...	16	Route1		Wed Aug 10 19:...	0
<?xml version=...	17	_choice1		Wed Aug 10 19:...	0
<?xml version=...	18	_log2		Wed Aug 10 19:...	3
<?xml version=...	19	_setHeader2		Wed Aug 10 19:...	3
<?xml version=...	20	_Fulfill	Dispatcher	Wed Aug 10 19:...	4
<?xml version=...	21	Route2	Dispatcher	Wed Aug 10 19:...	4

Hovering over the displayed metrics reveals additional metrics about message flow:



- Mean time the step took to process a message
- Maximum time the step took to process a message
- Minimum time the step took to process a message

#### 4. When done:

- In **JMX Navigator**, right-click `_context1` 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 as a **Local Camel Context**, instead of **Local Camel Context (without tests)**. See [Chapter 8, To Test a Route with JUnit](#) for details.

## CHAPTER 8. TO TEST A ROUTE WITH JUNIT

This tutorial shows you how to use the **New Camel Test Case** wizard to create a test case for your route and then test the route.

### OVERVIEW

The **New Camel Test Case** wizard generates a boilerplate JUnit test case. When you create or modify a route (for example, adding more processors to it), you create or modify the generated test case to add expectations and assertions specific to the route you created or updated. This ensures that the test is valid for the route.

### GOALS

In this tutorial you will:

- Create the `/src/test/` folder to store the JUnit test case
- Generate the 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 need the **CBRroute** project you used in [Chapter 7, To Trace a Message Through a Route](#)

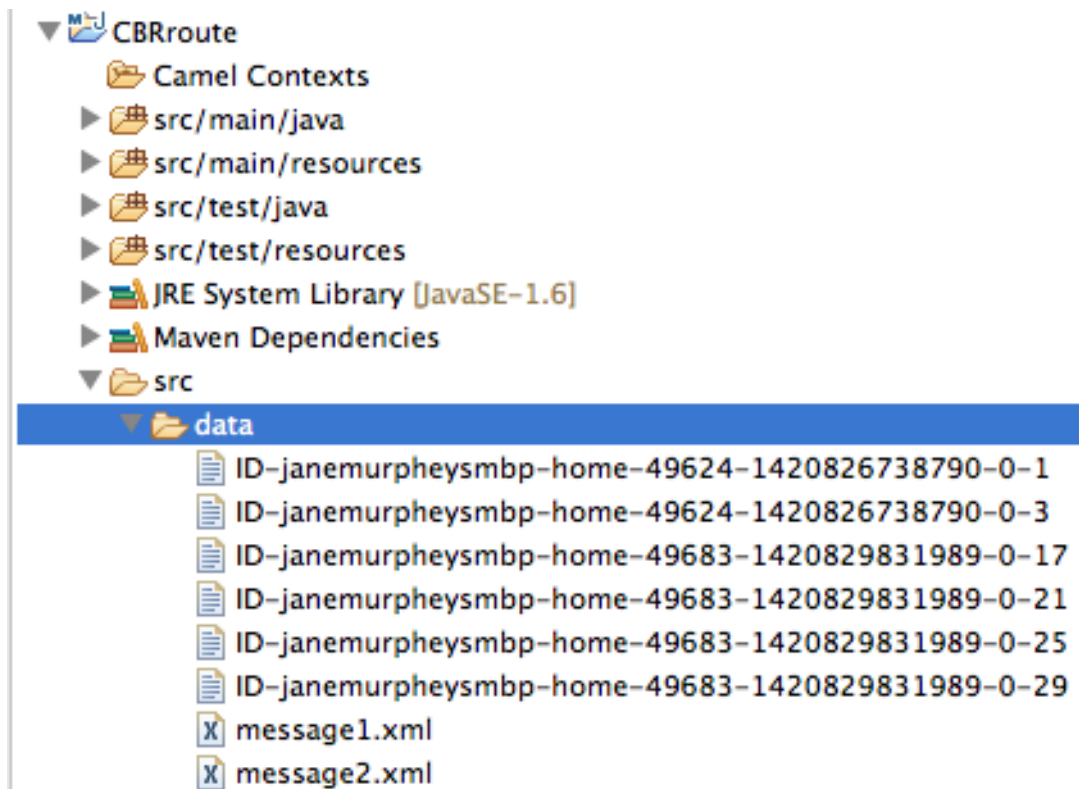


#### NOTE

If you skipped any tutorial after [Chapter 2, To Create a New Route](#), you can use the prefabricated `blueprintContext6.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

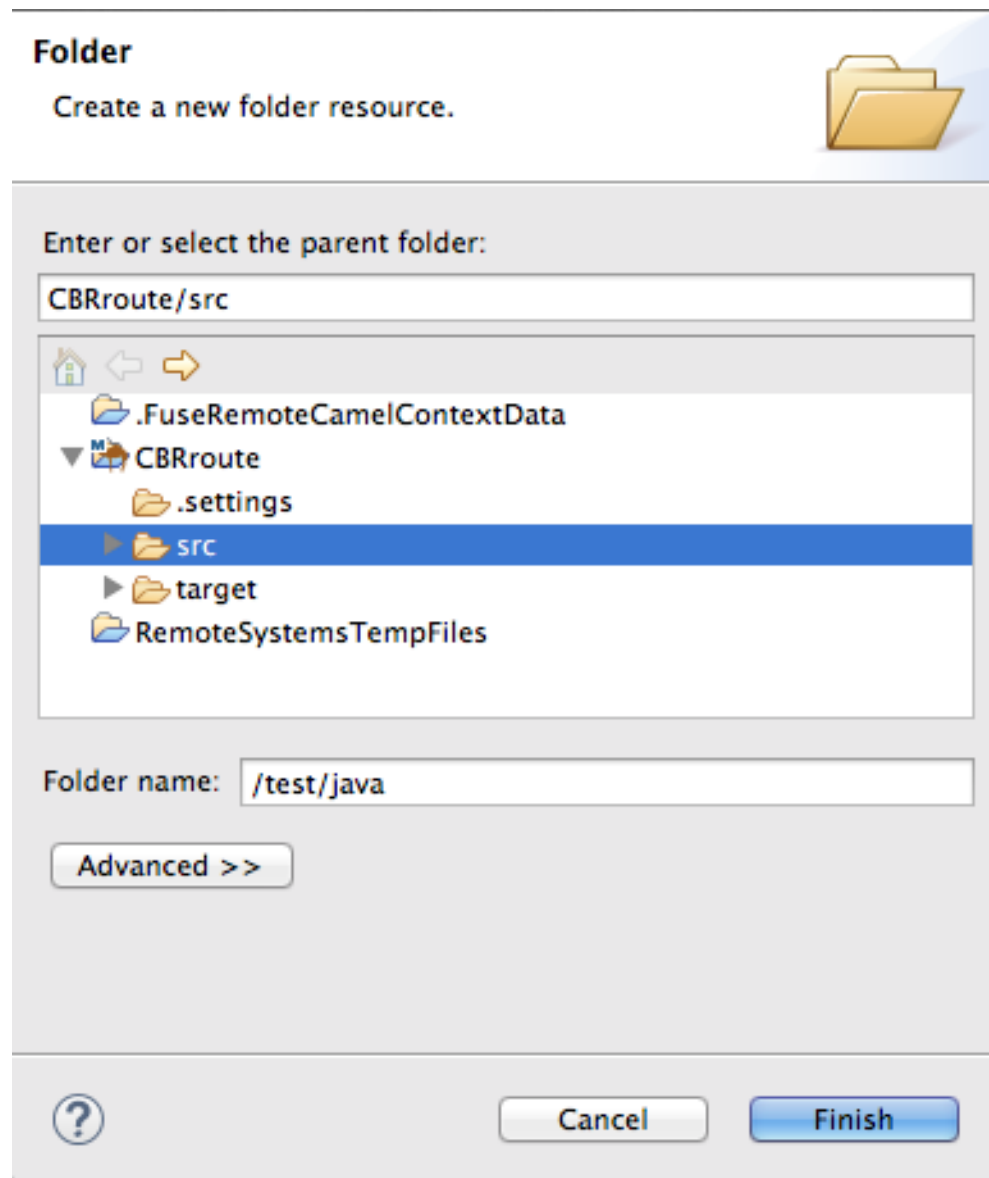


Select all trace-generated messages in batch, right-click to open the context menu, and select **Delete**.

