Red Hat Fuse 7.0

Tooling User Guide

Tooling User Guide

Last Updated: 2018-08-02

Red Hat Fuse 7.0 Tooling User Guide

Tooling User Guide

Legal Notice

Abstract

This guide describes how to use the Red Hat Fuse Tooling, which provides developer tools designed to increase your productivity when designing, developing, testing, and debugging your integration applications.

Table of Contents

PREFACE

Red Hat Fuse Tooling is an Eclipse-based IDE that simplifies and streamlines the process of developing integration applications within Red Hat Developer Studio. Fuse Tooling provides a set of developer tools that are specifically designed to work with:

- Red Hat Fuse
- Red Hat JBoss EAP
- Apache Camel
- Apache CXF
- Apache Karaf
- Spring Boot

This guide provides information about how to use Fuse Tooling to:

- Create a project for your application, including Maven dependencies
- Connect and configure Enterprise Integration Patterns to build routes
- Browse endpoints and routes
- Drag and drop messages onto running routes
- Browse and visualize runtime processes via JMX
- Debug locally running Camel contexts and routes
- Test your application by:
  - Creating and using JUnit test cases on Apache Camel routes
  - Using JMX to analyze running components
  - Tracing messages through Apache Camel routes
- Deploy your application

For new users, the Tooling Tutorials provide step-by-step instructions on how to create, debug, test, and deploy a sample Camel application.

PART I. DEVELOPING APPLICATIONS

CHAPTER 1. CREATING A NEW FUSE INTEGRATION PROJECT

OVERVIEW

Creating a new Fuse Integration project involves these main steps:

- Specifying the project name and workspace
- Configuring the project deployment environment
- Selecting a project template
- If necessary, resolving Maven dependency errors

After you configure the project, the tooling downloads all of the required Maven dependencies and creates the POM file needed to run and publish the project.

NOTE

The first time that you build a Fuse project in Developer Studio, it might take several minutes for the wizard to finish generating the project as it downloads dependencies from remote Maven repositories.

BEFORE YOU BEGIN

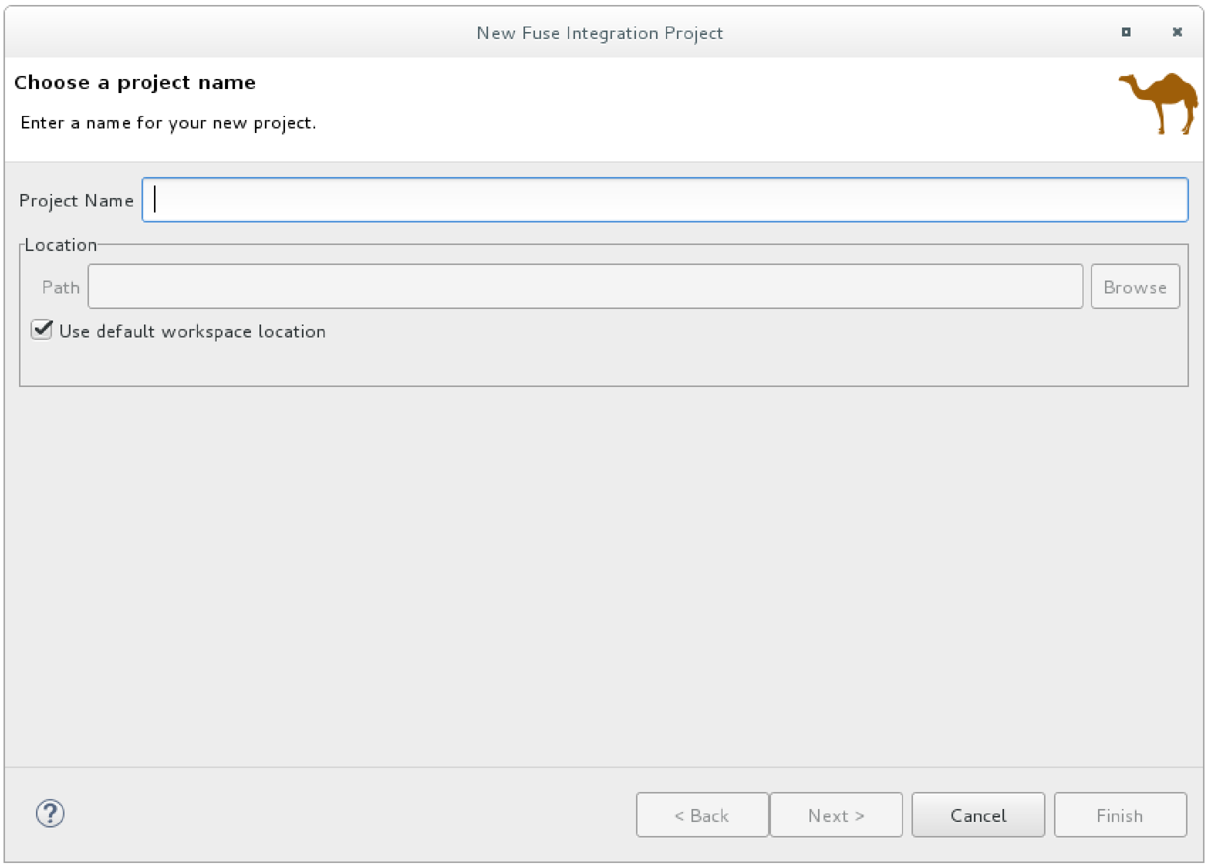
Before you create a new Fuse Integration project, you should have the following information:

