Red Hat Fuse 7.8

Tooling Tutorials

Tooling Tutorials

Last Updated: 2023-07-24

Red Hat Fuse 7.8 Tooling Tutorials

Tooling Tutorials

Legal Notice

Copyright © 2023 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, the Red Hat 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 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 Fuse Tooling to develop and test applications.

Table of Contents

CHAPTER 1. ABOUT THE FUSE TOOLING TUTORIALS 4
PREREQUISITES 4
OVERVIEW OF THE FUSE TOOLING TUTORIALS 4
ABOUT THE SAMPLE APPLICATION 5
ABOUT THE RESOURCE FILES 5

CHAPTER 2. SETTING UP YOUR ENVIRONMENT 6
GOALS 6
BEFORE YOUR BEGIN 6
CREATING A FUSE INTEGRATION PROJECT 6
SETTING COMPONENT LABELS TO DISPLAY ID VALUES 12
DOWNLOADING TEST MESSAGES FOR YOUR PROJECT 13
VIEWING THE TEST MESSAGES 14
NEXT STEPS 15

CHAPTER 3. DEFINING A ROUTE 16
GOALS 16
BEFORE YOU BEGIN 16
CONFIGURING THE SOURCE ENDPOINT 16
CONFIGURING THE SINK ENDPOINT 17
NEXT STEPS 19

CHAPTER 4. RUNNING A ROUTE 20
GOALS 20
PREREQUISITES 20
RUNNING THE ROUTE 20
VERIFYING THE ROUTE 21
NEXT STEPS 22

CHAPTER 5. ADDING A CONTENT-BASED ROUTER 23
GOALS 23
PREREQUISITES 23
ADDING AND CONFIGURING A CONTENT-BASED ROUTER 23
ADDING AND CONFIGURING LOGGING 27
ADDING AND CONFIGURING MESSAGE HEADERS 29
ADDING AND CONFIGURING A BRANCH TO HANDLE VALID ORDERS 32
VERIFYING THE CBR 38
NEXT STEPS 39

CHAPTER 6. ADDING ANOTHER ROUTE TO THE ROUTING CONTEXT 41
GOALS 41
PREREQUISITES 41
RECONFIGURING THE EXISTING ROUTE'S ENDPOINT 41
ADDING THE SECOND ROUTE 42
CONFIGURING A CHOICE BRANCH TO PROCESS USA ORDERS 43
CONFIGURING AN OTHERWISE BRANCH TO PROCESS GERMANY ORDERS 50
VERIFYING THE SECOND ROUTE 54
NEXT STEPS 58

CHAPTER 7. DEBUGGING A ROUTING CONTEXT 59
GOALS 59
PREREQUISITES 59
SETTING BREAKPOINTS 59

STEPPING THROUGH THE ROUTING CONTEXT 60
CHANGING THE VALUE OF A VARIABLE 64
NARROWING THE CAMEL DEBUGGER'S FOCUS 68
VERIFYING THE EFFECT OF CHANGING A MESSAGE VARIABLE VALUE 70
NEXT STEPS 71

CHAPTER 8. TRACING A MESSAGE THROUGH A ROUTE 72
GOALS 72
PREREQUISITES 72
SETTING UP YOUR FUSE INTEGRATION PERSPECTIVE 72
STARTING MESSAGE TRACING 75
DROPPING MESSAGES ON THE RUNNING ZOOORDERAPP PROJECT 78
CONFIGURING MESSAGES VIEW 79
STEPPING THROUGH MESSAGE TRACES 80
NEXT STEPS 83

CHAPTER 9. TESTING A ROUTE WITH JUNIT 84
OVERVIEW 84
GOALS 84
PREREQUISITES 84
CREATING THE SRC/TEST FOLDER 85
CREATING THE JUNIT TEST CASE 87
MODIFYING THE BLUEPRINTXMLTEST FILE 91
MODIFYING THE POM.XML FILE 95
RUNNING THE JUNIT TEST 95
FURTHER READING 96
NEXT STEPS 96

CHAPTER 10. PUBLISHING YOUR PROJECT TO RED HAT FUSE 97
GOALS 97
PREREQUISITES 97
DEFINING A RED HAT FUSE SERVER 97
CONFIGURING THE PUBLISHING OPTIONS 101
CONNECTING TO THE RUNTIME SERVER 106
UNINSTALLING THE ZOOORDERAPP PROJECT 107

CHAPTER 1. ABOUT THE FUSE TOOLING TUTORIALS

The Red Hat Fuse Tooling tutorials provide a hands-on introduction to using the Fuse Tooling to develop, run, test, and deploy an Apache Camel application.

PREREQUISITES

Before you begin, you should be familiar with the following software:

- Apache Camel
- Apache Maven

OVERVIEW OF THE FUSE TOOLING TUTORIALS

Here is a summary of the tutorials and what you accomplish in each one:

- **Chapter 2, *Setting up your environment***
  Create a Fuse Integration project and set up the tutorial resource files (example messages and routing context files). When you create a project, it auto-creates a routing context and a preliminary route.
- **Chapter 3, *Defining a Route***
  Define the endpoints for a simple route that retrieves messages from a folder and copies them to another folder.
- **Chapter 4, *Running a Route***
  View the test messages. Run the route and verify that it works by seeing that the test messages were copied from the source folder to the target folder.
- **Chapter 5, *Adding a Content-Based Router***
  Add a content-based router that filters the messages and copies them to different target folders based on content in the messages.
- **Chapter 6, *Adding another route to the routing context***
  Add another route that further filters the messages and copies them to different target folders based on content in the messages.
- **Chapter 7, *Debugging a routing context***
  Use the Camel debugger to set breakpoints and then step through a route to examine route and message variables.
- **Chapter 8, *Tracing a message through a route***
  Drop messages onto the route and track them through all route nodes.
- **Chapter 9, *Testing a route with JUnit***
  Create a JUnit test case for the route and then test the route.
- **Chapter 10, *Publishing your project to Red Hat Fuse***
  Walk through the process of publishing an Apache Camel project to Red Hat Fuse: define a local server, configure publishing options, start the server, publish the project, connect to the server, and verify that the project was successfully built and published.

For more details on Fuse Tooling features, see the Tooling User Guide.

ABOUT THE SAMPLE APPLICATION

The sample application that you build in the Fuse Tooling tutorials simulates a simple order application for zoos to order animals. Sample XML messages are provided - each XML message includes customer information (the name, city, and country of the zoo) and order information (the type and number of animals requested, and the maximum number of animals allowed).

Using the Fuse Tooling, you create a Blueprint project that takes incoming sample messages, filters them based on their content (valid versus invalid orders), and then further sorts the valid orders by the location (country) of the zoo. In the later tutorials, you use the sample application to debug a routing context, trace a message through a route, test a route with JUnit, and finally to publish a Fuse project.

ABOUT THE RESOURCE FILES

Each tutorial builds upon the previous one. The code generated by one tutorial is the starting point for the next tutorial so that you can complete the tutorials in sequence. Alternately, after you complete the first tutorial, you can do any other tutorial out of sequence by using one of the provided context files as a starting point.

The tutorials rely on resource files provided in the **Fuse-tooling-tutorials-jbds-10.3.zip** file located here. This zip file contains two folders:

Messages

This folder contains six message files named **message1.xml**, **message2.xml**, ... , **message6.xml**. In the first tutorial, Chapter 2, *Setting up your environment*, you create the directory in which to store these message files and you also view their contents. You need these message files for all tutorials.

blueprintContexts

This folder contains three routing context files:

- **Blueprint1.xml** - This is the solution routing context resulting from completing the Chapter 3, *Defining a Route* tutorial. You can use it as the starting point for the following tutorials:
  - Chapter 4, *Running a Route*
  - Chapter 5, *Adding a Content-Based Router*
- **Blueprint2.xml** - This is the solution context file for the Chapter 5, *Adding a Content-Based Router* tutorial. You can use **blueprint2.xml** as the starting point for the Chapter 6, *Adding another route to the routing context* tutorial.
- **Blueprint3.xml** - This is the solution context file for the Chapter 6, *Adding another route to the routing context* tutorial. You can use **blueprint3.xml** as the starting point for these tutorials:
  - Chapter 7, *Debugging a routing context*
  - Chapter 8, *Tracing a message through a route*
  - Chapter 9, *Testing a route with JUnit*
  - Chapter 10, *Publishing your project to Red Hat Fuse*

CHAPTER 2. SETTING UP YOUR ENVIRONMENT

This tutorial walks you through the process of creating a Fuse Integration project. The project includes an initial route and a default CamelContext. A route is a chain of processors through which a message travels. A CamelContext is a single routing rule base that defines the context for configuring routes, and specifies the policies to use during message exchanges between endpoints (message sources and targets).

You must complete this tutorial before you follow any of the other tutorials.

GOALS

In this tutorial you complete the following tasks:

- Create a Fuse Integration project
- Download test messages (XML files) for your project
- View the test messages

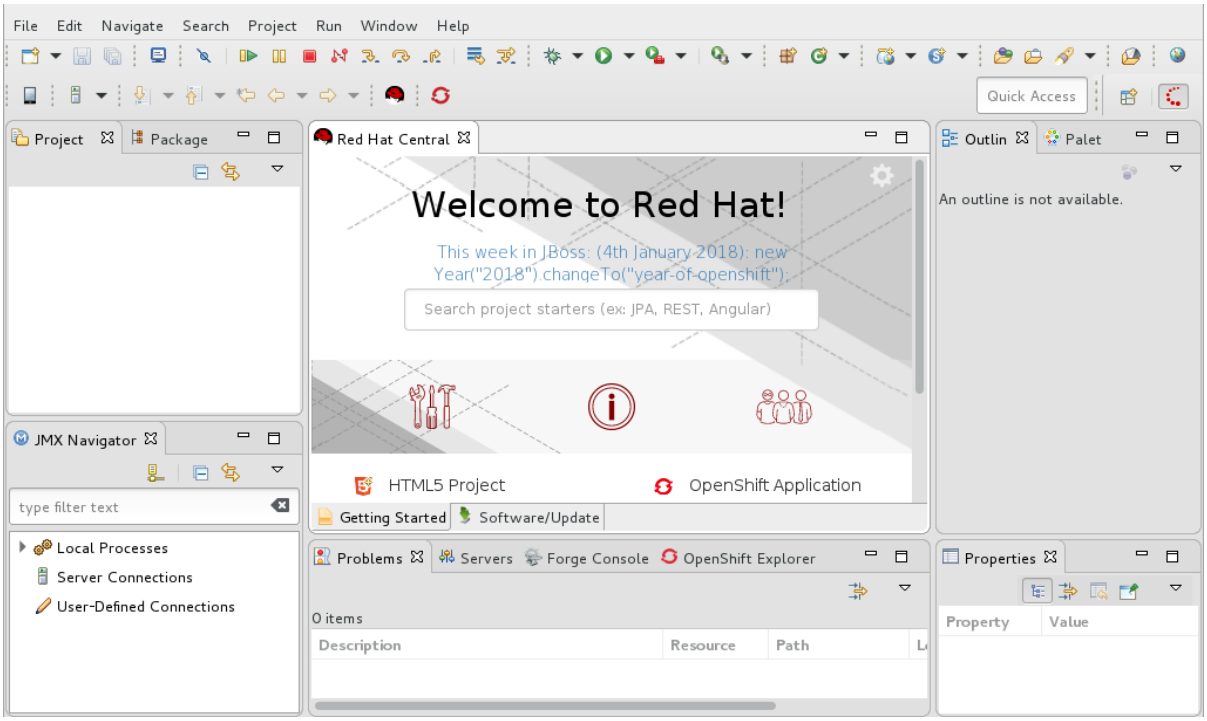
BEFORE YOUR BEGIN

Before you can set up a Fuse Integration project, you must install Red Hat CodeReady Studio with Fuse Tooling. For information on how to install CodeReady Studio, go to the Red Hat customer portal for the installation guide for your platform.

Before you can follow the steps in the Chapter 10, *Publishing your project to Red Hat Fuse* tutorial, you must install Java 8.

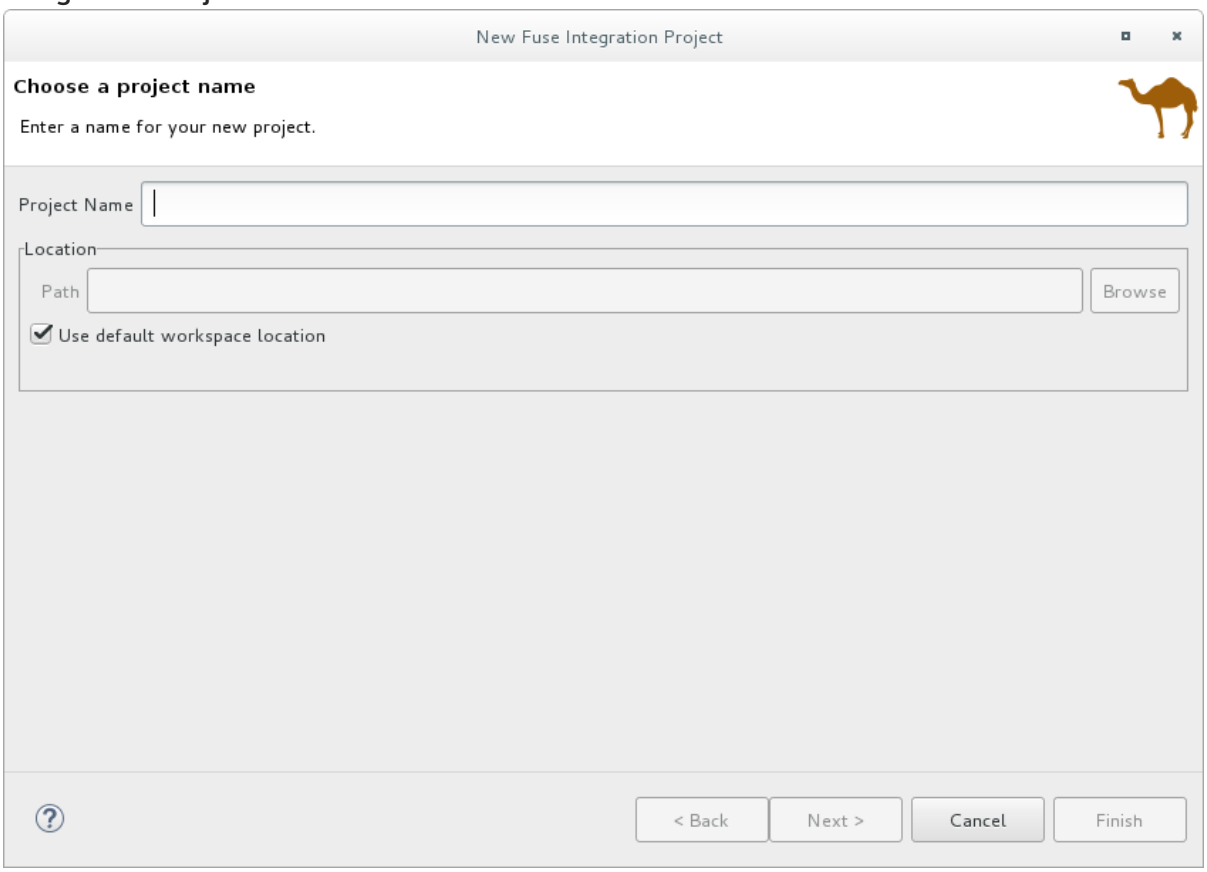
CREATING A FUSE INTEGRATION PROJECT

1. Open Red Hat CodeReady Studio.
   When you start CodeReady Studio for the first time, it opens in the **JBoss** perspective:



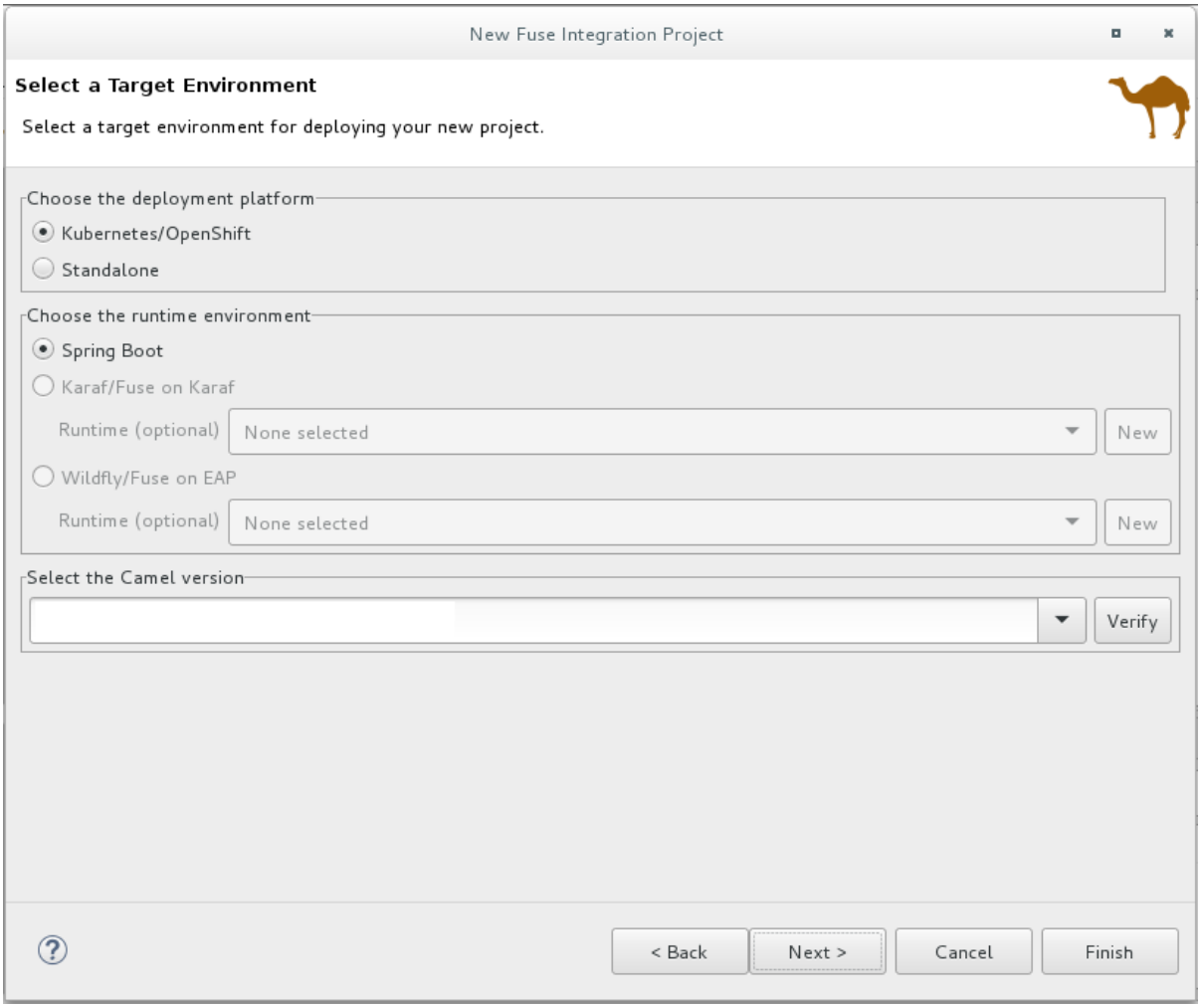
Otherwise, it opens in the perspective that you were using in your previous CodeReady Studio session.

2. From the menu , select **File → New → Fuse Integration Project** to open the **New Fuse Integration Project** wizard:



3. In the **Project Name** field, enter **ZooOrderApp**.
Leave the **Use default workspace location** option checked.

4. Click **Next** to open the **Select a Target Runtime** page:

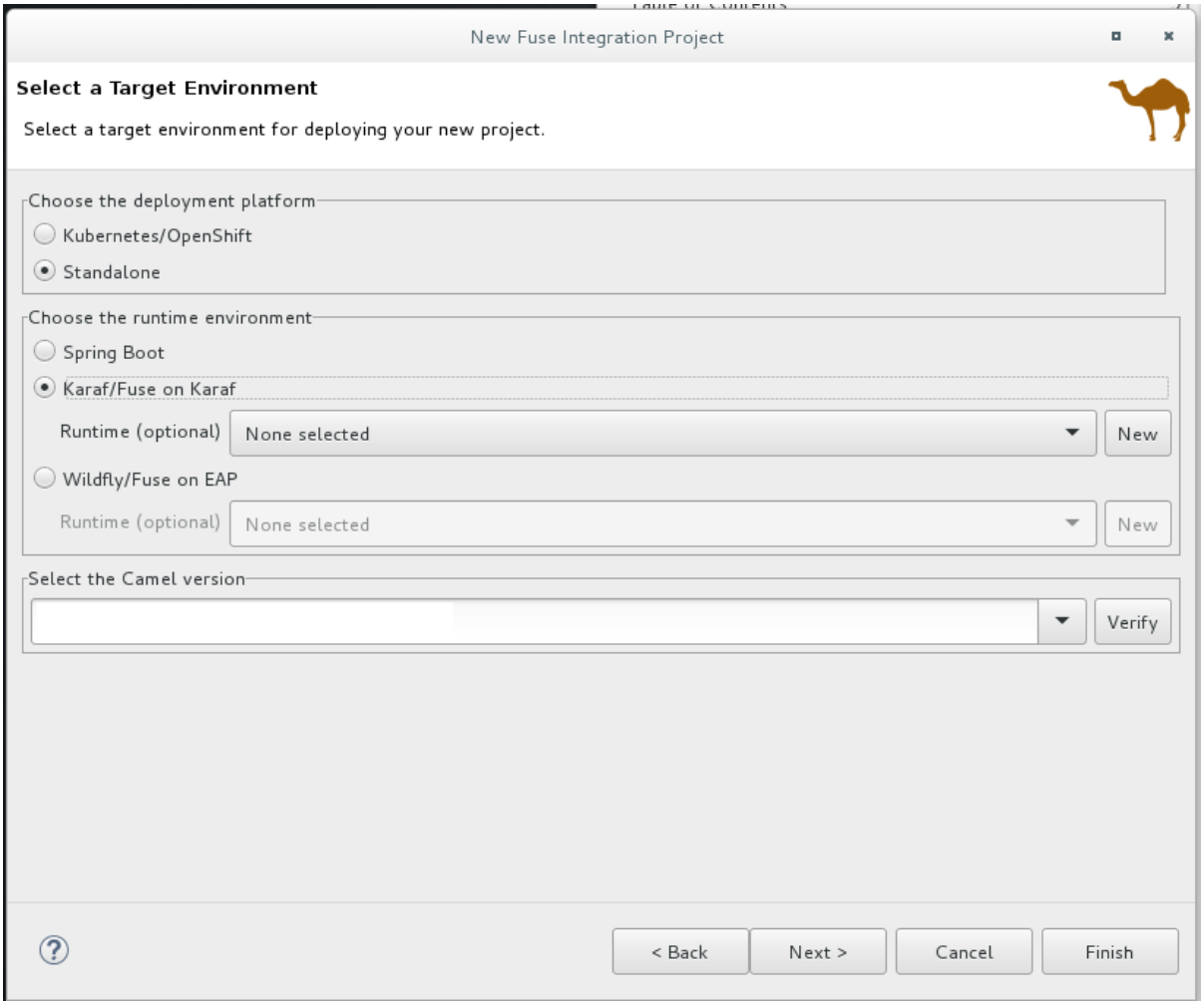


5. Select **Standalone** for the deployment platform.
6. Choose **Karaf/Fuse on Karaf** and accept **None selected** for the runtime.

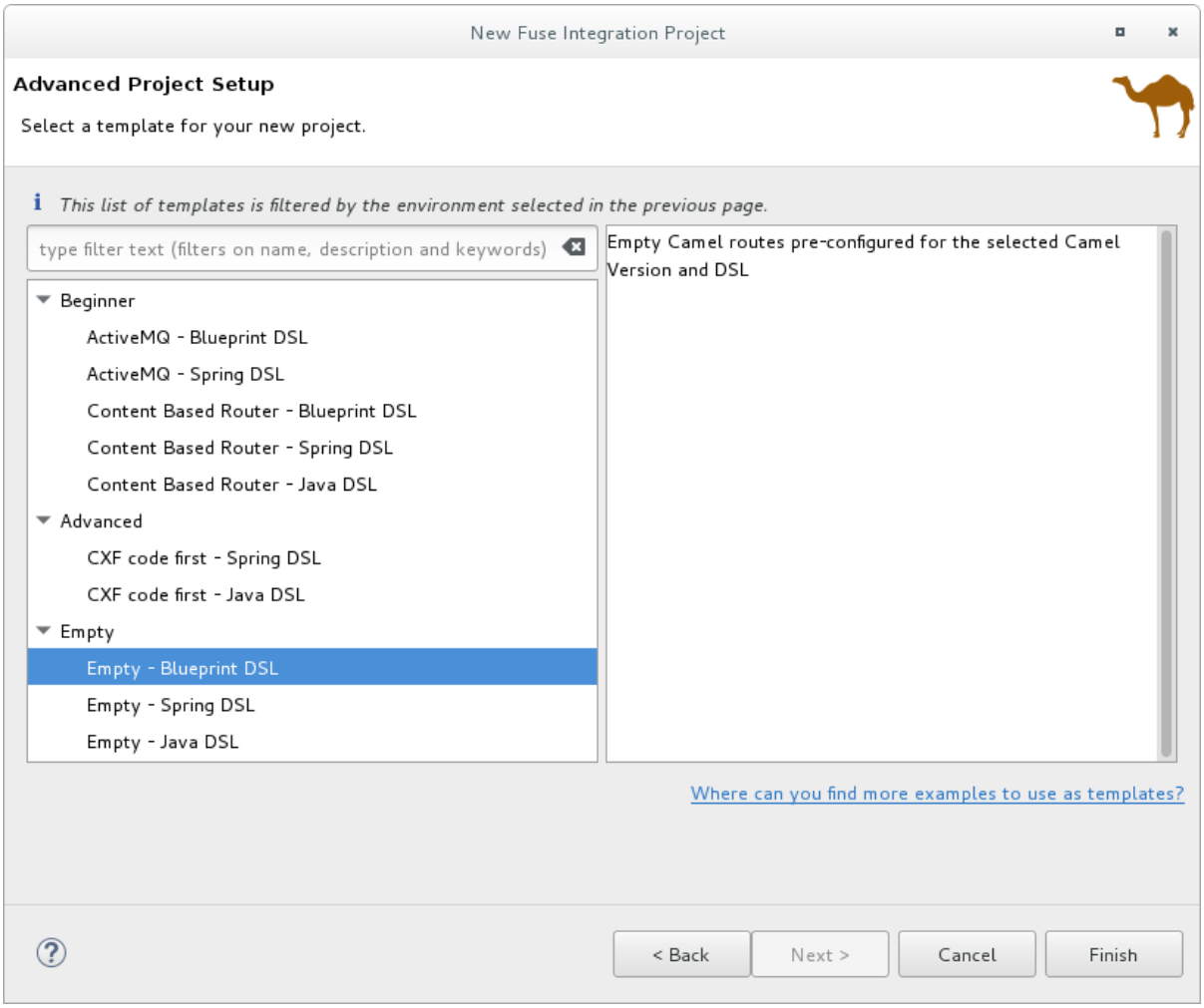
NOTE

You add the runtime later in the Chapter 10, *Publishing your project to Red Hat Fuse* tutorial .

7. Accept the default Apache **Camel version**.

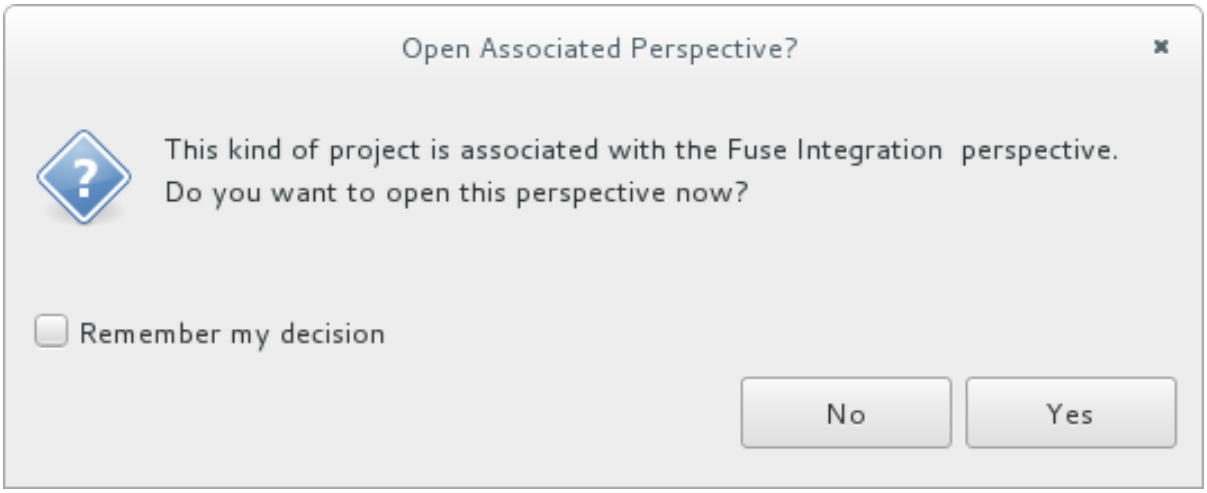


8. Click **Next** to open the **Advanced Project Setup** page, and then select the **Empty - Blueprint DSL** template:

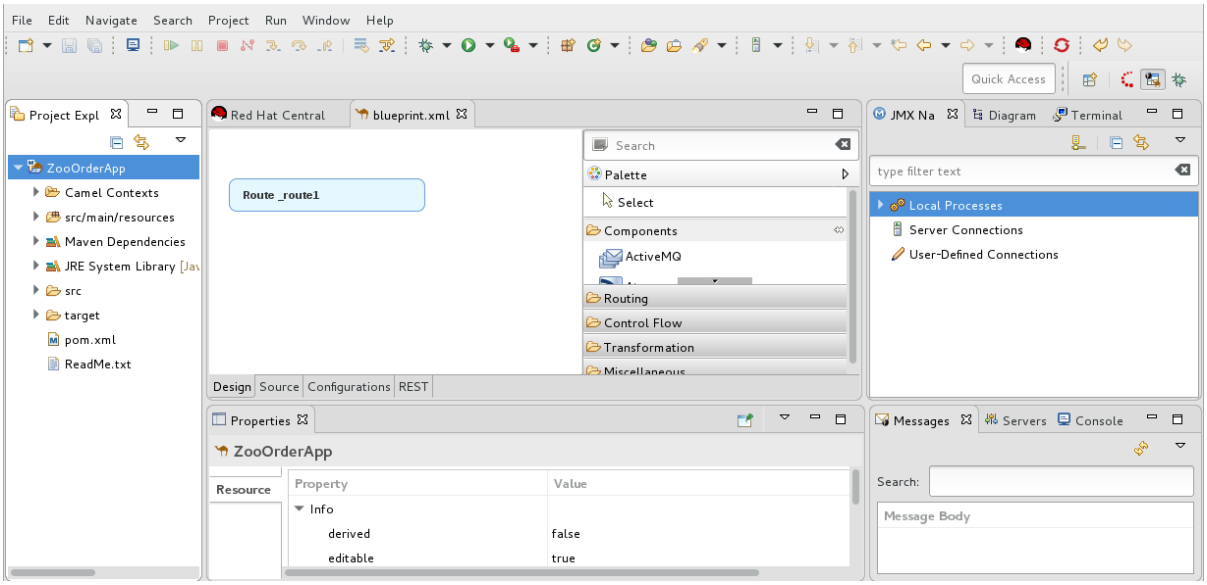


9. Click **Finish**.
   Fuse Tooling starts downloading-from the Maven repository-all of the files that it needs to build the project, and then it adds the new project to the **Project Explorer** view.

   If CodeReady Studio is not already showing the **Fuse Integration** perspective, it asks whether you want to switch to it now:

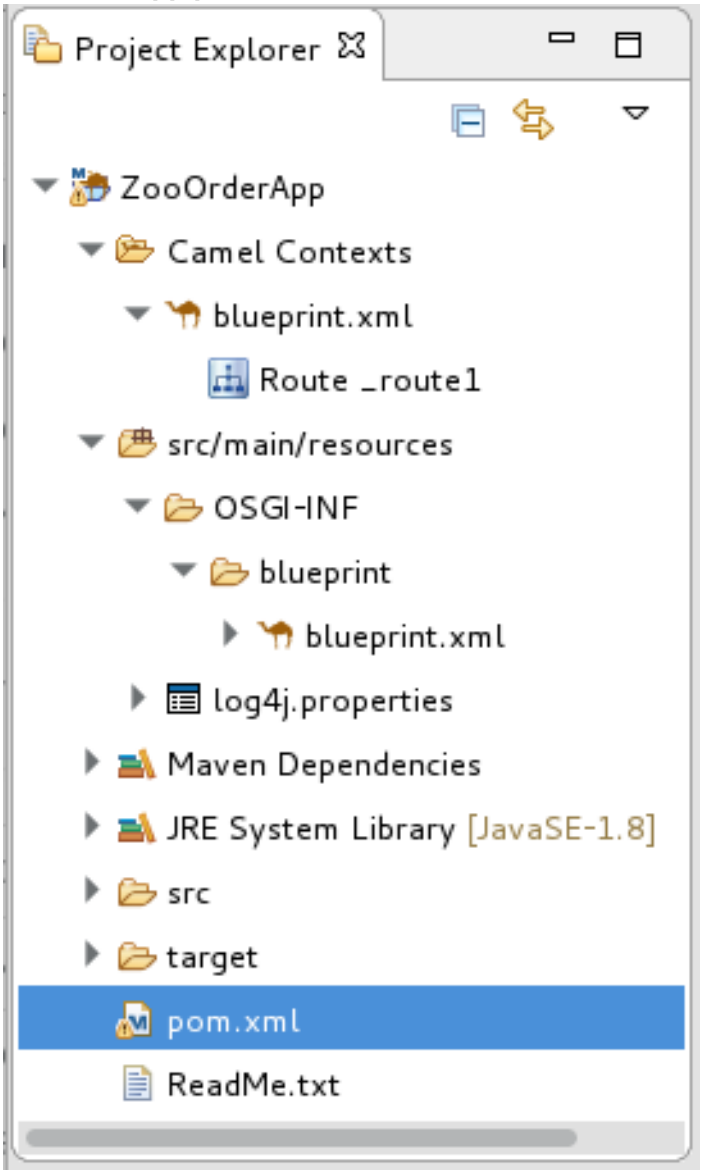


10. Click **Yes**.
    The new **ZooOrderApp** project opens in the **Fuse Integration** perspective:



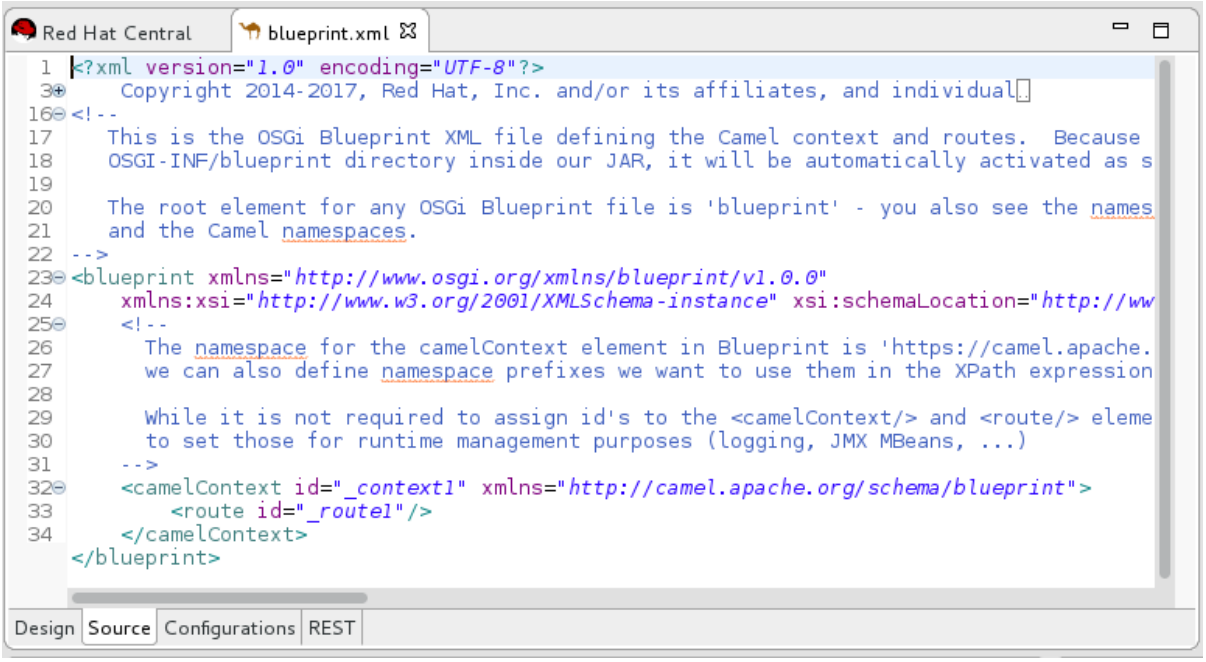
The **ZooOrderApp** project contains all of the files that you need to create and run routes, including:

- **ZooOrderApp/pom.xml** – A Maven project file.



- **ZooOrderApp/src/main/resources/OSGI-INF/blueprint/blueprint.xml** – A Blueprint XML file that contains a Camel routing context and an initial empty route.

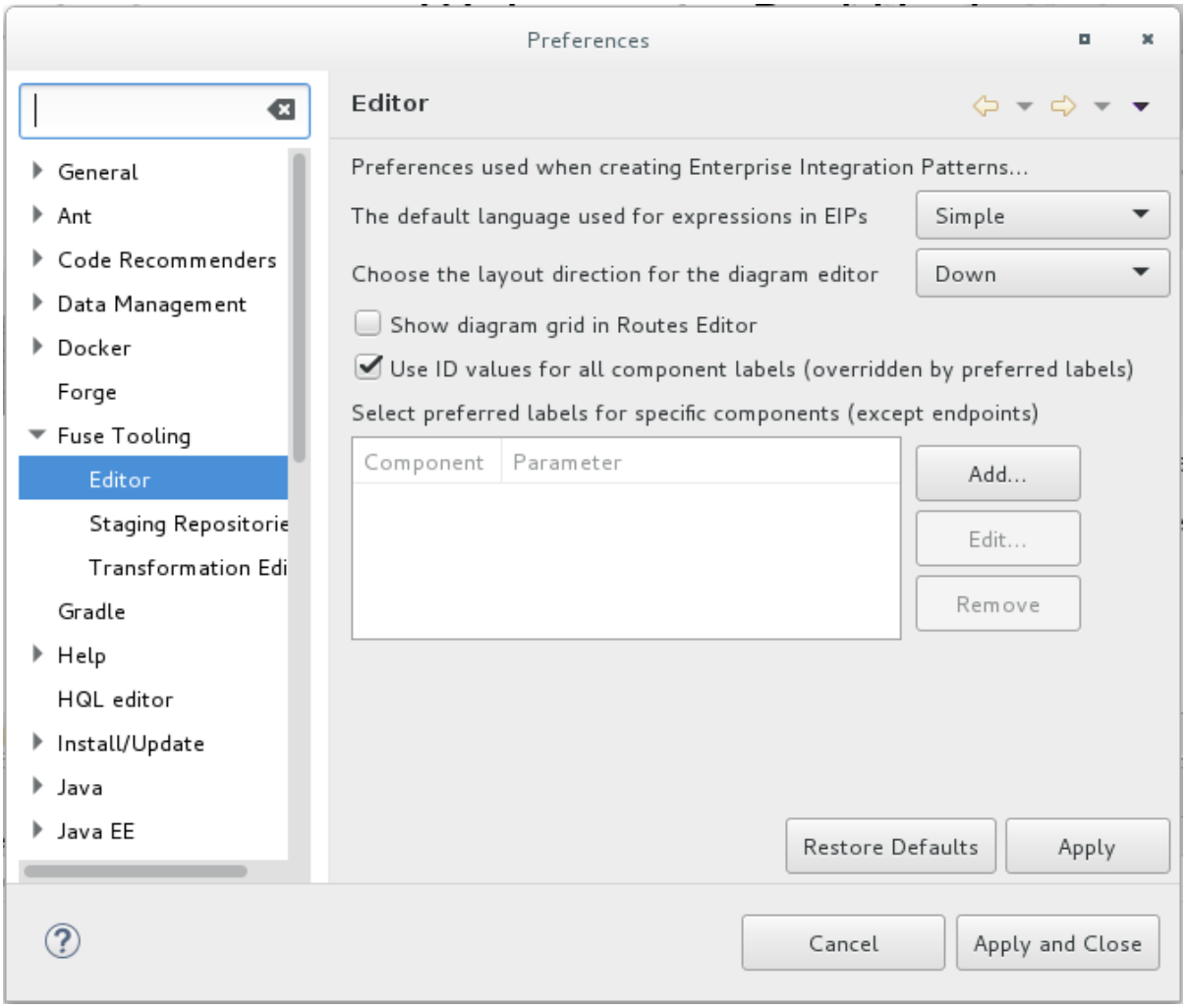
11. To view the preliminary routing context, open the **blueprint.xml** file in the Editor view, and then click the **Source** tab.



SETTING COMPONENT LABELS TO DISPLAY ID VALUES

To ensure that the labels of the patterns and components that you place on the Design canvas are the same as the labels shown in the Tooling Tutorials:

1. Open the Editor preferences page:
   - On Linux and Windows machines, select **Windows → Preferences → Fuse Tooling → Editor**.
   - On OS X, select **CodeReady Studio → Preferences → Fuse Tooling → Editor**.
2. Check the **Use ID values for all component labels** option.



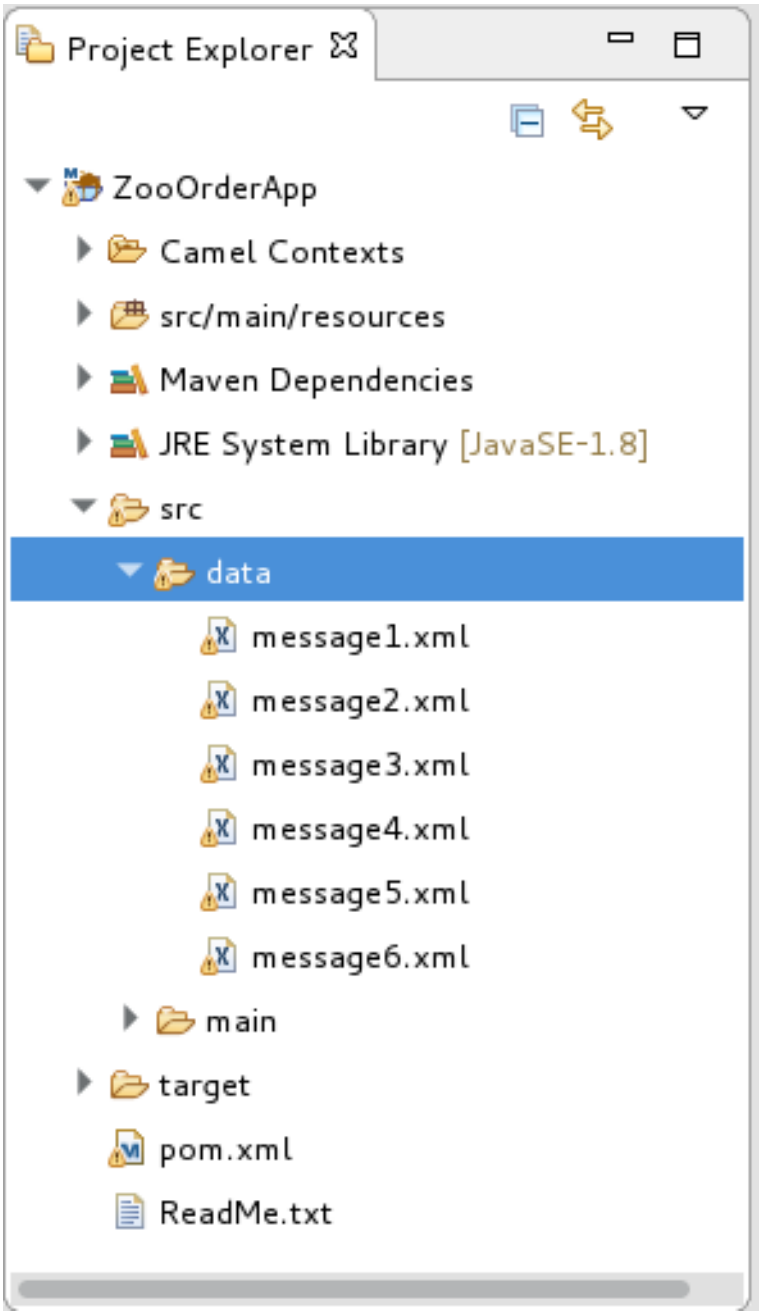
3. Click **Apply and Close**.