## CREATING THE SRC/TEST FOLDER

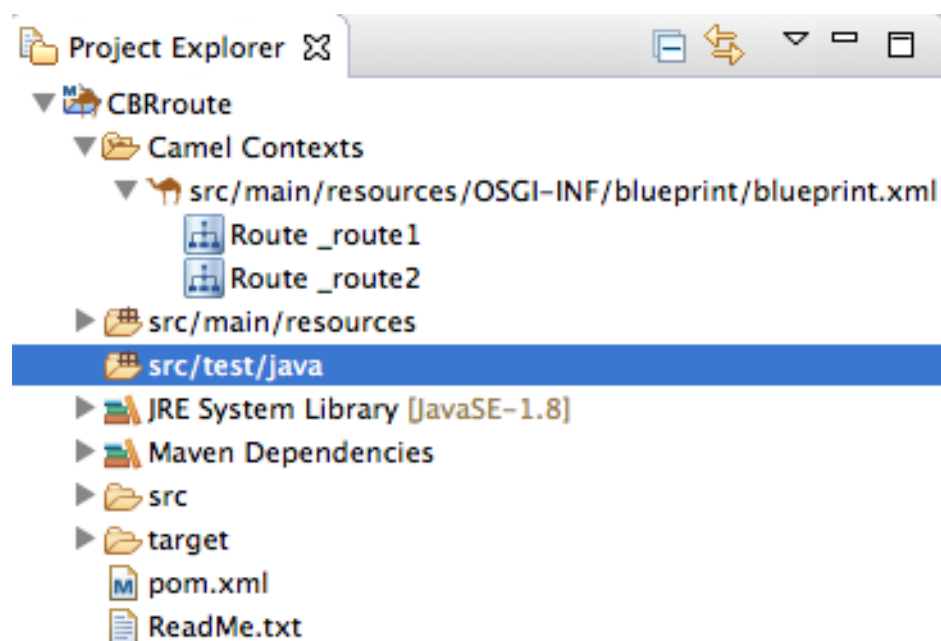
Before you create a JUnit test case for the CBRroute project, you must create a folder for it that is included in the build path:

1. In **Project Explorer**, right-click the CBRroute project's root to open the context menu, and then select menu:New[ > > Folder > ].
2. In the **New Folder** dialog, in the project tree pane, expand the CBRroute node and select the src folder.  
Make sure CBRroute/src appears in the **Enter or select the parent folder** field.
3. In **Folder name**, enter /test/java:

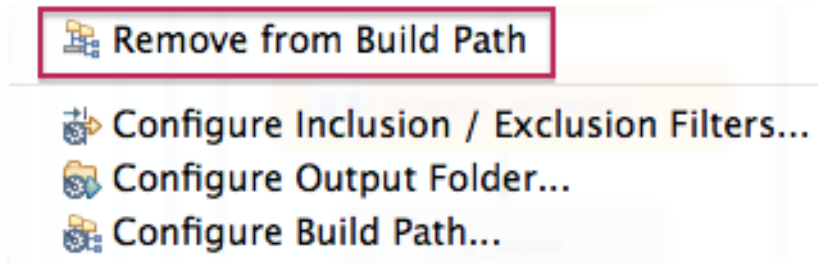


4. Click **Finish**.

In **Project Explorer**, the new `src/test/java` folder appears under the `src/main/resources` folder:



5. Verify that the new `/src/test/java` folder is included in the build path.
  - a. In **Project Explorer**, right-click the `/src/test/java` folder to open the context menu.
  - b. Select **Build Path** to see the menu options:  
The menu option **Remove from Build Path** verifies that the `/src/test/java` folder is currently included in the build path:



## CREATING THE JUNIT TEST CASE

To create a JUnit test case for the `CBRroute` project:

1. In **Project Explorer**, select `src/test/java`.
2. Right-click it to open the context menu, and then select menu:`New[ > > Camel Test Case > ]`:

**Camel JUnit Test Case**

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

Source folder:

Package:  (default)

Camel XML file under test:

Name:

Which method stubs would you like to create?

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

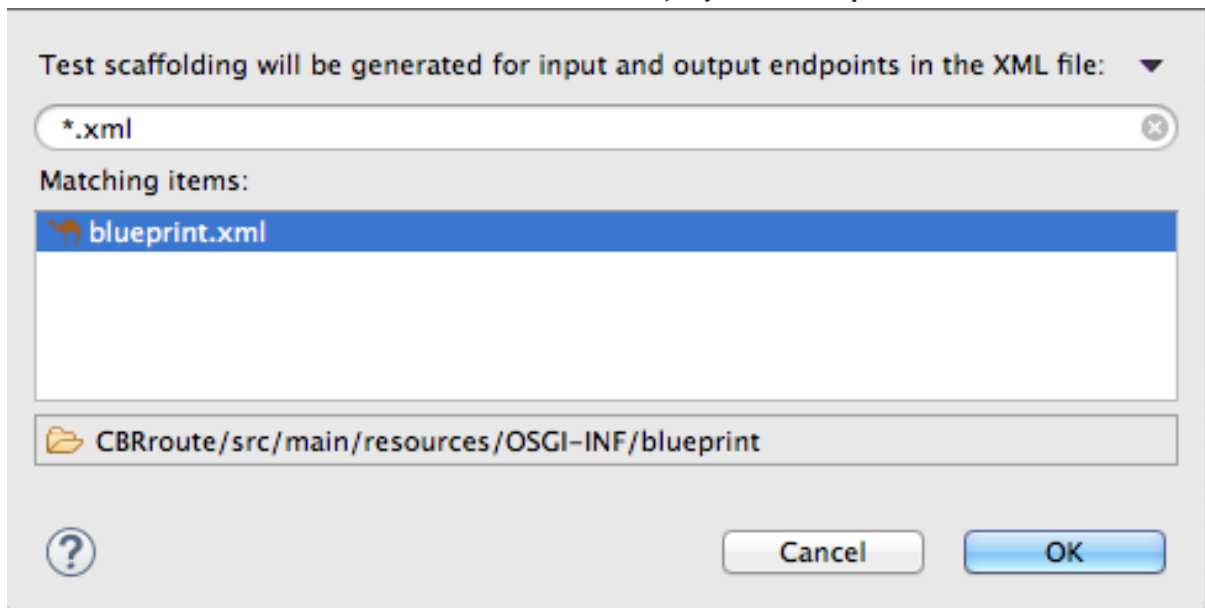
Do you want to add comments? (Configure templates and default value [here](#))

Generate comments

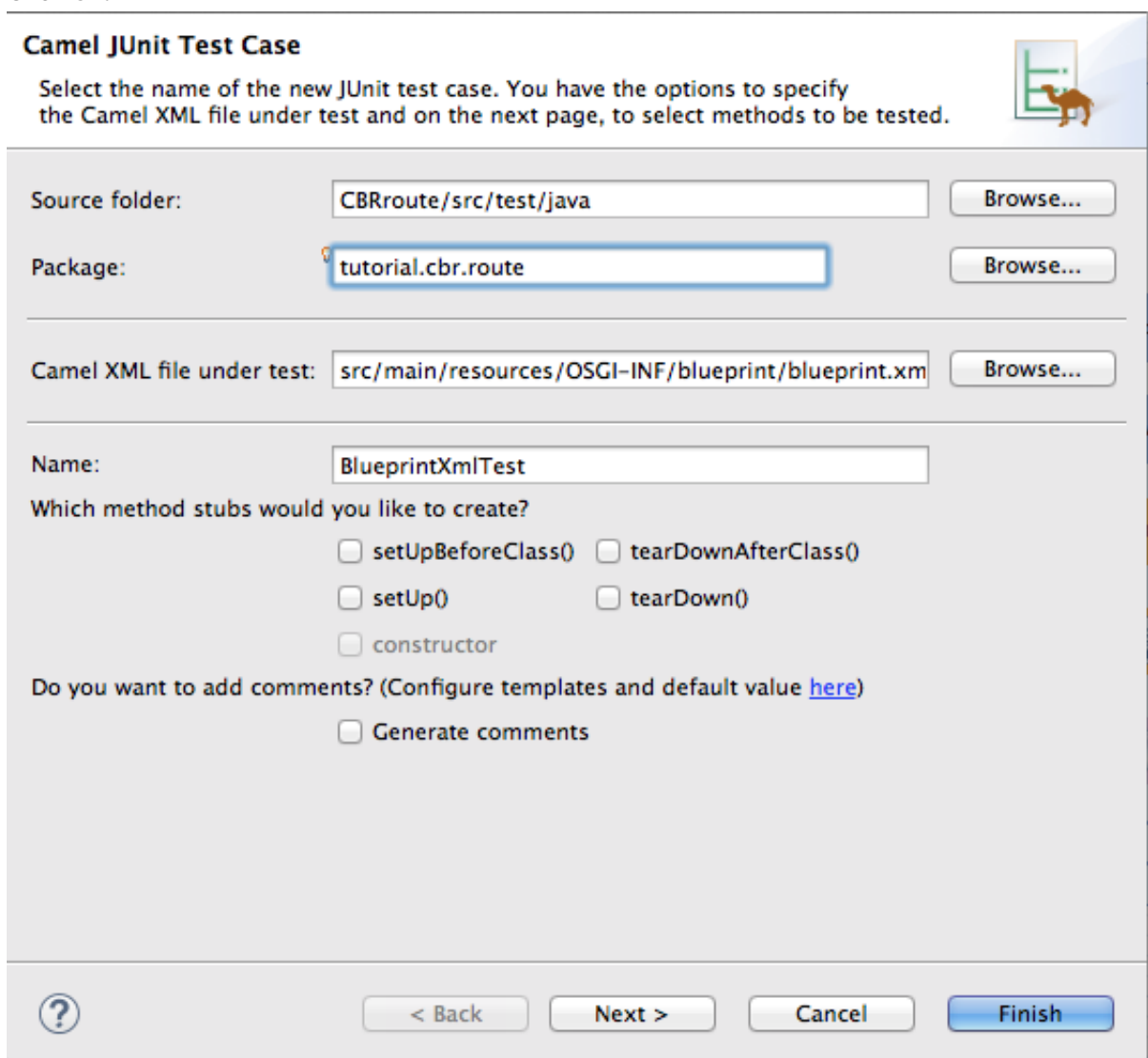
3. In the **Camel JUnit Test Case** wizard, make sure the **Source folder** field contains `CBRroute/src/test/java`. To find the proper folder, click  .