- Your target runtime environment: Fuse on OpenShift or Fuse standalone (Spring Boot, Fuse on Karaf, or Fuse on EAP)
- The Camel version (if other than the default used by the tooling)

SPECIFYING THE PROJECT NAME AND WORKSPACE

To create a new Fuse Integration project, follow these steps:

1. Select **New** → **Project** → **Red Hat Fuse** → **Fuse Integration Project** to open the **New Fuse Integration Project** wizard.
   The wizard opens with the **Use default workspace location** option selected in the **Location** pane.


New Fuse Integration Project
Choose a project name
Enter a name for your new project.
Project Name
Location
Path
Browse
Use default workspace location
< Back
Next >
Cancel
Finish

2. In **Project Name**, type a name for the new project, for example **`MySampleProject`**.

3. Specify the workspace location in which you want to store the data for the project.

   - To use the default workspace, leave the **Use default workspace location** option enabled.

   - To use an alternative location clear the **Use default workspace location** option and specify a location in the **Path** field.

     Click Browse... to quickly find and select an alternate workspace.

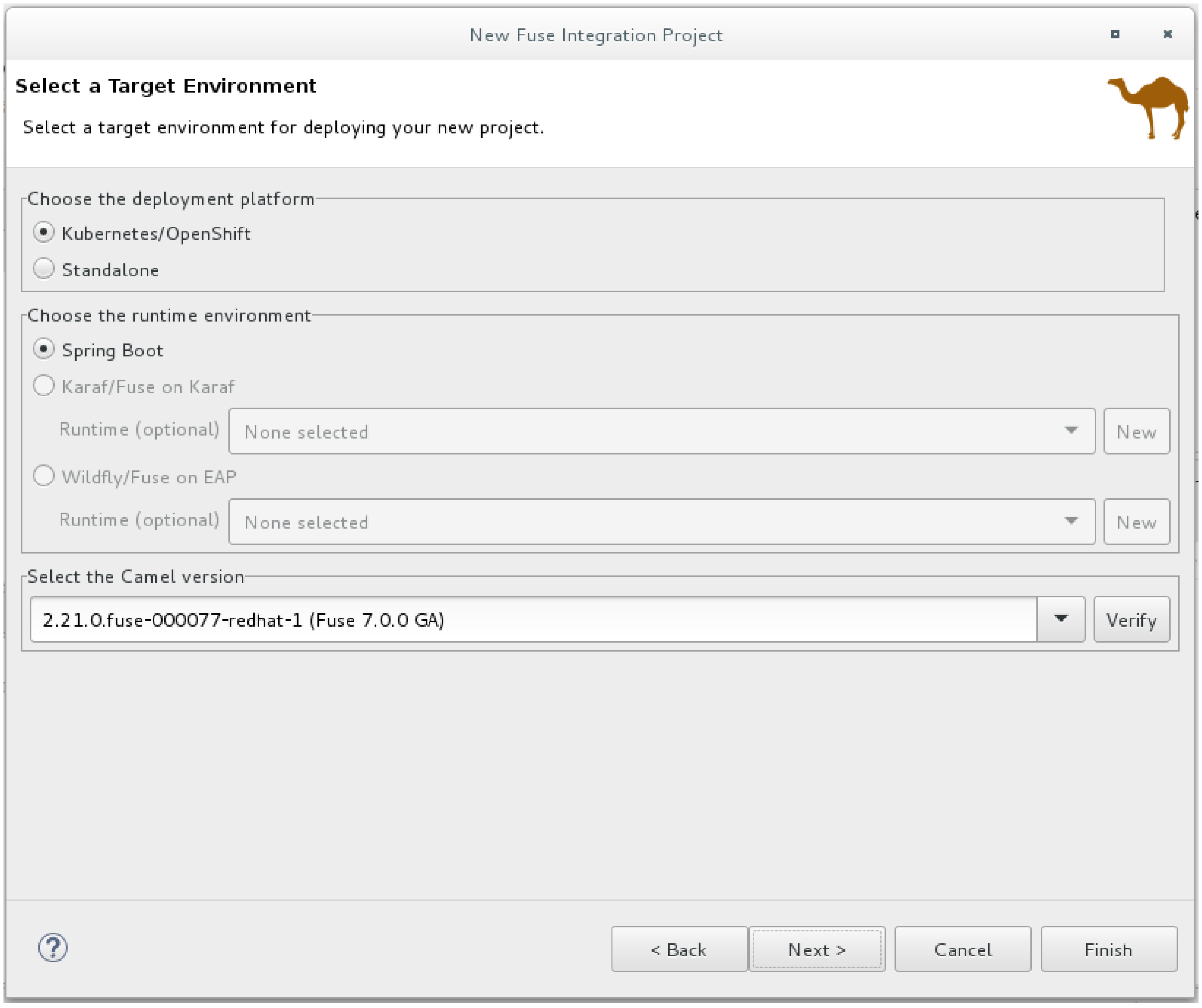
4. Click **Next** to open the **Select a Target Environment** page.