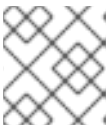
DOWNLOADING TEST MESSAGES FOR YOUR PROJECT

Sample XML message files are provided so that you can test your ZooOrderApp project as you work through the Tooling Tutorials. The messages contain order information for zoo animals. For example, an order of five wombats for the Chicago zoo.

To download and copy the provided test messages (XML files) to your project:

1. In the CodeReady Studio **Project Explorer** view, create a folder to contain the test messages:
   a. Right-click the **ZooOrderApp/src** folder and then select **New → Folder**. The **New Folder** wizard opens.
   b. For **Folder name**, type **data**.
   c. Click **Finish**.
2. Click here to open a web browser to the location of the provided Tooling Tutorial resource **Fuse-tooling-tutorials-jbds-10.3.zip** file.
   Download the **Fuse-tooling-tutorials-jbds-10.3.zip** file to a convenient location that is external to the ZooOrderApp project's workspace, and then unzip it. It contains two folders as described in Chapter 1, *About the Fuse Tooling Tutorials*.
3. From the **messages** folder, copy the six XML files to your **ZooOrderApp** project's **src/data** folder.





NOTE

You can safely ignore the on the XML files.

VIEWING THE TEST MESSAGES

Each XML message file contains an order from a zoo (a customer) for a quantity of animals. For example, the 'message1.xml' file contains an order from the Brooklyn Zoo for 12 wombats.

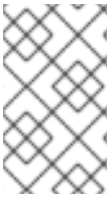
You can open any of the message XML files in the **Editor** view to examine the contents.

1. In the **Project Explorer** view, right-click a message file.
2. From the popup menu, select **Open**.
3. Click the **Source** tab.
   The XML file opens in the **Editor** view.

For example, the contents of the **message1.xml** file shows an order from the Bronx Zoo for 12 wombats:

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

<order>
  <customer>
    <name>Bronx Zoo</name>
    <city>Bronx NY</city>
    <country>USA</country>
  </customer>
  <orderline>
    <animal>wombat</animal>
    <quantity>12</quantity>
  </orderline>
</order>
```



NOTE

You can safely ignore the on the first 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.

The following table provides a summary of the contents of all six message files:

Table 2.1. Provided test messages

msg#	<name>	<city>	<country>	<animal>	<quantity>
1	Bronx Zoo	Bronx NY	USA	wombat	12
2	San Diego Zoo	San Diego CA	USA	giraffe	3
3	Sea Life Centre	Munich	Germany	penguin	15
4	Berlin Zoo	Berlin	Germany	emu	6
5	Philadelphia Zoo	Philapelphia PA	USA	giraffe	2
6	St Louis Zoo	St Loius MO	USA	penguin	10

NEXT STEPS

Now that you have set up your CodeReady Studio project, you can continue to the Chapter 3, *Defining a Route* tutorial in which you define the route that processes the XML messages.

CHAPTER 3. DEFINING A ROUTE

This tutorial walks you through adding and configuring endpoints to a route. Endpoints define the source and sink for messages traveling through the route. For your **ZooOrderApp** project, the starting (source) endpoint is the folder containing the XML message files. The sink (finishing) endpoint is another folder that you specify in your project.

GOALS

In this tutorial you complete the following tasks:

- Add source and sink endpoints to the route
- Configure the endpoints
- Connect the endpoints

BEFORE YOU BEGIN

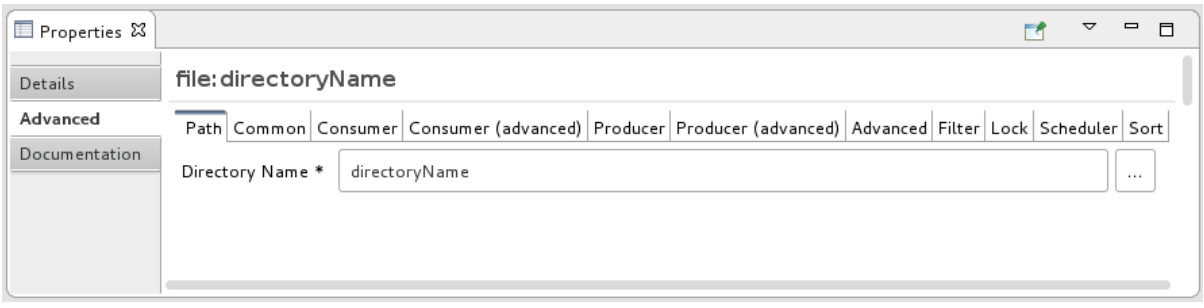
Before you start this tutorial:

1. You must set up your workspace environment, as described in the Chapter 2, *Setting up your environment* tutorial.
2. In CodeReady Studio, open your **ZooOrderApp** project's **/src/main/resources/OSGI-INF/blueprint/blueprint.xml** file in the **Editor** view.
3. If needed, click the **Design** tab at the bottom of the **Editor** view to see the graphic display of the initial route, labeled **Route_route1**.

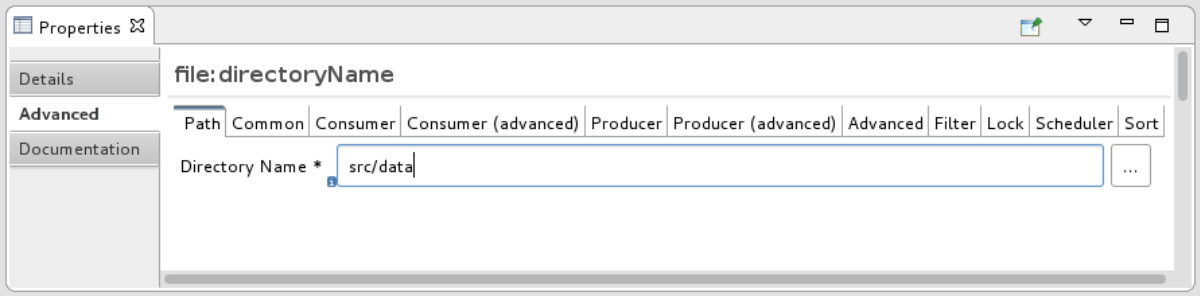
CONFIGURING THE SOURCE ENDPOINT

Follow these steps to configure the **src/data** folder as the route's source endpoint:

1. 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.
2. On the canvas, select the **From _from1** node.
   The **Properties** view, located below the canvas, displays the node's property fields for editing.
3. To specify the source directory for the message files, in the **Properties** view, click the **Advanced** tab:



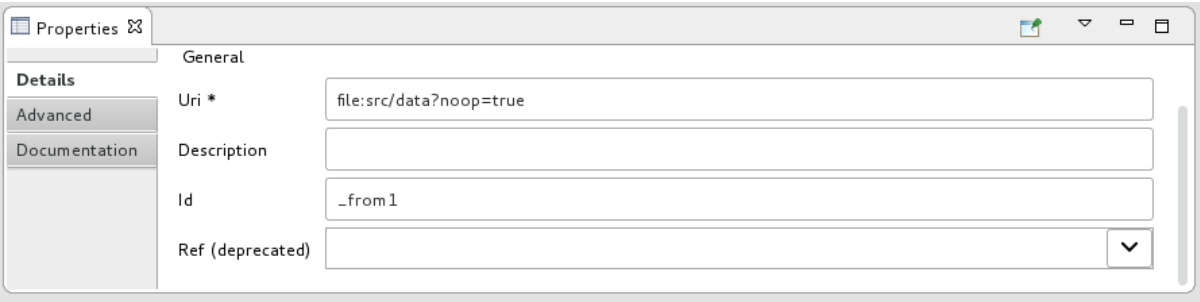
4. In the **Directory Name** field, enter **src/data**:

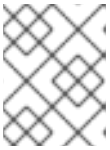


The path **src/data** is relative to the project's directory.

5. On the **Consumer** tab, enable the **Noop** option by clicking its check box.
The **Noop** option prevents the **message#.xml** files from being deleted from the **src/data** folder, and it enables idempotency to ensure that each **message#.xml** file is consumed only once.

6. Select the **Details** tab to open the file node's **Details** page.
Notice that 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**):





NOTE

The tooling prefixes autogenerated ID values with an underscore (_). You can optionally change the ID value. The underscore prefix is not a requirement.

Leave the autogenerated **Id** as is.

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

CONFIGURING THE SINK ENDPOINT

To add and configure the route's sink (target) endpoint:

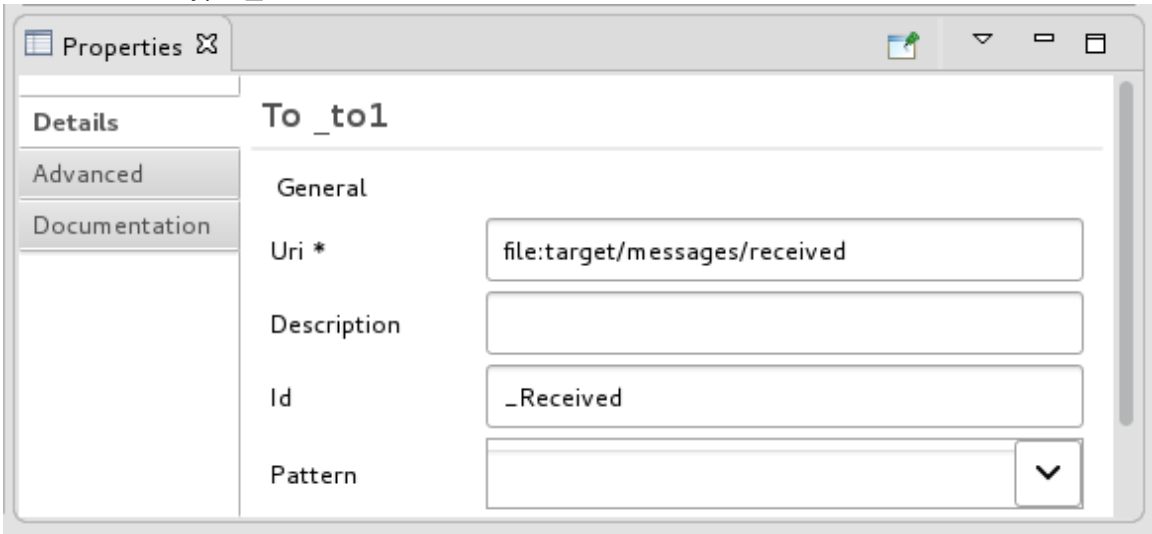
1. Drag another **File** component from the **Palette**'s **Components** drawer and drop it in the **Route_route1** container node.
The **File** component changes to a **To_to1** node inside the **Route_route1** container node.

2. On the canvas, select the **To_to1** node.
The **Properties** view, located below the canvas, displays the node's property fields for editing.

3. On the **Details** tab:

   a. In the **Uri** field, type **file:target/messages/received**.

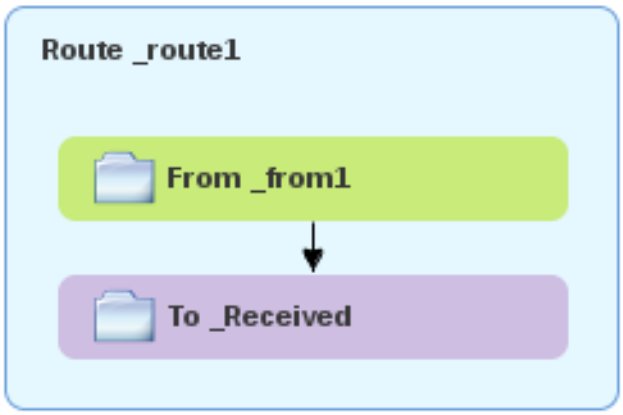
b. In the **Id** field, type **_Received**.

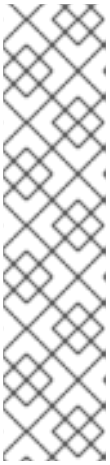


NOTE

The tooling will create the **target/messages/received** folder at runtime.

4. In the **Route_route1** container, select the **From _from1** node and drag its connector arrow ( ) over the **To_Received** node, and 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 **Down** (the default) and **Right**.

To access the route editor 's layout preference options:

- On Linux and Windows machines, select **Windows → Preferences → Fuse Tooling → Editor → Choose the layout direction for the diagram editor**
- On OS X, select **CodeReady Studio → Preferences → Fuse Tooling → Editor → Choose the layout direction for the diagram editor**

NOTE

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

5. **Save** the route.

6. Click the **Source** tab at the bottom of the canvas to display the XML for the route:

```
<?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="_Received" uri="file:target/messages/received"/>
      </route>
   </camelContext>
</blueprint>
```

NEXT STEPS

Now that you have added and configured endpoints in the route, you can run the route as described in the Chapter 4, *Running a Route* tutorial.

CHAPTER 4. RUNNING A ROUTE

This tutorial walks you through the process of running a route to verify that the route correctly transfers messages from the source endpoint to the sink endpoint.

GOALS

In this tutorial you complete the following tasks:

- Run a route as a local Camel context (without tests since you have not set up a test yet)
- Send messages through the route
- Examine the messages received by the sink endpoint to make sure that the route correctly processed the test messages

PREREQUISITES

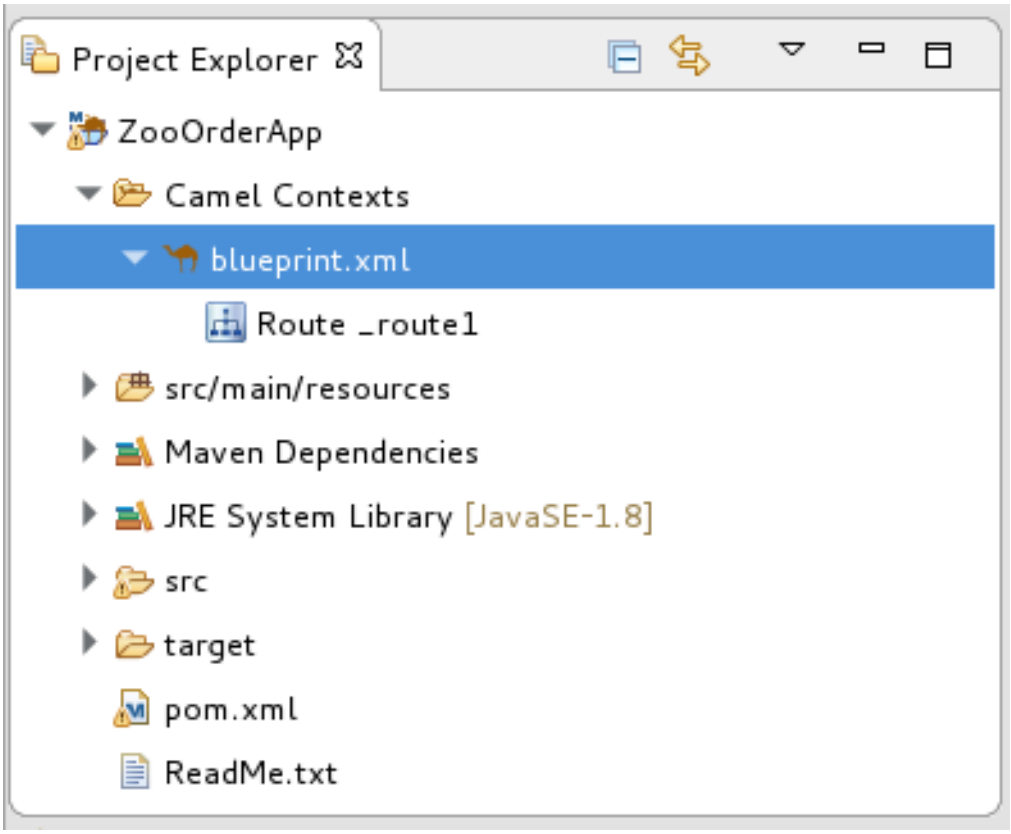
To start this tutorial, you need the **ZooOrderApp** project resulting from:

1. Completing the Chapter 2, *Setting up your environment* tutorial.
2. One of the following:
   - Completing the Chapter 3, *Defining a Route* tutorial.
     or
   - Replacing your project's **blueprint.xml** file with the provided **blueprintContexts/blueprint1.xml** file, as described in the section called "About the resource files".

RUNNING THE ROUTE

To run the route:

1. Open the **ZooOrderApp** project.
2. In **Project Explorer**, select **ZooOrderApp/Camel Contexts/blueprint.xml** :



3. Right-click the **blueprint.xml**, and then select **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 a JUnit test does not exist, the tooling reverts to running the routing context without tests. In the Chapter 9, *Testing a route with JUnit* tutorial, you create a JUnit test case to test the **ZooOrderApp** 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. The Maven download process can take a few minutes.

4. Wait for messages (similar to the following) to appear at the end of the output. These messages indicate that the route executed successfully:

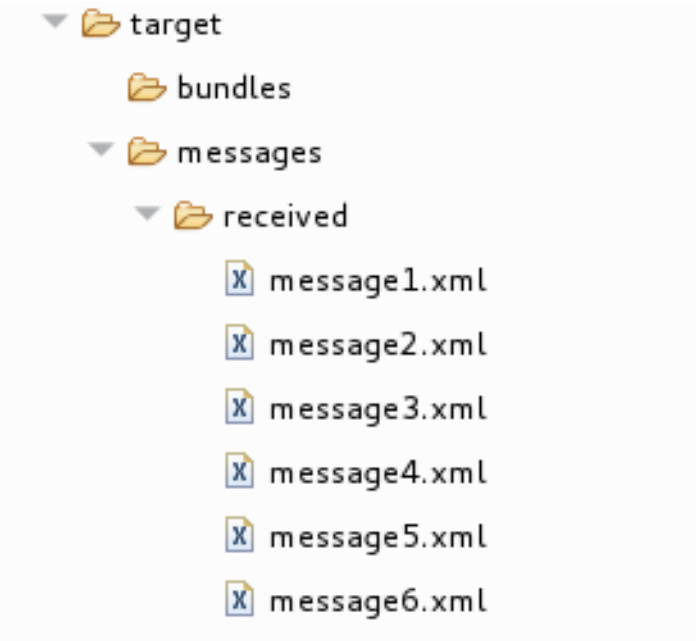
```
...
[Blueprint Event Dispatcher: 1] BlueprintCamelContext INFO  Route: _route1 started and
consuming from:Endpoint[file://src/data?noop=true]
[Blueprint Event Dispatcher: 1] BlueprintCamelContext INFO  Total 1 routes, of which 1 are
started.
[Blueprint Event Dispatcher: 1]BlueprintCamelContext INFO  Apache Camel 2.21.0.redhat-3
(CamelContext: ...) started in 0.163 seconds
[Blueprint Event Dispatcher: 1] BlueprintCamelContext INFO  Apache Camel 2.21.0.redhat-3
(CamelContext: ...) started in 0.918 seconds
```

5. To shutdown the route, click located at the top of the **Console** view.

VERIFYING THE ROUTE

To verify that the route executed properly, you check to see whether the message XML files were copied from the source folder (**src/data**) to the target folder (**target/messages/received**).

1. In **Project Explorer**, select **ZooOrderApp**.
2. Right-click and then select **Refresh**.
3. In **Project Explorer**, locate the **target/messages/** folder and expand it to verify that the **target/messages/received** folder contains the six message files, **message1.xml** through **message6.xml**:



4. Double-click **message1.xml** to open it in the route editor's **Design** tab, and then select the **Source** tab to see the XML code:

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

<order>
  <customer>
    <name>Bronx Zoo</name>
    <city>Bronx NY</city>
    <country>USA</country>
  </customer>
  <orderline>
    <animal>wombat</animal>
    <quantity>12</quantity>
  </orderline>
</order>
```

NEXT STEPS

In the Chapter 5, *Adding a Content-Based Router* tutorial you add a Content-Based Router that uses the content of a message to determine its destination.

CHAPTER 5. ADDING A CONTENT-BASED ROUTER

This tutorial shows how to add a Content-Based Router (CBR) and logging to a route.

A CBR routes a message to a destination based on its content. In this tutorial, the CBR that you create routes messages to different folders (valid or invalid) based on the value of each message's quantity field (the number of animals in the order). The maximum value of animals for each order is 10. The CBR routes the messages to different folders, depending on whether the quantity is greater than 10. For example, if a zoo orders five zebras and only three zebras are available, the order is copied to the invalid order target folder.

GOALS

In this tutorial you complete the following tasks:

- 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

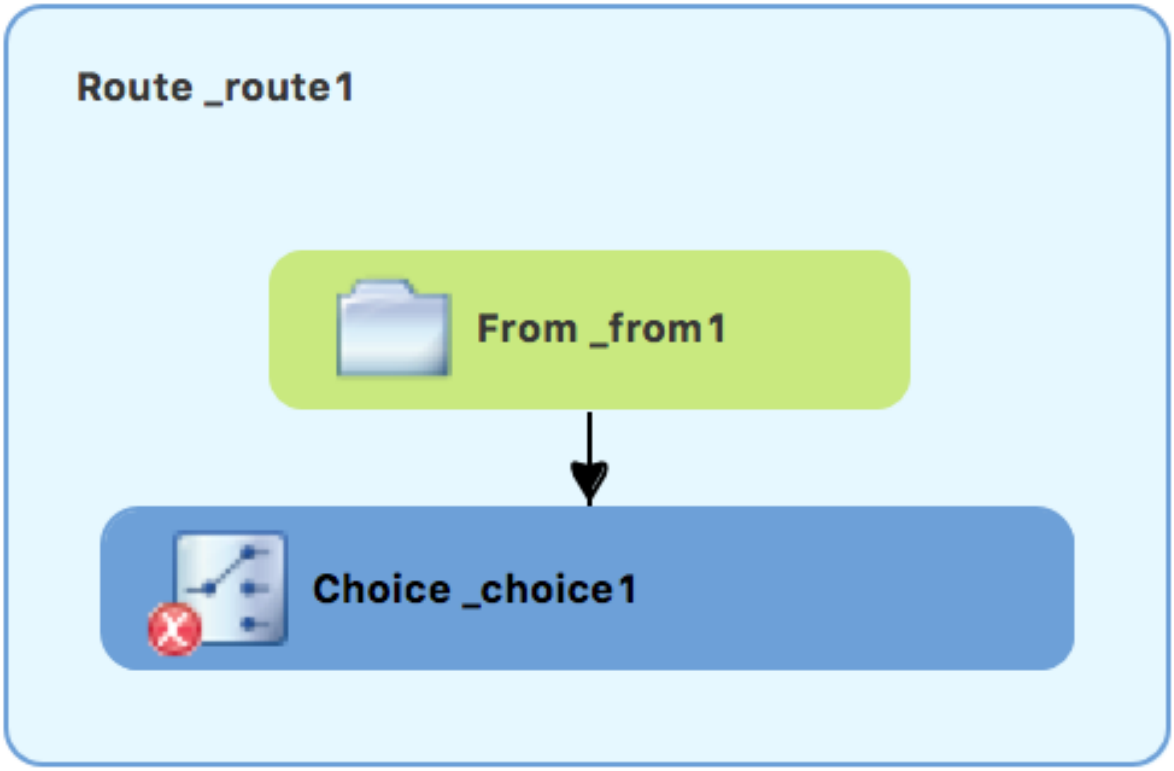
To start this tutorial, you need the **ZooOrderApp** project resulting from one of the following:

- Completing the Chapter 4, *Running a Route* tutorial.
  or
- Completing the Chapter 2, *Setting up your environment* tutorial and replacing your project's **blueprint.xml** file with the provided **blueprintContexts/blueprint1.xml** file, as described in the section called "About the resource files".

ADDING AND CONFIGURING A CONTENT-BASED ROUTER

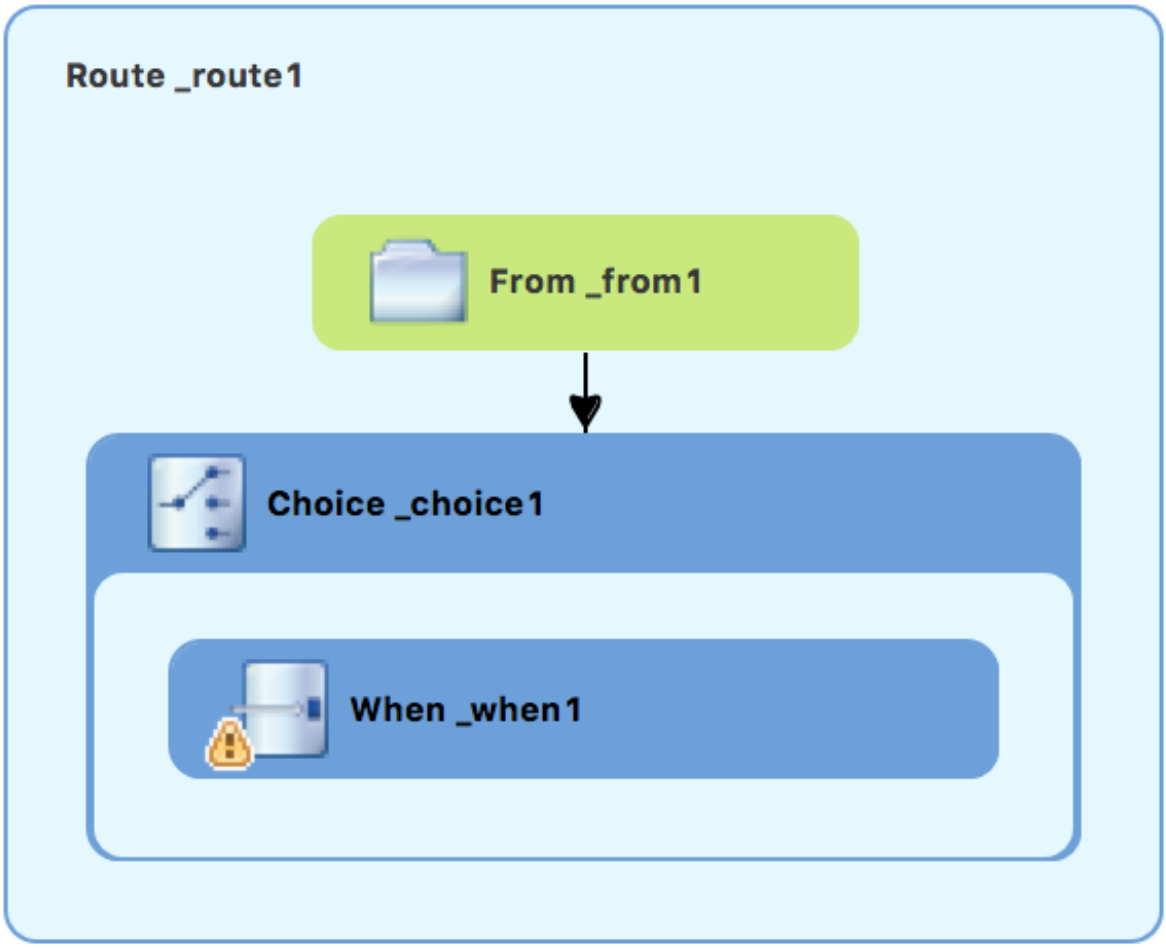
To add and configure a Content-Based Router for your route:

1. In **Project Explorer**, double-click **ZooOrderApp/src/main/resources/OSGI-INF/blueprint/blueprint.xml** to open it in the Editor view.
2. On the **Design** canvas, select the **To_Received** node and then select the trash can icon to delete it.
3. In the **Palette**, open the **Routing** drawer, click a **Choice** ( ) pattern, and then (in the **Design** canvas) click the **From_from1** node.



The **Route_route1** container expands to accommodate the **Choice_choice1** node. The error icon indicates that the **Choice_choice1** node requires a child node, which you add next.

4. From the **Routing** drawer, click the **When** ( ) pattern and then, in the canvas, click 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 adding a pattern to an invalid drop point in a Route container.

5. On the canvas, select the **When_when1** node, to open its properties in the **Properties** view:

6. Click the button in the **Expression** field to open the list of available options.

7. Select **xpath** (for the XML query language) because the test messages are written in XML.

NOTE

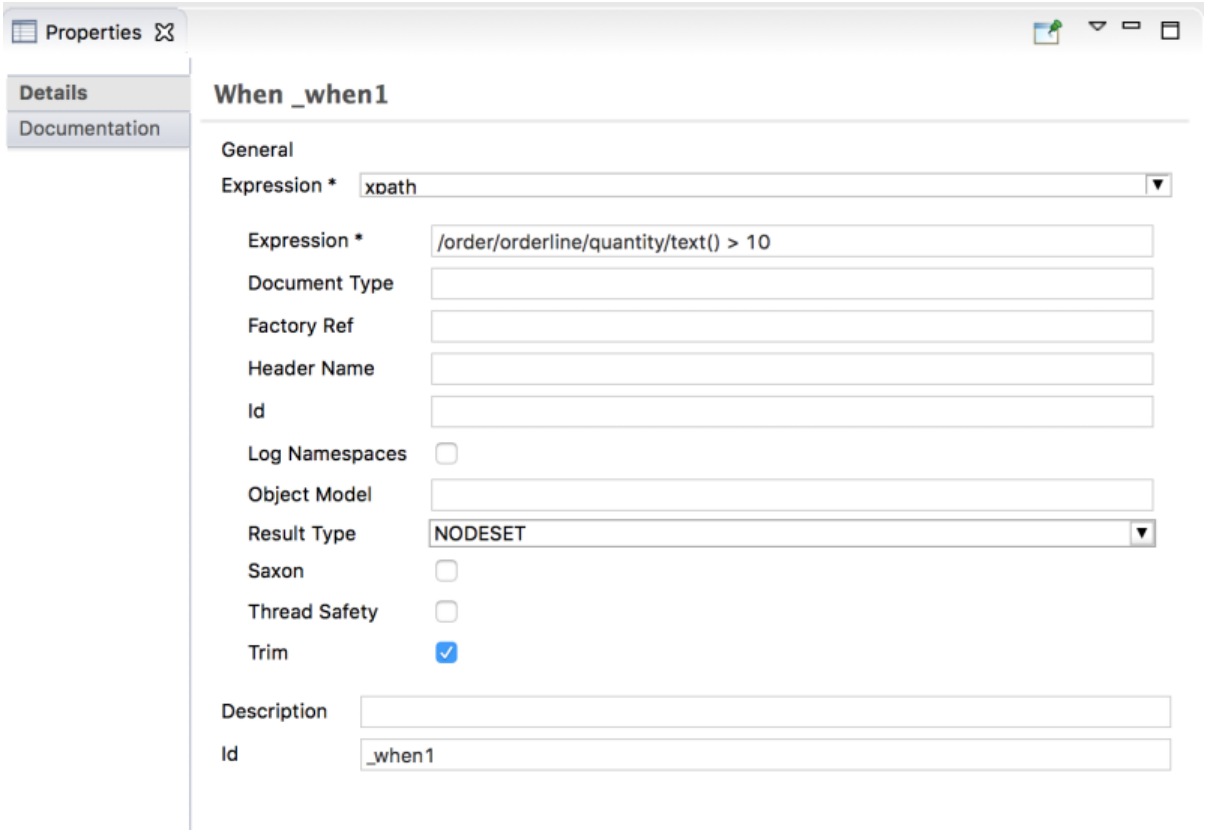
Once you select the **Expression** language, the **Properties** view displays its properties in an indented list directly below the **Expression** field. The **Id** property in this indented list sets the ID of the expression. The **Id** property following the **Description** field sets the ID of the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

8. In the indented **Expression** field, type: **/order/orderline/quantity/text() > 10**
This expression specifies that only messages in which the value of the **quantity** field is greater than 10 travel this path in the route (to the invalidOrders folder).

9. Leave each of the remaining properties as they are.

NOTE

The **Trim** option (enabled by default) removes any leading or trailing white spaces and line breaks from the message.



10. **Save** the routing context file.

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

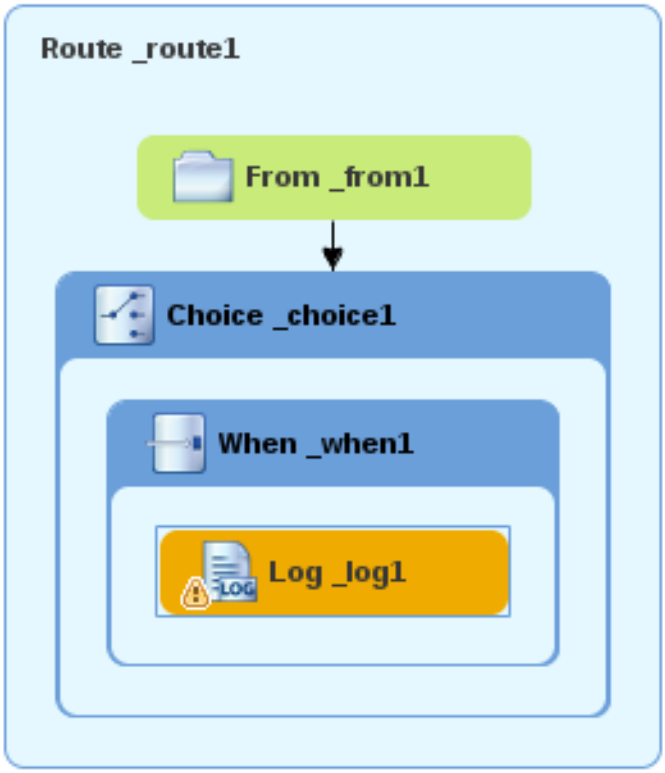
ADDING AND CONFIGURING LOGGING

For the ZooOrder application example, you add a log message so that you can track an XML message as it passes through the route. When you run the route, the log message appears in the **Console** view.

Follow these steps to add logging to your CBR route:

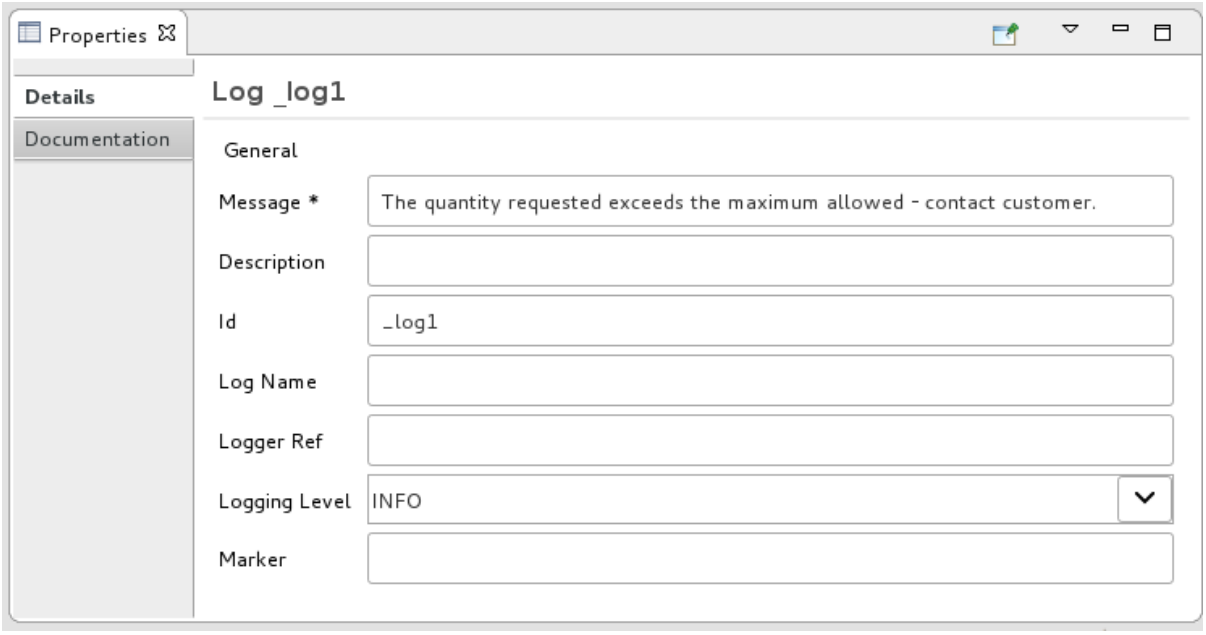
1. In the **Design** tab's **Palette**, open the **Components** drawer and click the **Log** component ( ).

2. In the canvas, click the **When_when1** node.
   The **When_when1** container expands to accommodate the **Log_log1** node:



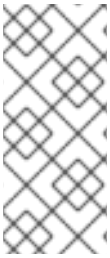
3. On the canvas, select the **Log_log1** node to open its properties in the **Properties** view.

4. In the **Message** field, type: **The quantity requested exceeds the maximum allowed - contact customer.**



Leave the remaining properties as they are.

+



NOTE

The tooling auto-generates a log node **id** value. 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 the Chapter 8, *Tracing a message through a route* tutorial). In the **Console**, it adds the contents of the log node's **Message** field to the log data whenever the route runs.

1. **Save** the routing context file.

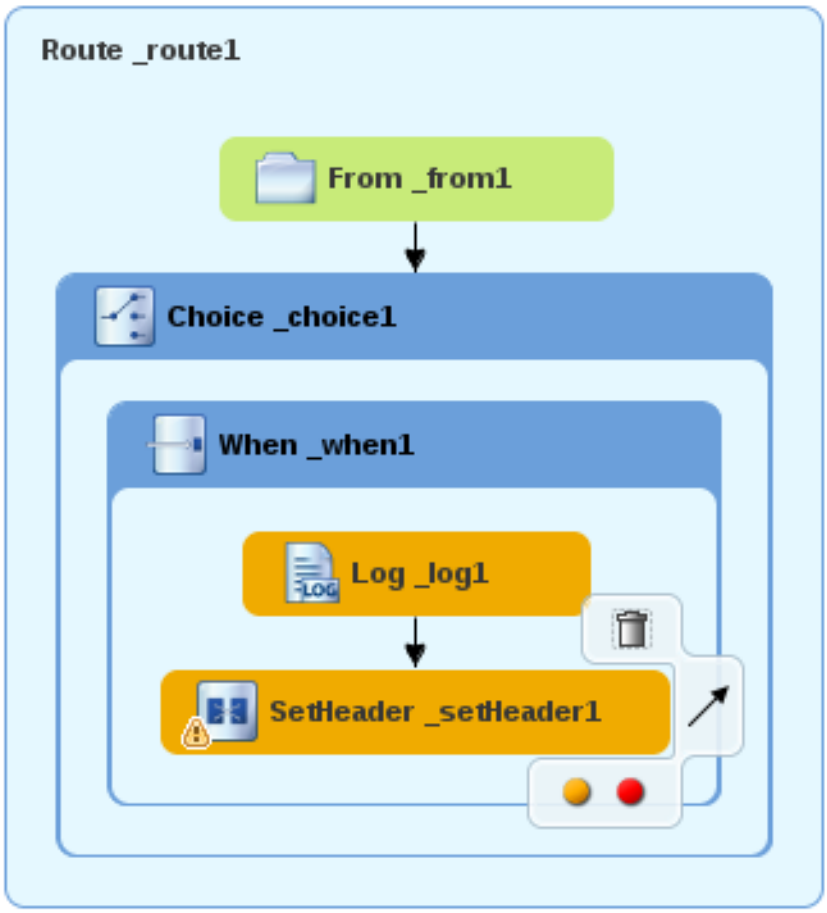
ADDING AND CONFIGURING MESSAGE HEADERS

A message header contains information to process a message.

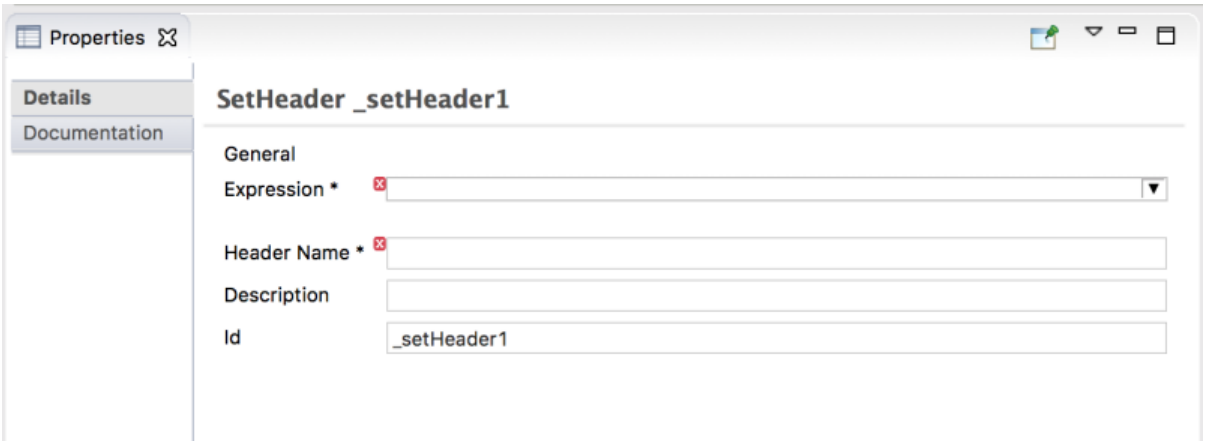
To add and configure message headers:

1. In the **Palette**, open the **Transformation** drawer and then click the **Set Header** ( ) pattern.

2. In the canvas, click the **Log_log1** node.
The **When_when1** container expands to accommodate the **SetHeader_setHeader1** node:



3. On the canvas, select the **SetHeader_setHeader1** node to open its properties in the **Properties** view:

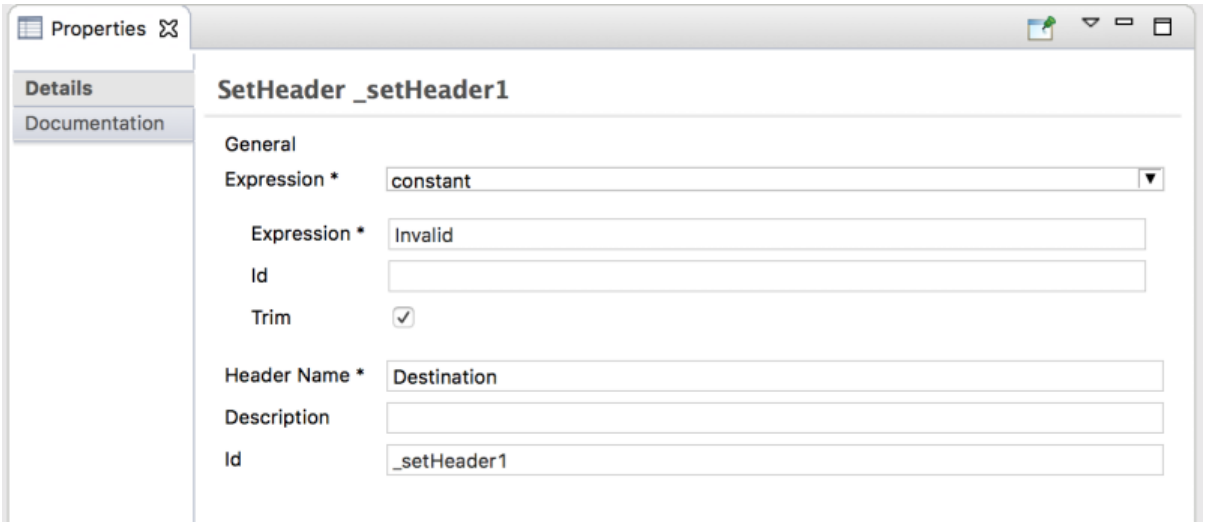


4. Click the button in the **Expression** field to open the list of available languages, and then select **constant**.

5. In the indented **Expression** field, type **Invalid**.

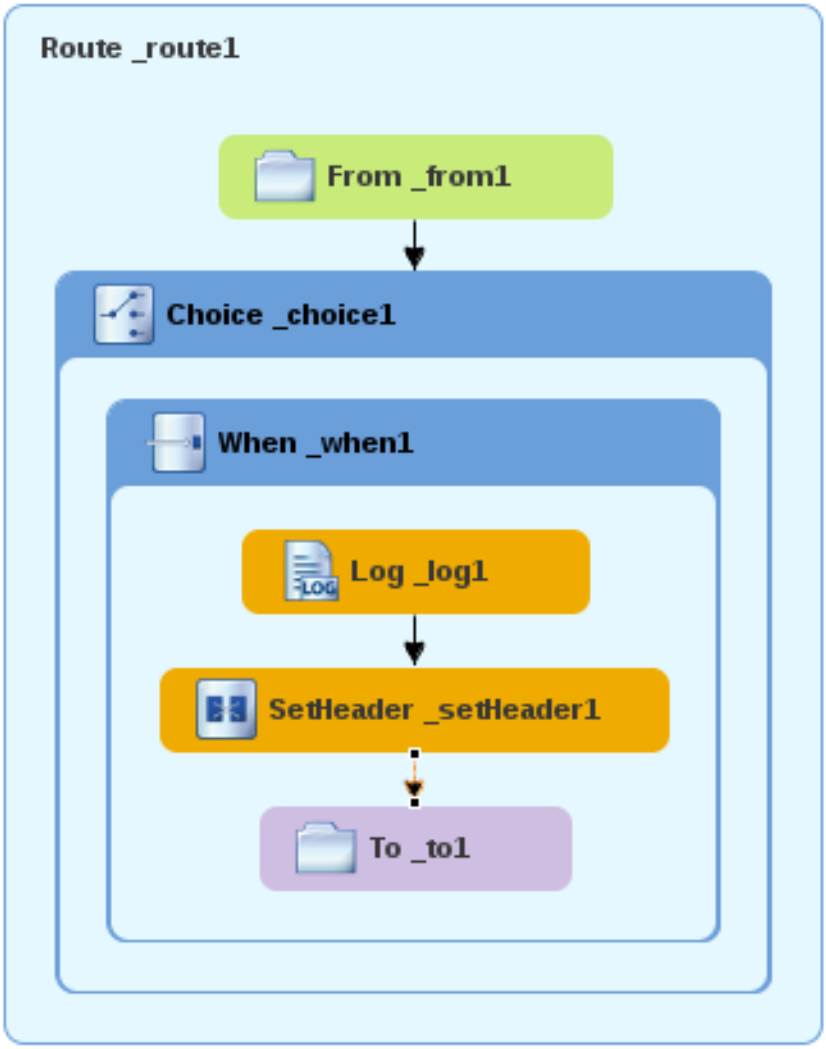
6. In the **Header Name** field, type **Destination**.