- In the **Package** field, enter `tutorial.cbr.route`. This is the package that will include the new test case.
- Next to the **Camel XML file under test** field, click **Browse** to open a file explorer configured to screen for XML files, and then select the **CBRRoute** project's `blueprint.xml` file:



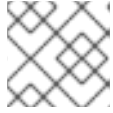
- Click **OK**.



**NOTE**

The **Name** field defaults to *BlueprintXmlTest*.

7. Click **Next** to open the **Test Endpoints** page.
8. By default, all endpoints are selected and will be included in the test case. 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 BlueprintXmlTest extends CamelBlueprintTestSupport {

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

    protected Object[] expectBodies = {
        "<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;
    @Produce(uri = "direct:OrderFulfillment")
    protected ProducerTemplate input2Endpoint;

    // 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;
    @EndpointInject(uri = "mock:output6")
    protected MockEndpoint output6Endpoint;
}
```

```

@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/invalidOrders").to(outputEndpoint);
            from("file:target/messages/GreatBritain").to(output3Endpoint);
            from("file:target/messages/Germany").to(output4Endpoint);
            from("file:target/messages/USA").to(output2Endpoint);
            from("file:target/messages/France").to(output5Endpoint);
        }
    });

    // Define some expectations

    // TODO Ensure expectations make sense for the route(s) we're testing
    outputEndpoint.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/blueprint.xml";
}
}

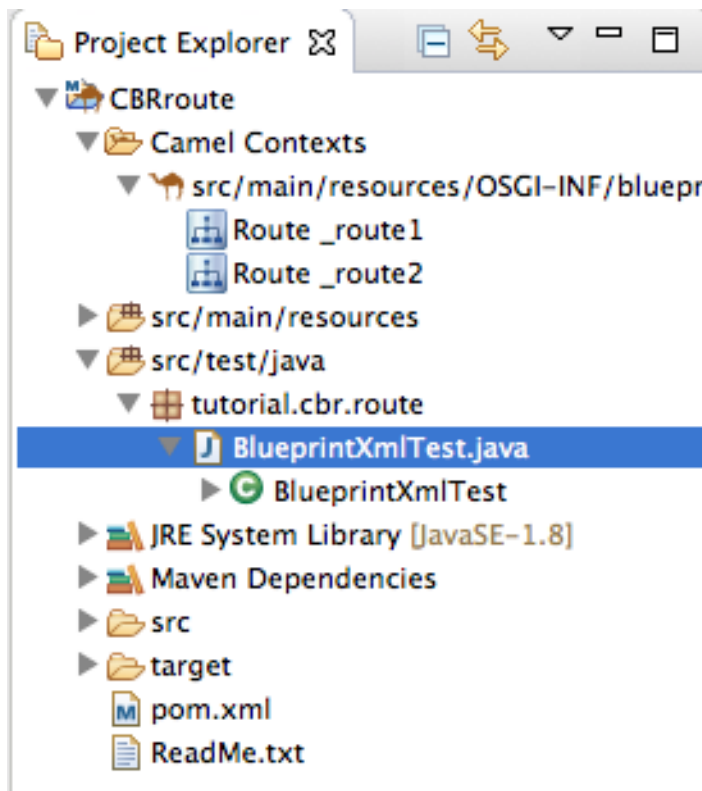
```

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 BlueprintXmlTest file”](#) and [the section called “Modifying the pom.xml file”](#).

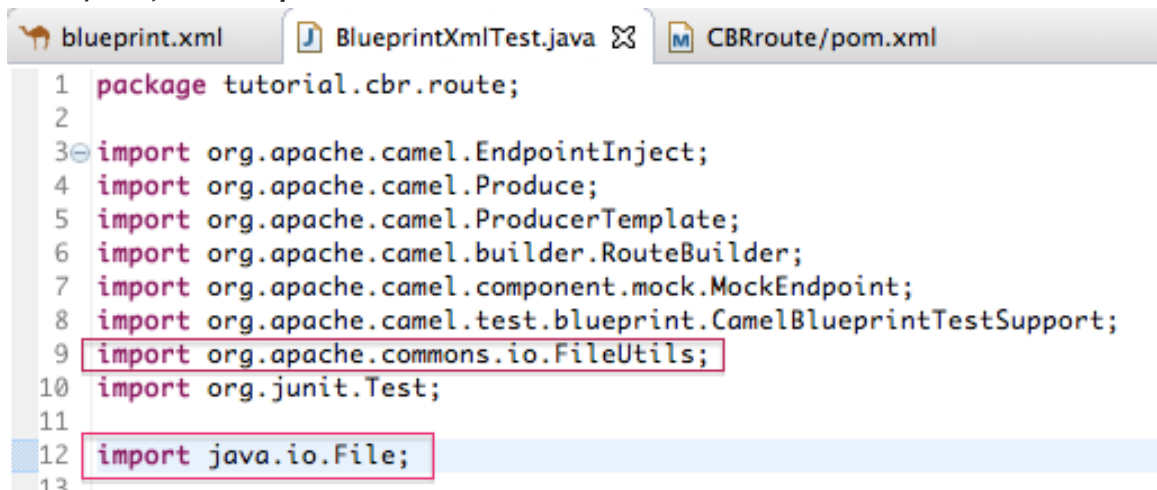
## MODIFYING THE BLUEPRINTXMLTEST FILE

You must modify the `BlueprintXmlTest.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 `BlueprintXmlTest.java` file:



2. Double-click `BlueprintXmlTest.java` to open the file in the tooling's Java editor.
3. In the Java editor, click the expand button next to `import org.apache.camel.EndpointInject;` to expand the list.
4. Add the two lines shown below. Adding the first line will cause an error that will be resolved when you update the `pom.xml` file as instructed in the next section.



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 BlueprintXmlTest 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 the 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"), "UTF-8");` shown below. These lines will indicate an error until you update the `pom.xml` file as instructed in the next section.

```
@Test
public void testCamelRoute() throws Exception {
    // Easy way of reading content of xml files to a String object. But you must add
    body1 = FileUtils.readFileToString(new File("src/data/message1.xml"), "UTF-8");
    body3 = FileUtils.readFileToString(new File("src/data/message3.xml"), "UTF-8");
    body5 = FileUtils.readFileToString(new File("src/data/message5.xml"), "UTF-8");
    body6 = FileUtils.readFileToString(new File("src/data/message6.xml"), "UTF-8");

    // Invalid orders
    body2 = FileUtils.readFileToString(new File("src/data/message2.xml"), "UTF-8");
    body4 = FileUtils.readFileToString(new File("src/data/message4.xml"), "UTF-8");
}
```

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 `outputEndpoint.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 `BlueprintXmlTest.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;

public class BlueprintXmlTest 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 the 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 String object,
        // but you must
        // add a dependency to the commons-io project to the CBRroute
        // pom.xml file
        body1 = FileUtils.readFileToString(new
        File("src/data/message1.xml"), "UTF-8");
        body3 = FileUtils.readFileToString(new
        File("src/data/message3.xml"), "UTF-8");
        body5 = FileUtils.readFileToString(new
        File("src/data/message5.xml"), "UTF-8");
        body6 = FileUtils.readFileToString(new
        File("src/data/message6.xml"), "UTF-8");

        // Invalid Orders
        body2 = FileUtils.readFileToString(new
        File("src/data/message2.xml"), "UTF-8");
        body4 = FileUtils.readFileToString(new
        File("src/data/message4.xml"), "UTF-8");
```

```

context.addRoutes(new RouteBuilder() {
    @Override
    public void configure() throws Exception {

from("file:target/messages/invalidOrders").to(outputEndpoint);

from("file:target/messages/GreatBritain").to(output3Endpoint);
    from("file:target/messages/Germany").to(output4Endpoint);
    from("file:target/messages/USA").to(output2Endpoint);
    from("file:target/messages/France").to(output5Endpoint);
    }
});

// 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/blueprint.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 tooling's XML 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>2.5</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>co</groupId>
  <artifactId>camel-blueprint</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <packaging>bundle</packaging>
  <name>Camel Blueprint Quickstart</name>
  <description>Empty Camel Blueprint Example</description>

  <licenses>
    <license>
      <name>Apache License, Version 2.0</name>
      <url>http://www.apache.org/licenses/LICENSE-2.0.html</url>
      <distribution>repo</distribution>
    </license>
  </licenses>

  <properties>
    <camel.version>2.18.1.redhat-000015</camel.version>
    <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>
    <version.maven-bundle-plugin>2.3.7<</version.maven-bundle-
plugin>
    <jboss.fuse.bom.version>6.3.0.redhat-
187</jboss.fuse.bom.version>
    <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
  </properties>

  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.jboss.fuse.bom</groupId>
        <artifactId>jboss-fuse-parent</artifactId>
        <version>${jboss.fuse.bom.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>

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



```

<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-blueprint</artifactId>
  <version>${camel.version}</version>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-api</artifactId>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-log4j12</artifactId>
</dependency>
<dependency>
  <groupId>log4j</groupId>
  <artifactId>log4j</artifactId>
</dependency>
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-test-blueprint</artifactId>
  <version>${camel.version}</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>commons-io</groupId>
  <artifactId>commons-io</artifactId>
  <version>2.5</version>
  <scope>test</scope>
</dependency>
</dependencies>

<repositories>
  <repository>
    <releases>
      <enabled>>true</enabled>
      <updatePolicy>never</updatePolicy>
    </releases>
    <snapshots>
      <enabled>>false</enabled>
    </snapshots>
    <id>fuse-public-repository</id>
    <name>FuseSource Community Release Repository</name>
  </repository>
  <repository>
    <releases>
      <enabled>>true</enabled>
      <updatePolicy>never</updatePolicy>
    </releases>
    <snapshots>
      <enabled>>false</enabled>
    </snapshots>
    <id>red-hat-ga-repository</id>
    <name>Red Hat GA Repository</name>
    <url>https://maven.repository.redhat.com/ga</url>
  </repository>
  <repository>
    <releases>
      <enabled>true</enabled>
      <updatePolicy>never</updatePolicy>
    </releases>
    <snapshots>
      <enabled>>false</enabled>
    </snapshots>
    <id>fuse-public-repository</id>
    <name>FuseSource Community Release Repository</name>
    <url>https://repo.fusesource.com/nexus/content/groups/public</url>
  </repository>
  <repository>
    <releases>
      <enabled>true</enabled>
      <updatePolicy>never</updatePolicy>
    </releases>
    <snapshots>
      <enabled>>false</enabled>
    </snapshots>
    <id>red-hat-ga-repository</id>
    <name>Red Hat GA Repository</name>
    <url>https://maven.repository.redhat.com/ga</url>
  </repository>
</repositories>

```

```

    </repository>
    <repository>
      <id>fuse-ea</id>
      <url>http://download.eng.brq.redhat.com/brewroot/repos/jb-
fuse-6.2-build/latest/maven</url>
    </repository>
    <repository>
      <id>redhat-ea-repository</id>
      <url>https://maven.repository.redhat.com/earlyaccess/all</url>
    </repository>
  </repositories>

  <pluginRepositories>
    <pluginRepository>
      <releases>
        <enabled>true</enabled>
        <updatePolicy>never</updatePolicy>
      </releases>
      <snapshots>
        <enabled>>false</enabled>
      </snapshots>
      <id>fuse-public-repository</id>
      <name>FuseSource Community Release Repository</name>

      <url>https://repo.fusesource.com/nexus/content/groups/public</url>
    </pluginRepository>
    <pluginRepository>
      <releases>
        <enabled>true</enabled>
        <updatePolicy>never</updatePolicy>
      </releases>
      <snapshots>
        <enabled>>false</enabled>
      </snapshots>
      <id>red-hat-ga-repository</id>
      <name>Red Hat GA Repository</name>
      <url>https://maven.repository.redhat.com/ga</url>
    </pluginRepository>
    <pluginRepository>
      <id>fuse-ea</id>
      <url>http://download.eng.brq.redhat.com/brewroot/repos/jb-
fuse-6.2-build/latest/maven</url>
    </pluginRepository>
    <pluginRepository>
      <id>redhat-ea-repository</id>
      <url>https://maven.repository.redhat.com/earlyaccess/all</url>
    </pluginRepository>
    <pluginRepository>
      <id>camelStaging</id>

      <url>https://repository.jboss.org/nexus/content/repositories/fusesou
rce_releases_external-2384</url>
    </pluginRepository>
  </pluginRepositories>

  <build>

```

```

<defaultGoal>install</defaultGoal>
<plugins>
  <plugin>
    <groupId>org.apache.felix</groupId>
    <artifactId>maven-bundle-plugin</artifactId>
    <version>${version.maven-bundle-plugin}</version>
    <extensions>true</extensions>
    <configuration>
      <instructions>
        <Bundle-SymbolicName>CBRRoute</Bundle-SymbolicName>
        <Bundle-Name>Empty Camel Blueprint Example [CBRRoute]
</Bundle-Name>
      </instructions>
    </configuration>
  </plugin>
  <plugin>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.5.1</version>
    <configuration>
      <source>1.8</source>
      <target>1.8</target>
    </configuration>
  </plugin>
  <plugin>
    <artifactId>maven-resources-plugin</artifactId>
    <version>3.0.1</version>
    <configuration>
      <encoding>UTF-8</encoding>
    </configuration>
  </plugin>
  <plugin>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-maven-plugin</artifactId>
    <version>${camel.version}</version>
    <configuration>
      <useBlueprint>true</useBlueprint>
    </configuration>
  </plugin>
</plugins>
</build>