7. Leave the remaining properties as they are.

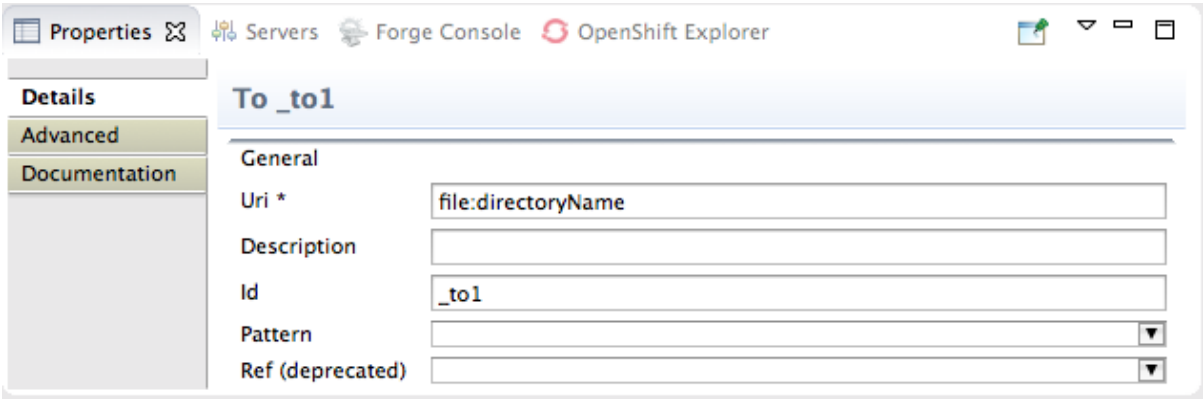


8. In the **Palette**, open the **Components** drawer and then click the **File** ( ) component.

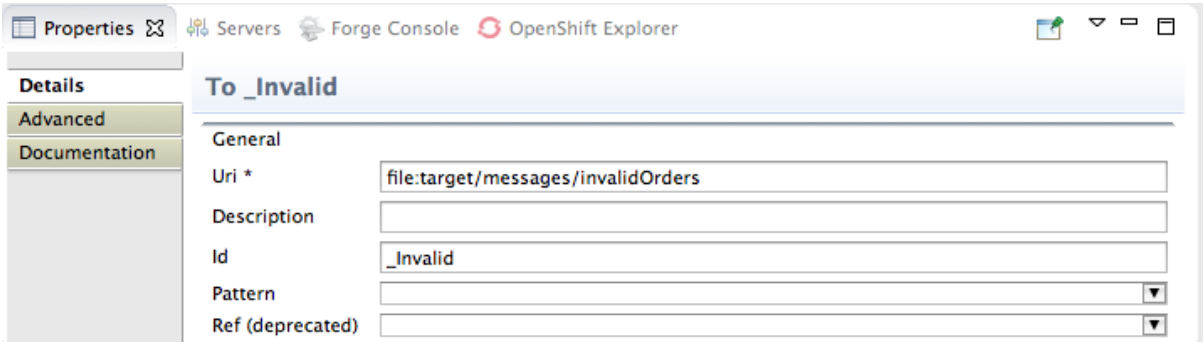
9. In the canvas, click the **SetHeader_setHeader1** node.
The **When_when1** container expands to accommodate the **To_to1** node.



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

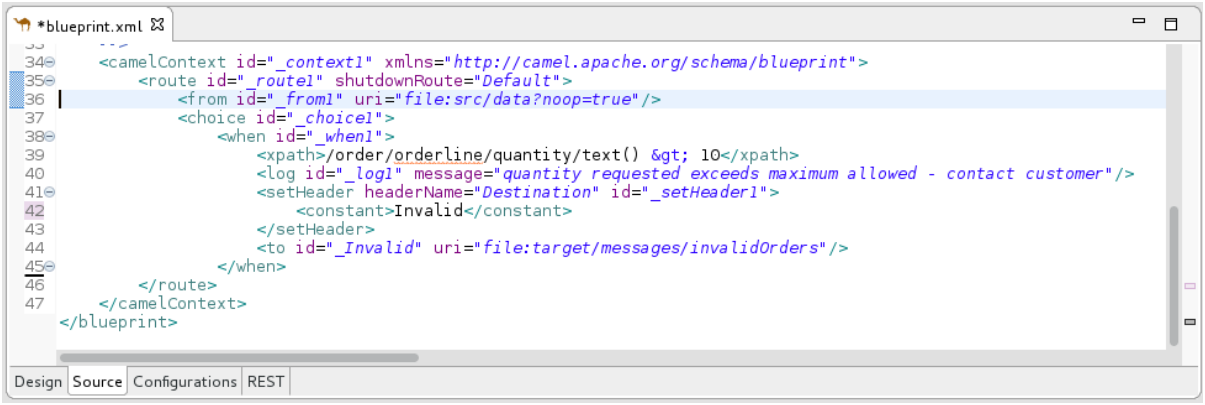


11. On the **Details** tab, replace *directoryName* with **target/messages/invalidOrders** in the **Uri** field, and type **_Invalid** in the **Id** field:



12. **Save** the routing context file.

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



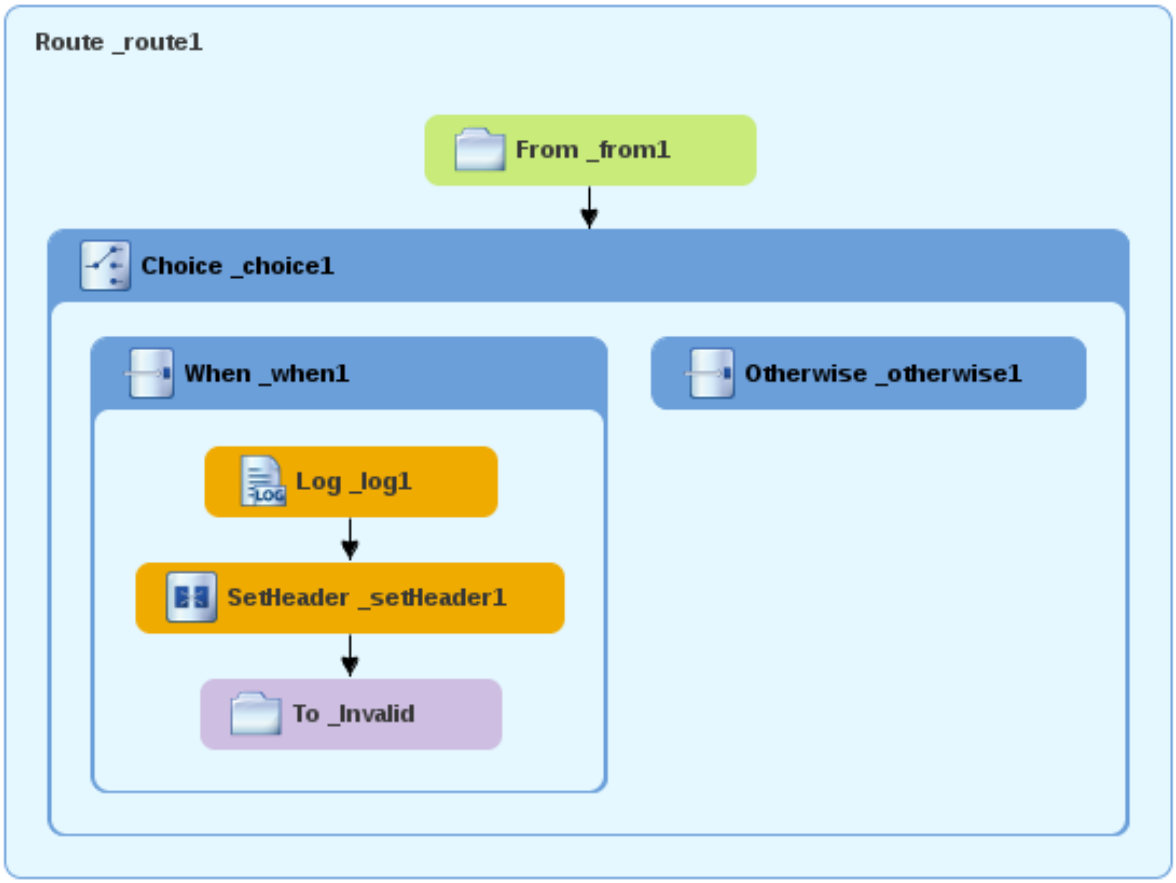
ADDING AND CONFIGURING A BRANCH TO HANDLE VALID ORDERS

So far, the CBR handles messages that contain invalid orders (orders where the quantity value is greater than 10).

To add and configure an otherwise branch of your route to handle valid orders (that is, any XML messages that do not match the XPath expression set for the **When_when1** node):

1. In the **Palette**, open the **Routing** drawer and click the **Otherwise** ( ) pattern.

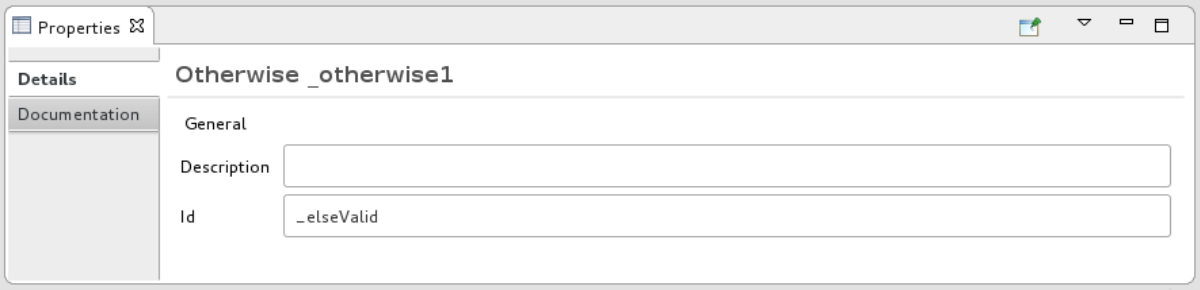
2. In the canvas, click the **Choice_choice1** container:



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

3. On the canvas, select the **Otherwise_otherwise1** node to open its properties in the **Properties** view.

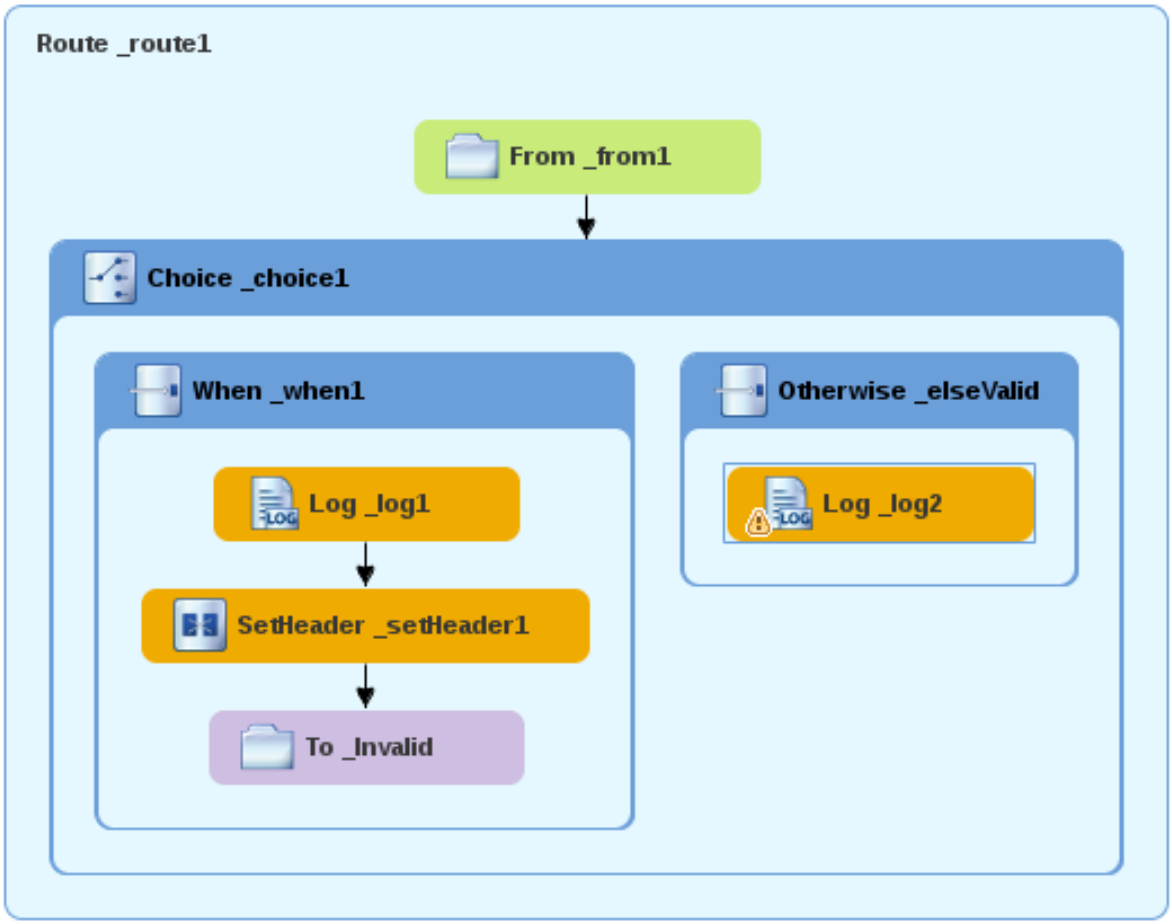
4. In the **Id** field, change **_otherwise1** to **_elseValid**:



To configure logging for the otherwise branch:

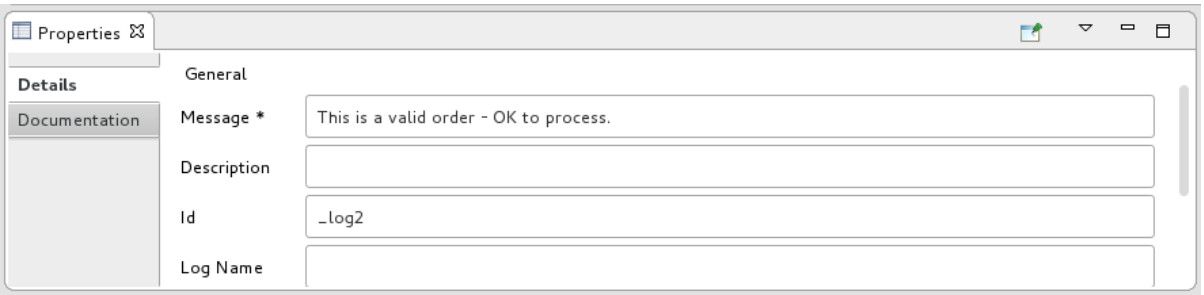
1. In the **Palette**, open the **Components** drawer and and then click the **Log** ( ) component.

2. In the canvas, click the **Otherwise_elseValid** node:
The **Otherwise-elseValid** container expands to accommodate the **Log_log2** node.



3. On the canvas, select the **Log_log2** node to open its properties in the **Properties** view.

4. In the **Message** field, type **This is a valid order - OK to process.**

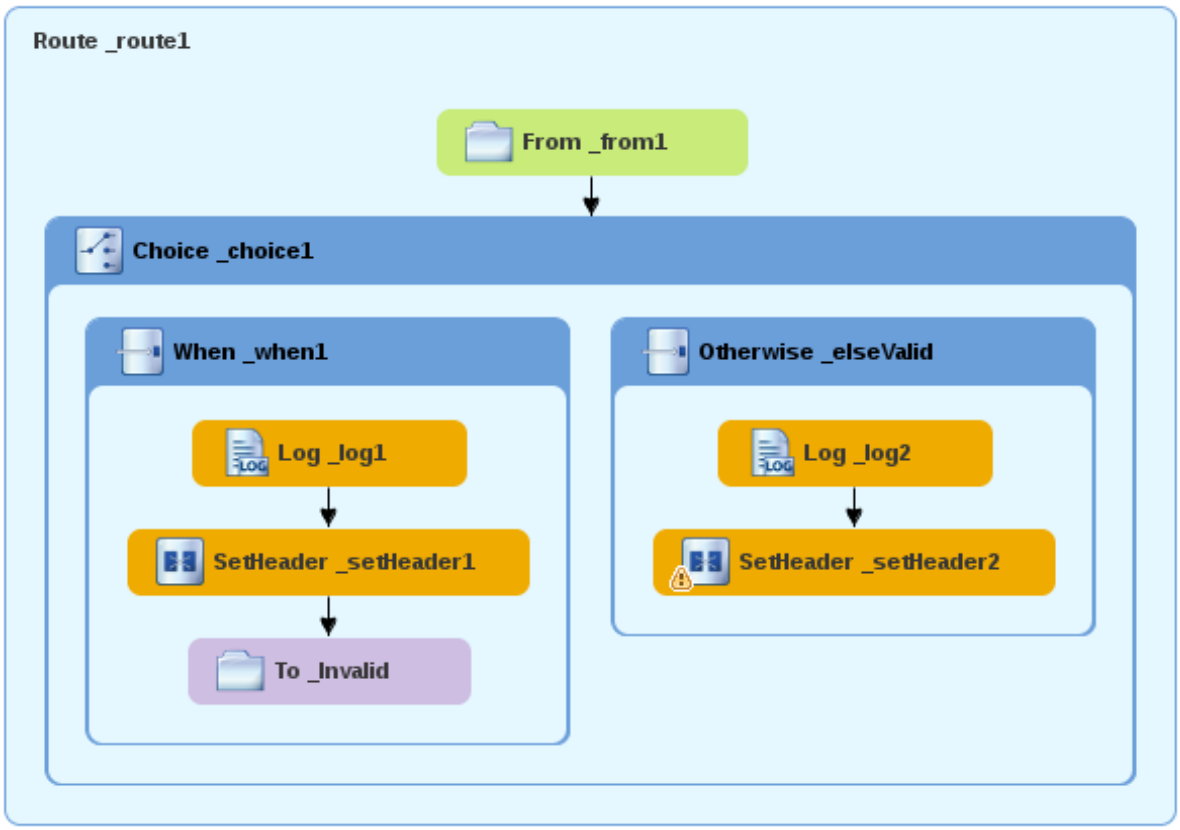


Leave the remaining properties as they are.

5. **Save** the route.

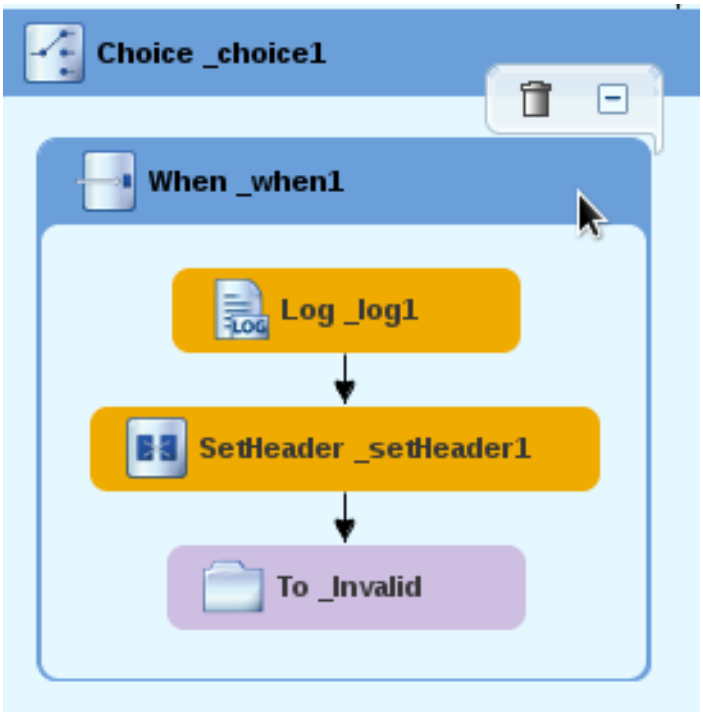
To configure a message header for the otherwise branch:

1. In the **Palette**, open the **Transformation** drawer and then click the **Set Header** pattern.
2. In the canvas, click the **Log_log2** node.
   The **Otherwise_elseValid** container expands to accommodate the **SetHeader_setHeader2** node.

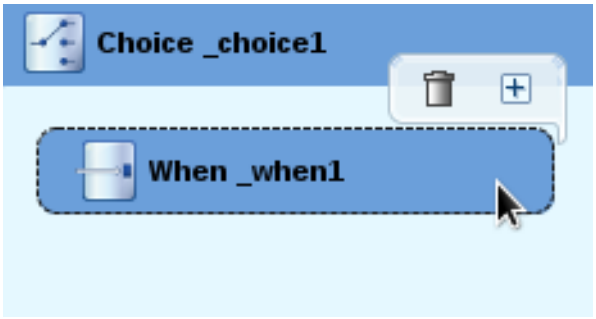


NOTE

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.

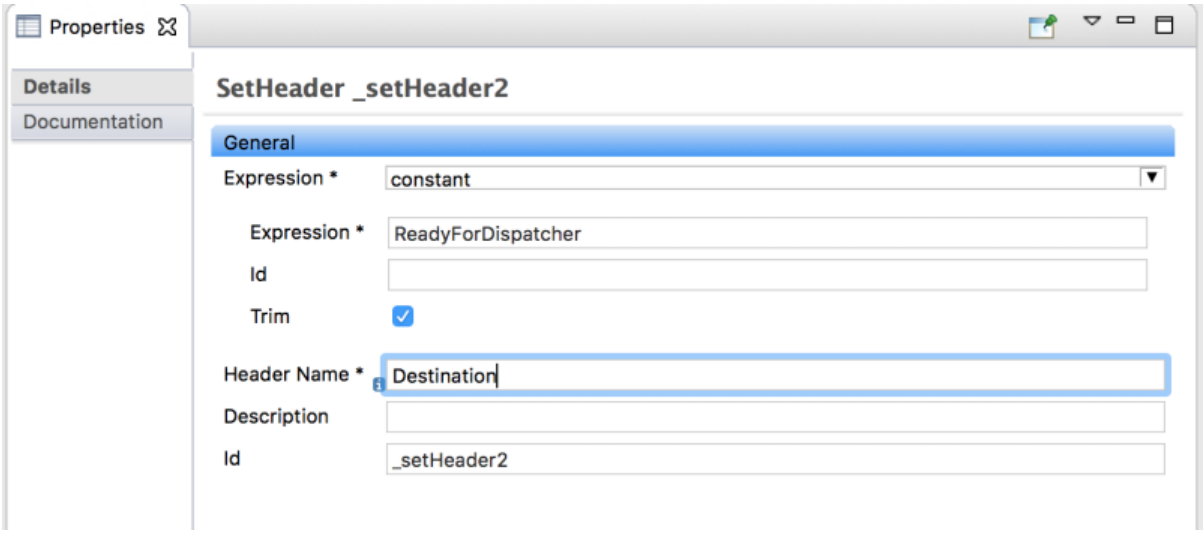
3. On the canvas, select the **SetHeader_setHeader2** node to open its properties in the **Properties** view.

4. Click the button in the **Expression** field to open the list of available languages, and select **constant**.

5. In the indented **Expression** field, type **ReadyForDispatcher**.

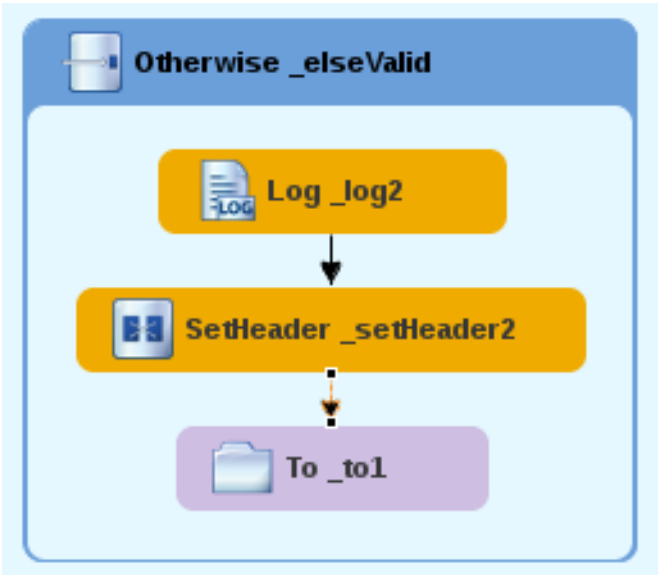
6. In the **Header Name** field, type **Destination**.

7. Leave the remaining properties as they are.

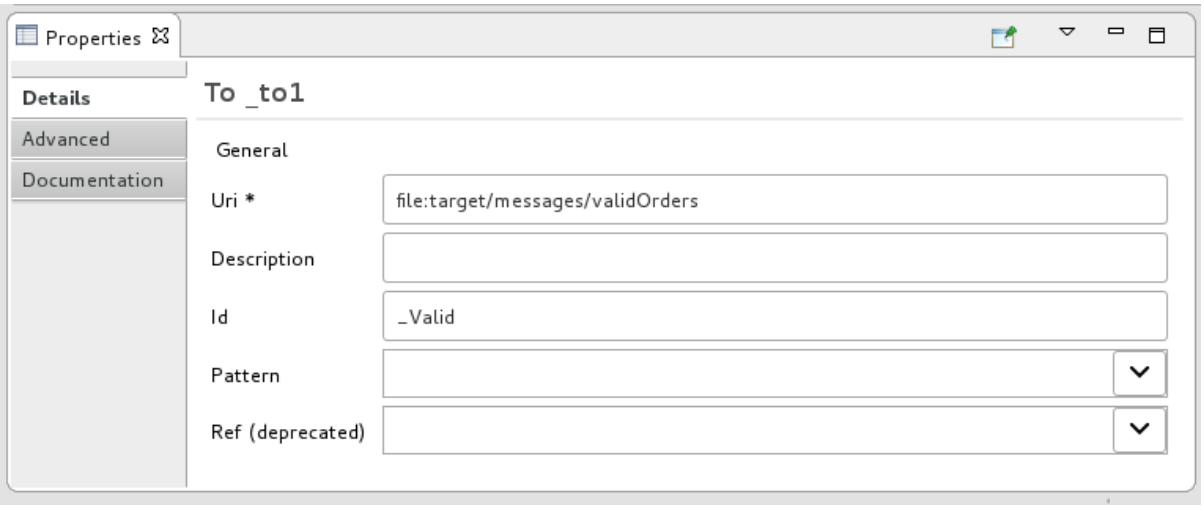


To specify the target folder for the valid messages:

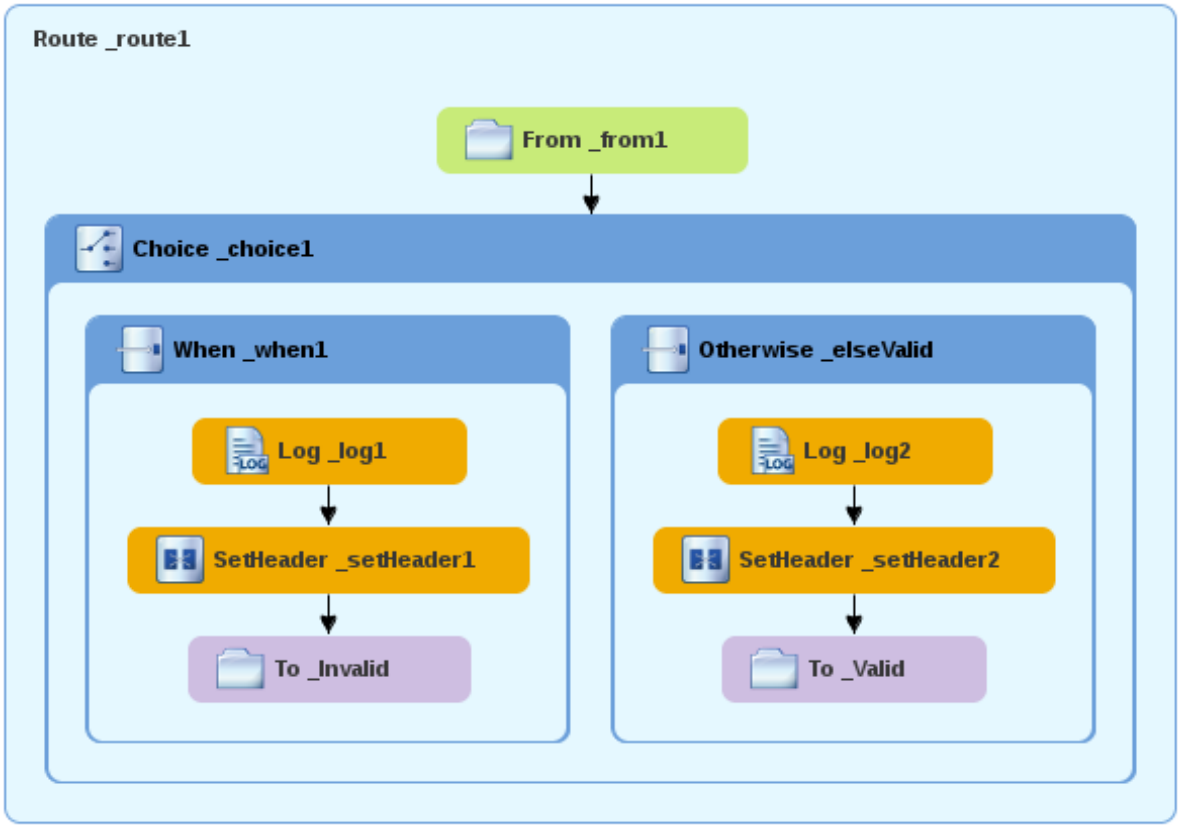
1. In the **Palette**, open the **Components** drawer and then select the **File** ( ) component.
2. In the canvas, click the **SetHeader_setHeader2** node.
   The **Otherwise_elseValid** container expands to accommodate the **To_to1** node.



3. On the canvas, select the **To_to1** node to open its properties in the **Properties** view.
4. In the **URI** field, replace *directoryName* with **target/messages/validOrders**, and in the **Id** field, type **_Valid**.



5. **Save** the routing context file.
   The completed content-based router should look like this:



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

```
<?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"/>
```

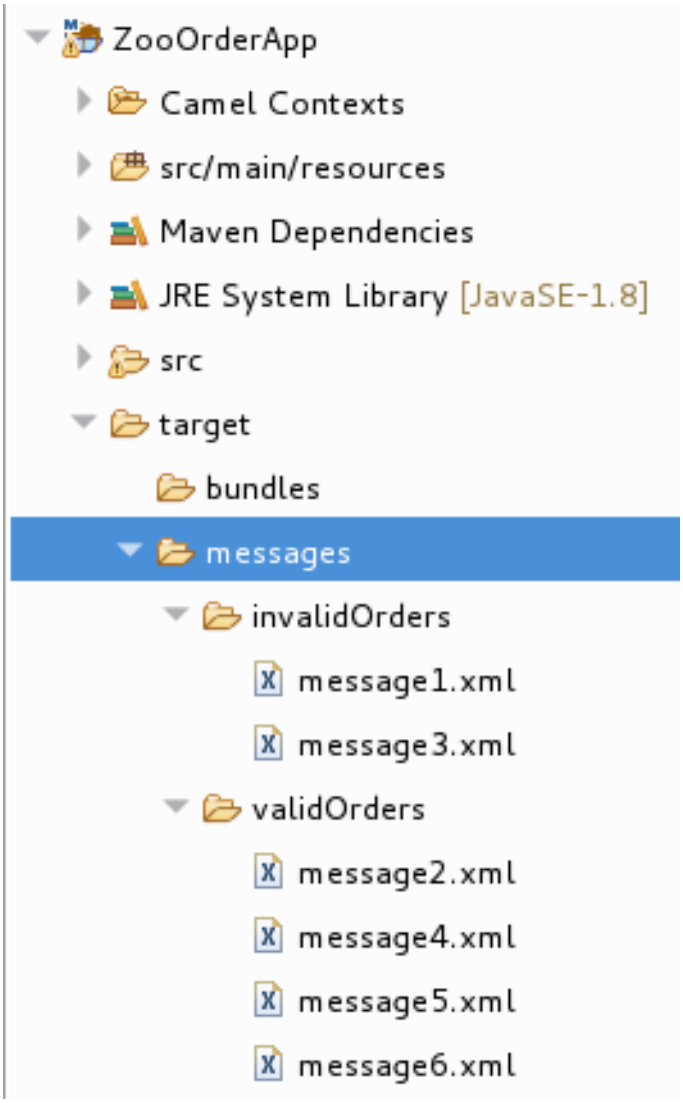
```
        <choice id="_choice1">
          <when id="_when1">
            <xpath>/order/orderline/quantity/text() &gt; 10</xpath>
            <log id="_log1" message="The quantity requested exceeds the maximum
allowed - contact customer."/>
            <setHeader headerName="Destination" id="_setHeader1">
              <constant>Invalid</constant>
            </setHeader>
            <to id="_Invalid" uri="file:target/messages/invalidOrders"/>
          </when>
          <otherwise id="_elseValid">
            <log id="_log2" message="This is a valid order - OK to process."/>
            <setHeader headerName="Destination" id="_setHeader2">
              <constant>ReadyForDispatcher</constant>
            </setHeader>
            <to id="_Valid" uri="file:target/messages/validOrders"/>
          </otherwise>
        </choice>
      </route>
    </camelContext>
</blueprint>
```

VERIFYING THE CBR

You can run the new route as described in the the section called “Running the route” tutorial and look at the **Console** view to see the log messages.

After you run it, to verify whether the route executed properly, check the target destination folders in the **Project Explorer**:

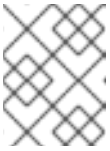
1. Select **ZooOrderApp**.
2. Right-click it to open the context menu, and then select **Refresh**.
3. Under the project root node (**ZooOrderApp**), locate the **target**/**messages**/ folder and expand it.



4. Check that the **target/messages/invalidOrders** folder contains **message1.xml** and **message3.xml**.
   In these messages, the value of the **quantity** element exceeds 10.

5. Check that the **target/messages/validOrders** folder contains the four message files that contain valid orders:

   - **message2.xml**
   - **message4.xml**
   - **message5.xml**
   - **message6.xml**
     In these messages, the value of the **quantity** element is less than or equal to 10.



NOTE

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

NEXT STEPS

In the next tutorial, Chapter 6, *Adding another route to the routing context* , you add a second route that further processes valid order messages.

CHAPTER 6. ADDING ANOTHER ROUTE TO THE ROUTING CONTEXT

This tutorial shows you how to add a second route to the camel context in the **ZooOrderApp** project's **blueprint.xml** file. The second route:

- Takes messages (valid orders) directly from the terminal end of the first route's **otherwise** branch.
- Sorts the valid messages according to the customer's country.
- Sends each message to the corresponding **country** folder in the **ZooOrderApp/target/messages** folder. For example, an order from the Chicago zoo is copied to the USA folder.

GOALS

In this tutorial you complete the following tasks:

- Reconfigure the existing route for direct connection to a second route
- Add a second route to your Camel context
- Configure the second route to take messages directly from the otherwise branch of the first route
- Add a content-based router to the second route
- Add and configure a message header, logging, and target destination to each output branch of the second route's content-based router

PREREQUISITES

To start this tutorial, you need the **ZooOrderApp** project resulting from one of the following:

- Complete the Chapter 5, *Adding a Content-Based Router* tutorial.
  or
- Complete the Chapter 2, *Setting up your environment* tutorial and replace your project's **blueprint.xml** file with the provided **blueprintContexts/blueprint2.xml** file, as described in the section called "About the resource files".

RECONFIGURING THE EXISTING ROUTE'S ENDPOINT

The existing route sends all valid orders to the **target/messages/validOrders** folder.

In this section, you reconfigure the endpoint of the existing route's **Otherwise _elseValid** branch to instead connect to a second route (which you create in the next section).

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

1. Open your **ZooOrderApp/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 then select **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

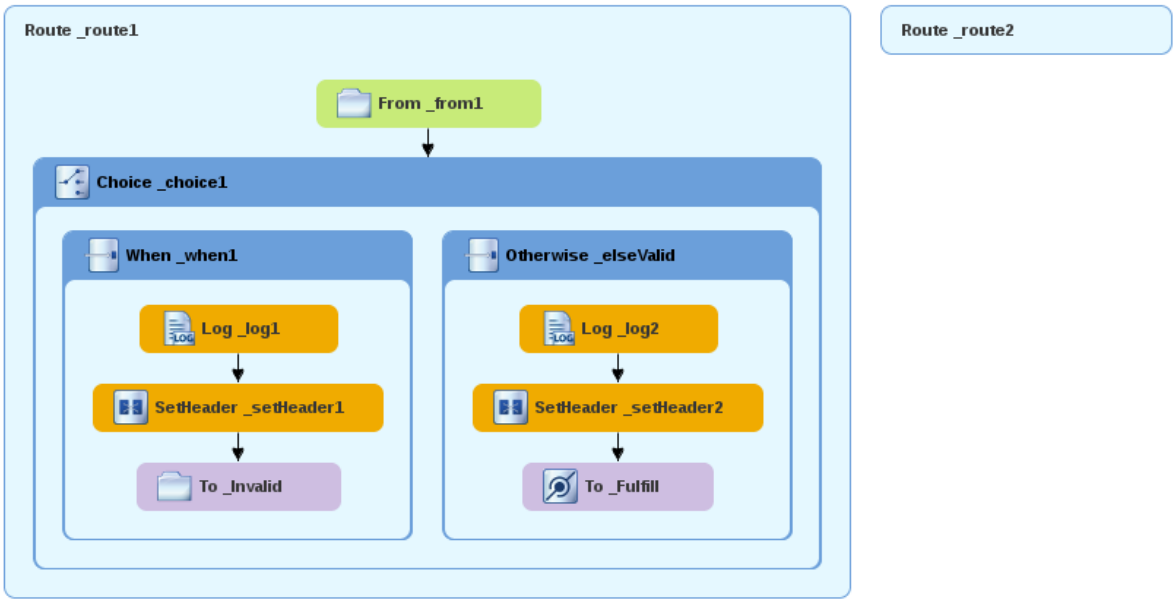
Instead of repurposing the existing **To_Valid** terminal file node, you could have replaced it with a **Components → Direct** component, configuring it with the same property values as the repurposed **To_Valid** node.

To learn more about the **Direct** component see the Apache Camel Component Reference.

ADDING THE SECOND ROUTE

To add another route to the routing context:

1. In the **Palette**, open the **Routing** drawer and then click the **Route** ( ) pattern.
2. In the canvas, click to the right of the **Route_route1** container:

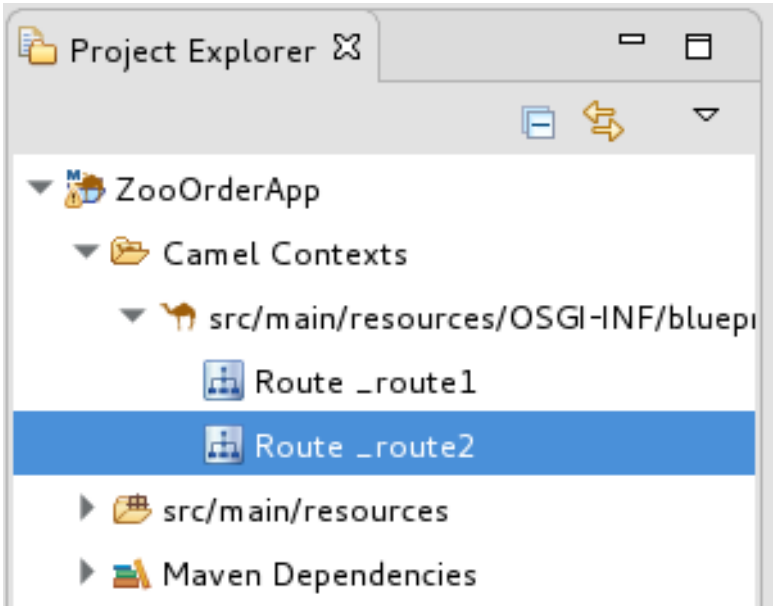


The **Route** pattern becomes the **Route_route2** container node on the canvas.

3. Click the **Route_route2** container node to display its properties in the **Properties** view. Leave the properties as they are.
4. **Save** the file.

NOTE

As your routing context grows in complexity, you might 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 that you want the route editor to display on the canvas; for example **Route_route2**:



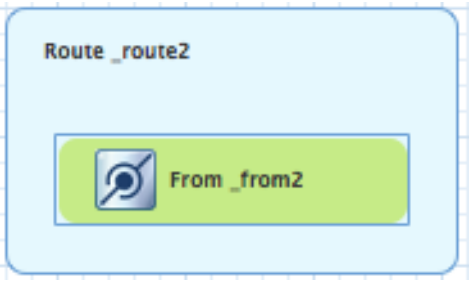
To display all routes in the routing context on the canvas, double-click the project's **.xml** context file entry (**src/main/resources/OSGI-INF/...**) at the top of the **Camel Contexts** folder.

CONFIGURING A CHOICE BRANCH TO PROCESS USA ORDERS

In this section, you add a Choice branch to the route and configure the route to send USA orders to a new **target/messages/validOrders/USA** folder. You also set a message header and a log file component.

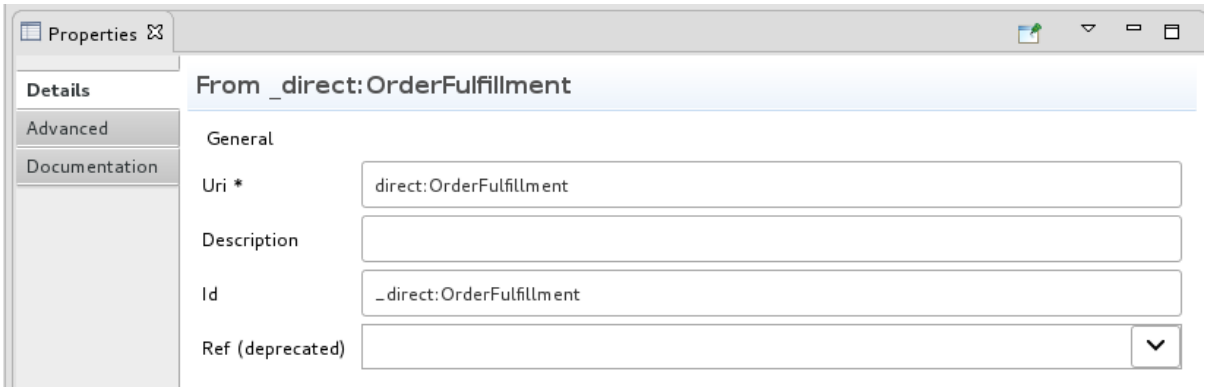
1. In the **Palette**, open the **Components** drawer and then select the **Direct** component ( ).

2. In the canvas, click the **Route_route2** container:
   The **Route_route2** container expands to accommodate the **Direct** component (the **From_from2** node):



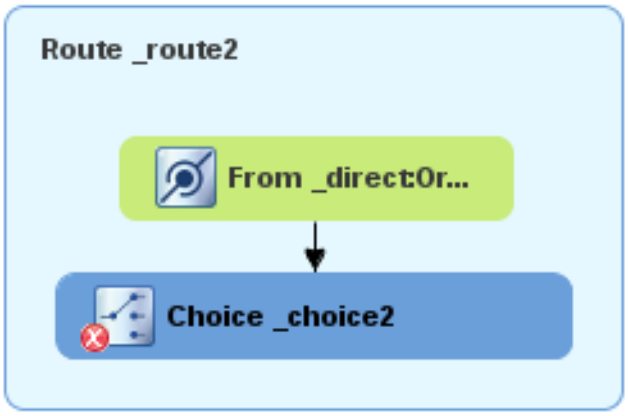
3. On the canvas, click the **From_from2** node to open its properties in the **Properties** view.

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



5. In the **Palette**, open the **Routing** drawer and then select the **Choice** ( ) pattern.

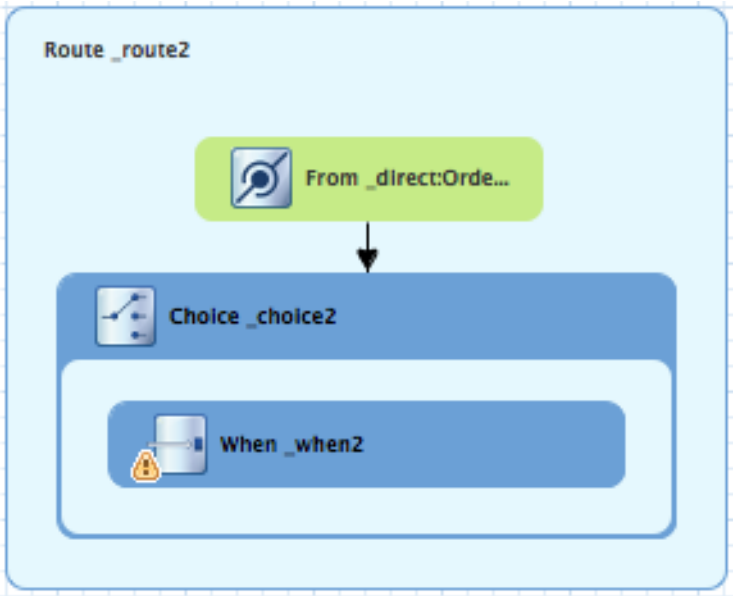
6. In the canvas, click the **From _direct:OrderFulfillment** node.
The **Route_route2** container expands to accommodate the **Choice_choice2** node:



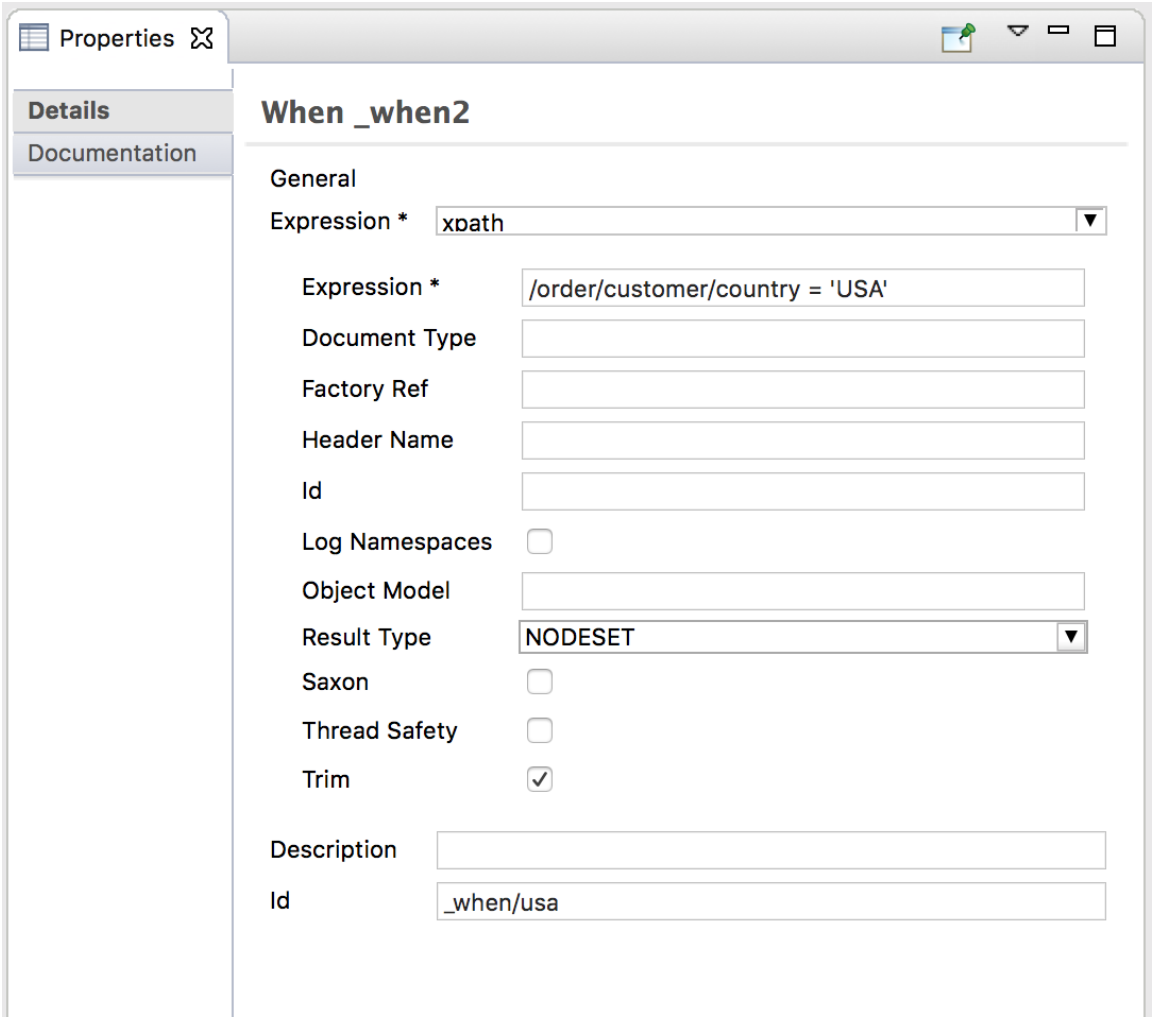
In the **Properties** view, leave the **Choice_choice2** node's properties as they are.

7. In the Palette, open the **Routing** drawer and then select the **When** ( ) pattern.

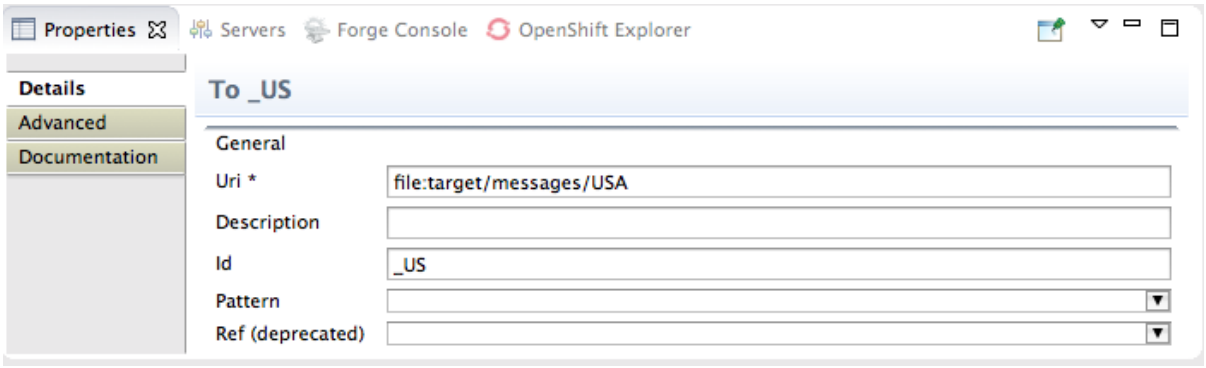
8. In the canvas, click the **Choice_choice2** node.
The **Choice_choice2** container expands to accommodate the **When_when2** node.



9. On the canvas, select the **When_when2** node to open its properties in the **Properties** view.
10. Set the **When_when2** node's properties as follows:
    - Select **xpath** from the **Expression** drop-down list.
    - In the indented **Expression** field, type **/order/customer/country = 'USA'**.
    - Leave **Trim** enabled.
    - In the second **Id** field, type **_when/usa**



11. In the **Palette**, open the **Components** drawer and then select the **File** component ( ).
12. In the canvas, click the **When_when/usa** container.
    The **When_when/usa** container expands to accommodate the **To_to1** node.
13. In the **Properties** view:

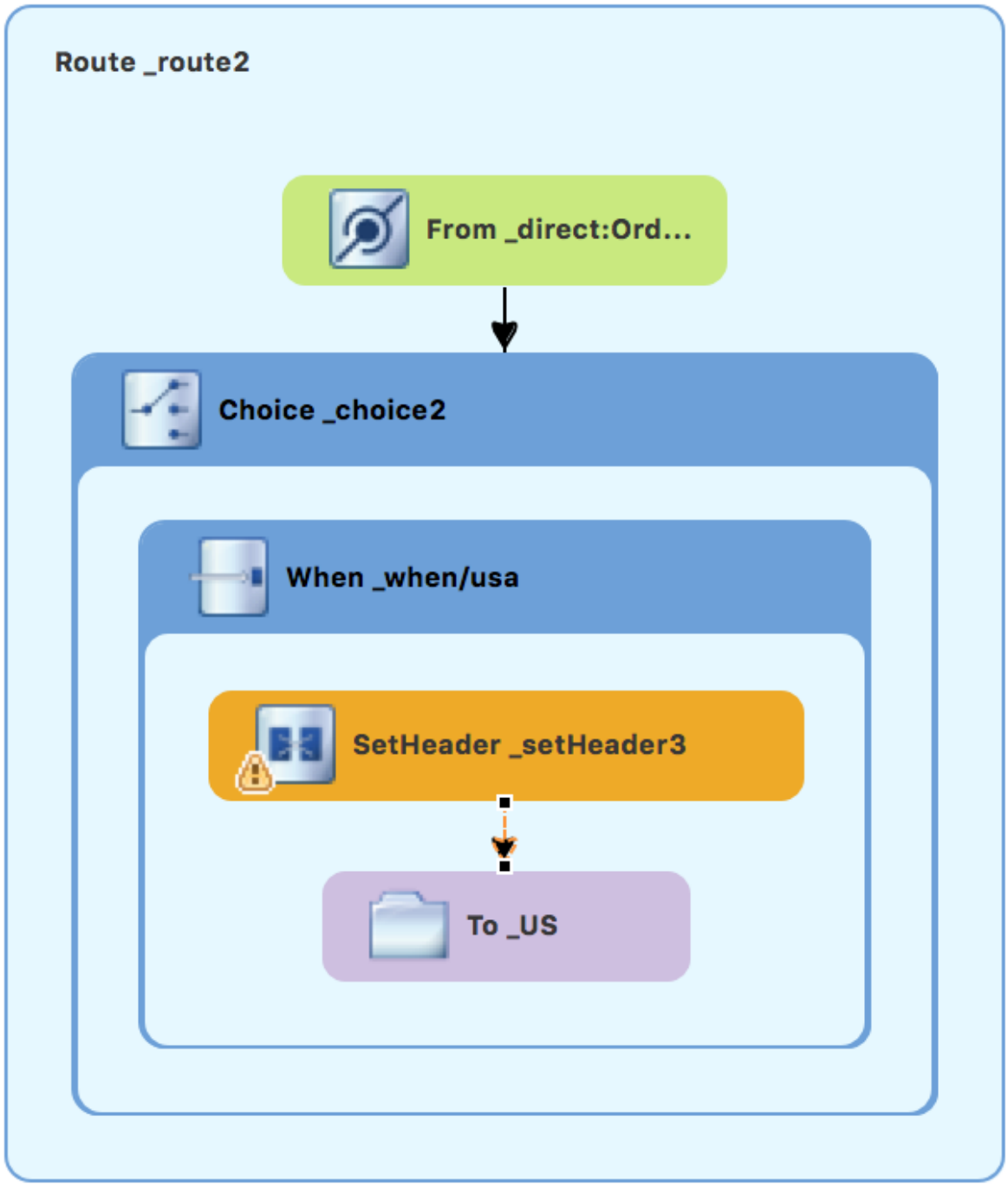


- In the **Uri** field, replace **directoryName** with **target/messages/validOrders/USA** .
- In the **Id** field, type **_US**.

14. **Save** the file.

To set a message header and add a log component:

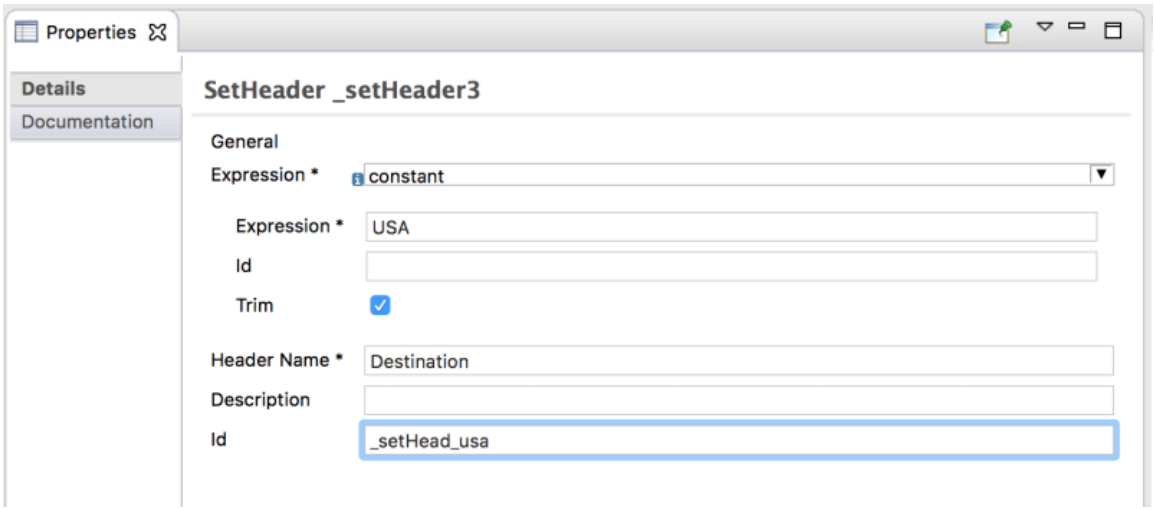
1. In the **Palette**, open the **Transformation** drawer and then select the **Set Header** pattern.
2. In the canvas, click the **When_when/usa** node.
   The **When_when/usa** container expands to accommodate the **SetHeader_setHeader3** node:



3. On the canvas, select the **SetHeader_setHeader3** node to open its properties in the **Properties** view.

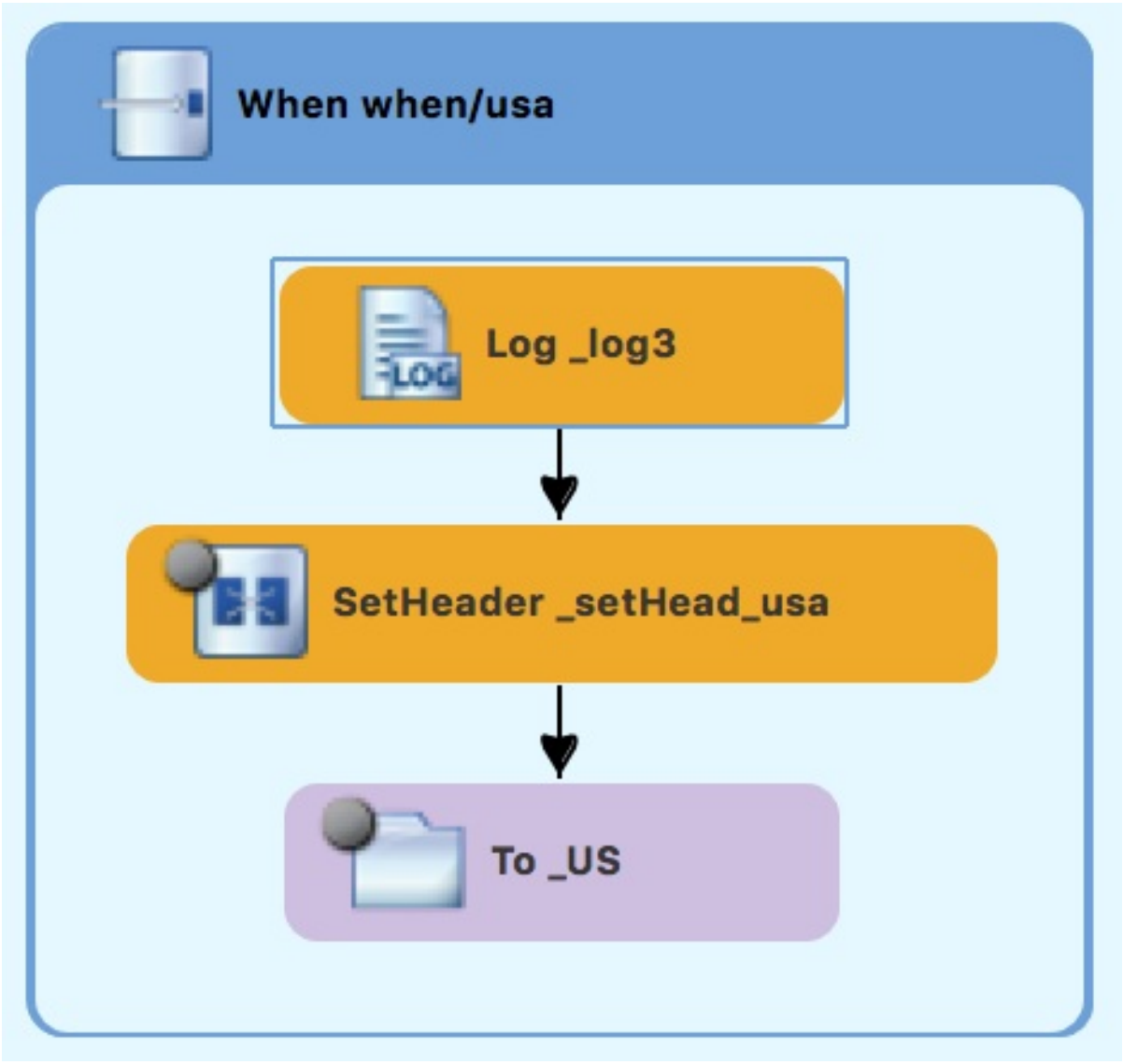
4. Set the node's properties as follows:

   - From the **Expression** drop-down menu, select **constant**.
   - In the indented **Expression** field, type: **USA**
   - Leave **Trim** enabled.
   - In the **Header Name** field, type: **Destination**
   - In the second **Id** field, type: **_setHead_usa**

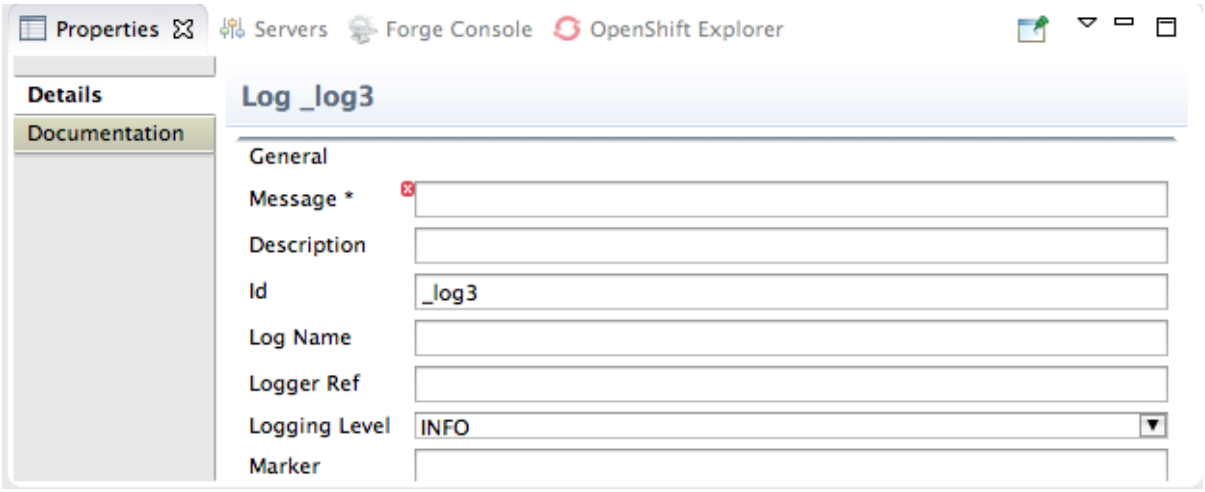


5. In the **Palette**, open the **Components** drawer and then select the **Log** component ( ).

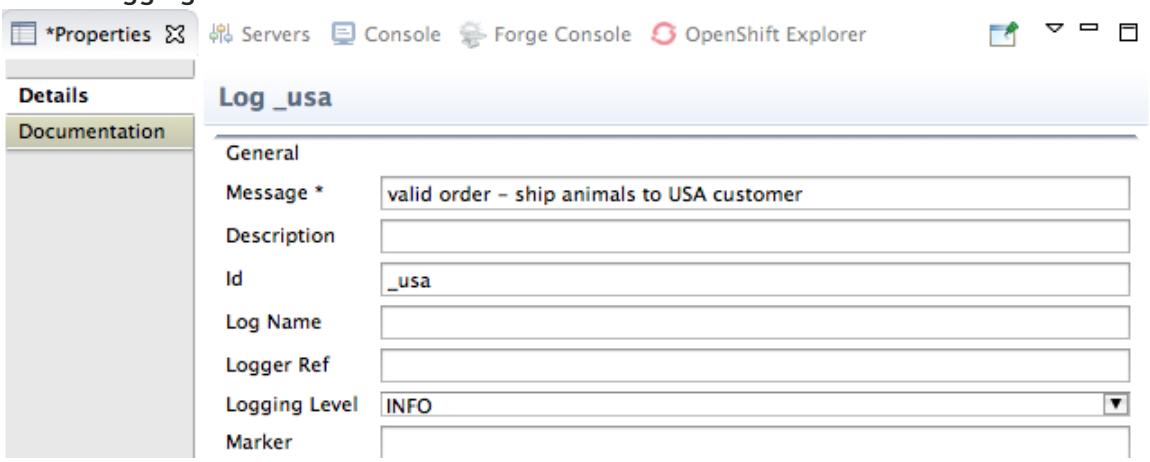
6. In the canvas, click above the **SetHeader** node.
   The **When_when/usa** container expands to accommodate the **Log_log3** node.



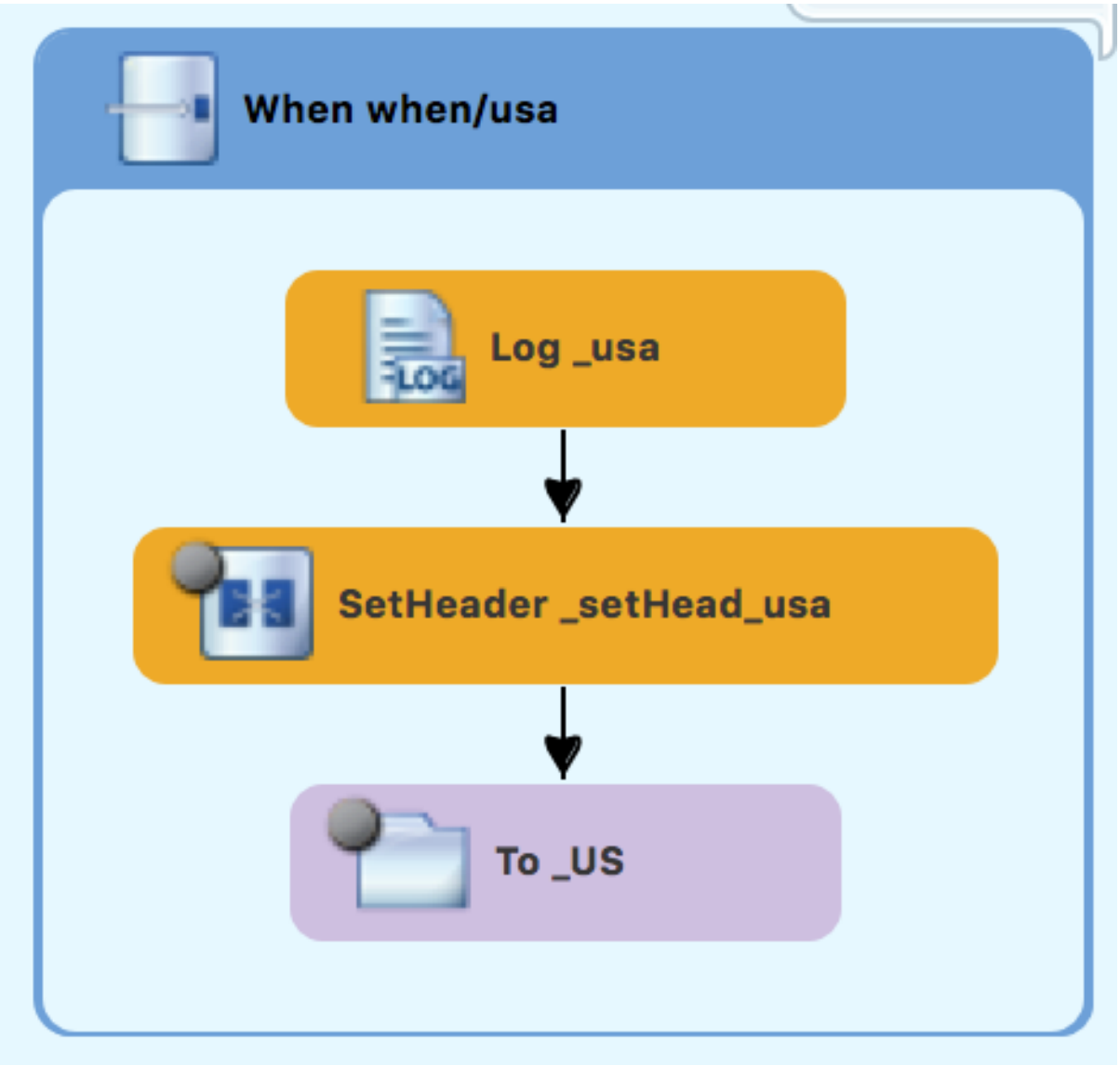
7. On the canvas, select the **Log_log3** node to open its properties in the **Properties** view:



8. In the **Properties** view:
   - In the **Message** field, type **Valid order - ship animals to USA customer** .
   - In the **Id** field, type **_usa**.
   - Leave **Logging Level** as is.



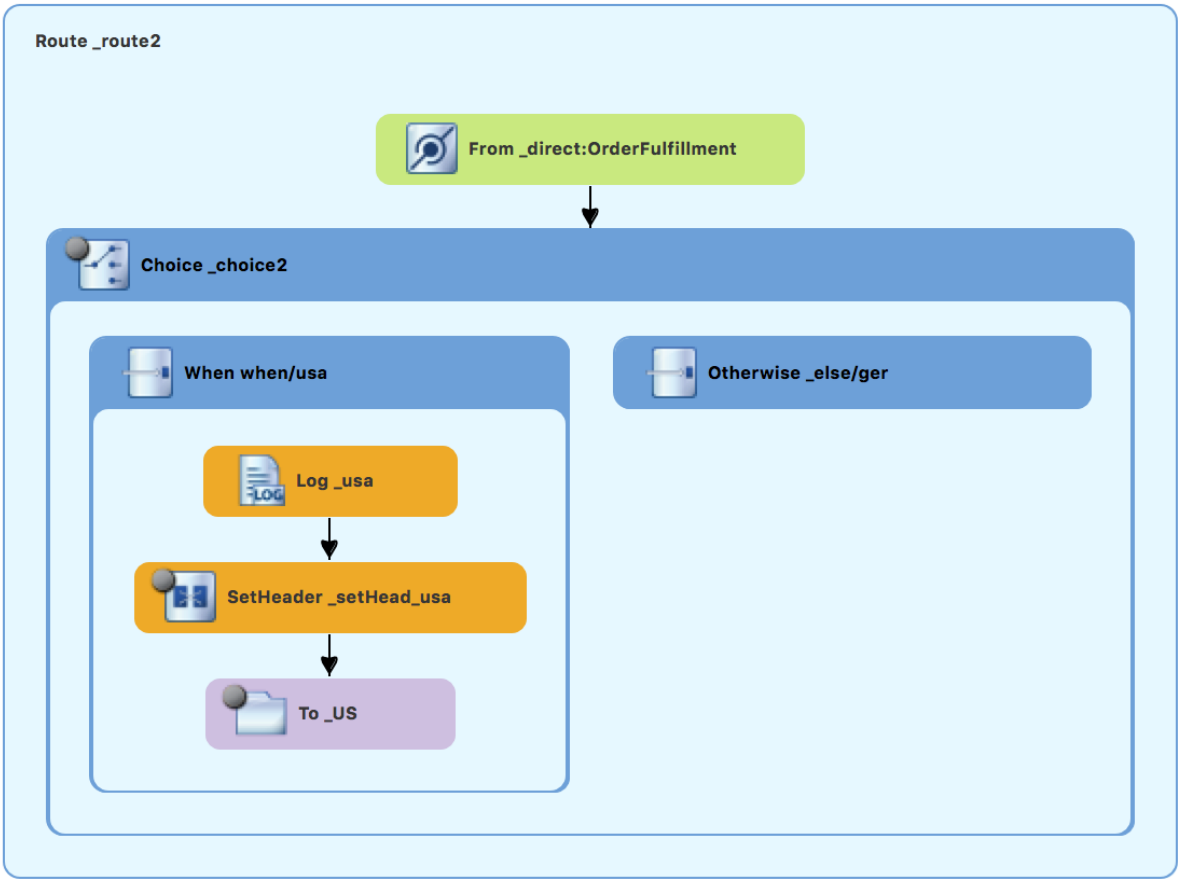
9. **Save** the file.
   The USA branch of **Route_route2** should look like this:



CONFIGURING AN OTHERWISE BRANCH TO PROCESS GERMANY ORDERS

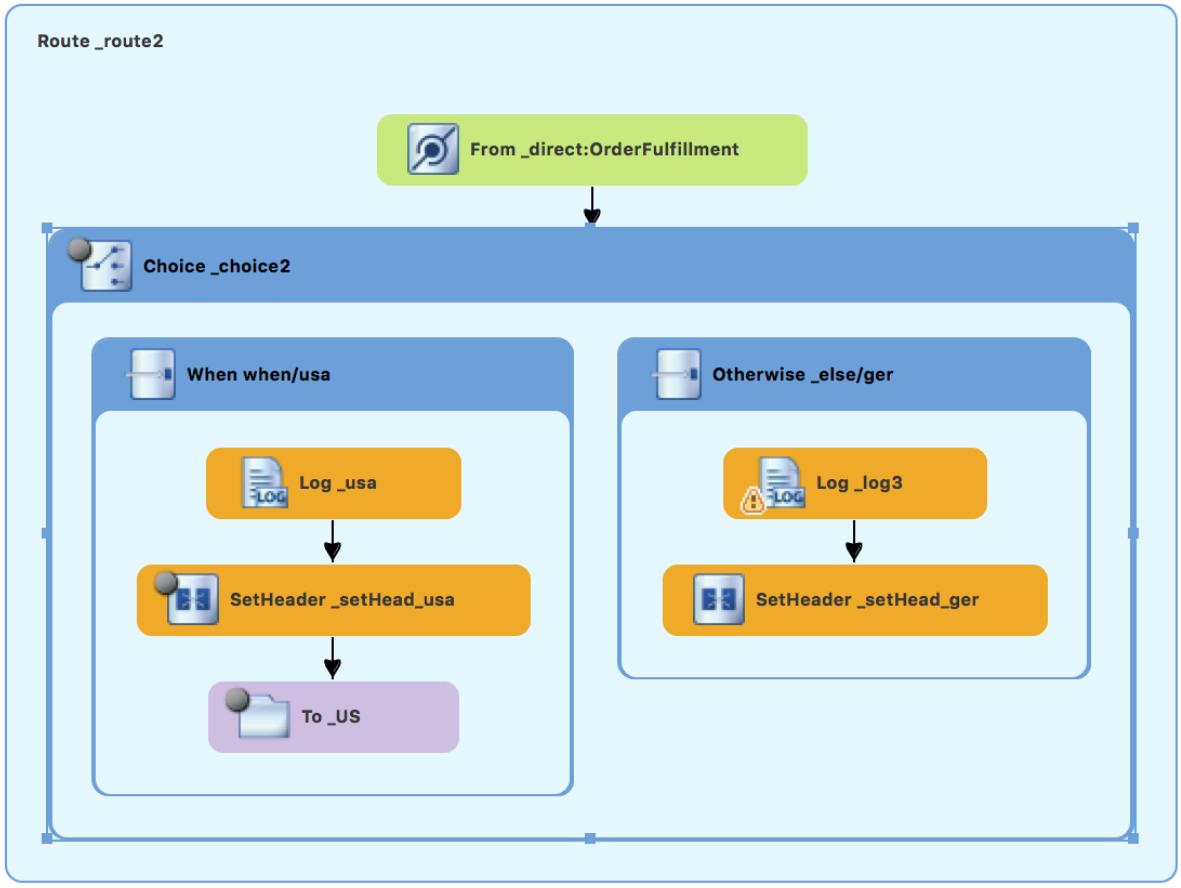
With **Route_route2** displayed on the canvas:

1. In the **Palette**, open the **Routing** drawer and then select the **Otherwise** pattern ( ).
2. In the canvas, click the **Choice_choice2** container.
   The **Choice_choice2** container expands to accommodate the **Otherwise_otherwise1** node.



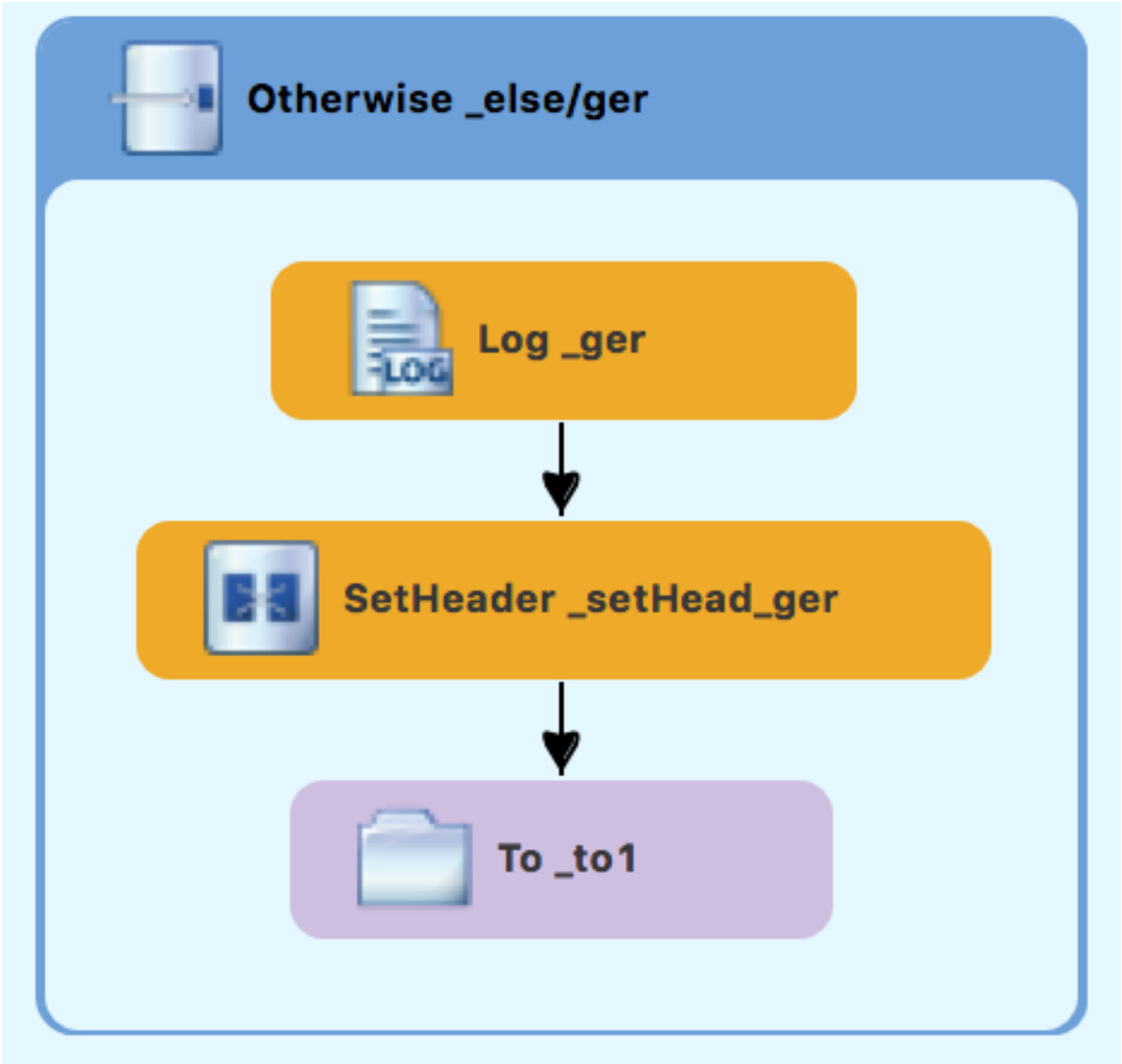
3. Select the **Otherwise_otherwise1** node to open its properties in the **Properties** view.
4. In the **Properties** view, enter **_else/ger** for the **Id** field.
5. In the **Palette**, open the **Transformation** drawer and then select the **Set Header** pattern ( ).
6. In the canvas, click the **Otherwise_else/ger** node.
   The **Otherwise_else/ger** container expands to accommodate the **SetHeader_setHeader3** node.

7. On the canvas, select the **SetHeader_setHeader3** node to open its properties in the **Properties** view.
8. In the **Properties** view:
   - From the **Expression** drop-down list, select **constant**.
   - In the second **Expression** field, type **Germany**.
   - Leave **Trim** as is.
   - In the **Header Name** field, type **Destination**.
   - In the second **Id** field, type **_setHead_ger**.
9. In the **Palette**, open the **Components** drawer and then select the **Log** pattern ( ).
10. In the canvas, click below the **SetHeader_setHead_ger** node.
    The **Otherwise_else/ger** container expands to accommodate the **Log_log3** node. If needed, drag the connector error from the **Log_log3** node to the **SetHeader_setHead_ger** node:



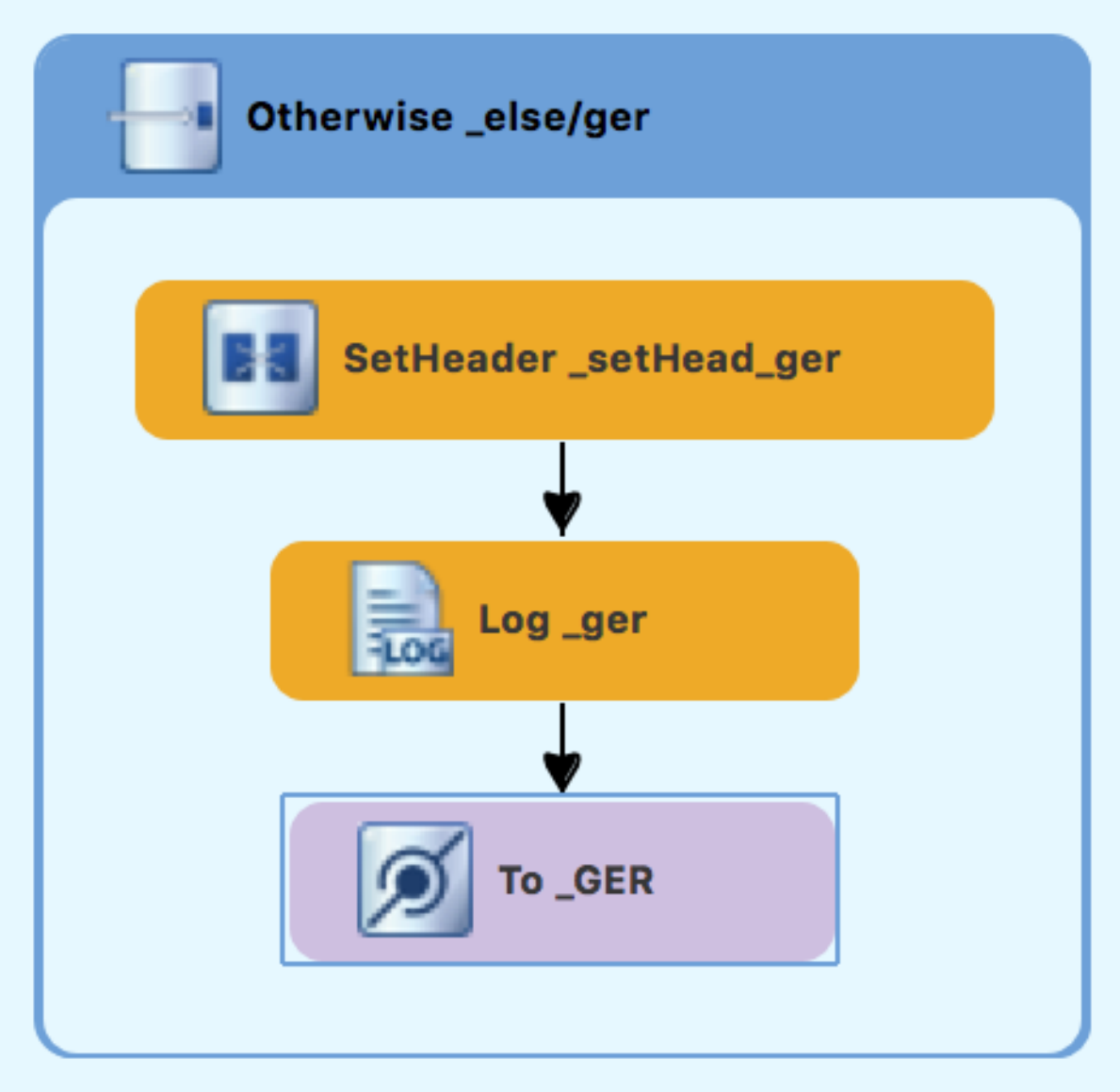
11. On the canvas, select the **Log_log3** node to open its properties in the **Properties** view.
12. In the **Properties** view:
    - In the **Message** field, type **Valid order - ship animals to Germany customer**.
    - In the **Id** field, type **_ger**.
    - Leave the **Logging Level** as is.

13. In the **Components** drawer, select a **File** pattern ( ) and then click below the **Log_ger** node.
The **Otherwise_else/ger** container expands to accommodate the **To_to1** node. If needed, drag the connector error from the **SetHeader_setHead_ger** node to the **To_to1** node:



14. On the canvas, select the **To_to1** node to open its properties in the **Properties** view.
15. In the **Properties** view:
    - In the **Uri** field, replace **directoryName** with **target/messages/validOrders/Germany**
    - In the **Id** field, type **_GER**.
16. **Save** the file.