</project>

```

## RUNNING THE JUNIT TEST

To run the test:

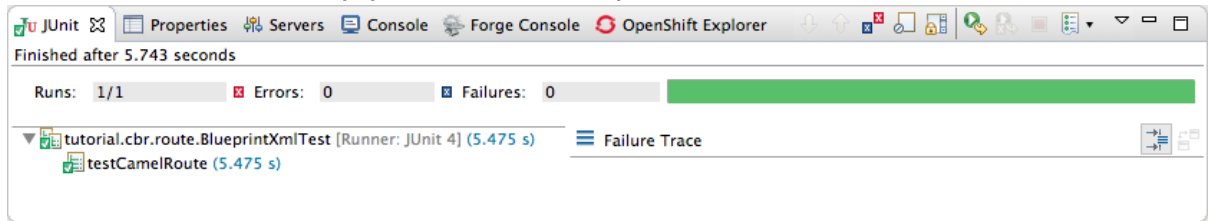
1. Switch to **JBoss** perspective to free up more workspace.
2. Select the project root, **CBRRoute**, in the **Project Explorer**.
3. Open the context menu.
4. Select menu:Run As[ > JUnit Test].



## NOTE

By default, the **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.)

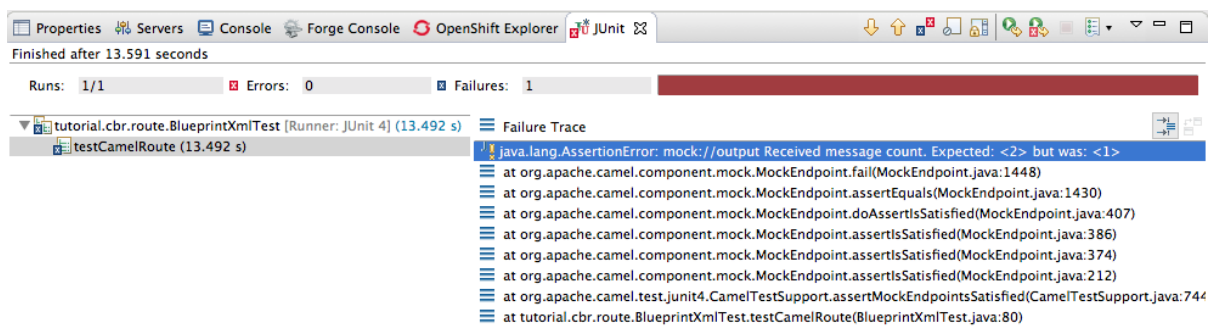
### 5. If the test runs successfully, you'll see something like this:



## NOTE

Sometimes the test fails the first time JUnit is run on a project. Rerunning the test usually results in a successful outcome.

When the test does fail, you'll see something like this:



## NOTE

JUnit will fail if your execution environment is not set to Java SE 8 or 7. 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 menu:Run As[ > > Run Configurations > > JRE > ]. Click the **Environments** button next to the **Execution environment** field to locate and select a Java SE 8 or 7 environment.

### 6. 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.

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 JBOSS FUSE

This tutorial walks you through the process of publishing an Apache Camel project to 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 6.3 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:

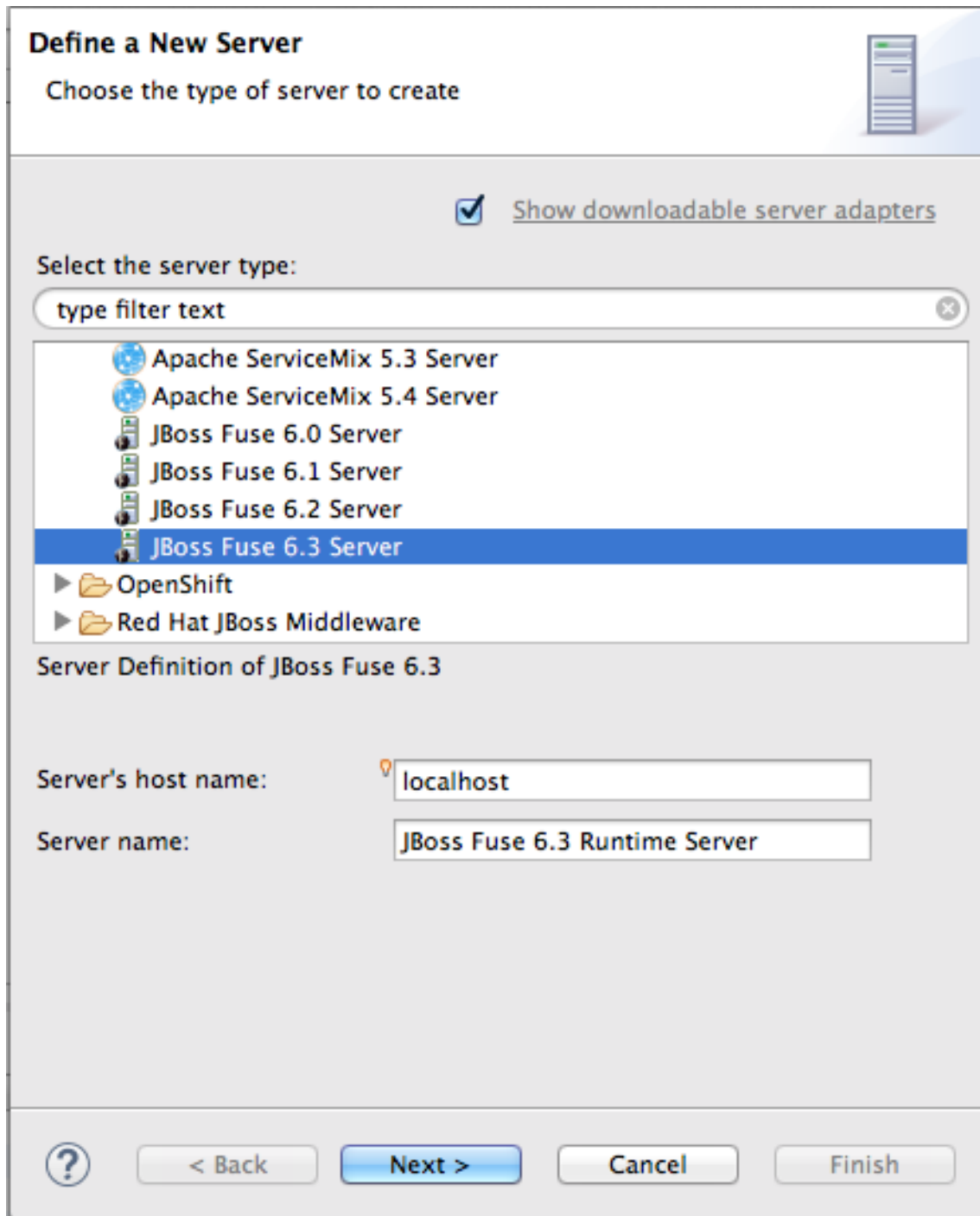
1. Open **Fuse Integration** perspective.
2. Click the **Servers** tab in the lower, right panel to open the **Servers** view.
3. Click the link **No servers are available. Click this link to create a new server...** to open the **Define a New Server** page.



#### NOTE

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


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



5. Click **JBoss Fuse 6.3 Server**.
6. Accept the defaults for **Server's host name** (*localhost*) and **Server name** (*JBoss Fuse 6.3 Runtime Server*), and then click **Next** to open the **JBoss Fuse Runtime** page:

## JBoss Fuse Runtime

Runtime definition for JBoss Fuse 6.3



Please point to a JBoss Fuse installation.

Name

Home Directory [Download and install runtime...](#)

Runtime JRE

Execution Environment:

Alternate JRE:



### NOTE

If you do not have JBoss Fuse 6.3 already installed, you can download it now using the **Download and install runtime...** link.



### NOTE

If you have already defined a JBoss Fuse 6.3 server, the tooling skips this page, and instead displays the configuration details page shown in [\[configDetails\]](#).

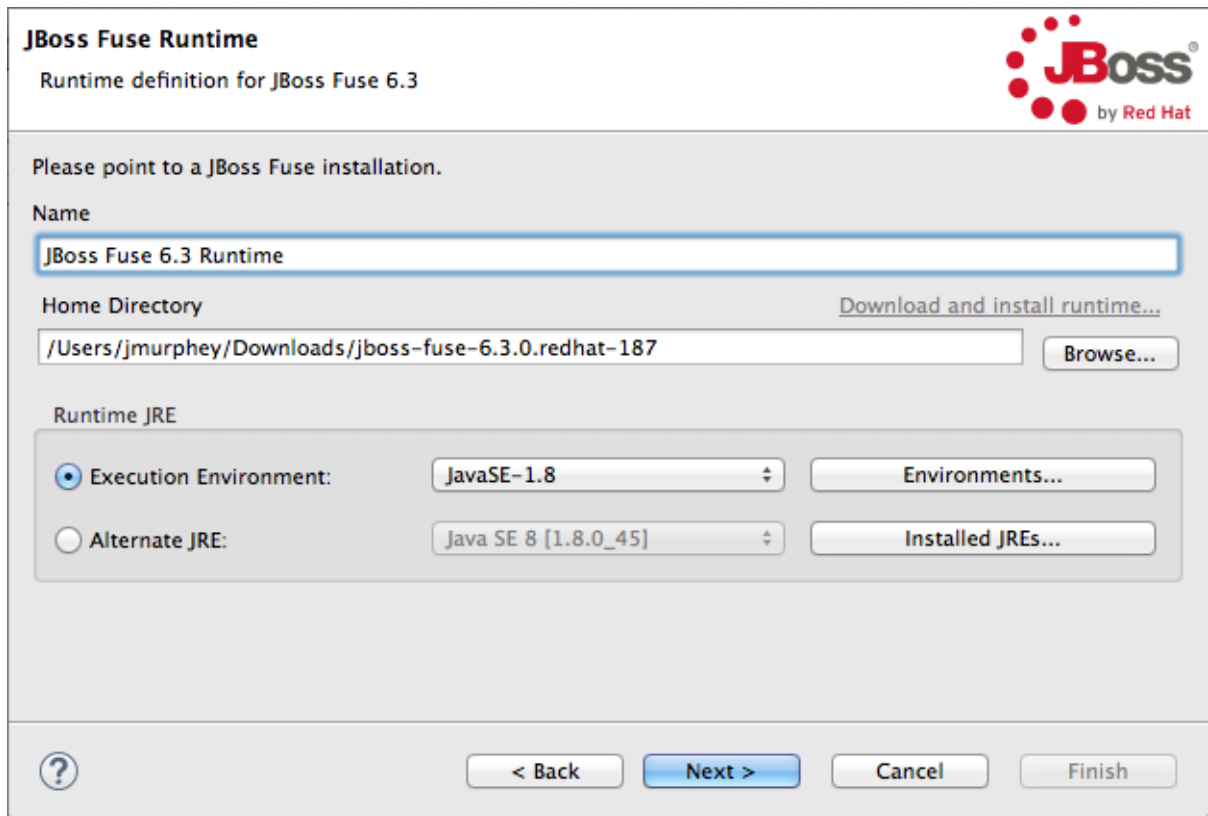
7. Accept the default for Name (*JBoss Fuse 6.3 Runtime*).
8. Click **Browse** next to the **Home Directory** field, to navigate to the JBoss Fuse 6.3 installation and select it.
9. Select the runtime JRE from the drop-down menu next to **Execution Environment**. Select either JavaSE-1.8 (recommended) or JavaSE-1.7. If neither appears as an option, click the **Environments...** button and select either version from the list.