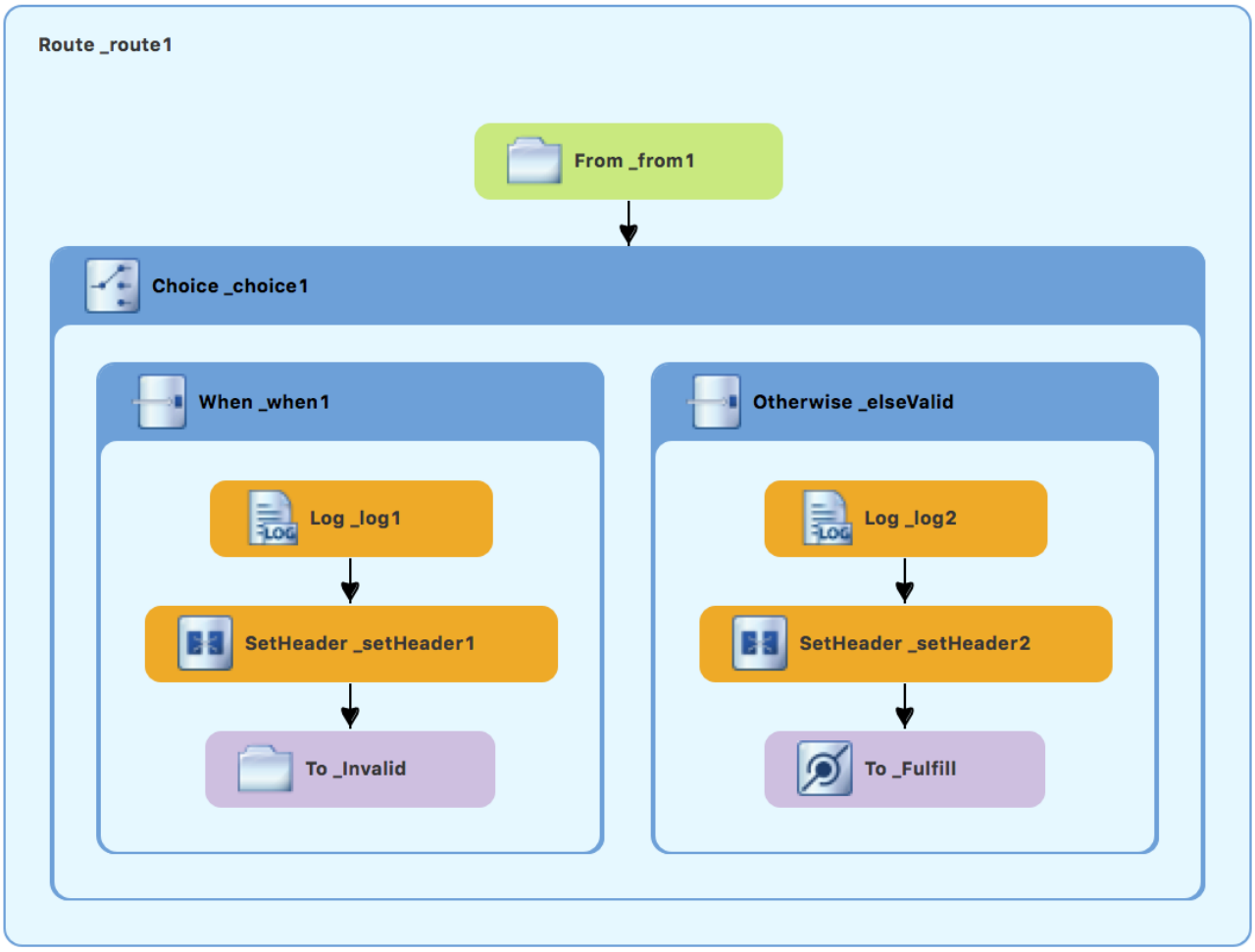
The Germany branch of **Route_route2** should look like this:



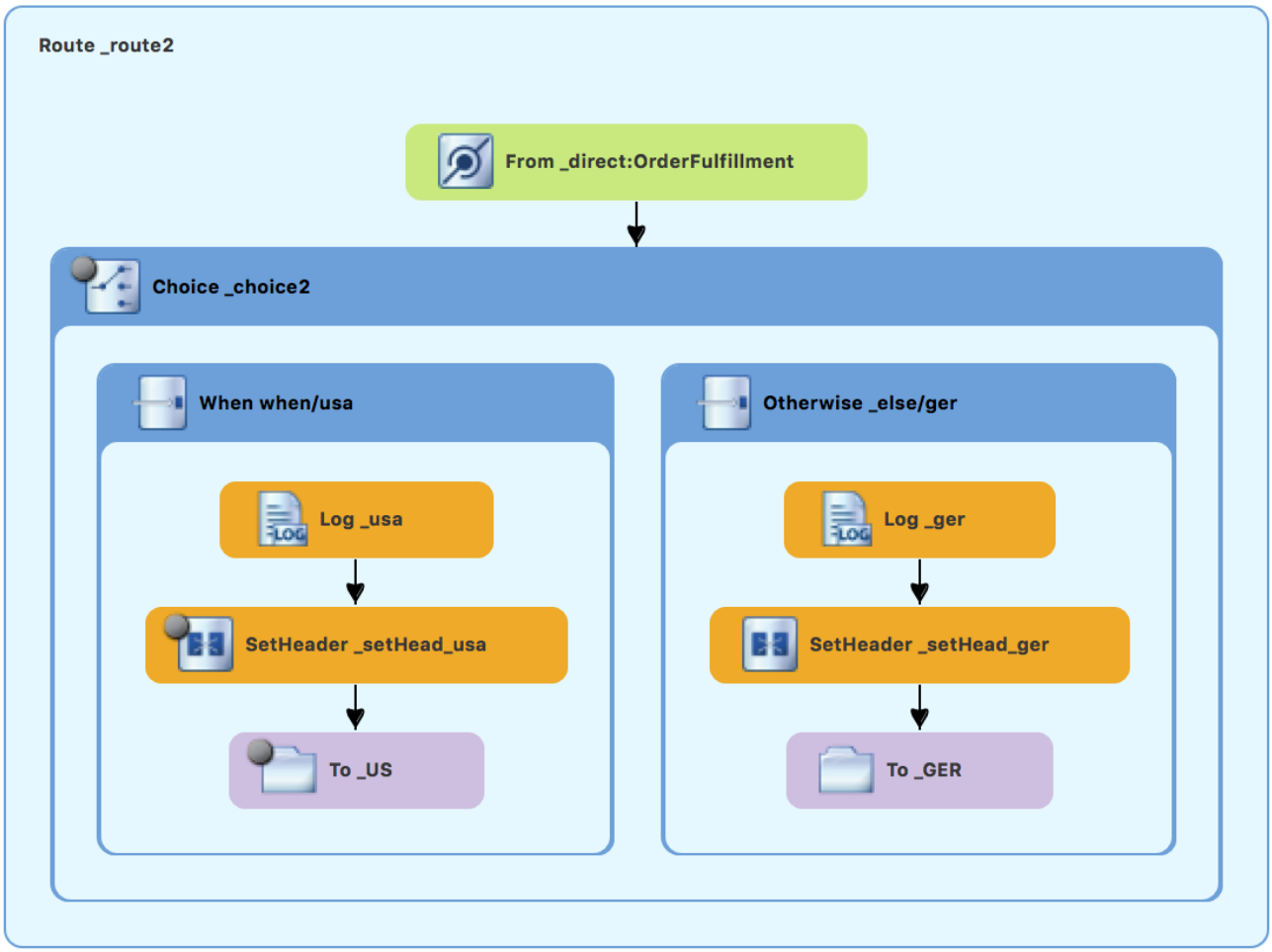
VERIFYING THE SECOND ROUTE

The routes on the canvas should look like this:

Completed route1



Completed route2



In the **Source** tab at the bottom of the canvas, the XML for the camelContext element should look like that shown in Example 6.1, "XML for dual-route content-based router" :

Example 6.1. 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() &gt; 10</xpath>
           <log id="_log1" message="The quantity requested exceeds the maximum allowed -
contact customer."/>
           <setHeader headerName="Destination" id="_setHeader1">
             <constant>Invalid</constant>
           </setHeader>
           <to id="_Invalid" uri="file:target/messages/invalidOrders"/>
         </when>
```

```
            <otherwise id="_elseValid">
              <log id="_log2" message="This is a valid order - OK to process."/>
              <setHeader headerName="Destination" id="_setHeader2">
                <constant>ReadyForDispatcher</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>
              <log id="_usa" message="Valid order - ship animals to USA customer"/>
              <setHeader headerName="Destination" id="_setHead_usa">
                <constant>USA</constant>
              </setHeader>
              <to id="_US" uri="file:target/messages/validOrders/USA"/>
            </when>
            <otherwise id="_else/ger">
              <log id="_ger" message="Valid order - ship animals to Germany customer"/>
              <setHeader headerName="Destination" id="_setHead_ger">
                <constant>Germany</constant>
              </setHeader>
              <to id="_GER" uri="file:target/messages/validOrders/Germany"/>
            </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 **ZooOrderApp** project might fail to run.

To make sure that your updated project works as expected, follow these steps:

1. Run the **ZooOrderApp/Camel Contexts/blueprint.xml** as a local Camel Context (without tests).

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

```
[ Blueprint Event Dispatcher: 1] BlueprintCamelContext          INFO  Route: _route1 started and consuming from: file://src/data?noop=true
[ Blueprint Event Dispatcher: 1] BlueprintCamelContext          INFO  Route: _route2 started and consuming from: direct://OrderFulfillment
[ Blueprint Event Dispatcher: 1] BlueprintCamelContext          INFO  Total 2 routes, of which 2 are started
[ Blueprint Event Dispatcher: 1] BlueprintCamelContext          INFO  Apache Camel 2.21.0.fuse-000112-redhat-3 (CamelContext: _context1) started in 0.318
[2.redhat.com:1099/jmxrmi/camel] DefaultManagementAgent         INFO  JMX Connector thread started and listening at: service:jmx:rmi:///jndi/rmi://ovpn-1
[1) thread #4 - file://src/data] _route1                        INFO  This is a valid order - OK to process.
[1) thread #4 - file://src/data] _route2                        INFO  Valid order - ship animals to USA customer
[1) thread #4 - file://src/data] _route1                        INFO  This is a valid order - OK to process.
[1) thread #4 - file://src/data] _route2                        INFO  Valid order - ship animals to Germany customer
[1) thread #4 - file://src/data] _route1                        INFO  This is a valid order - OK to process.
[1) thread #4 - file://src/data] _route2                        INFO  Valid order - ship animals to USA customer
[1) thread #4 - file://src/data] _route1                        INFO  The quantity requested exceeds the maximum allowed - contact customer.
[1) thread #4 - file://src/data] _route1                        INFO  This is a valid order - OK to process.
[1) thread #4 - file://src/data] _route2                        INFO  Valid order - ship animals to USA customer
[1) thread #4 - file://src/data] _route1                        INFO  The quantity requested exceeds the maximum allowed - contact customer.
```

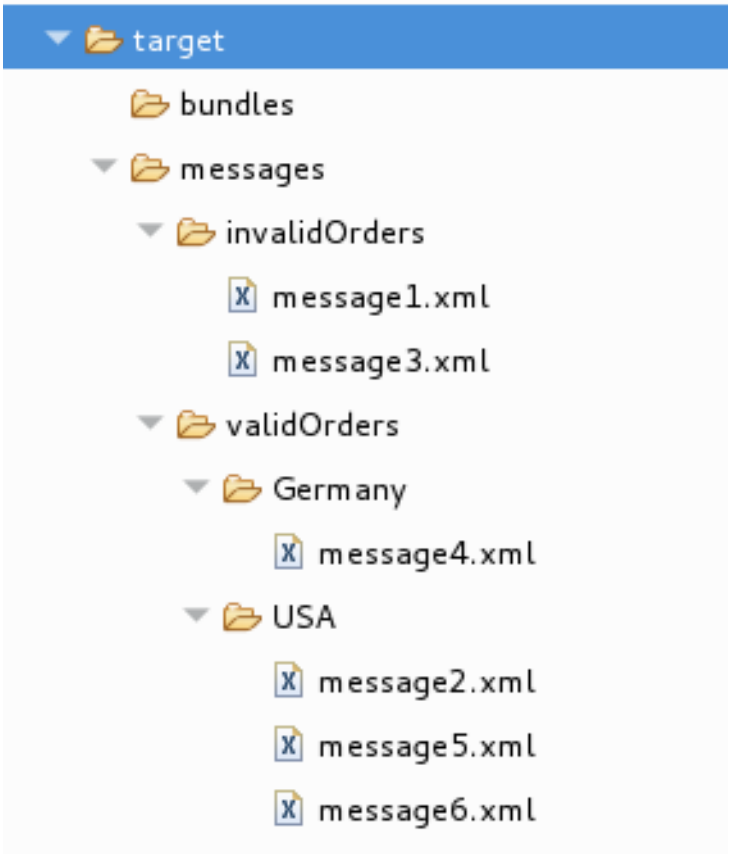
3. Check the target destination folders to verify that the routes executed properly:

   a. In **Project Explorer**, right-click **ZooOrderApp** and then select **Refresh**.

a. In **Project Explorer**, right-click **ZooOrderApp** and then select **Refresh**.

b. Expand the **target/messages/** folder.
The **message*.xml** files should be dispersed in your the destinations as shown:

Figure 6.1. Target message destinations in Project Explorer



NEXT STEPS

In the next tutorial, Chapter 7, *Debugging a routing context* , you learn how to use the Fuse Tooling debugger.

CHAPTER 7. DEBUGGING A ROUTING CONTEXT

This tutorial shows how to use the Camel debugger to find logic errors for a locally running routing context.

GOALS

In this tutorial you complete the following tasks:

- Set breakpoints on the nodes of interest in the two routes
- In the Debug perspective, step through the routes and examine the values of message variables
- Step through the routes again, changing the value of a message variable and observing the effect

PREREQUISITES

To start this tutorial, you need the **ZooOrderApp** project resulting from one of the following:

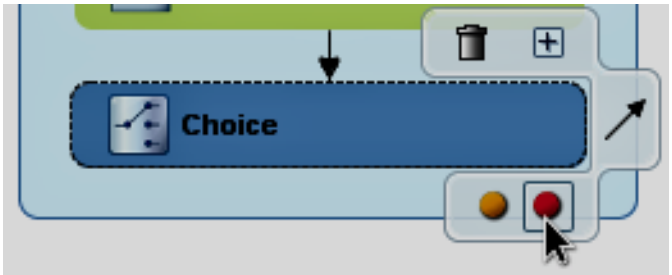
- Complete the Chapter 6, *Adding another route to the routing context* tutorial.
  or
- Complete the Chapter 2, *Setting up your environment* tutorial and replace your project's **blueprint.xml** file with the provided **blueprintContexts/blueprint3.xml** file, as described in the section called "About the resource files".

SETTING BREAKPOINTS

In the Debugger, you can set both conditional and unconditional breakpoints. In this tutorial, you only set unconditional breakpoints. To learn how to set conditional breakpoints (that are triggered when a specific condition is met during the debugging session), see the Tooling User Guide.

To set unconditional breakpoints:

1. If necessary, open your **ZooOrderApp/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/blueprint.xml** 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 icon, respectively. You can delete all set breakpoints by right-clicking the canvas and selecting **Delete all breakpoints**.

5. Set unconditional breakpoints 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. Set unconditional breakpoints on the following **Route_Route2** nodes:
   - **Choice_choice2**
   - **SetHeader_setHead_usa**
   - **Log_usa**
   - **To_US**
   - **SetHeader_setHead_ger**
   - **Log_ger**
   - **To_GER**

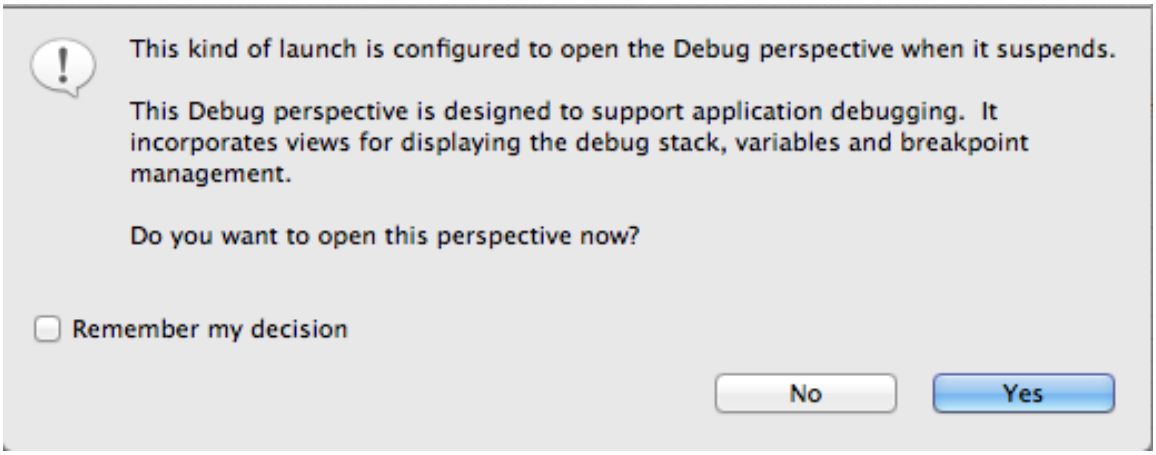
STEPPING THROUGH THE ROUTING CONTEXT

You can step through the routing context in two ways:

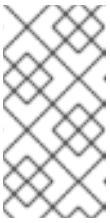
- Step over ( ) - Jumps to the next node of execution in the routing context, regardless of breakpoints.
- Resume ( ) - Jumps to the next active breakpoint in the routing context.

1. In **Project Explorer**, expand the **ZooOrderApp** project's **Camel Contexts** folder to expose the **blueprint.xml** file.

2. Right-click the **blueprint.xml** file to open its context menu, and then click **Debug As → Local Camel Context (without tests)**
   The Camel debugger suspends execution at the first breakpoint it encounters and asks whether you want to open the **Debug** perspective now:



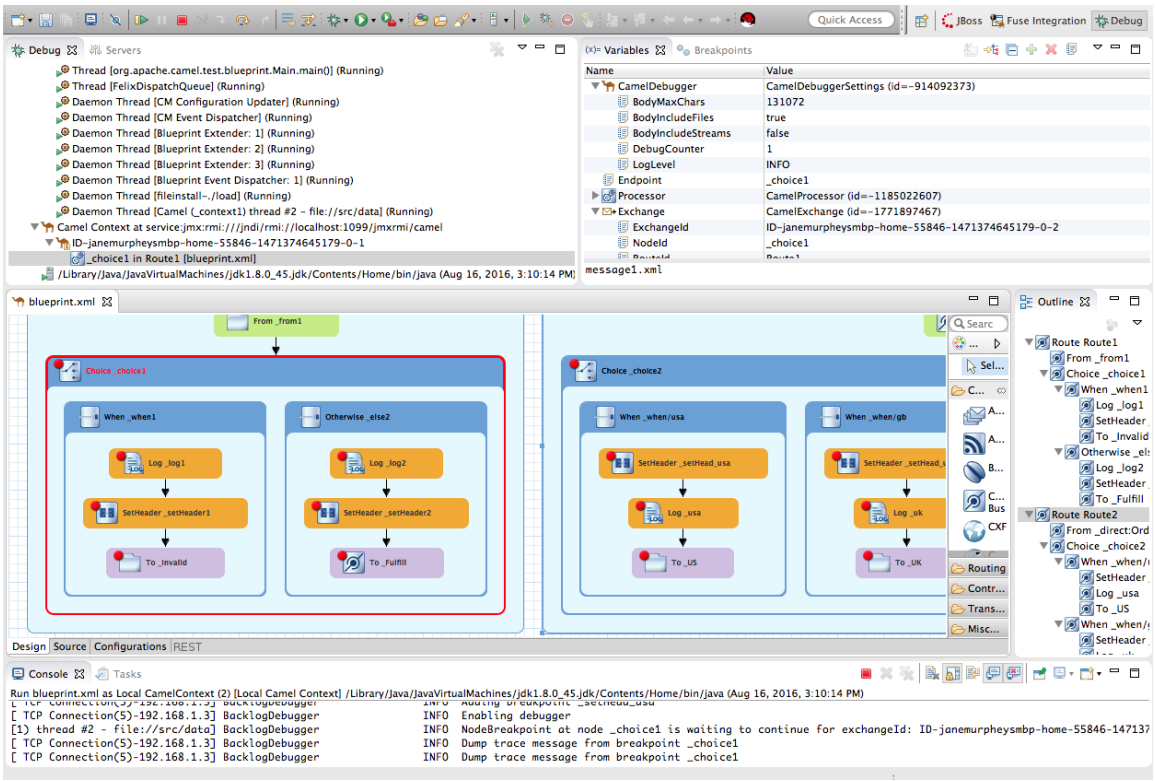
3. Click **Yes**.



NOTE

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

The **Debug** perspective opens with the routing context suspended at **_choice1 in _route1 [blueprint.xml]** as shown in the **Debug** view:

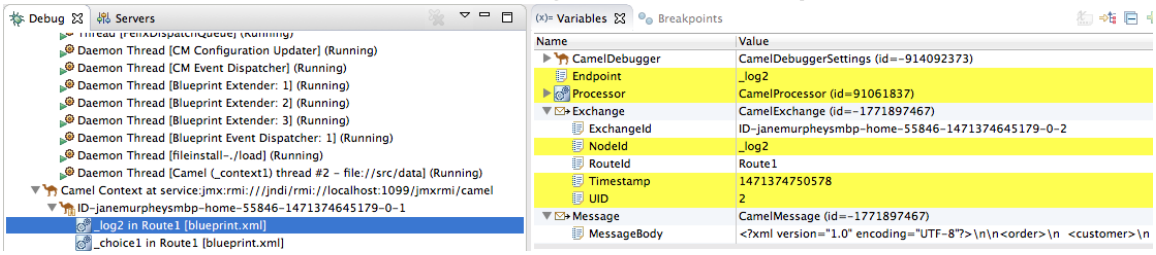


NOTE

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 might need to expand the nodes at each breakpoint to reveal variables that have changed.

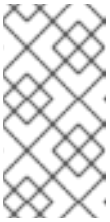
5. Click to step to the next breakpoint, **_log2 in _route1 [blueprint.xml]**:



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 (highlighted in yellow) since the breakpoint at **_log2 in Route1 [blueprint.xml]**.

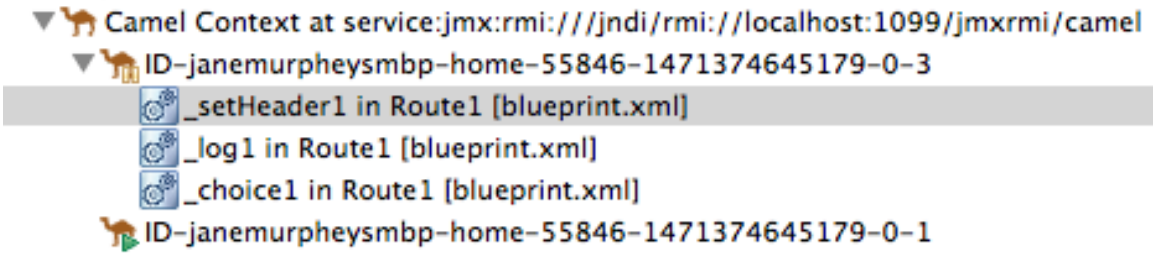
8. In the **Debug** view, click **_log2 in _route1 [blueprint.xml]** to populate the **Variables** view with the variable values from the breakpoint **_log2 in _route1 [blueprint.xml]** for a quick comparison.
   In the **Debug** view, you can switch between breakpoints within the same message flow to 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.

9. 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**.

10. 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)

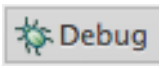
Timing metrics are in milliseconds.

11. 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.

12. In the Menu bar, click to terminate the Camel debugger. The Console terminates, but you must manually clear the output.

NOTE

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

13. In the Menu bar, right-click 

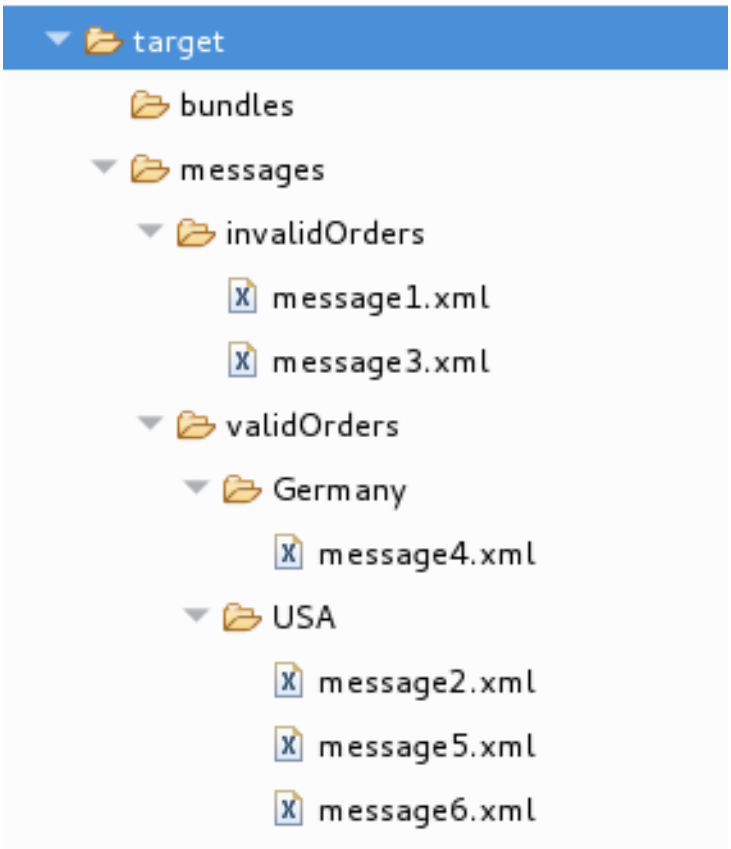
to open the context menu, and then select **Close** to close **Debug** perspective.
CodeReady Studio automatically returns to the perspective from which you launched the Camel debugger.

14. In **Project Explorer**, right-click the project and then 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 **ZooOrderApp/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.

15. Expand the **ZooOrderApp/target/messages/** directories to check that the messages were delivered to their expected destinations:



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

CHANGING THE VALUE OF A VARIABLE

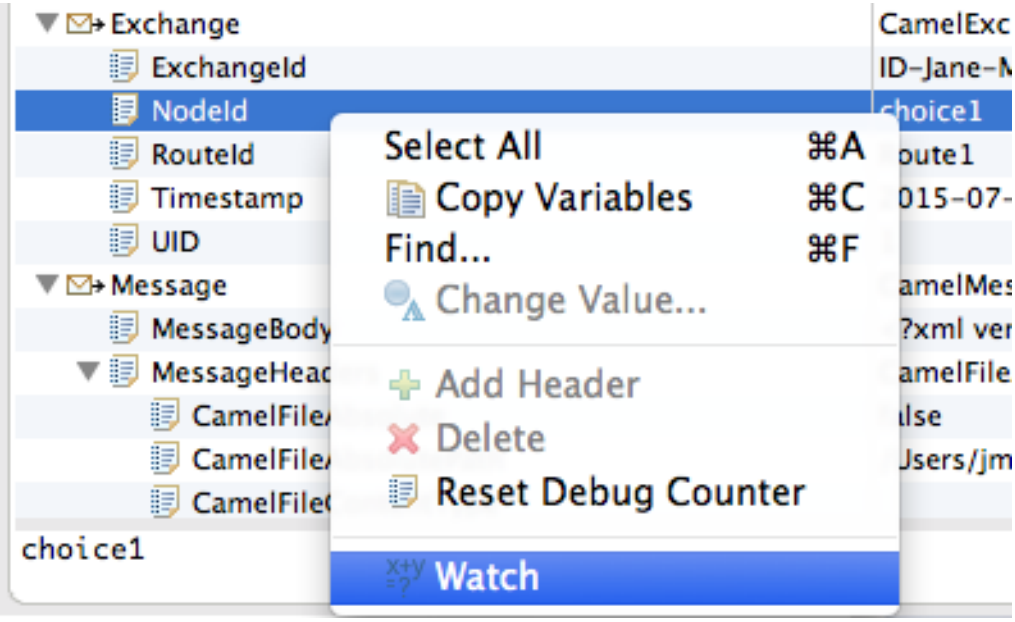
In this section, you add variables to a watch list to easily check how their values change as messages pass through the routing context. You change the value of a variable in the body of a message and then observe how the change affects the message's route through the routing context.

1. To rerun the Camel debugger on the **ZooOrderApp** project, right-click the **blueprint.xml** file and then click **Debug As → Local Camel Context (without tests)**

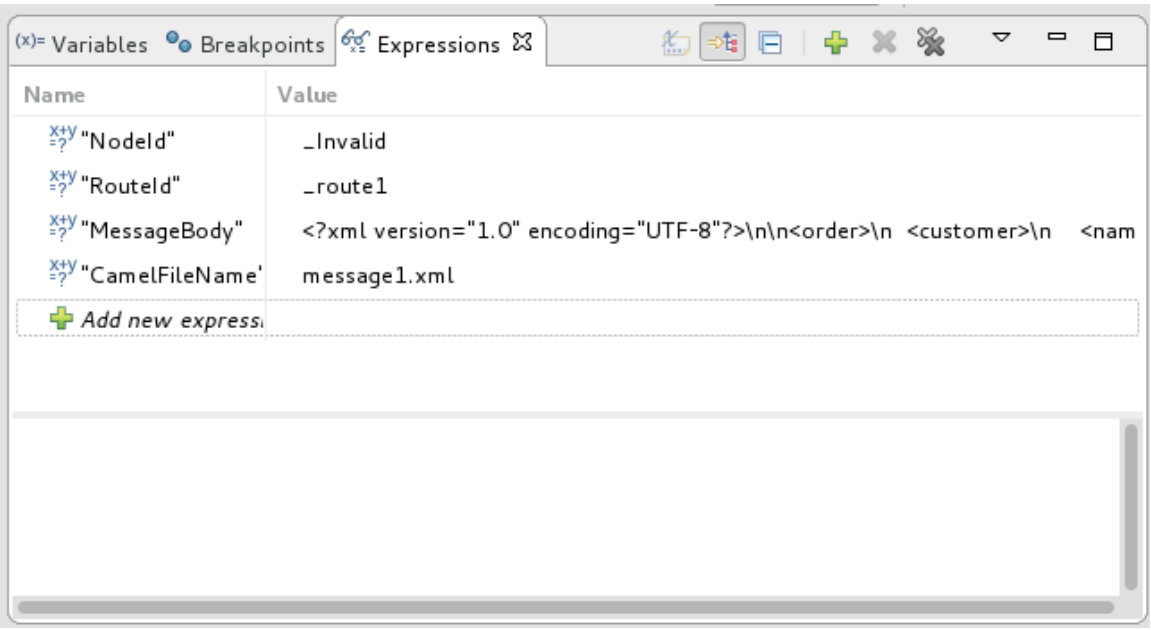
2. With **message1** stopped at the first breakpoint, **_choice1 in _route1 [blueprint.xml]**, add the variables **NodeId** and **RouteId** (in the **Exchange** category) and **MessageBody** and **CamelFileName** (in the **Message** category) to the watch list.
   For each of the four variables:

   a. In the **Variables** view, expand the appropriate category to expose the target variable:

   b. 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:



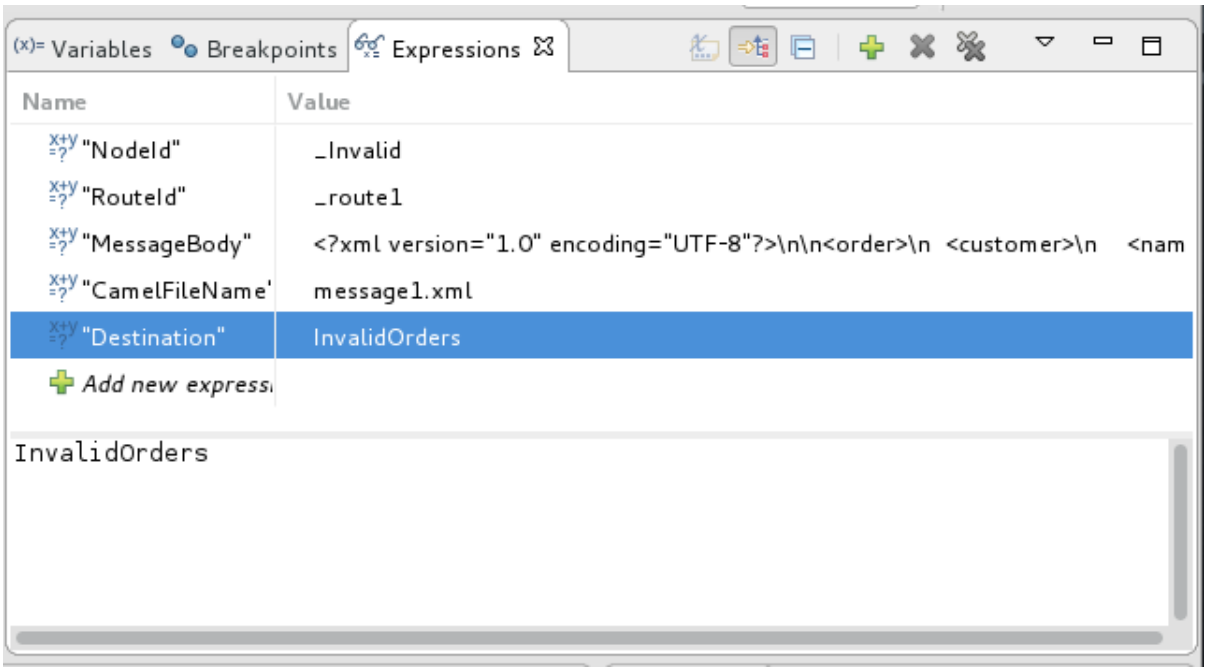
NOTE

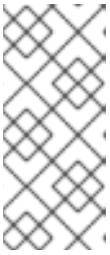
Creating a watch list makes it easy for you to quickly check the current value of multiple variables of interest.

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. Add the variable **Destination** to the watch list.
The **Expressions** view should now contain these variables:





NOTE

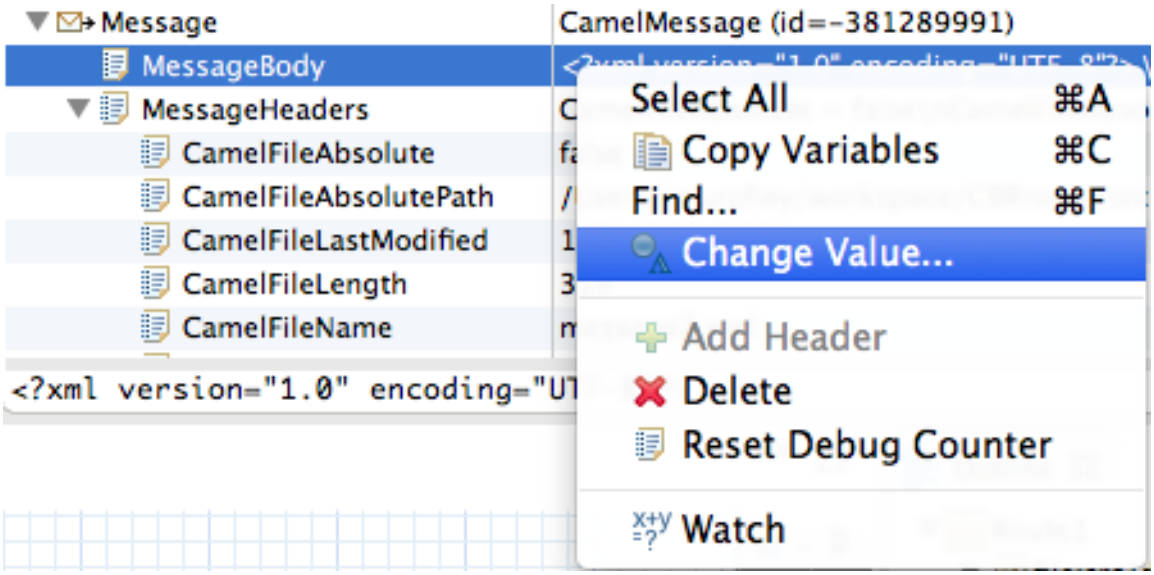
- The pane below the list of variables displays the value of the selected variable.
- The **Expressions** view retains all variables that you add to the list until you explicitly remove them.

6. Step **message1** through the rest of the routing context and then step **message2** all of the way through.

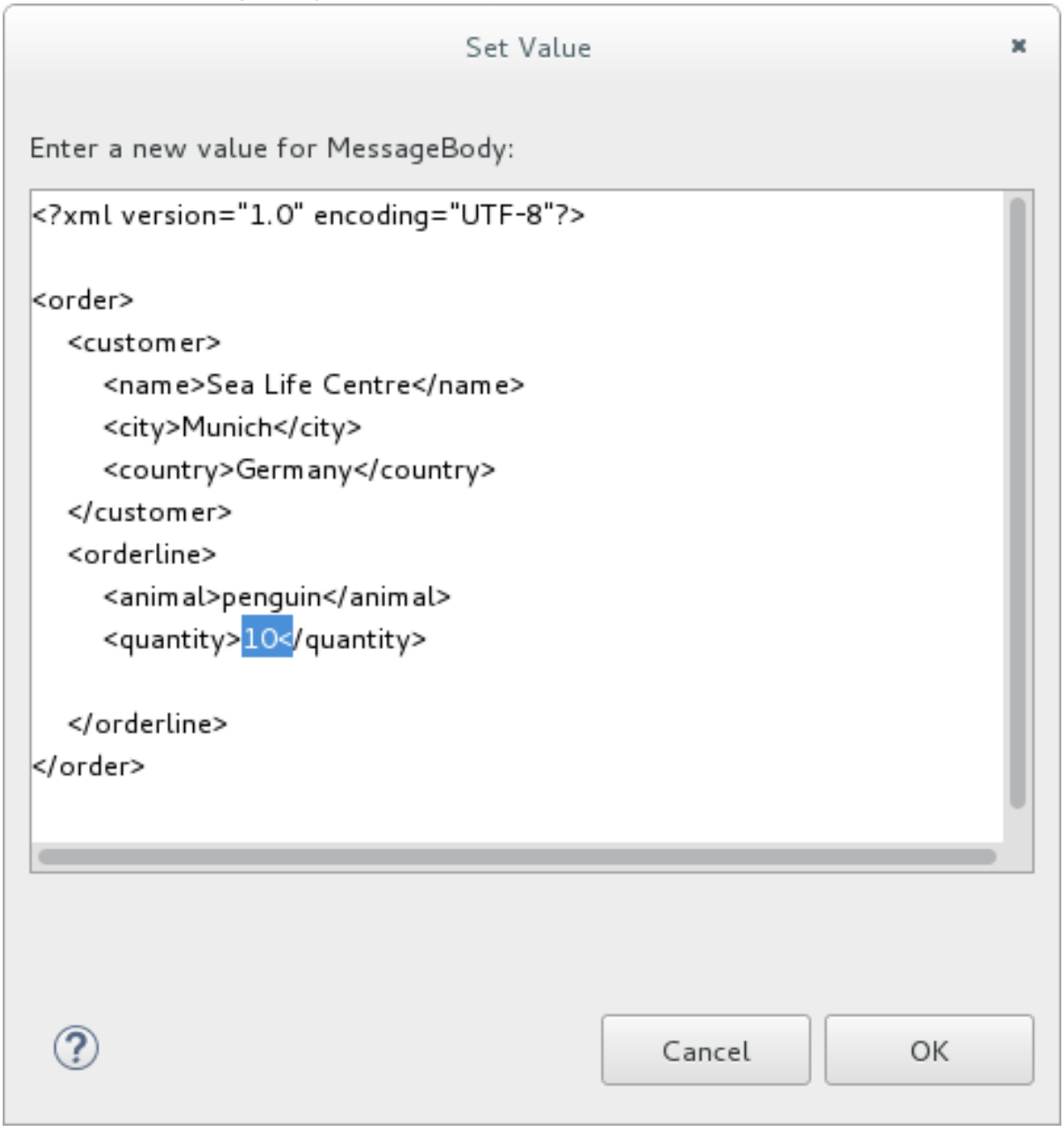
7. Stop **message3** at **_choice1 in _route1 [blueprint.xml]**.

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**:



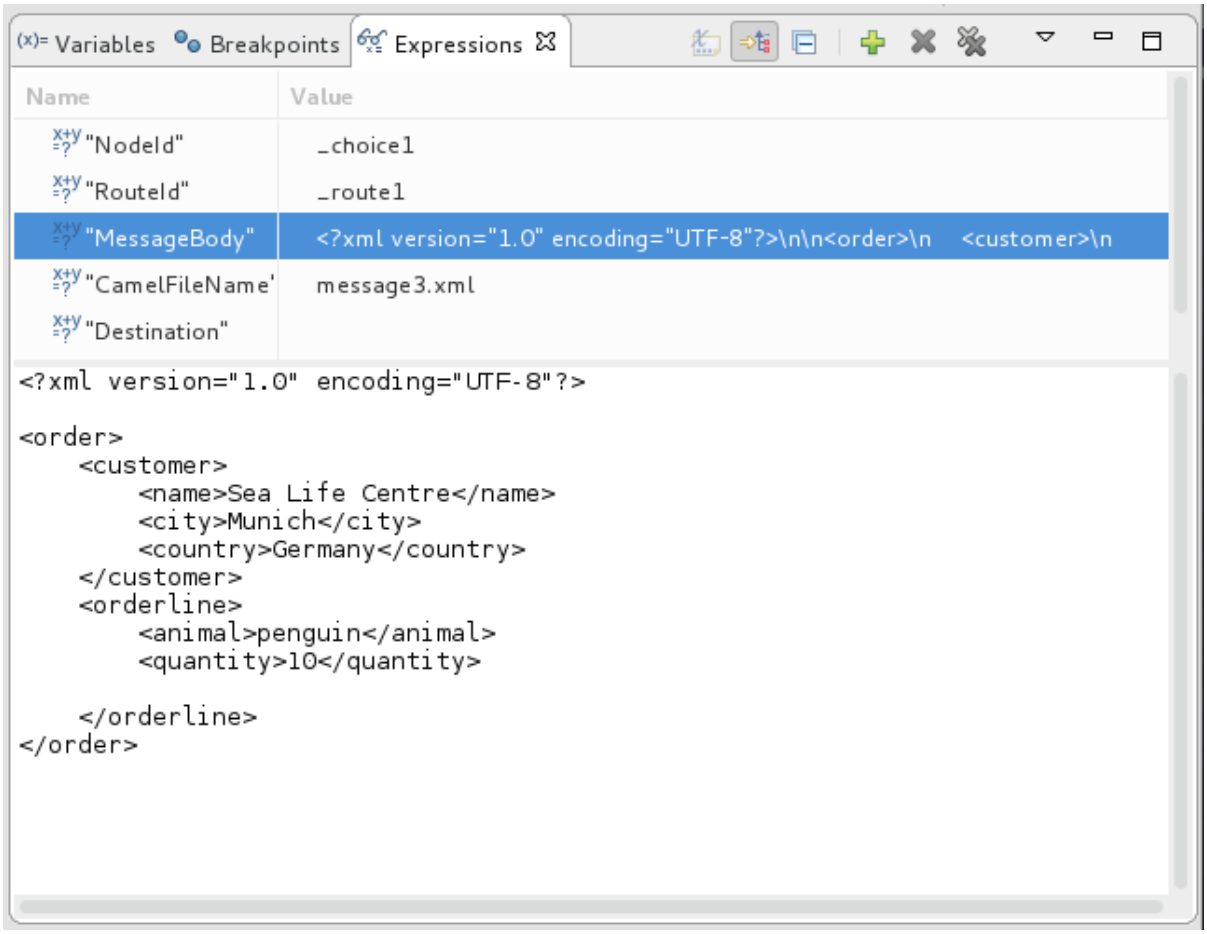
10. Change the value of **quantity** from **15** to **10** (to change it from an invalid order to a valid order):



This changes the in-memory value only (it does not edit the **message3.xml** file).

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 **message3**, making it easy to check the current value of order items:



13. Click to step to the next breakpoint.
Instead of following the branch leading to **To_Invalid**, **message3** now follows the branch leading to**To_Fulfill** and **Route_route2**.