### NOTE

The JBoss Fuse 6.3 server requires Java 8 (recommended) or Java 7. To select either version for the **Execution Environment**, you must have previously installed it.

10. Leave the **Alternate JRE** option as is.



**JBoss Fuse Runtime**  
Runtime definition for JBoss Fuse 6.3

Please point to a JBoss Fuse installation.

Name  
JBoss Fuse 6.3 Runtime

Home Directory [Download and install runtime...](#)  
/Users/jmurphey/Downloads/jboss-fuse-6.3.0.redhat-187 Browse...

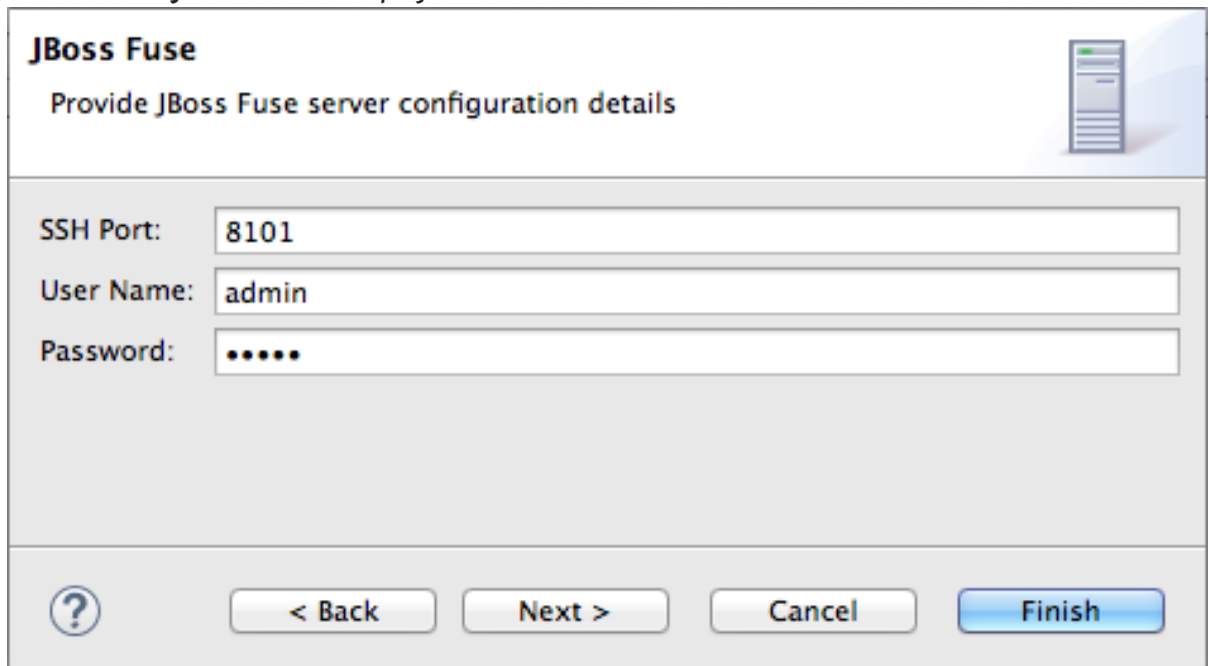
Runtime JRE

Execution Environment: JavaSE-1.8 Environments...

Alternate JRE: Java SE 8 [1.8.0\_45] Installed JREs...

? < Back Next > Cancel Finish

- Click **Next** to save the runtime definition for JBoss Fuse 6.3 Server and open the **JBoss Fuse server configuration details** page:



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

SSH Port: 8101

User Name: admin

Password: .....

? < Back Next > Cancel Finish

- 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.
- 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.



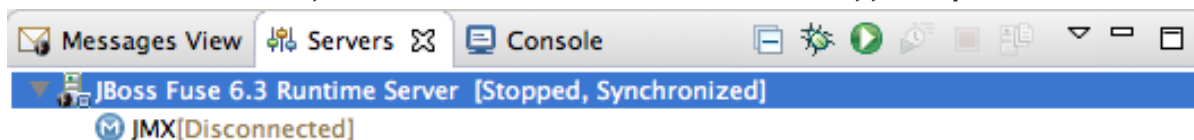
**NOTE**

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 [\[configDetails\]](#).

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 **Finish**.  
**JBoss Fuse 6.3 Runtime Server [stopped, Synchronized]** appears in the **Servers** view.
16. In the **Servers** view, expand **JBoss Fuse 6.3 Runtime Server [stopped, Synchronized]** :



**JMX[Disconnected]** appears as a node under **JBoss Fuse 6.3 Runtime Server [stopped, Synchronized]** entry.

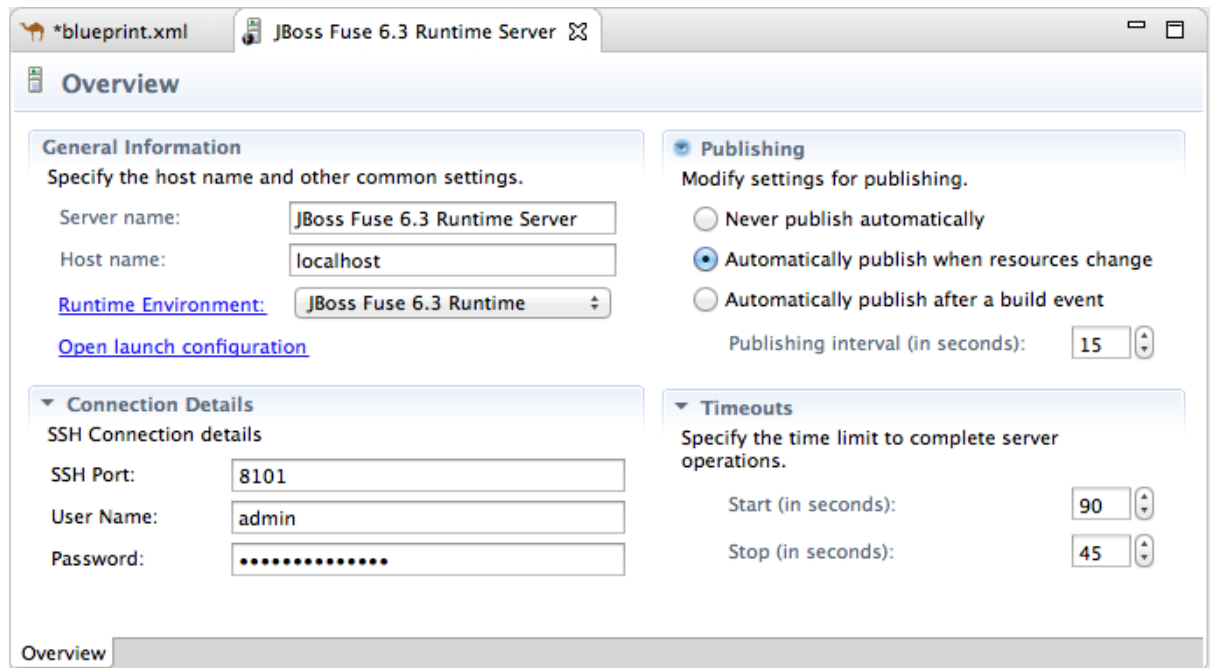
## CONFIGURING THE PUBLISHING OPTIONS

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 the **Servers** view, double-click the **JBoss Fuse 6.3 Runtime Server [stopped, Synchronized]** entry to display its overview:



2. On the server'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.

## NOTE

To configure manual publishing:

- Enable the **Never publish automatically** option on the server's **Overview** page.
- Disable the **If server started, publish changes immediately** option on the server's **Add and Remove** page.

Then to manually publish changes made to selective resources configured on the running server, use the **Full Publish** option on the resource's context menu in the **Servers** view. The **Incremental Publish** option is not supported and clicking it results in a full publish.

## STARTING THE RED HAT JBOSS FUSE SERVER

This section provides instructions for starting the Fuse server and then assigning the **CBRoute** module to it for immediate publishing.

1. In the **Servers** view, select **JBoss Fuse 6.3 Runtime Server** and click  to start it.

## IMPORTANT

A warning that the host identification has changed may appear. Click **yes** to replace the key **only** if the JBoss Fuse 6.3 server runtime is installed on the same machine where Red Hat JBoss Fuse Tooling is running! Otherwise click **no** and contact your system administrator.

2. Wait a few seconds for JBoss Fuse 6.3 Server to start up. When it does:

- The **Terminal** view displays the JBoss Fuse splash screen:

```

Terminal
SSH admin@localhost:8101 (9/27/16 9:41 PM)

  JBoss Fuse
  JBoss Fuse (6.3.0.redhat-187)
  http://www.redhat.com/products/jbossenterprisemiddleware/fuse/

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

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

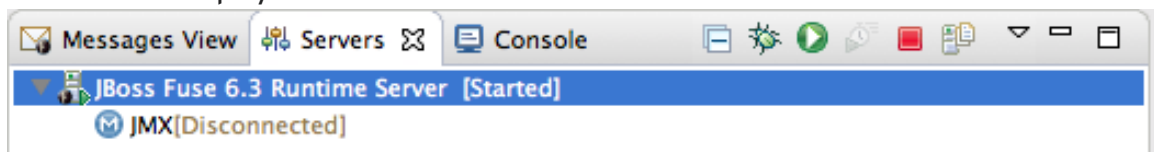
Create a new Fabric via 'fabric:create'
or join an existing Fabric via 'fabric:join [someUrls]'

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

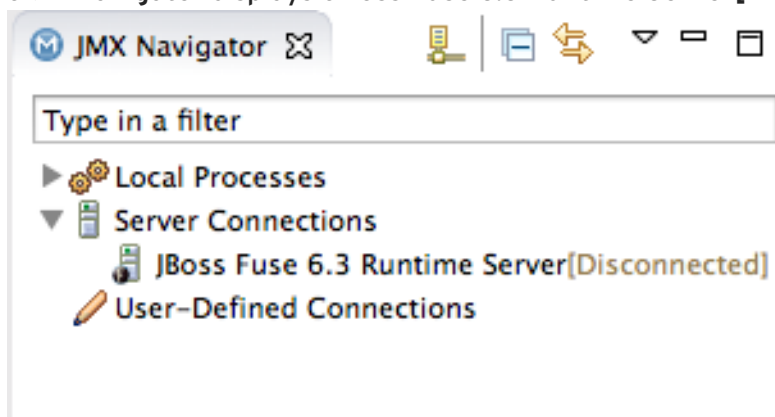
JBossFuse:admin@root>

```

- **Servers** view displays:



- **JMX Navigator** displays **JBoss Fuse 6.3 Runtime Server[Disconnected]**:



3. In the **Servers** view, right-click **JBoss Fuse 6.3 Runtime Server [Started]** to open the context menu.


4. Select **Add and Remove** to open the **Add and Remove** page:

### Add and Remove

Modify the resources that are configured on the server

Move resources to the right to configure them on the server

Available: Configured:

 CBRroute

Add >

< Remove

Add All >>

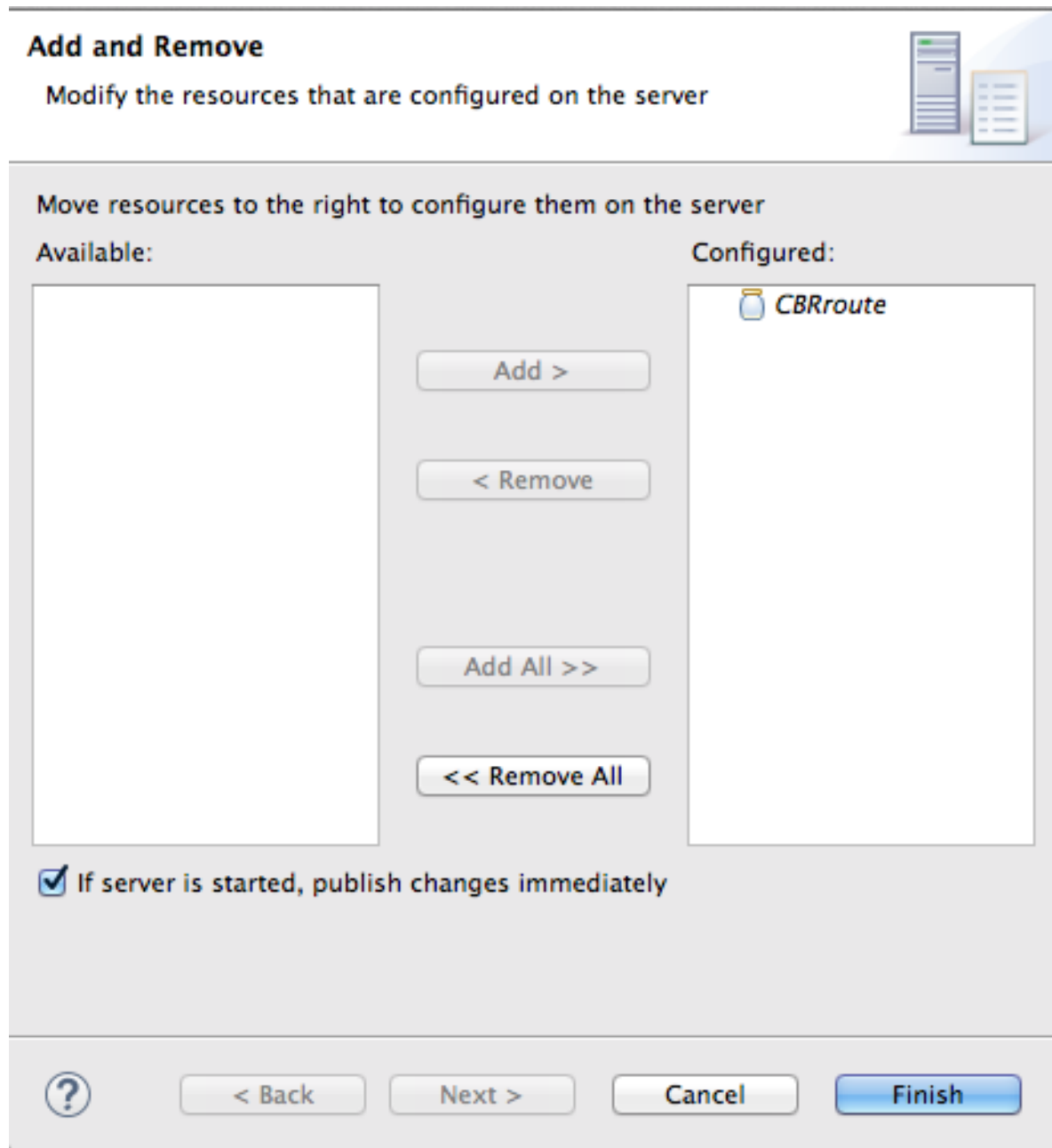
<< Remove All

If server is started, publish changes immediately

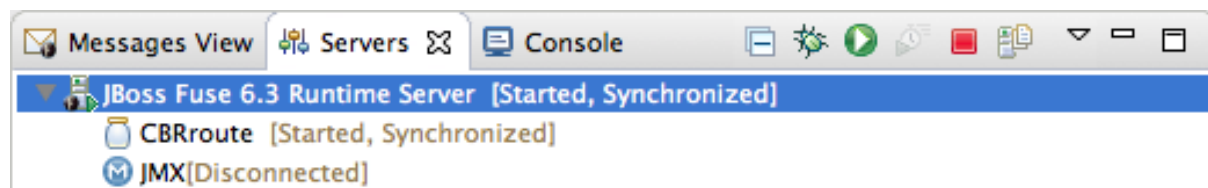
?< BackNext >CancelFinish

Make sure the option **If server is started, publish changes immediately** is checked.

5. Select **CBRroute** and click **Add** to assign it to the JBoss Fuse server;



6. Click **Finish**.



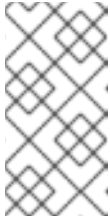
- **JBoss Fuse 6.3 Runtime Server [Started, Synchronized]**



#### NOTE

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

- **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**).

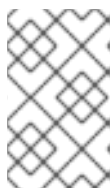
- **JMX[Disconnected]**

## CONNECTING TO THE JOSS FUSE 6.3 RUNTIME SERVER

When you connect to the **JBoss Fuse 6.3 Runtime Server**, you can see the published elements of your **CBRroute** project and interact with them. The instructions in this section will show a display such as the following:

Identifier	Symbolic Name	Version	State
283	jline	2.12.1.redhat-002	ACTIVE
284	io.hawt.hawtio-karaf-terminal	1.4.0.redhat-630187	ACTIVE
285	io.fabric8.support.support-core	1.2.0.redhat-630187	ACTIVE
286	io.fabric8.support.support-commands	1.2.0.redhat-630187	ACTIVE
287	io.fabric8.support.support-karaf	1.2.0.redhat-630187	ACTIVE
288	io.fabric8.support.support-webapp	1.2.0.redhat-630187	ACTIVE
289	io.fabric8.support.support-fabric8	1.2.0.redhat-630187	ACTIVE
290	io.hawt.hawtio-redhat-fuse-branding	1.4.0.redhat-630187	ACTIVE
291	org.apache.servicemix.specs.jsr311-api-1.1.1	2.7.0	ACTIVE
292	auth	0.0.0	ACTIVE
294	CBRroute	1.0.0.SNAPSHOT	ACTIVE

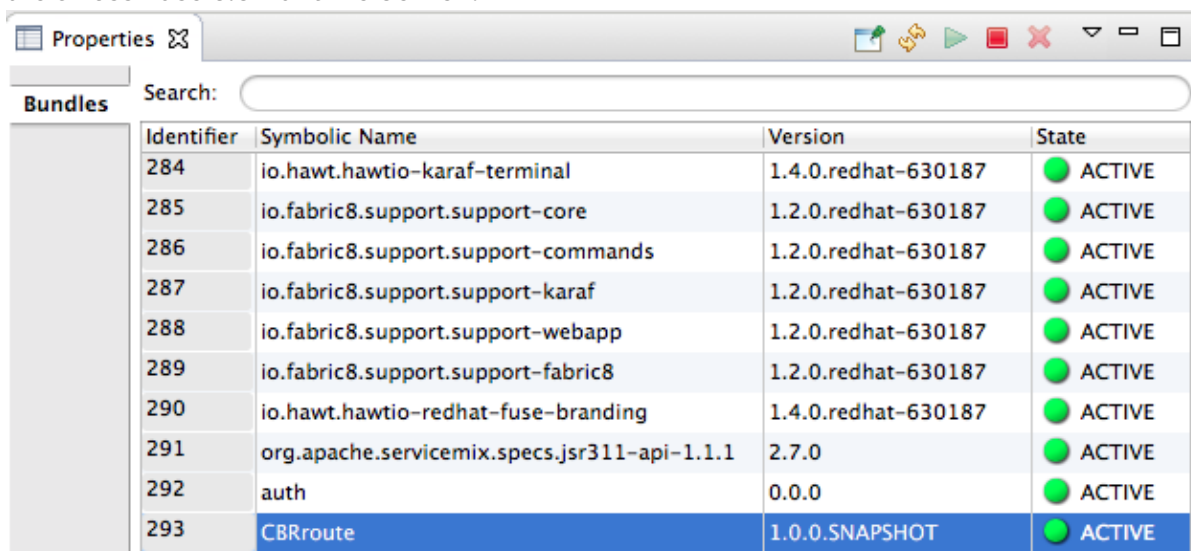
1. In the **Servers** view, double-click **JMX[Disconnected]** to connect to the runtime server.
2. Expand the **Camel** folder in the **Servers** view or **JMX Navigator** to expose the elements of the **CBRroute**.  
You can interact with the **CBRroute** routing context using either the **Servers** 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 **\_context1** node appears in **JMX Navigator** under **Server Connections** (or in the **Servers** view under **JMX[Connected]**), you can start tracing on it, as described in [Chapter 7, To Trace a Message Through a Route](#).

- Click the **Bundles** node to populate the **Properties** view with the list of bundles installed on the JBoss Fuse 6.3 Runtime Server :



Identifier	Symbolic Name	Version	State
284	io.hawt.hawtio-karaf-terminal	1.4.0.redhat-630187	ACTIVE
285	io.fabric8.support.support-core	1.2.0.redhat-630187	ACTIVE
286	io.fabric8.support.support-commands	1.2.0.redhat-630187	ACTIVE
287	io.fabric8.support.support-karaf	1.2.0.redhat-630187	ACTIVE
288	io.fabric8.support.support-webapp	1.2.0.redhat-630187	ACTIVE
289	io.fabric8.support.support-fabric8	1.2.0.redhat-630187	ACTIVE
290	io.hawt.hawtio-redhat-fuse-branding	1.4.0.redhat-630187	ACTIVE
291	org.apache.servicemix.specs.jsr311-api-1.1.1	2.7.0	ACTIVE
292	auth	0.0.0	ACTIVE
293	<b>CBRroute</b>	1.0.0.SNAPSHOT	ACTIVE

Start typing **CBRroute** in the **Properties** view's **Search** field to quickly determine whether your project's **CBRroute** bundle is included in the list. Note that it is the last bundle in the list, identified by its **Symbolic Name**, **CBRroute**, which is the name you gave your project when you created it.

## NOTE

Alternatively, you can issue the `osgi:list` command in the **Terminal** view to see a generated list of bundles installed on the JBoss Fuse server runtime. The tooling uses a different naming scheme for OSGi bundles displayed by the `osgi:list` command. In this case, the command returns **Empty Camel Blueprint Project [CBRroute]**, which appears at the end of the list of installed bundles.

In the `<build>` section of project's `pom.xml` file, you can find the bundle's symbolic name and its bundle name (OSGi) listed in the `maven-bundle-plugin` entry:

```
<build>
  <defaultGoal>install</defaultGoal>
  <plugins>
    <plugin>
      <groupId>org.apache.felix</groupId>
      <artifactId>maven-bundle-plugin</artifactId>
      <version>${version.maven-bundle-plugin}</version>
      <extensions>true</extensions>
      <configuration>
        <instructions>
          <Bundle-SymbolicName>CBRroute</Bundle-SymbolicName>
          <Bundle-Name>Empty Camel Blueprint Example [CBRroute]</Bundle-Name></instructions>
        </configuration>
      </plugin>
```

## UNINSTALLING THE CBRROUTE PROJECT

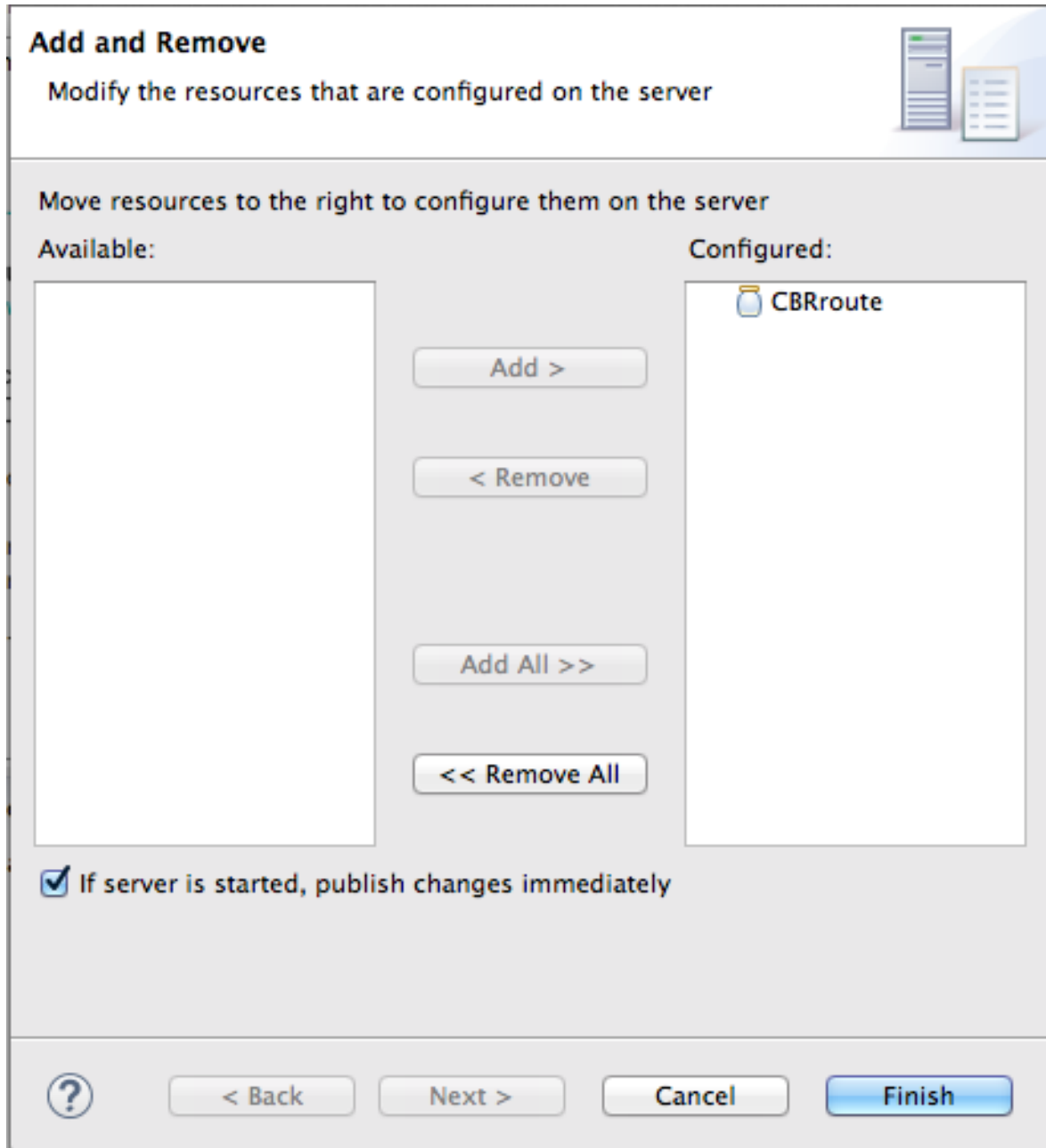
### NOTE

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



To remove the **CBRRoute** resource from the JBoss Fuse runtime server:

1. In the **Servers** view, right-click **JBoss Fuse 6.3 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 the **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 6.3 Runtime Server[Connected]** also disappears.



6. With the **Bundles** page displayed, start typing **CBRroute`** in the **Properties** view's **Search** field to verify that the bundle has been removed.