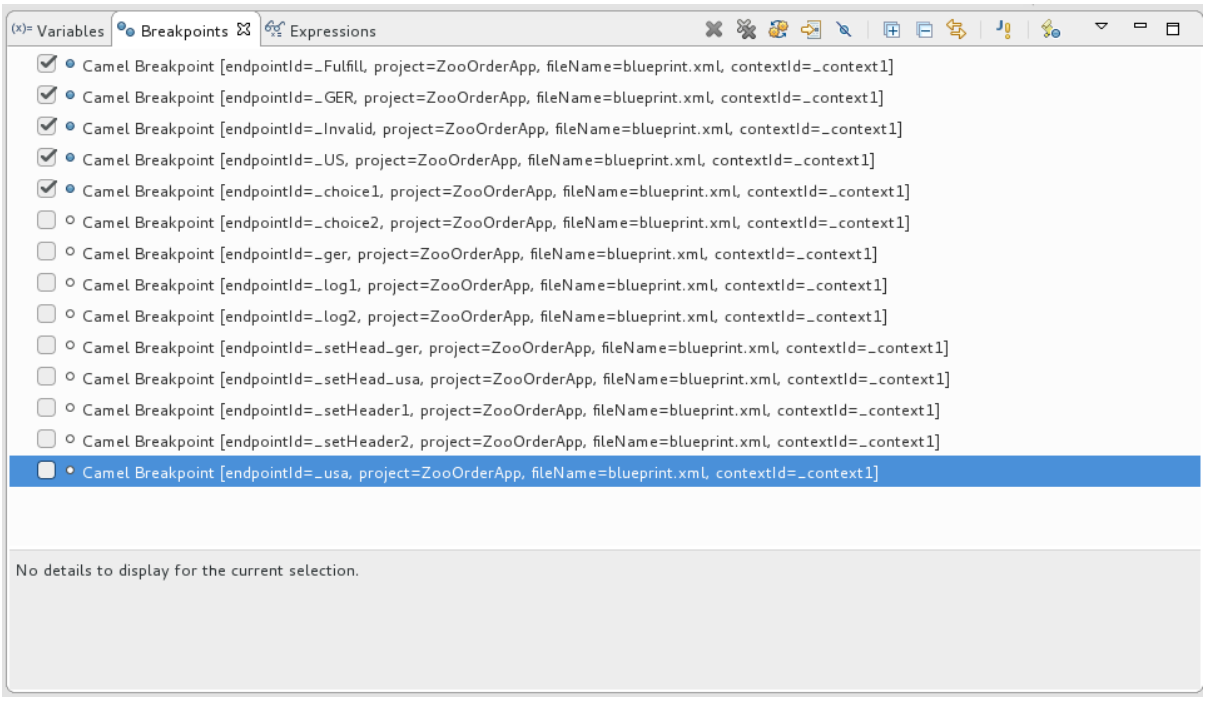
NARROWING THE CAMEL DEBUGGER'S FOCUS

You can temporarily narrow and then re-expand the debugger's focus by disabling and re-enabling breakpoints:

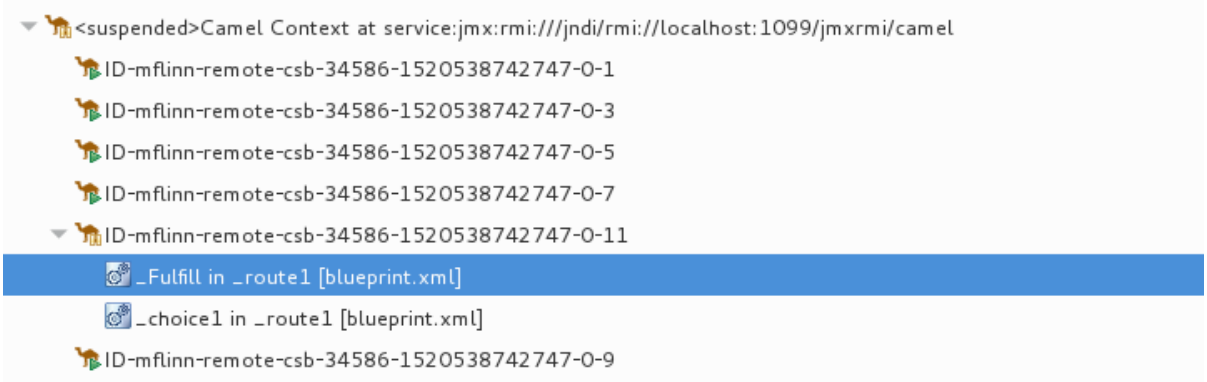
1. Step **message4** through the routing context, checking the **Debug** view, the **Variables** view, and the **Console** output at each step.

2. Stop **message4** at **_choice1 in _route1 [blueprint.xml]**.

3. Switch to the **Breakpoints** view, and clear each check box next to the breakpoints listed below **_choice1**. Clearing the check box of a breakpoint temporarily disables it.

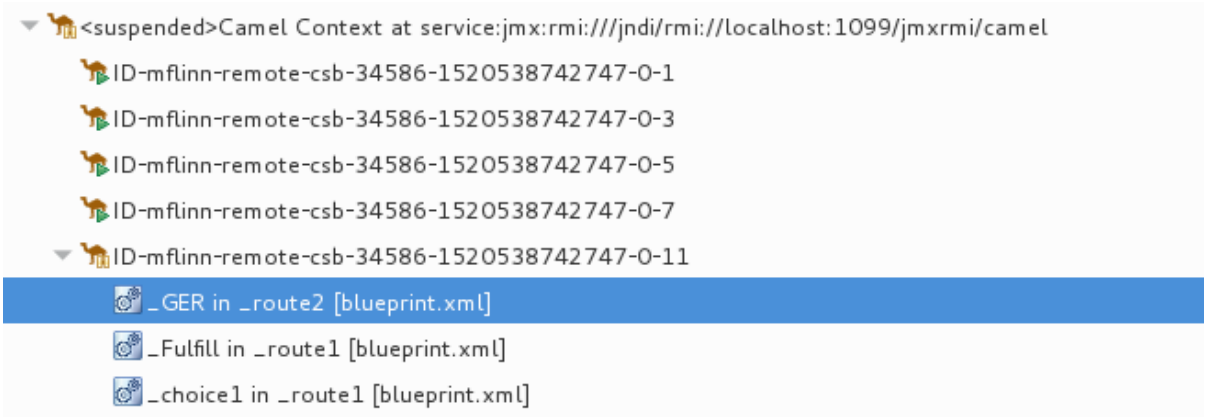


4. Click to step to the next breakpoint:



The debugger skips over the disabled breakpoints and jumps to **_FulFill in _route1 [blueprint.xml]**.

5. Click again to step to the next breakpoint:



The debugger jumps to **_GER in _route2 [blueprint.xml]**.

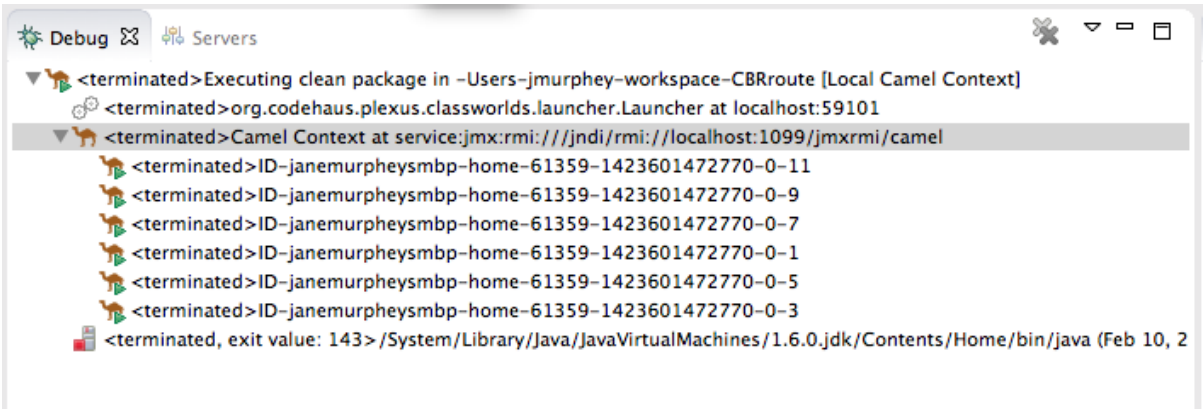
6. Click repeatedly to quickly step **message5** and **message6** through the routing context.

7. Switch to the **Breakpoints** view, and check the boxes next to all breakpoints to reenable them.

VERIFYING THE EFFECT OF CHANGING A MESSAGE VARIABLE VALUE

To stop the debugger and check the results of changing the value of `message1's quantity variable:

1. In the tool bar, click to terminate the Camel debugger:

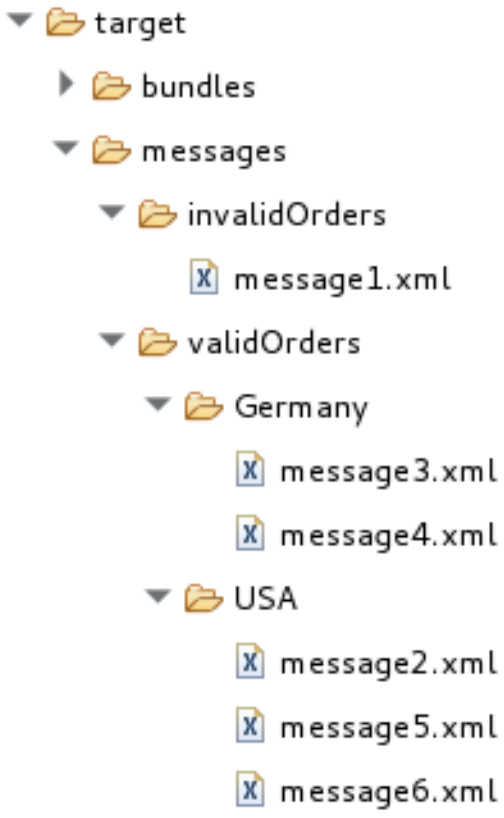


2. Click the Console's button to clear the output.

3. Close the **Debug** perspective and return to the perspective from which you launched the Camel debugger.

4. In **Project Explorer**, refresh the display.

5. Expand the **ZooOrderApp/target/messages/** directories to check whether the messages were delivered as expected:



You should see that only **message1** was sent to the **invalidOrders** and that **message3.xml** appears in the **validOrders/Germany** folder.

NEXT STEPS

In the Chapter 8, *Tracing a message through a route* tutorial, you trace messages through your routing context to determine where you can optimize and fine tune your routing context's performance.

CHAPTER 8. TRACING A MESSAGE THROUGH A ROUTE

Tracing allows you to intercept a message as it is routed from one node to another. You can trace messages through your routing context to see where you can optimize and fine tune your routing context's performance. This tutorial shows you how to trace a message through a route.

GOALS

In this tutorial you complete the following tasks:

- Run the **ZooOrderApp** in the **Fuse Integration** perspective
- Enable tracing on the **ZooOrderApp**
- Drop messages onto the **ZooOrderApp** and track them through all route nodes

PREREQUISITES

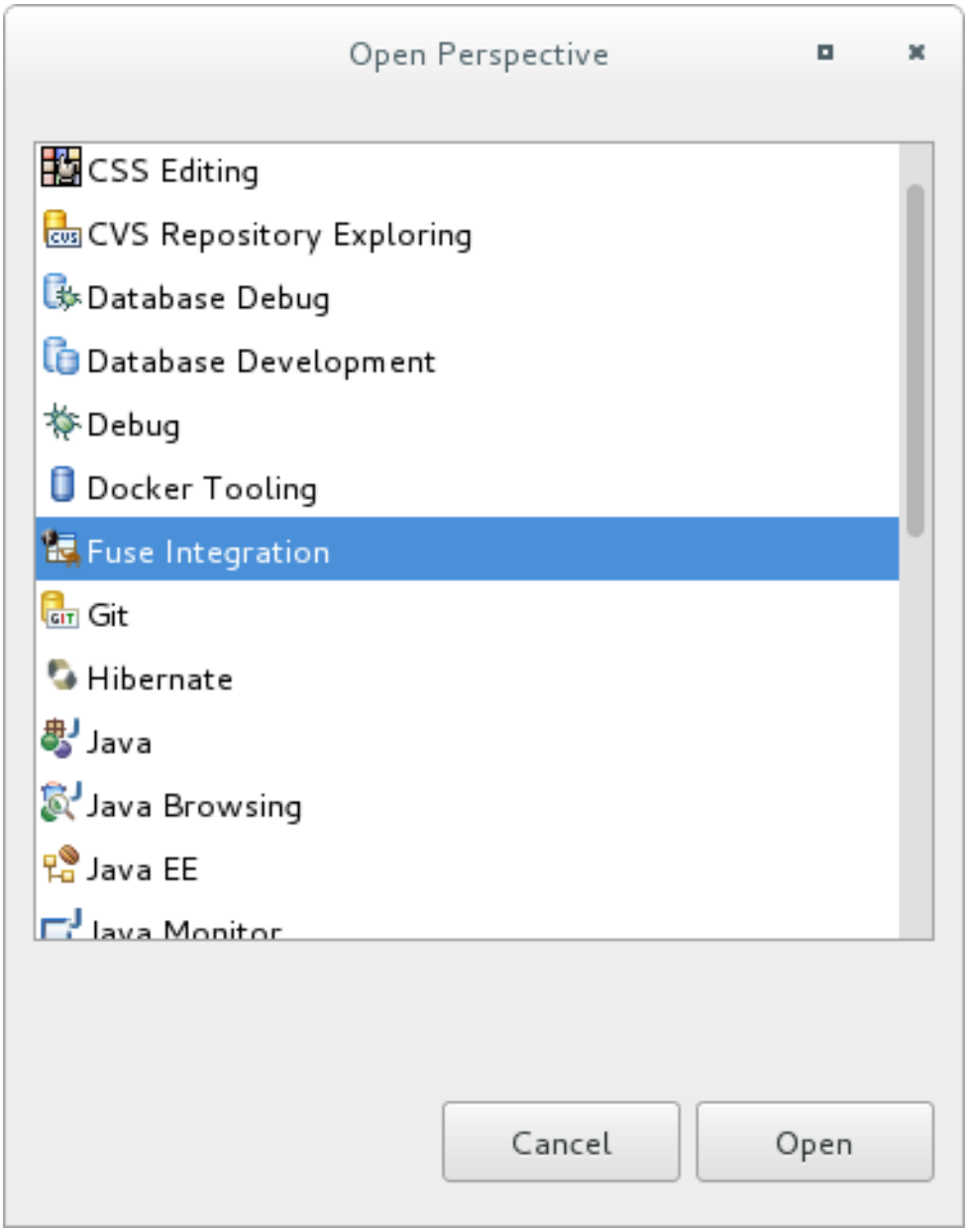
To start this tutorial, you need the **ZooOrderApp** project resulting from one of the following:

- Complete the Chapter 6, *Adding another route to the routing context* tutorial.
  or
- Complete the Chapter 2, *Setting up your environment* tutorial and replace your project's **blueprint.xml** file with the provided **blueprintContexts/blueprint3.xml** file, as described in the section called "About the resource files".

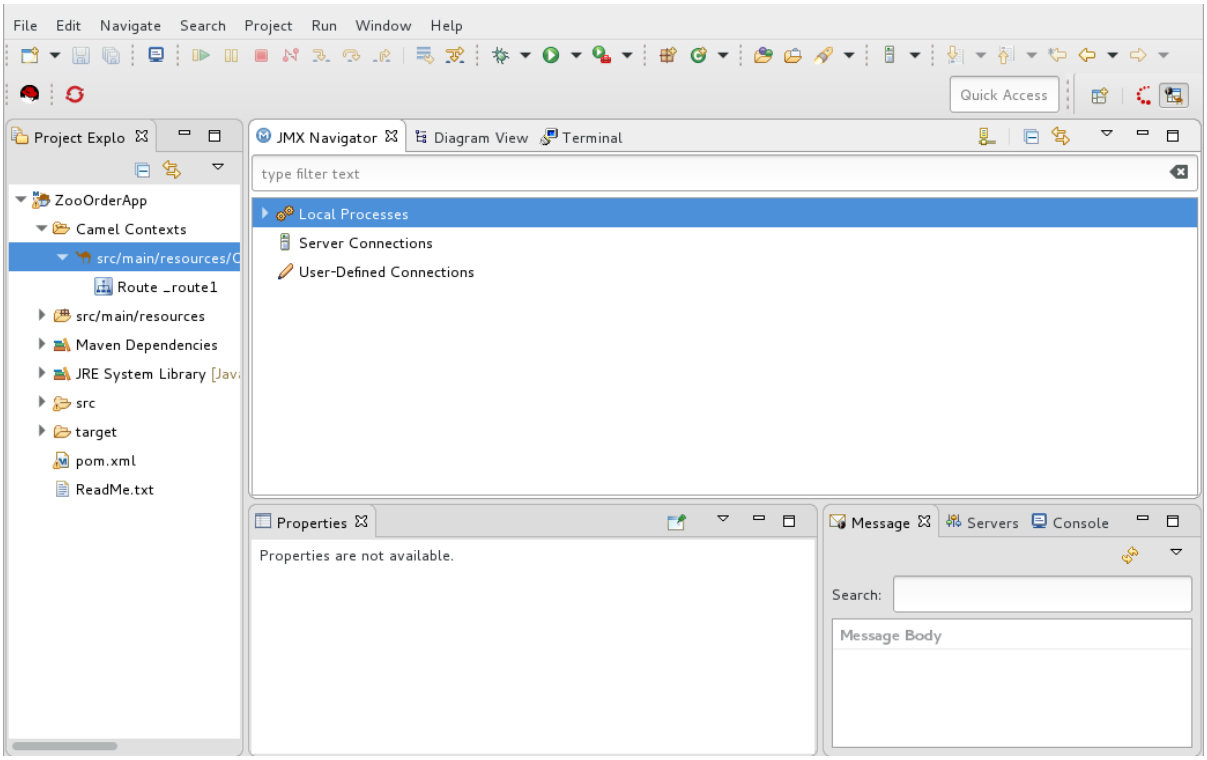
SETTING UP YOUR FUSE INTEGRATION PERSPECTIVE

To set up your workspace to facilitate message tracing:

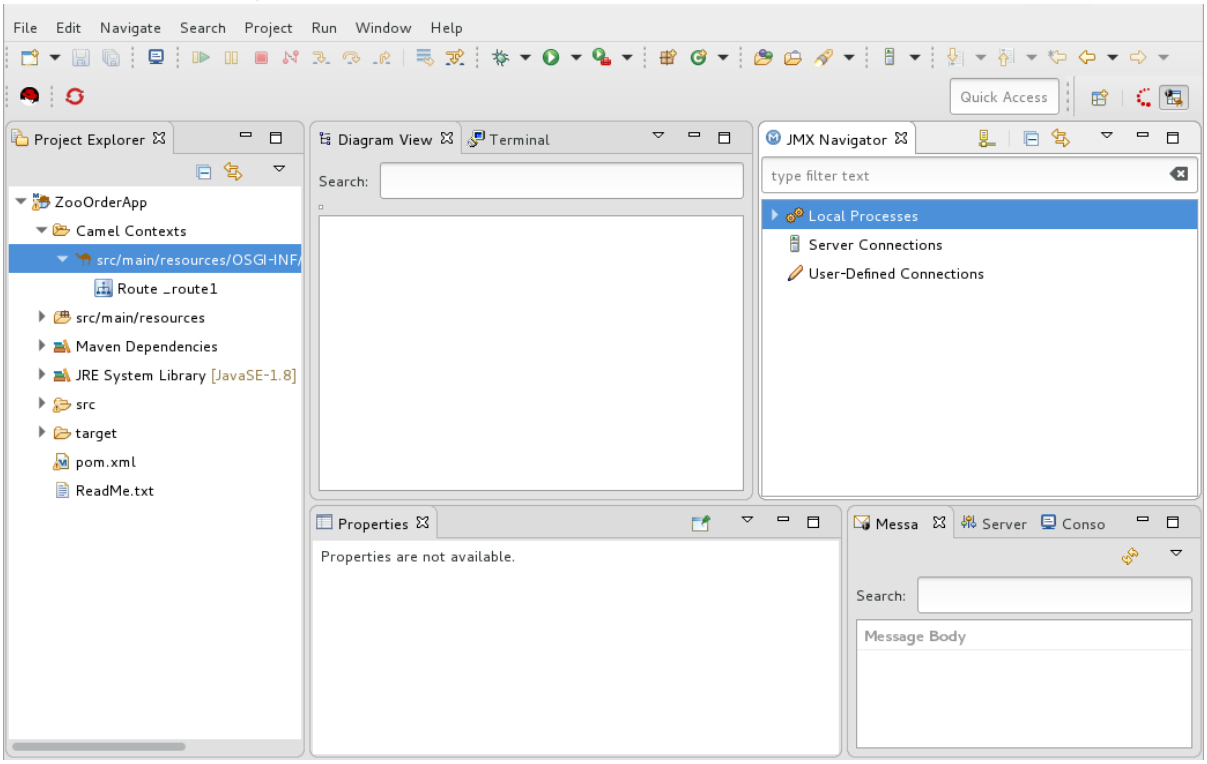
1. Click the button on the right side of the tool bar, and then select **Fuse Integration** from the list:



The **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:

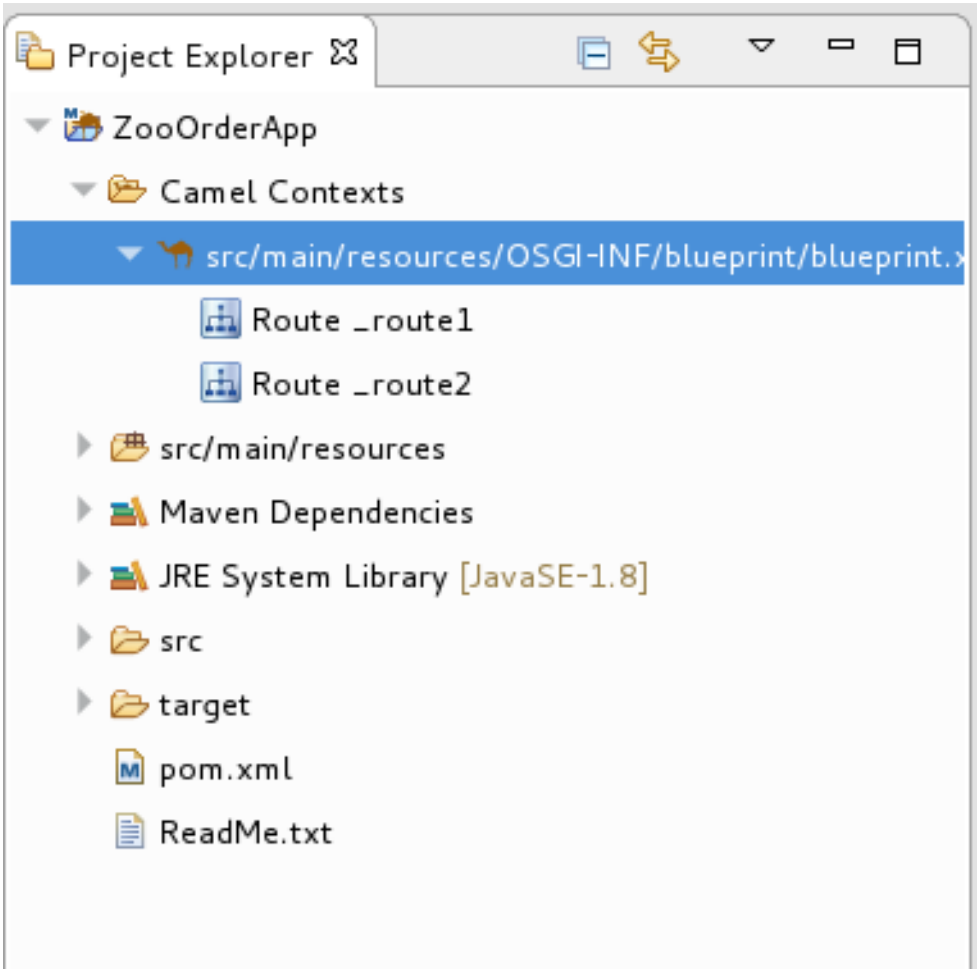


This arrangement provides more space for **Diagram View** to display the routing context's nodes graphically, which makes it easier for you to visually trace the path that messages take in traversing the routing context.

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**.

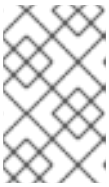
Additionally, all routes in a routing context are displayed as icons directly under their context file entry. To display a single route in the routing context on the canvas, double-click its icon in **Project Explorer**. To display all routes in the routing context, double-click the context file entry.



STARTING MESSAGE TRACING

To start message tracing on the **ZooOrderApp** project:

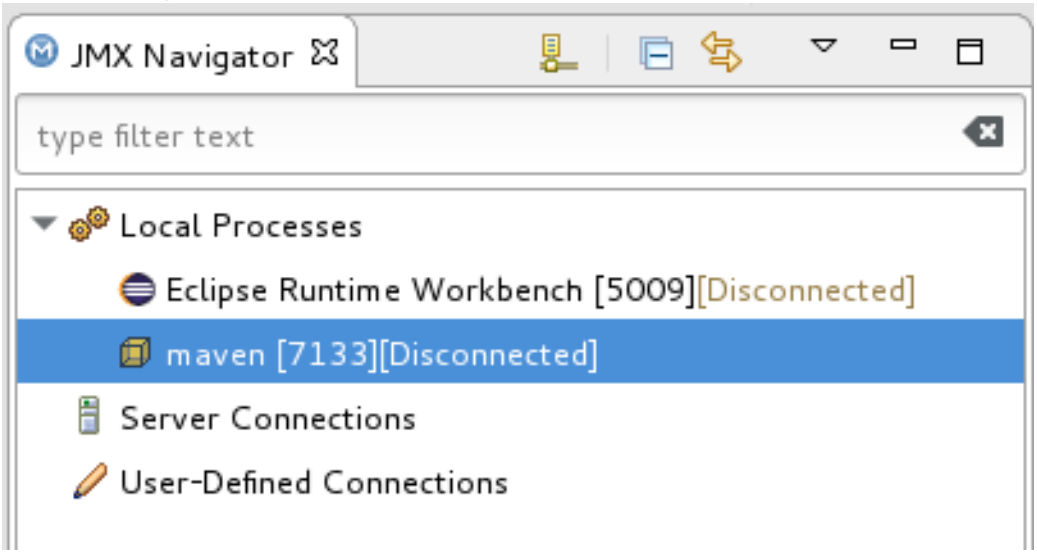
1. In **Project Explorer**, expand the **ZooOrderApp** project to expose **src/main/resources/OSGI-INF/blueprint/blueprint.xml**.
2. Right-click **src/main/resources/OSGI-INF/blueprint/blueprint.xml** to open the context menu.
3. Select **Run As → Local Camel Context (without tests)**



NOTE

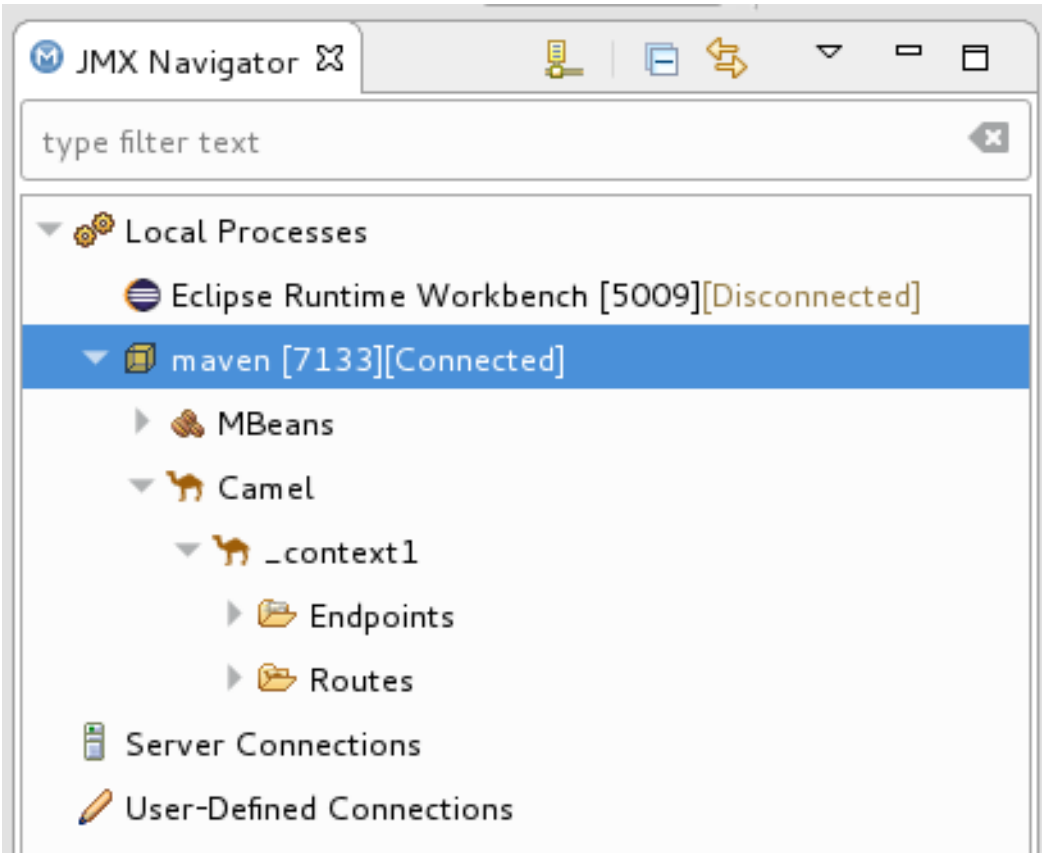
If you select **Local Camel Context**, the tooling reverts to running without tests because you have not yet created a JUnit test for the **ZooOrderApp** project. You will do that later in Chapter 9, *Testing a route with JUnit* .

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

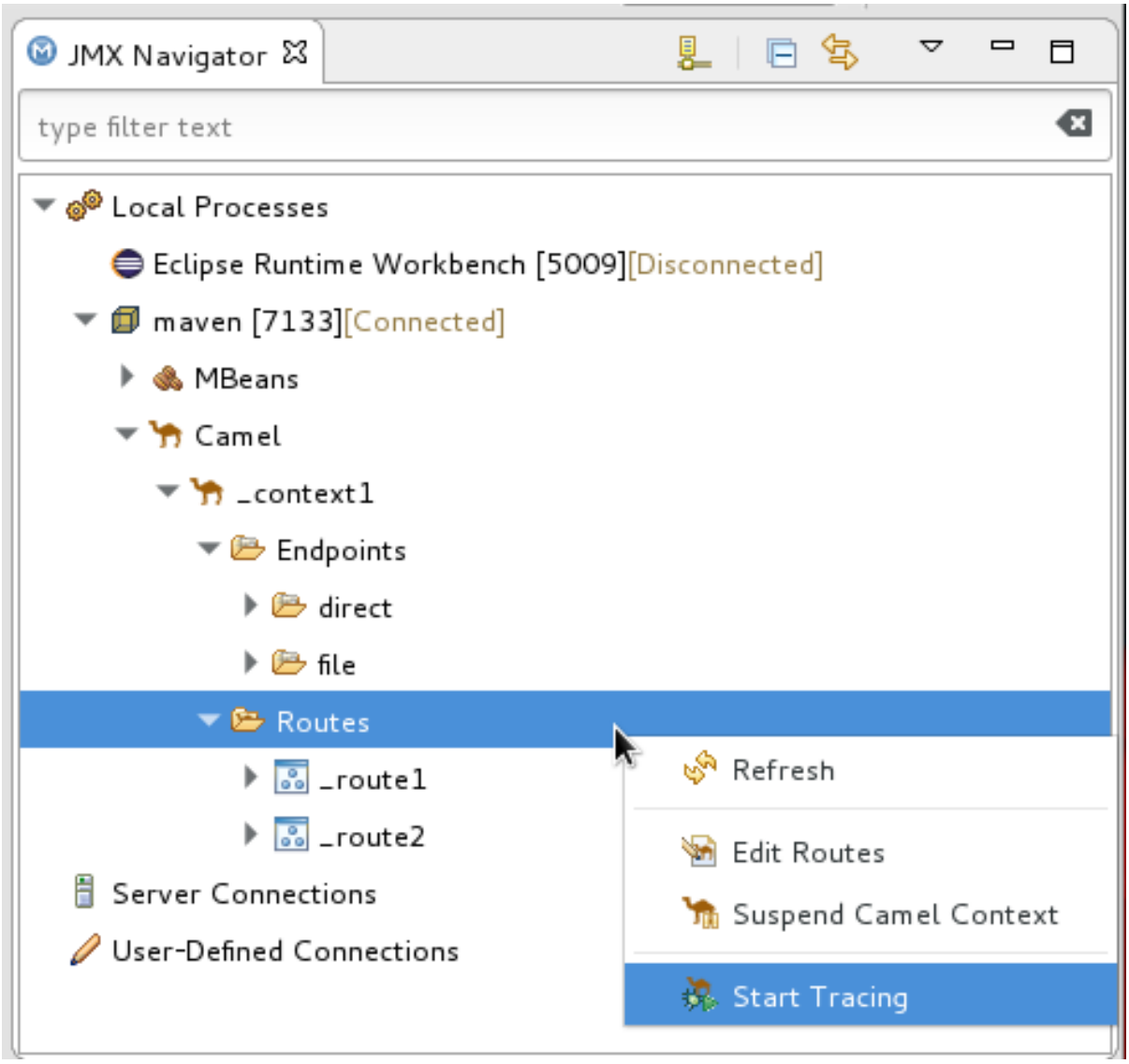


5. Right-click the **maven [ID]** node and then select **Connect**.

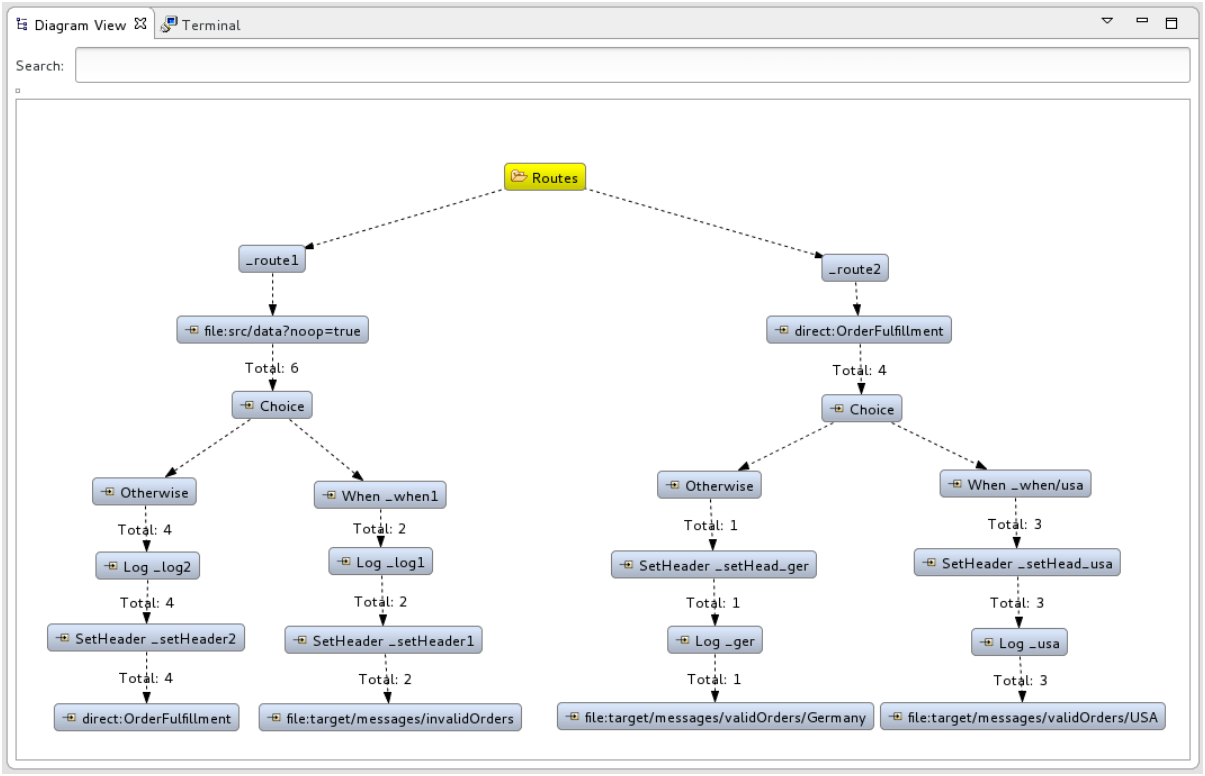
6. Expand the elements of your route:



7. Right-click the **Routes** node and then select **Start Tracing**.



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

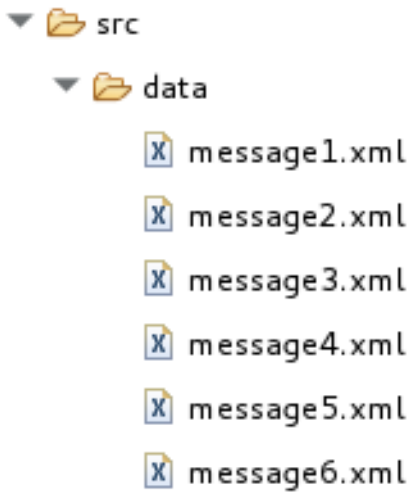


To see all message flow paths clearly, you probably need to rearrange the nodes by dragging them to fit neatly in the **Diagram View** tab. You may also need to adjust the size of the other views and tabs in Red Hat CodeReady Studio to allow the **Diagram View** tab to expand.

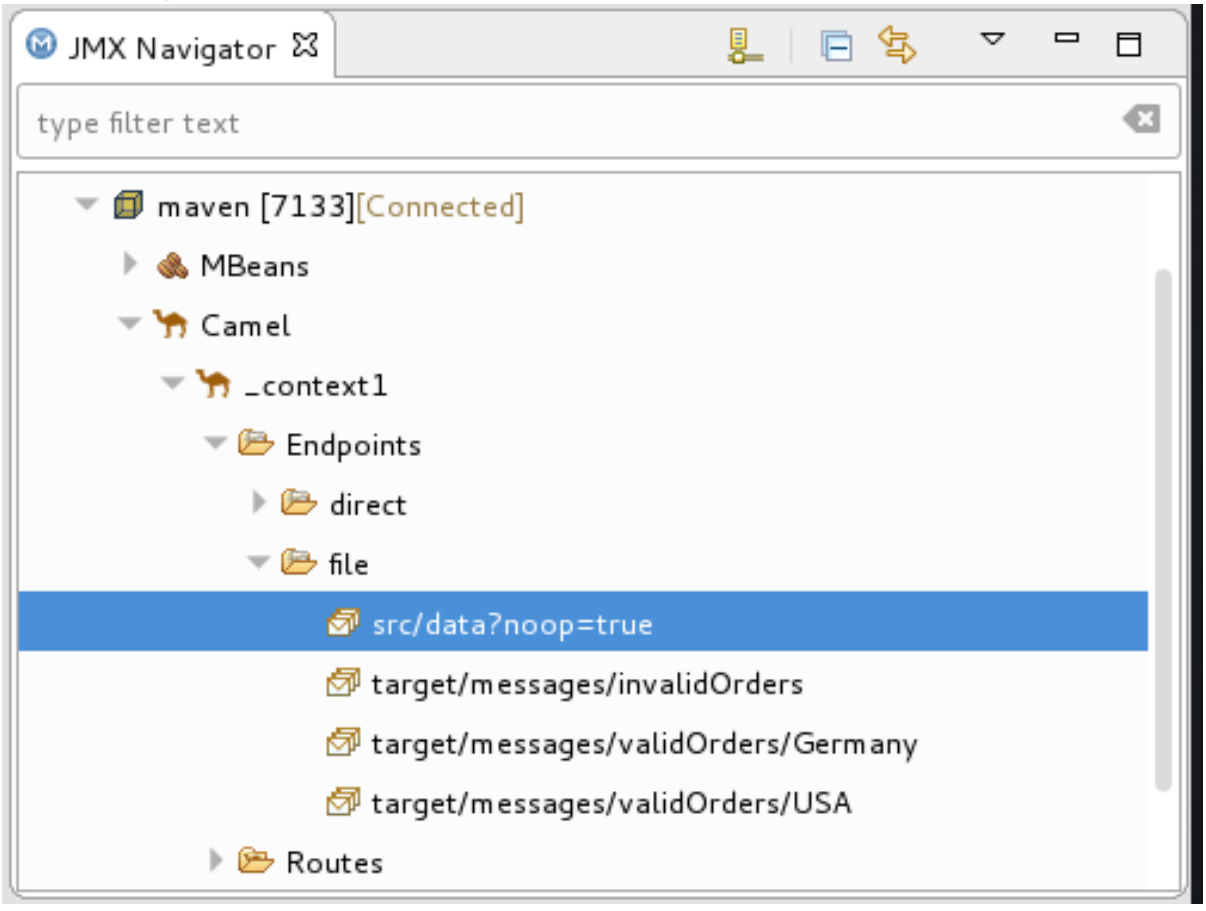
DROPPING MESSAGES ON THE RUNNING ZOOORDERAPP PROJECT

To drop messages on the running ZooOrderApp project:

1. In **Project Explorer**, expand **ZooOrderApp/src/data**, so that 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.

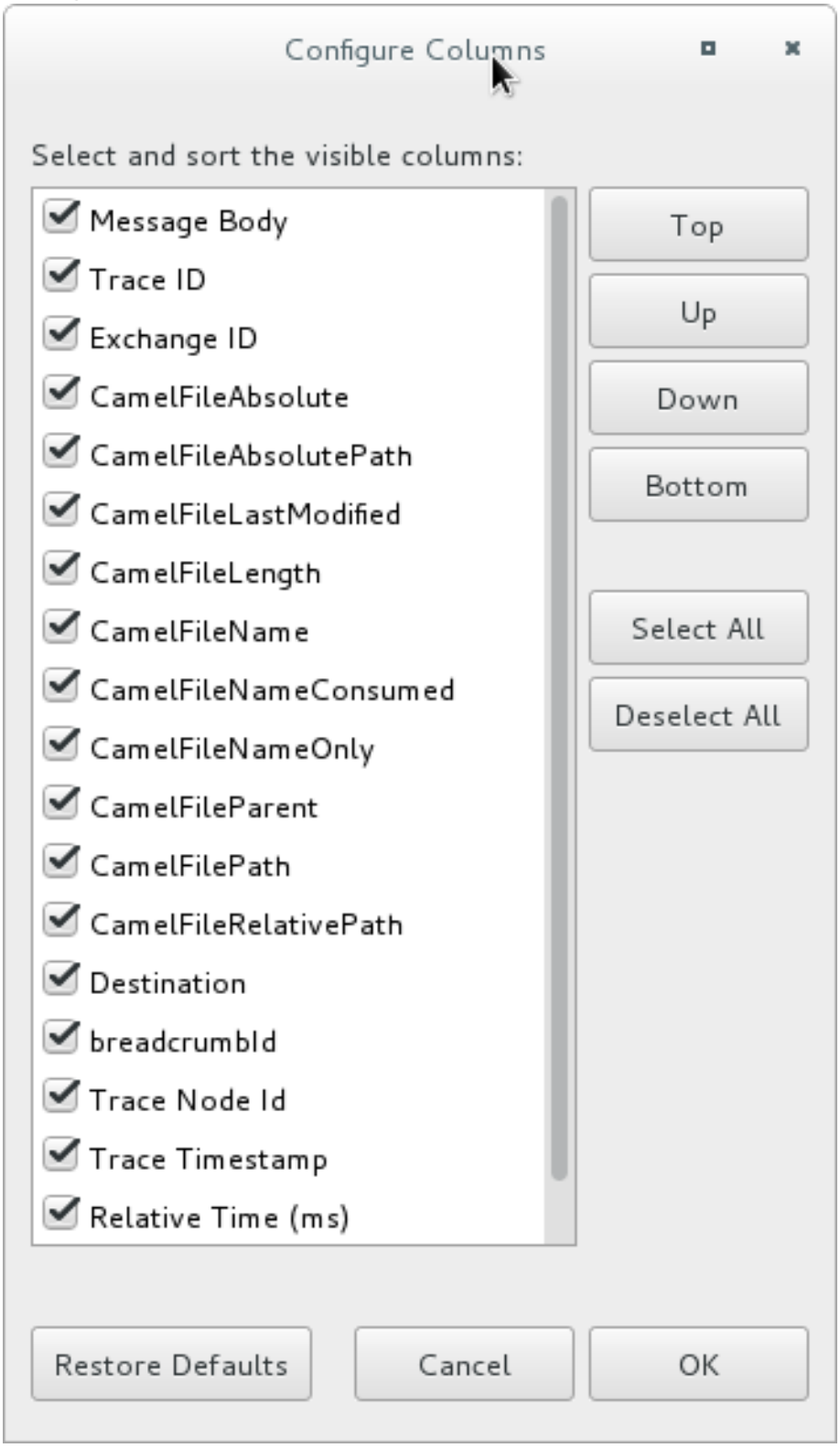
CONFIGURING MESSAGES VIEW

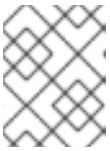
You must refresh the **Messages View** before it will display message traces. You also need to configure the columns in **Messages View** if you want them to persist across all message traces.

1. Open the **Messages View**.

2. Click the (Refresh button) on top, right of the panel's menu bar to populate the view with **message1.xml**'s message traces.

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



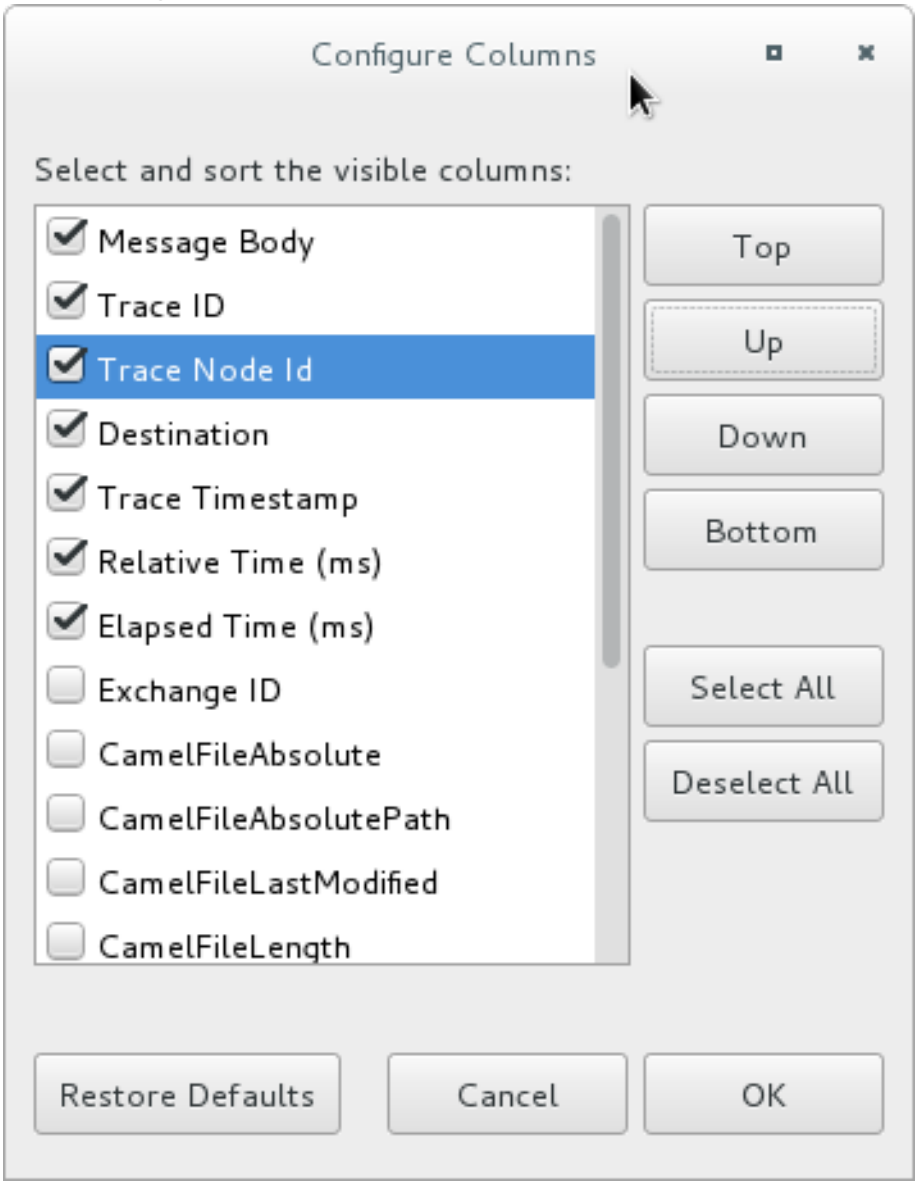


NOTE

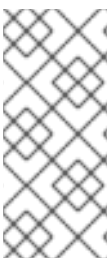
Notice that the message header, **Destination**, which you set for the messages in your routing context, appears in the list.

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.



NOTE

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

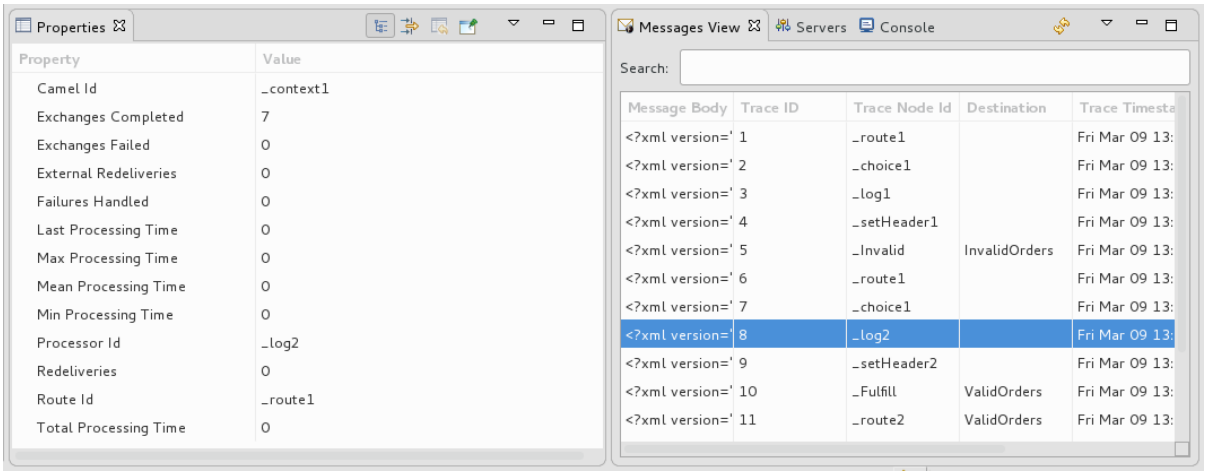
STEPPING THROUGH MESSAGE TRACES

To step through the message traces:

1. Drag **message2.xml** and drop it on the **_context1>Endpoints>file>src/data?noop=true** node in **JMX Navigator**.

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

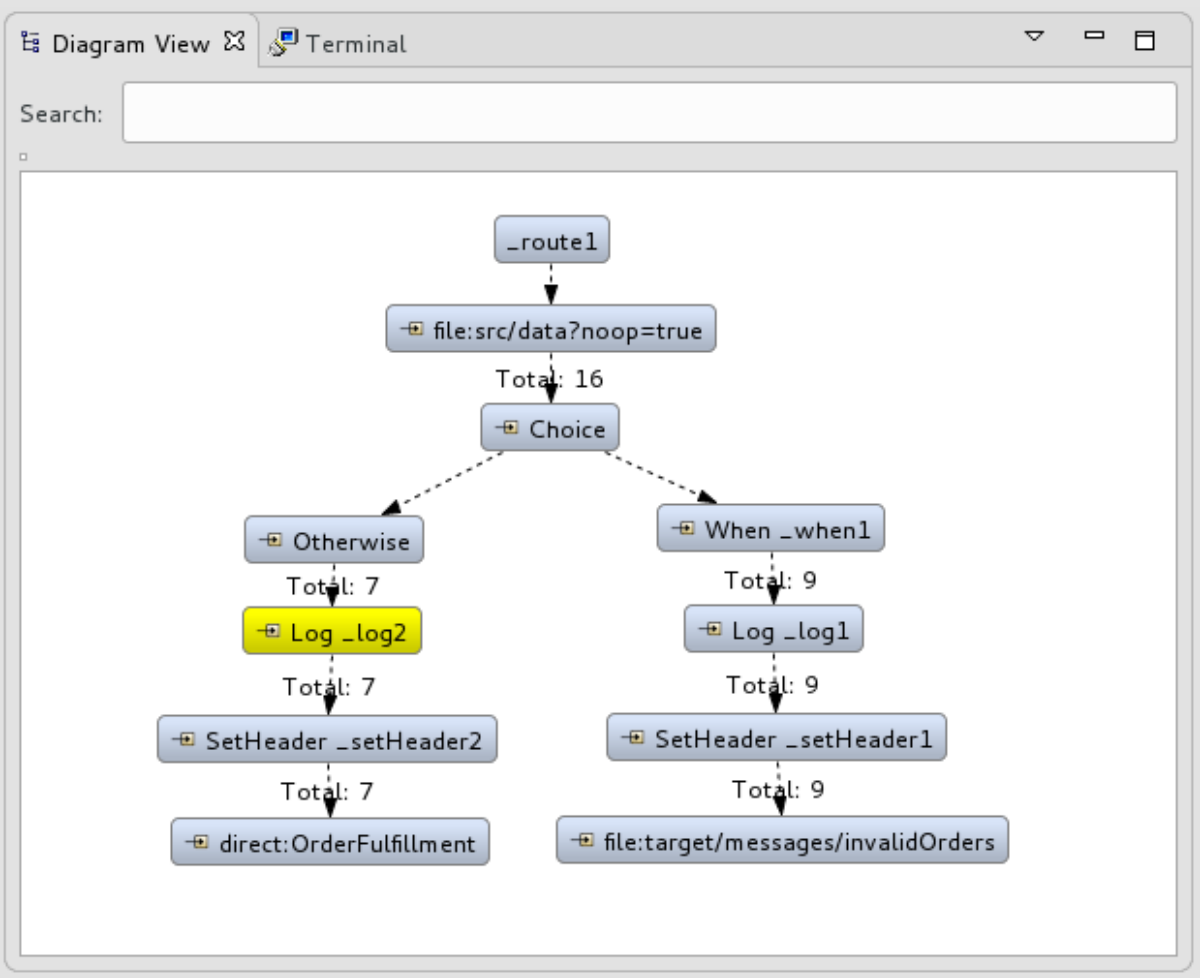
3. In **Messages View**, click the (Refresh button) to populate the view with **message2.xml** message traces.
   Each time you drop a message on in **JMX Navigator**, you need to refresh **Messages View** to populate it with the message traces.

4. Click one of the message traces to see more details about it in **Properties** view:
   

   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.

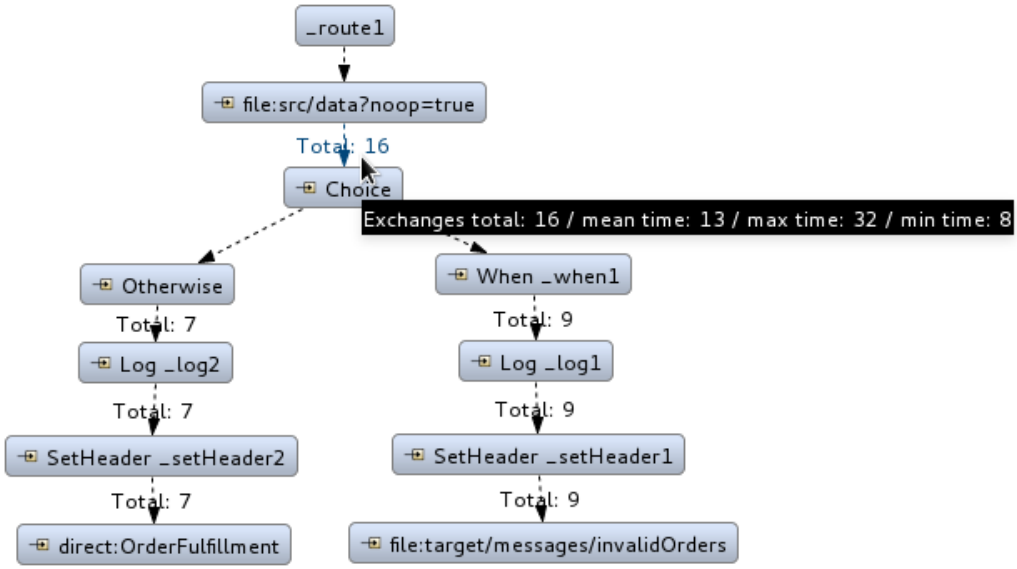
   You can step through the message instances by highlighting each one to see how a particular message traversed the route and whether it was processed as expected at each step in the route.

5. Open **Diagram View**, to see that the associated step in the route is highlighted:



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.

6. Hover the mouse pointer over the displayed metrics to reveal 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

7. Optionally, you can drag and drop the remaining messages in **ZooOrderApp/src/data/** into the **_context1>Endpoints>file>src/data?noop=true** node in **JMX Navigator** at any time, as long as tracing remains enabled.

   On each subsequent drop, remember to click the (Refresh button) to populate **Messages View** with the new message traces.

8. When done:
   - In **JMX Navigator**, right-click **_context1** and select **Stop Tracing Context**.
   - 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

In the Chapter 9, *Testing a route with JUnit* tutorial, you create a JUnit test case for your project and run your project as a **Local Camel Context**.

CHAPTER 9. TESTING 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 should create or modify the generated test case to add expectations and assertions specific to the route that you created or updated. This ensures that the test is valid for the route.

GOALS

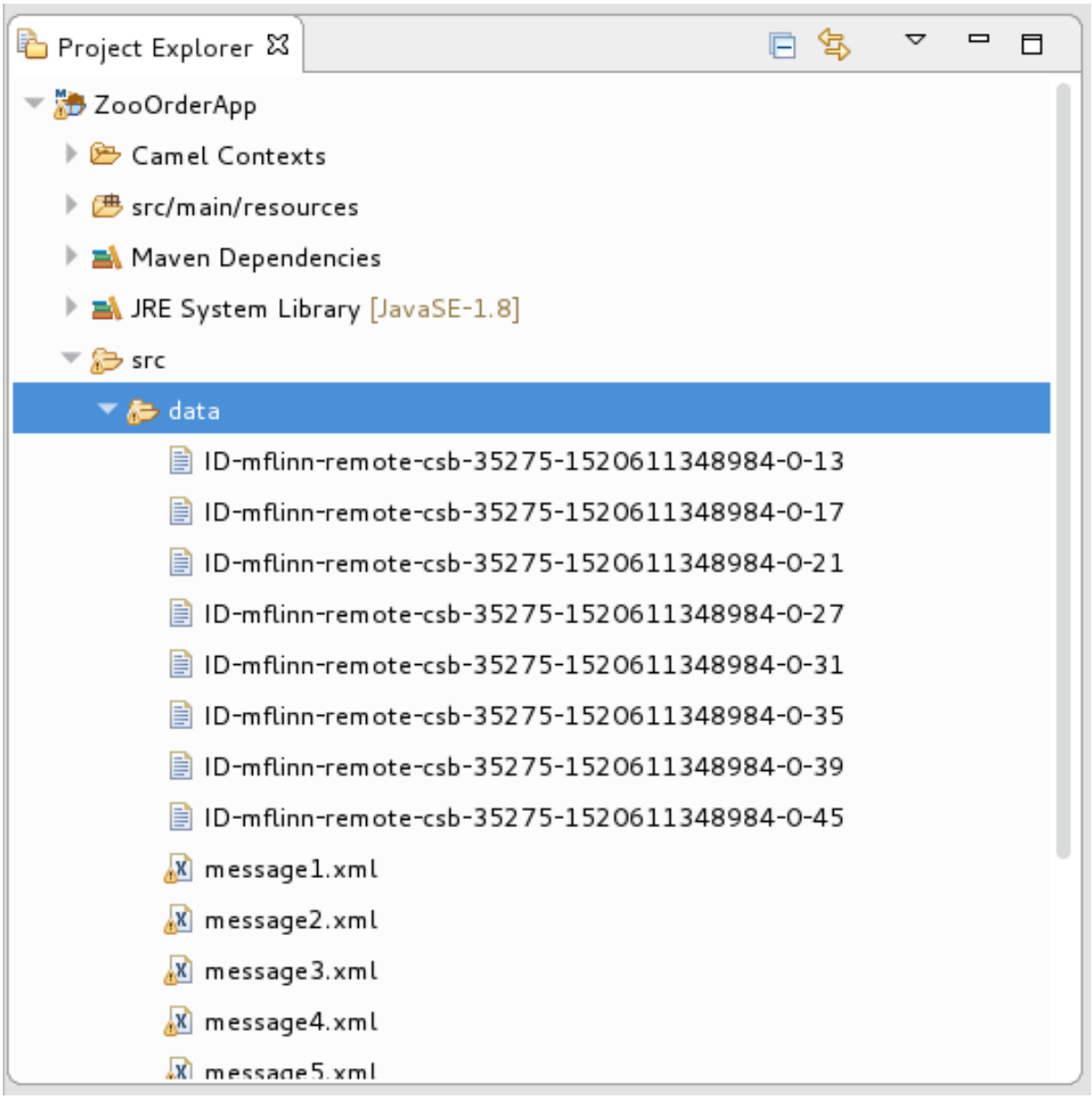
In this tutorial you complete the following tasks:

- Create the **/src/test/** folder to store the JUnit test case
- Generate the JUnit test case for the **ZooOrderApp** project
- Modify the newly generated JUnit test case
- Modify the **ZooOrderApp** project's **pom.xml** file
- Run the **ZooOrderApp** with the new JUnit test case
- Observe the output

PREREQUISITES

1. To start this tutorial, you need the **ZooOrderApp** project resulting from one of the following:
   - Complete the Chapter 8, *Tracing a message through a route* tutorial.
     or
   - Complete the Chapter 2, *Setting up your environment* tutorial and replace your project's **blueprint.xml** file with the provided **blueprintContexts/blueprint3.xml** file, as described in the section called "About the resource files".
2. Delete any trace-generated messages from the **ZooOrderApp** project's **/src/data/** directory and **/target/messages/** subdirectories in **Project Explorer**. Trace-generated messages begin with the **ID-** prefix. For example, Figure 9.1, "Trace-generated messages" shows eight trace-generated messages:

Figure 9.1. Trace-generated messages

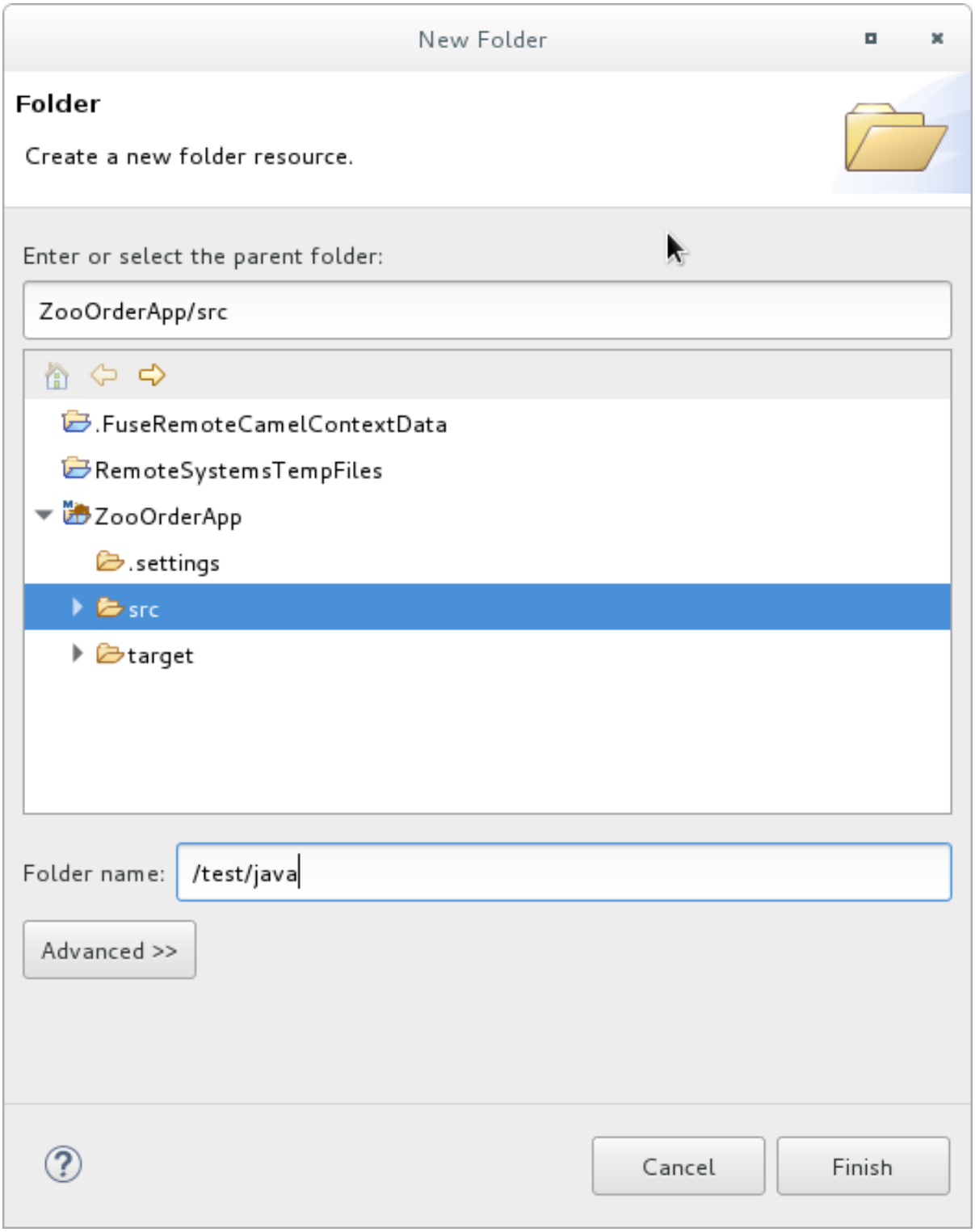


Select all trace-generated messages in batch, right-click and then select **Delete**.

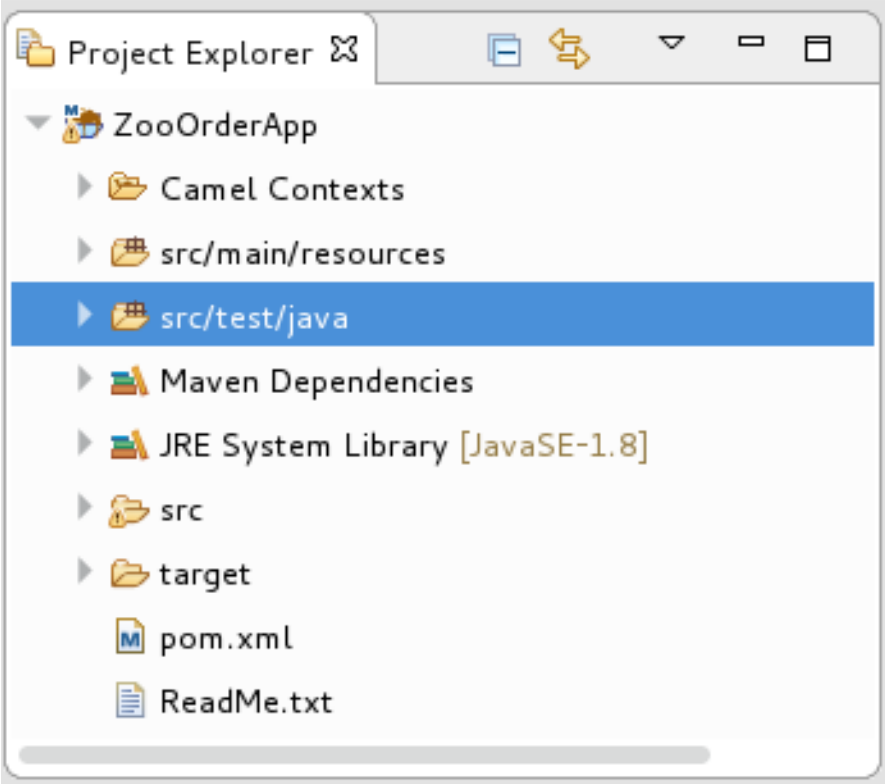
CREATING THE SRC/TEST FOLDER

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

1. In **Project Explorer**, right-click the **ZooOrderApp** project and then select **New → Folder**.
2. In the **New Folder** dialog, in the project tree pane, expand the **ZooOrderApp** node and select the **src** folder.
   Make sure **ZooOrderApp/src** appears in the **Enter or select the parent folder**field.
3. In **Folder name**, enter **/test/java**:



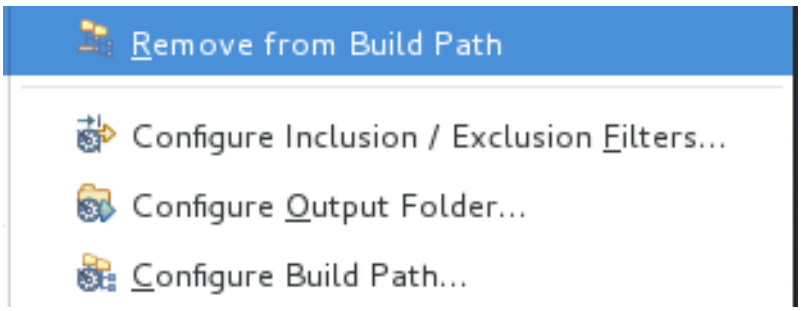
4. Click **Finish**.
   In **Project Explorer**, the new **src/test/java** folder appears under the **src/main/resources** folder:



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.

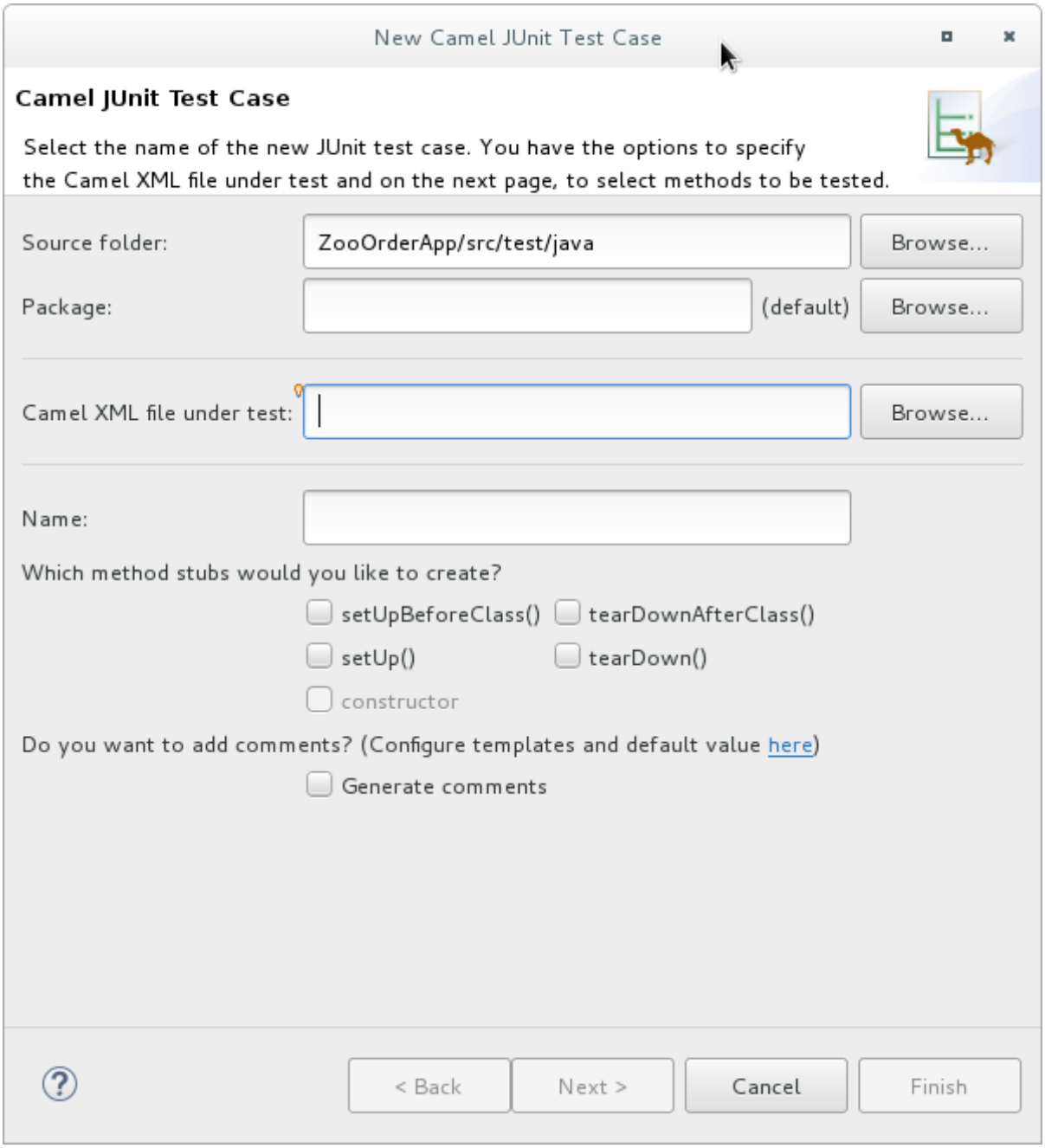
   b. Select Build Path to see the menu options:
   The menu option **Remove from Build Path**verifies that the **/src/test/java** folder is currently included in the build path:



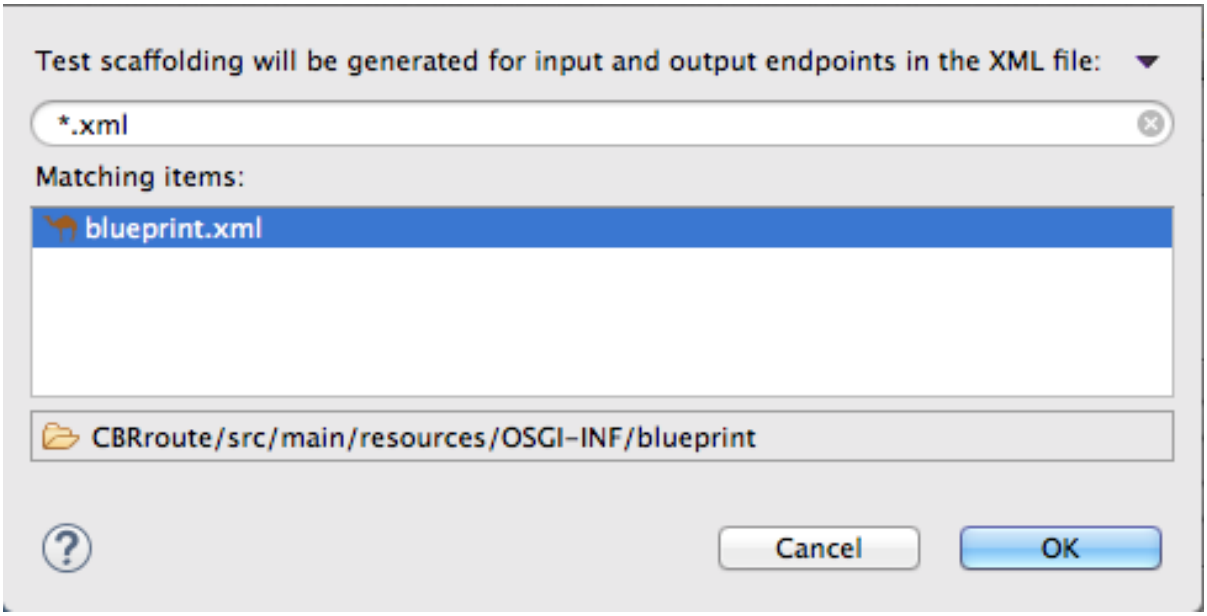
CREATING THE JUNIT TEST CASE

To create a JUnit test case for the **ZooOrderApp** project:

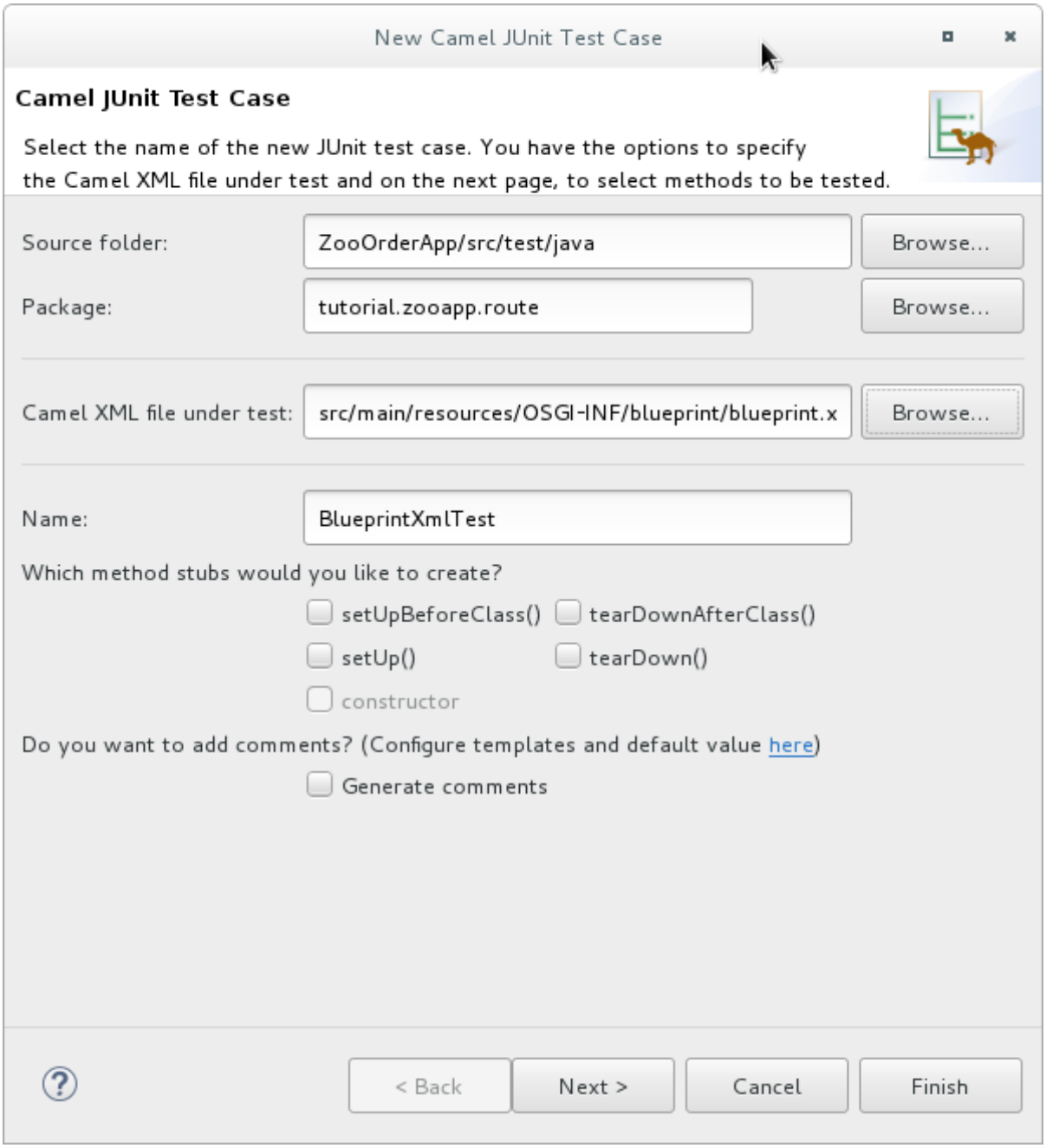
1. In **Project Explorer**, select **src/test/java**.
2. Right-click and then select **New → Camel Test Case**.



3. In the **Camel JUnit Test Case** wizard, make sure the **Source folder** field contains **ZooOrderApp/src/test/java**. To find the proper folder, click Browse... .
4. In the **Package** field, enter **tutorial.zooapp.route**. This package will include the new test case.
5. In the **Camel XML file under test**field, click Browse... to open a file explorer configured to filter for XML files, and then select the **ZooOrderApp** project's **blueprint.xml** file:



6. Click **OK**. The **Name** field defaults to *BlueprintXmlTest*.



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

8. 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.zooapp.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[] expectedBodies = { "<something id='1'>expectedBody1</something>",
   "<something id='2'>expectedBody2</something>" };
 // Templates to send to input endpoints
 @Produce(uri = "file:src/data?noop=true")
 protected ProducerTemplate inputEndpoint;
 @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;

 @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/validOrders/USA").to(output3Endpoint);
    from("file:target/messages/validOrders/Germany").to(output4Endpoint);
   }
  });
```

```
    // 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 **ZooOrderApp** 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" .

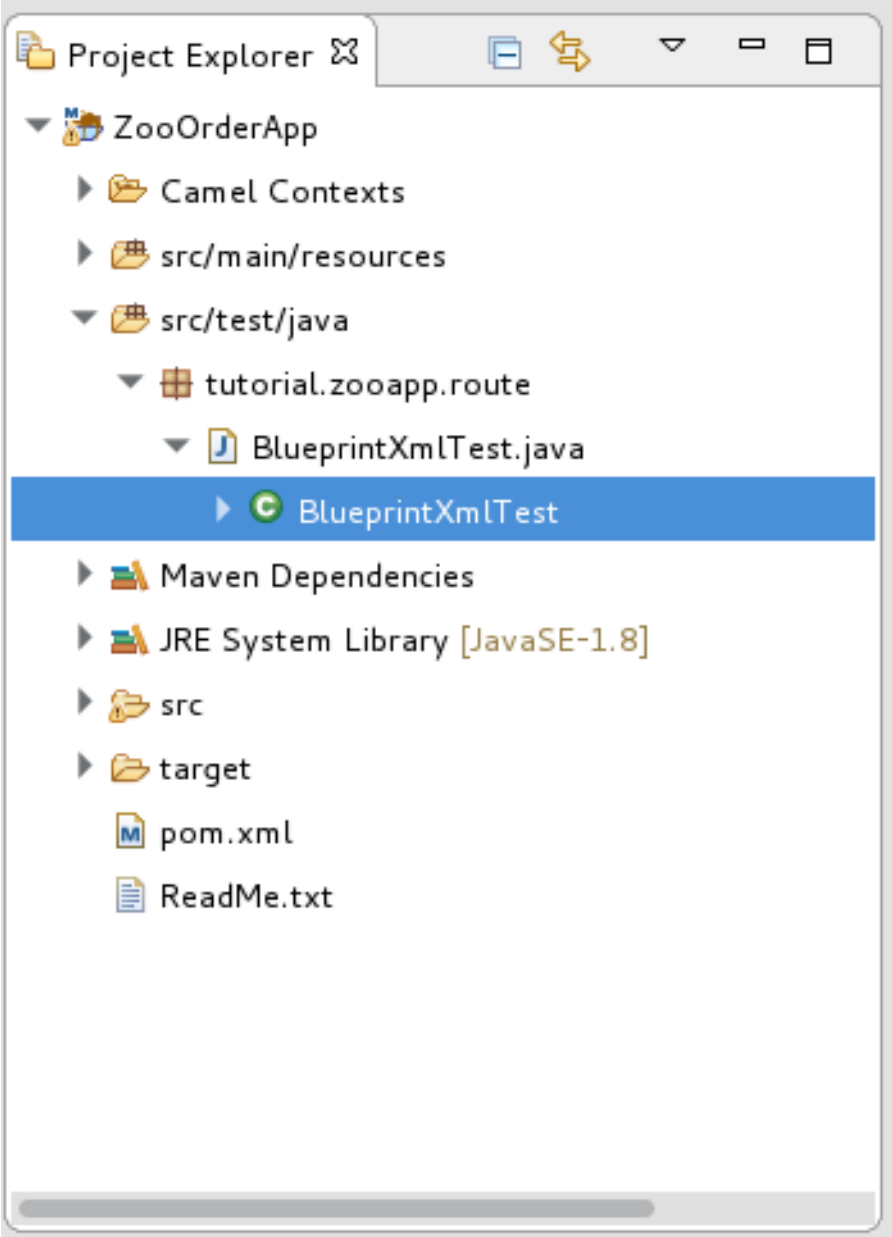
MODIFYING THE BLUEPRINTXMLTEST FILE

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

Follow these steps to modify the **BlueprintXmlTest.java** file:

1. In **Project Explorer**, expand the **ZooOrderApp** project to expose the **BlueprintXmlTest.java** file:



2. Open the **BlueprintXmlTest.java** file.

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 in bold text. Adding the first line causes an error that will be resolved when you update the **pom.xml** file as instructed in the next section.

```
package tutorial.zooapp.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;
```

5. Scroll down to the lines that follow directly after // **Expected message bodies**.

6. Replace those lines – **protected Object[] expectedBodies={ …… expectedBody2</something>"};** – with these **protected String body#;** lines:

```
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.

```
// Valid orders body2 = FileUtils.readFileToString(new File("src/data/message2.xml"), "UTF-8"); body4 = FileUtils.readFileToString(new File("src/data/message4.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 body1 = FileUtils.readFileToString(new File("src/data/message1.xml"), "UTF-8"); body3 = FileUtils.readFileToString(new File("src/data/message3.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:

```
// Invalid orders outputEndpoint.expectedBodiesReceived(body1, body3); // Valid orders for USA output3Endpoint.expectedBodiesReceived(body2, body5, body6); // Valid order for Germany output4Endpoint.expectedBodiesReceived(body4);
```

   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.zooapp.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
 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;
 @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;

 @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 {
    // Valid orders
    body2 = FileUtils.readFileToString(new File("src/data/message2.xml"), "UTF-8");
    body4 = FileUtils.readFileToString(new File("src/data/message4.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
    body1 = FileUtils.readFileToString(new File("src/data/message1.xml"), "UTF-8");
    body3 = FileUtils.readFileToString(new File("src/data/message3.xml"), "UTF-8");

    from("file:target/messages/invalidOrders").to(outputEndpoint);
    from("file:target/messages/validOrders/USA").to(output3Endpoint);
    from("file:target/messages/validOrders/Germany").to(output4Endpoint);
    from("direct:OrderFulfillment").to(output2Endpoint);
   }
  });

  // Define some expectations

  // TODO Ensure expectations make sense for the route(s) we're testing
  // Invalid orders
  outputEndpoint.expectedBodiesReceived(body1, body3);

  // Valid orders for USA
  output3Endpoint.expectedBodiesReceived(body2, body5, body6);

  // Valid order for Germany
  output4Endpoint.expectedBodiesReceived(body4);
```

```
 // 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 ZooOrderApp project's **pom.xml** file:

1. In **Project Explorer**, select the **pom.xml**, located below the **target** folder, and open it 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.

RUNNING THE JUNIT TEST

To run the test:

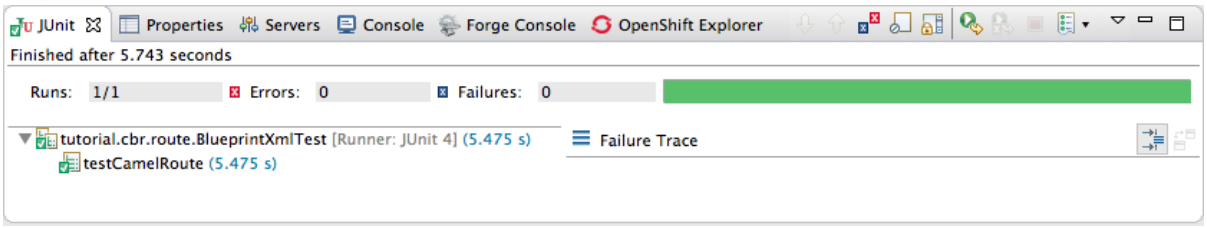
1. Switch to the **JBoss** perspective to free up more workspace.
2. In the **Project Explorer**, right-click the **ZooOrderApp** project.
3. Select **Run As → JUnit Test**.
   By default, the **JUnit** view opens in the sidebar. (To provide a better view, drag it to the bottom, right panel that displays the **Console**, **Servers**, and **Properties** tabs.)

NOTE

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

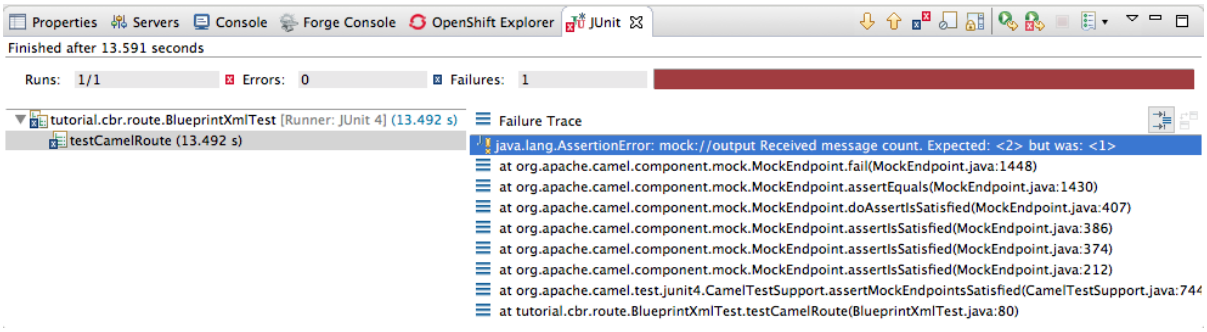
If the test runs successfully, you'll see something like this:

Figure 9.2. Successful JUnit run



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

Figure 9.3. Failed JUnit run



NOTE

JUnit will fail if your execution environment is not set to Java SE 8. The message bar at the top of the **JUnit** tab will display an error message indicating that it cannot find the correct SDK.

To resolve the issue, open the project's context menu, and select **Run As → Run Configurations → JRE**. Click the **Environments] button next to the *Execution environment** field to locate and select a Java SE 8 environment.

4. 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 ZooOrderApp project's **/src/data** folder in **Project Explorer** (see Figure 9.1, "Trace-generated messages").

FURTHER READING

To learn more about JUnit testing see JUnit.

NEXT STEPS

In the Chapter 10, *Publishing your project to Red Hat Fuse* tutorial, you learn how to publish your Apache Camel project to Red Hat Fuse.

CHAPTER 10. PUBLISHING YOUR PROJECT TO RED HAT FUSE

This tutorial walks you through the process of publishing your project to Red Hat Fuse. It assumes that you have an instance of Red Hat Fuse installed on the same machine on which you are running the Red Hat Fuse Tooling.

GOALS

In this tutorial you complete the following tasks:

- Define a Red Hat Fuse server
- Configure the publishing options
- Start up the Red Hat Fuse server and publish the **ZooOrderApp** project
- Connect to the Red Hat Fuse server
- Verify whether the **ZooOrderApp** project's bundle was successfully built and published
- Uninstall the **ZooOrderApp** project

PREREQUISITES

Before you start this tutorial you need:

- Access to a Red Hat Fuse instance
- Java 8 installed on your computer
- The **ZooOrderApp** project resulting from one of the following:
  - Complete the Chapter 9, *Testing a route with JUnit* tutorial.
    or
  - Complete the Chapter 2, *Setting up your environment* tutorial and replace your project's **blueprint.xml** file with the provided **blueprintContexts/blueprint3.xml** file, as described in the section called "About the resource files" .

DEFINING A RED HAT FUSE SERVER

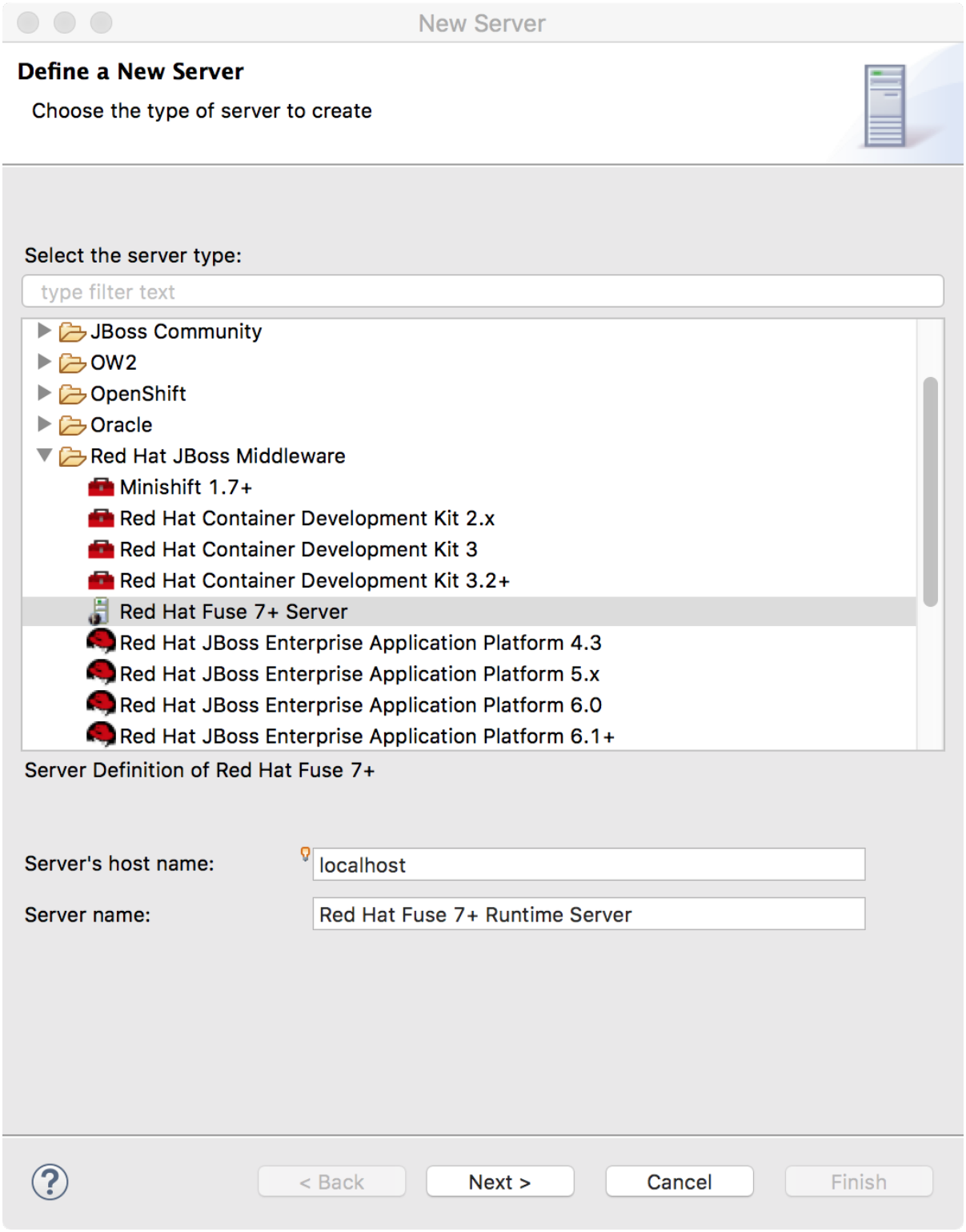
To define a server:

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

NOTE

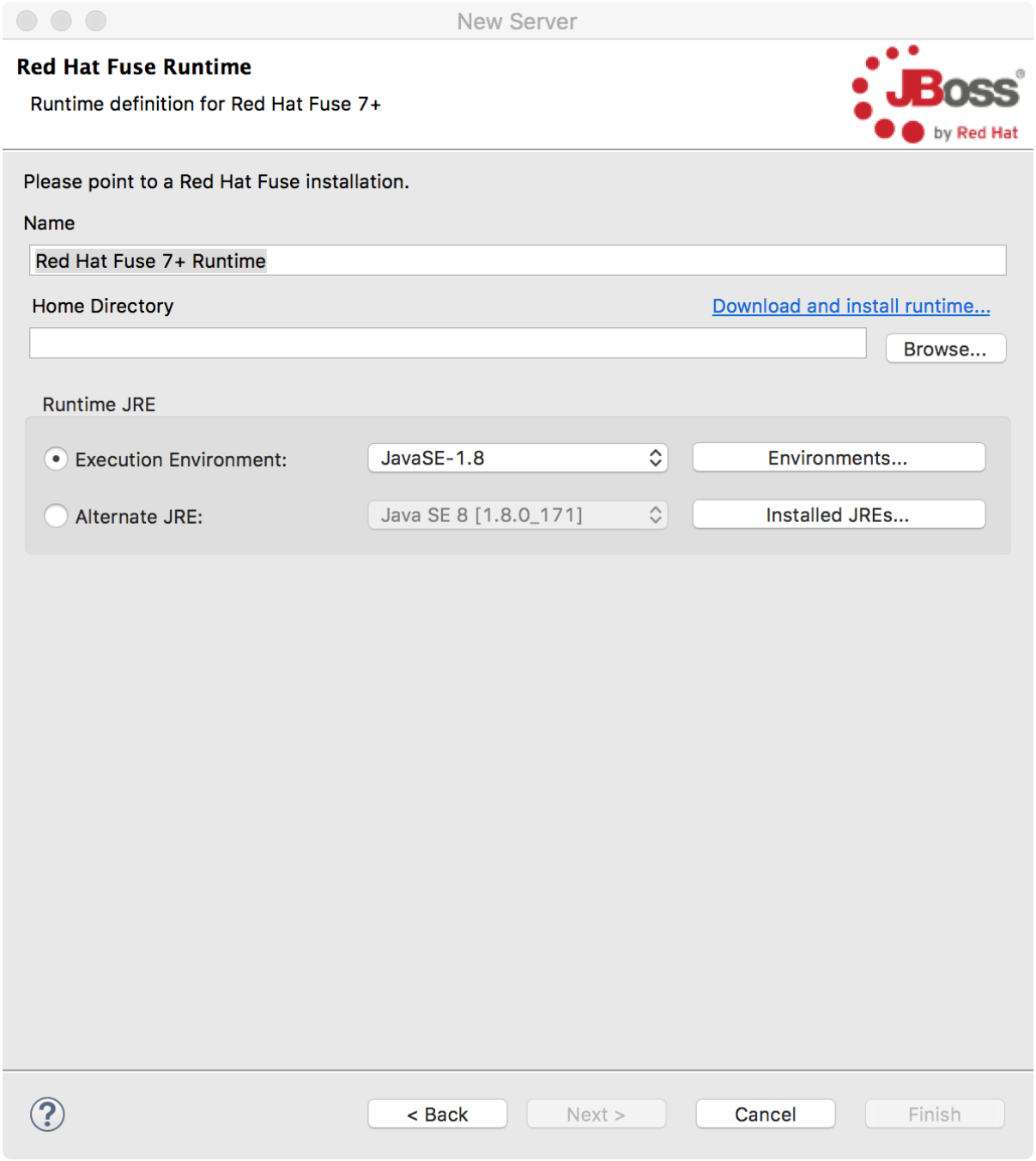
To define a new server when one is already defined, right-click inside the **Servers** view and then select **New → Server**.

4. Expand the **Red Hat JBoss Middleware** node to expose the available server options:



5. Select a Red Hat Fuse server.

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

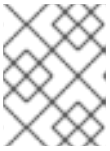


NOTE

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

If you have already defined a server, the tooling skips this page, and instead displays the configuration details page.

7. Accept the default for **Name**.
8. Click **Browse** next to the **Home Directory** field, to navigate to the installation and select it.
9. Select the runtime JRE from the drop-down menu next to **Execution Environment**. Select JavaSE-1.8 (recommended). If necessary, click the **Environments** button to select it from the list.

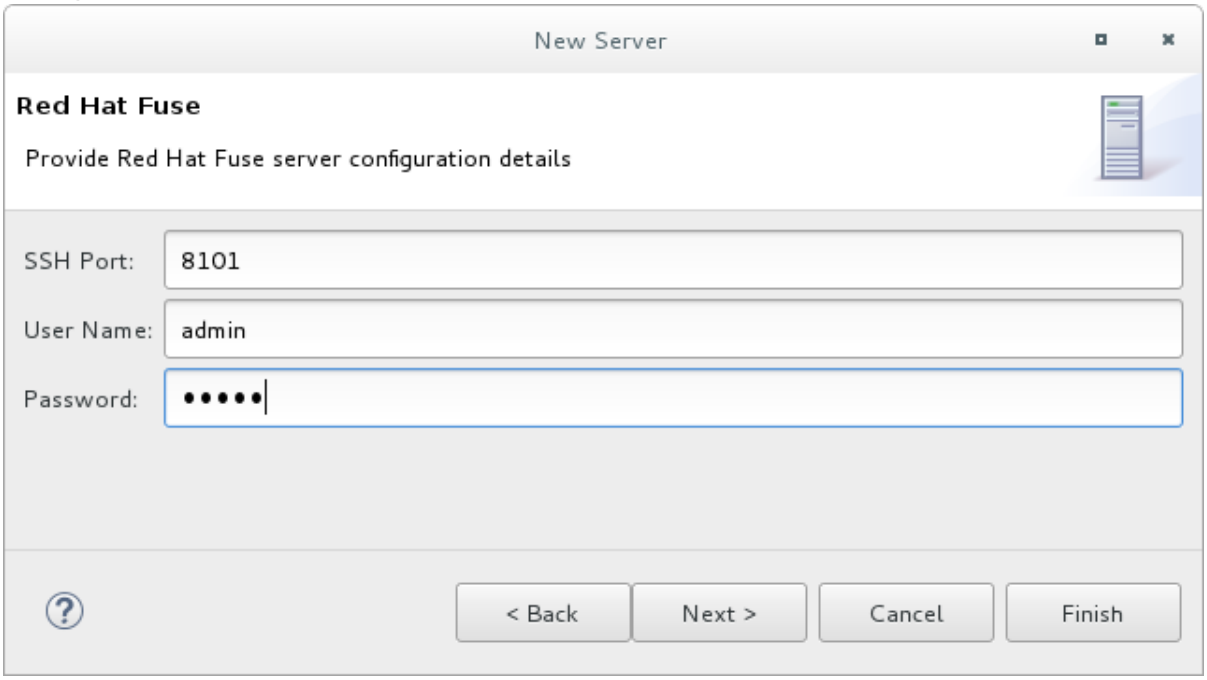


NOTE

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

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

11. Click **Next** to save the runtime definition for the Fuse Server and open the **Fuse server configuration details** page:



12. Accept the default for **SSH Port** (*8101*).
The runtime uses the SSH port to connect to the server's Karaf shell. If this default is incorrect, you can discover the correct port number by looking in the Red Hat Fuse ***installDir*/etc/org.apache.karaf.shell.cfg** file.

13. In **User Name**, enter the name used to log into the server.
This is a user name stored in the Red Hat 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.

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**, type the password required for **User name** to log into the server.
    This is the password set either in Red Hat Fuse's ***installDir*/etc/users.properties** file or by the karaf **jaas** commands.

15. Click **Finish**.
    Runtime Server [stopped, Synchronized] appears in the **Servers** view.

16. In the **Servers** view, expand the Runtime Server:

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

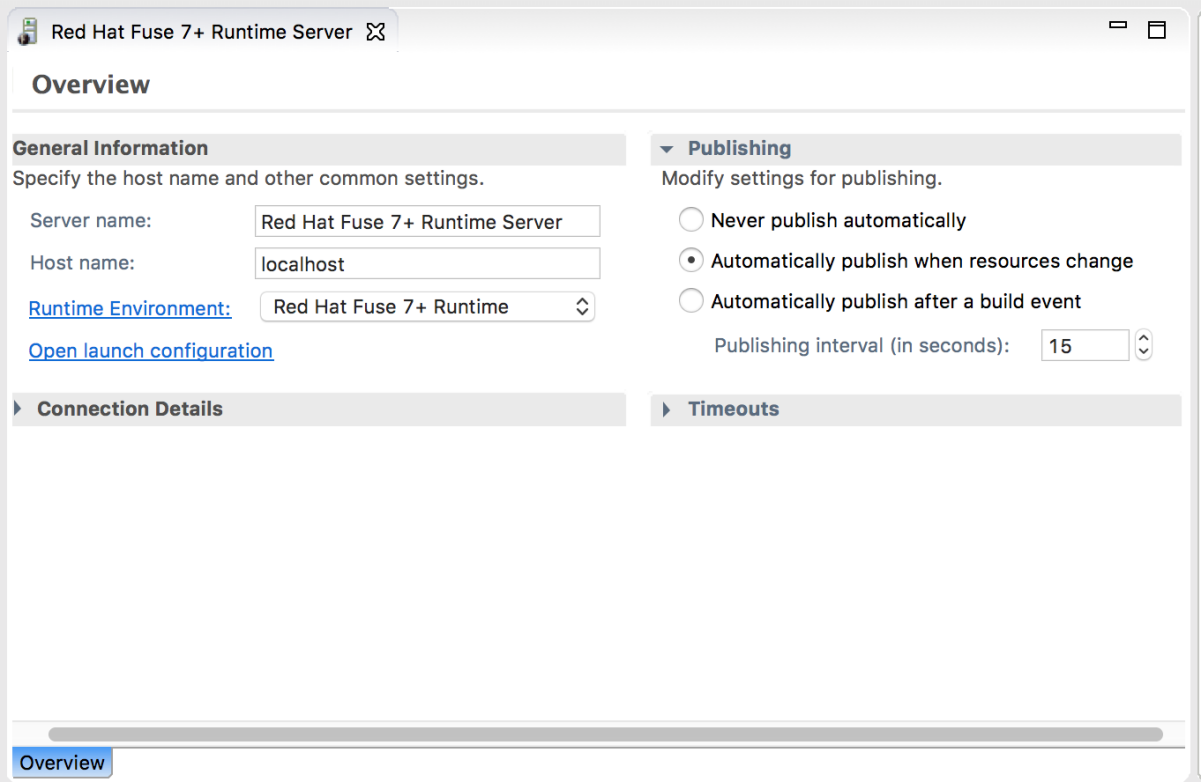
CONFIGURING THE PUBLISHING OPTIONS

Using publishing options, you can configure how and when your **ZooOrderApp** 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 configure immediate publishing upon saving changes to the **ZooOrderApp** project. To do so:

1. In the **Servers** view, double-click the **Runtime Server [stopped, Synchronized]** entry to display its overview.
2. On the server's **Overview** page, expand the **Publishing** section to expose the options.



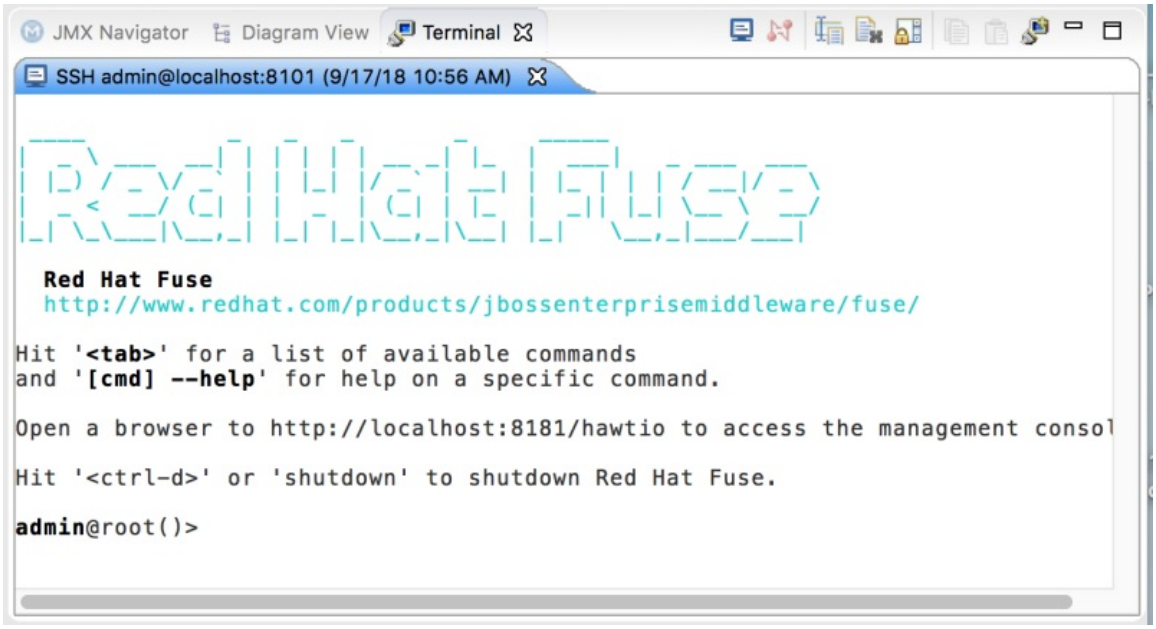
Make sure that the option **Automatically publish when resources change**is enabled.

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

3. In the **Servers** view, click .

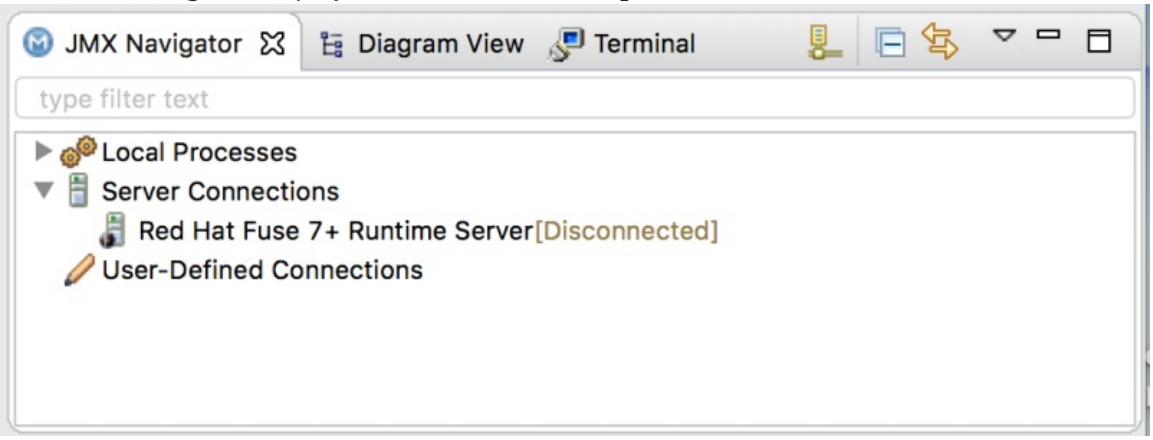
4. Wait a few seconds for the server to start. When it does:

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

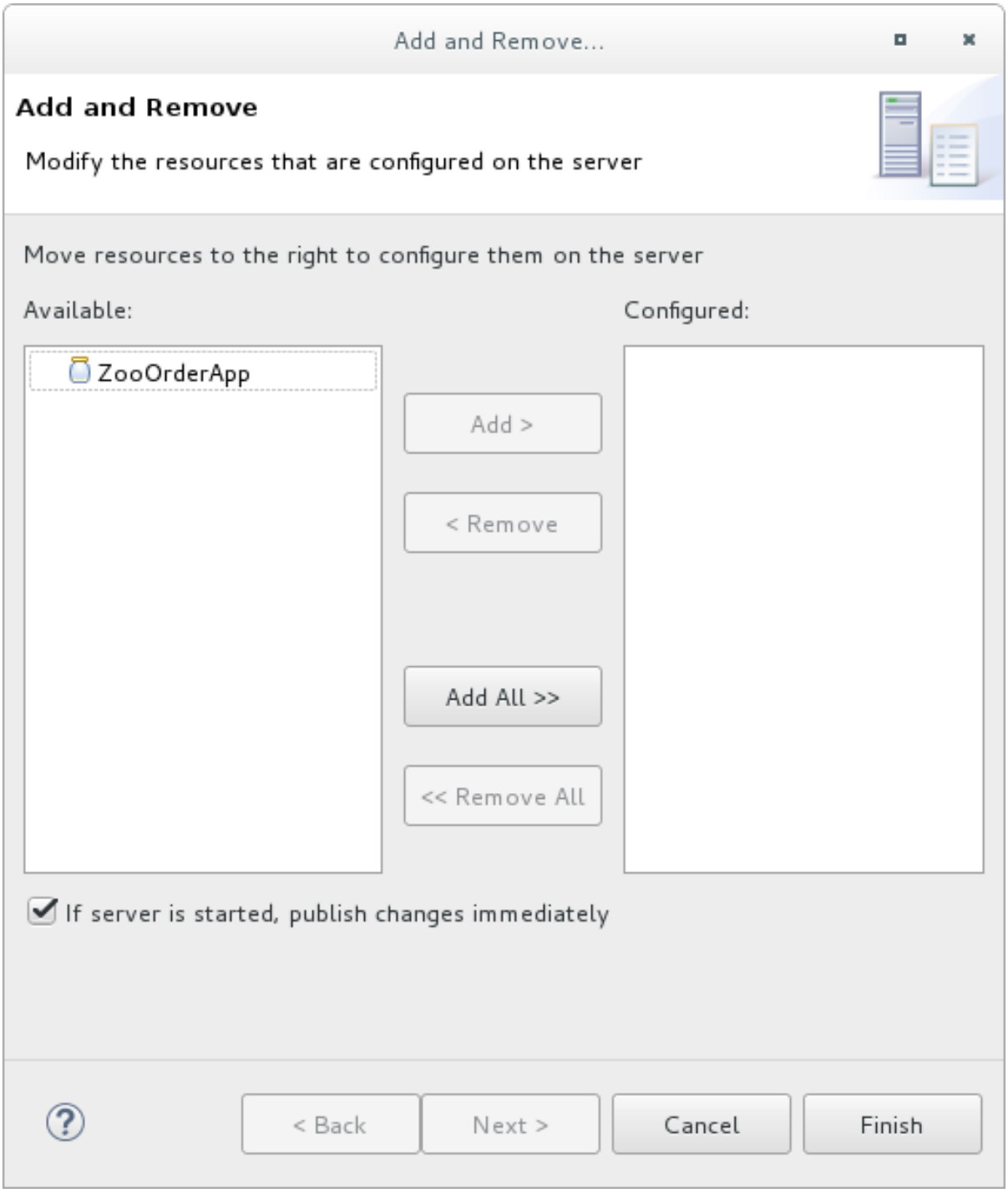


   - The **Servers** view displays:

- The **JMX Navigator** displays ***n.n* Runtime Server[Disconnected**

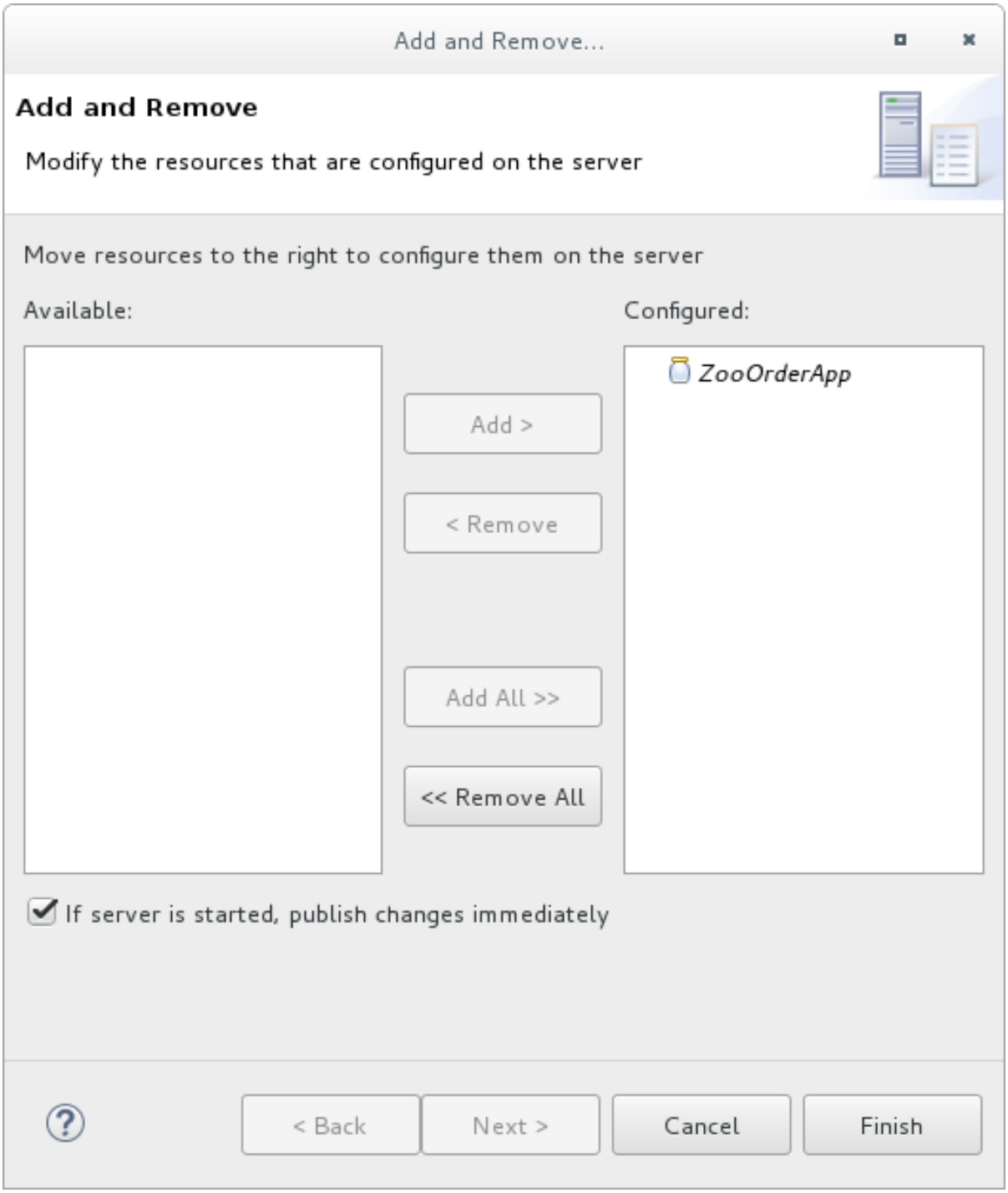


5. In the **Servers** view, right-click ***n.n* Runtime Server [Started]**and then select **Add and Remove** to open the **Add and Remove** page:

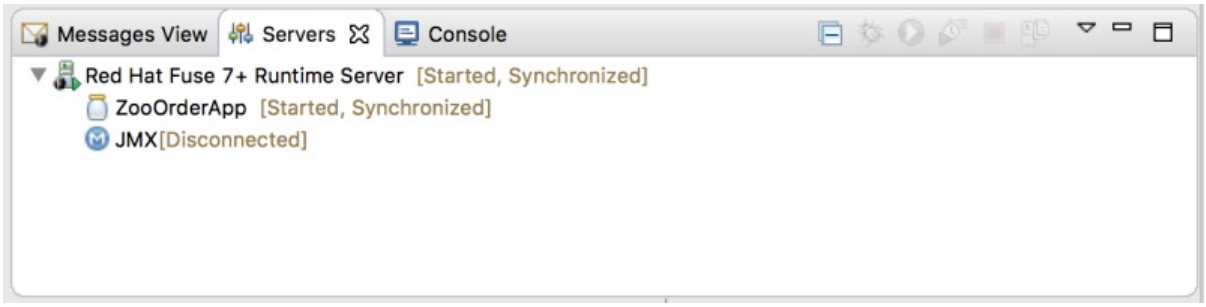


Make sure the option **If server is started, publish changes immediately** is checked.

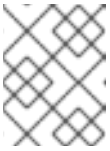
6. Select **ZooOrderApp** and click **Add** to assign it to the Fuse server:



7. Click **Finish**.
The **Servers** view should show the following:



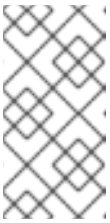
- **Runtime Server [Started, Synchronized]**



NOTE

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

- **ZooOrderApp [Started, Synchronized]**



NOTE

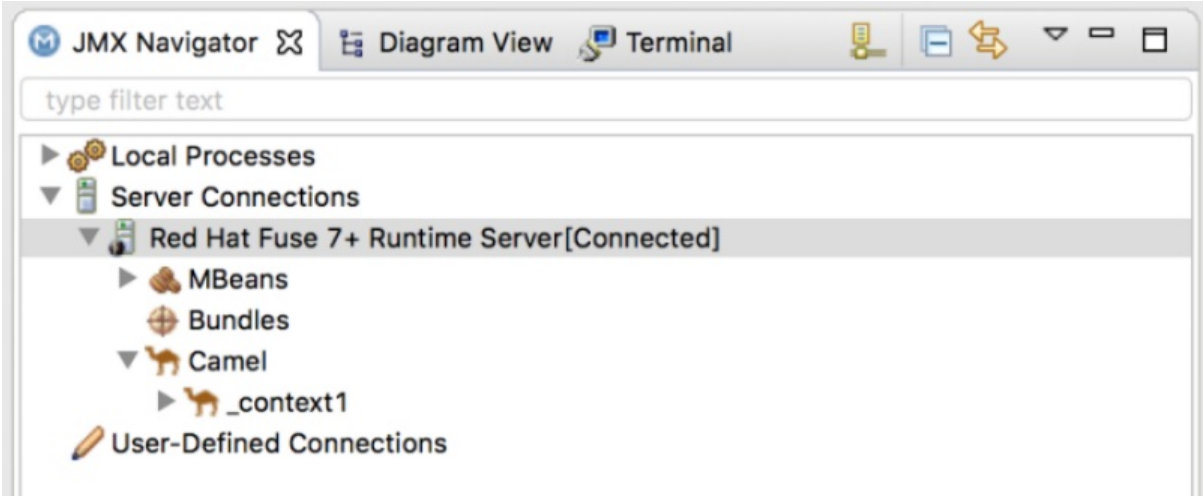
For a module, **synchronized** means that the published module is identical to its local counterpart. Because automatic publishing is enabled, changes made to the ZooOrderApp project are published in seconds (according to the value of the **Publishing interval**).

- **JMX[Disconnected]**

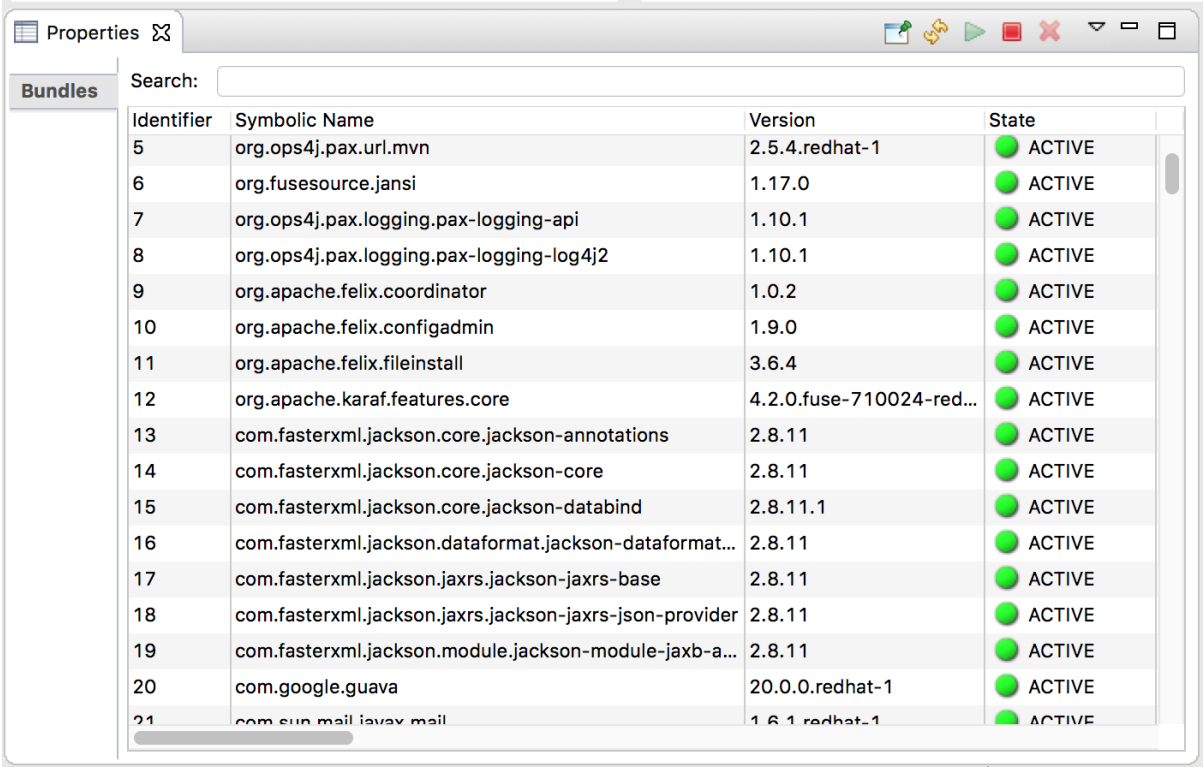
CONNECTING TO THE RUNTIME SERVER

After you connect to the runtime server, you can see the published elements of your **ZooOrderApp** project and interact with them.

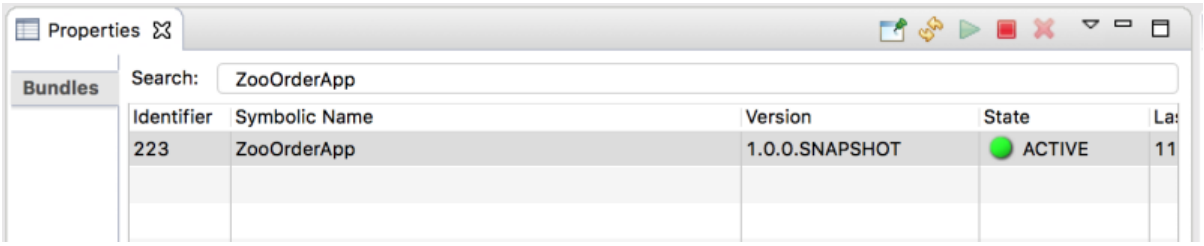
1. In the **Servers** view, double-click **JMX[Disconnected]** to connect to the runtime server.
2. In the **JMX Navigator**, expand the **Camel** folder to expose the elements of the **ZooOrderApp**.

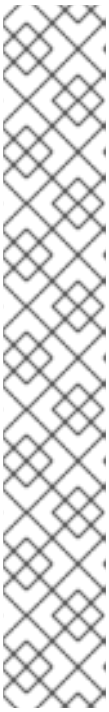


3. Click the **Bundles** node to populate the **Properties** view with the list of bundles installed on the runtime server:



4. In the **Search** field, type ZooOrderApp. The corresponding bundle is shown:

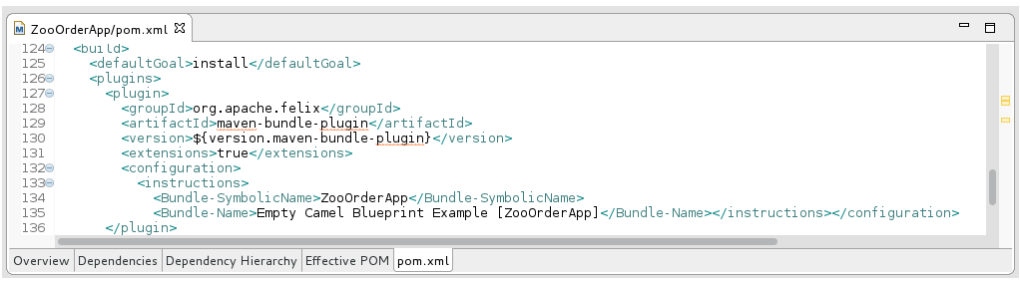




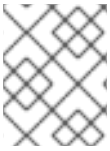
NOTE

Alternatively, you can issue the **osgi:list** command in the **Terminal** view to see a generated list of bundles installed on the server runtime. The tooling uses a different naming scheme for OSGi bundles displayed by the **osgi:list** command. In this case, the command returns **Camel Blueprint Quickstart**, 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:



UNINSTALLING THE ZOOORDERAPP PROJECT

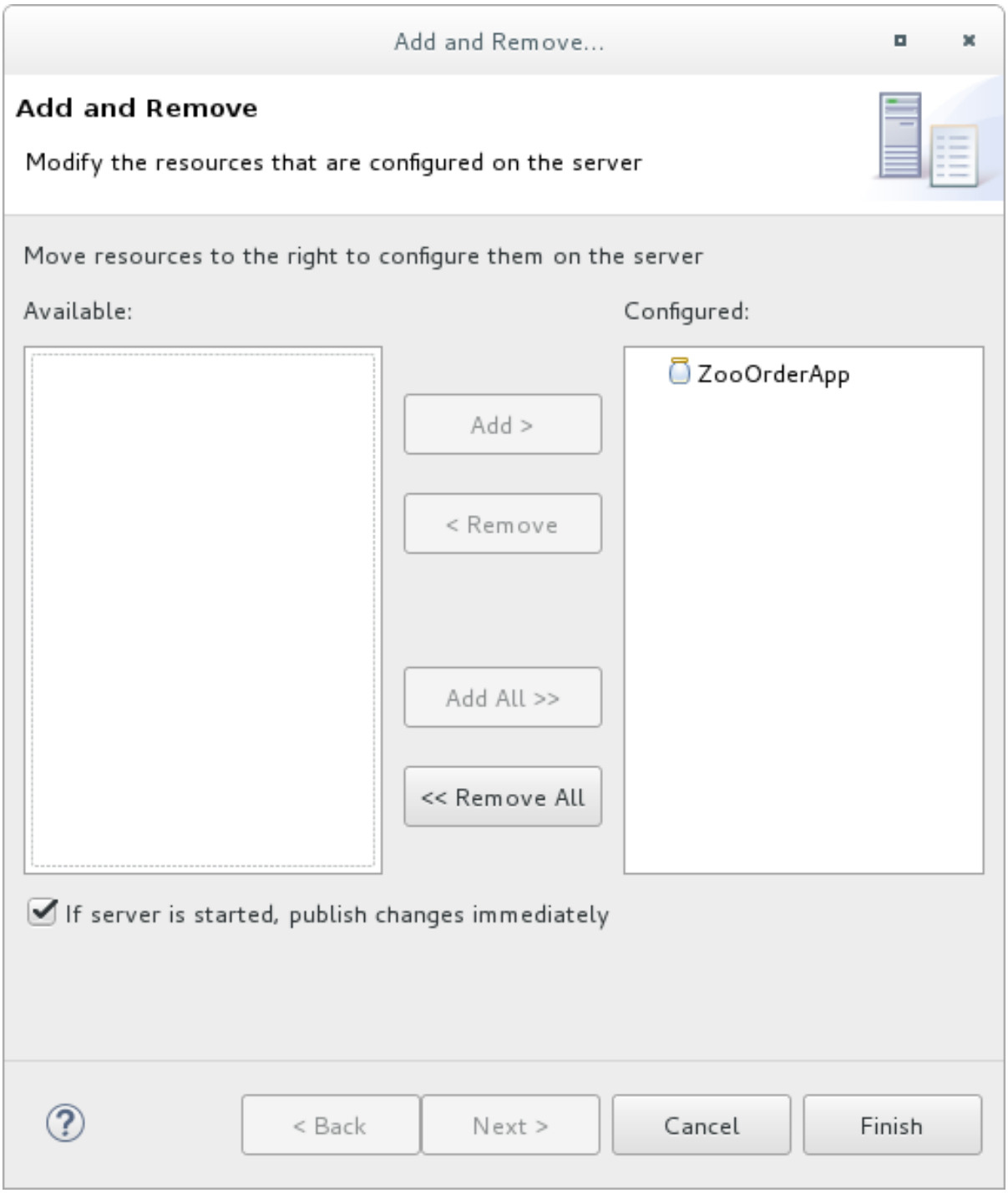


NOTE

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

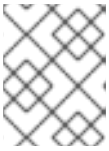
To remove the **ZooOrderApp** resource from the runtime server:

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



3. In the **Configured** column, select **ZooOrderApp**, and then click **Remove** to move the **ZooOrderApp** resource to the **Available** column.
4. Click **Finish**.

5. In the **Servers** view, right-click **JMX[Connected]** and then click **Refresh**.
   The **Camel** tree under **JMX[Connected]** disappears.



NOTE

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

6. With the **Bundles** page displayed in the **Properties** view, scroll down to the end of the list to verify that the ZooOrderApp's bundle is no longer listed.