CONFIGURING THE PROJECT DEPLOYMENT ENVIRONMENT

When you create a new project, you specify the project's target deployment environment so that your project has the resources it needs at runtime. You must select a deployment platform and a Camel version. Optionally, you can specify a runtime configuration.

With the **Select a Target Environment** page open:

1. Select whether you want to deploy the project on **Kubernetes/OpenShift** or on a **Standalone** platform.


New Fuse Integration Project
Select a Target Environment
Select a target environment for deploying your new project.
Choose the deployment platform
Kubernetes/OpenShift
Standalone
Choose the runtime environment
Spring Boot
Karaf/Fuse on Karaf
Runtime (optional)
None selected
New
Wildfly/Fuse on EAP
Runtime (optional)
None selected
New
Select the Camel version
2.21.0.fuse-000077-redhat-1 (Fuse 7.0.0 GA)
Verify
< Back
Next >
Cancel
Finish

If you select **Kubernetes/OpenShift** for the deployment platform, the **Sprint Boot** runtime is automatically selected and you can skip to Step 3.

2. If you select **Standalone** for the deployment platform:

    a. Choose a target runtime environment:

        - **Spring Boot**
        - **Karaf/Fuse on Karaf**
        - **Wildfly/Fuse on EAP**

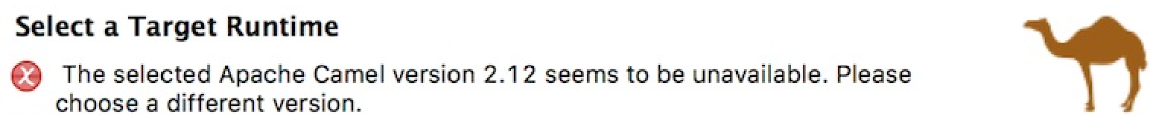
    b. For the Karaf and EAP standalone runtime environments, choose one of the following options for the runtime configuration:

        - Accept the **None selected** option (you can define the runtime configuration later).
        - Select an existing runtime configuration from the drop-down menu.
        - Create a new runtime configuration as described in the section called "Creating a new target runtime (optional)".

3. In the **Select the Camel version for your new project** pane, accept the default Camel version associated with the runtime or change the default by:

    - Selecting a Camel version from the drop-down list. Fuse Tooling supports the listed productized versions.

- Typing a different Camel version if you want to experiment with non-productized versions (that are not supported).
  You can click the **Verify** button to check whether the tooling can access the specified version. If not, a notification similar to the following example appears in the **Select a Target Runtime** page header:



Select a Target Runtime

The selected Apache Camel version 2.12 seems to be unavailable. Please choose a different version.

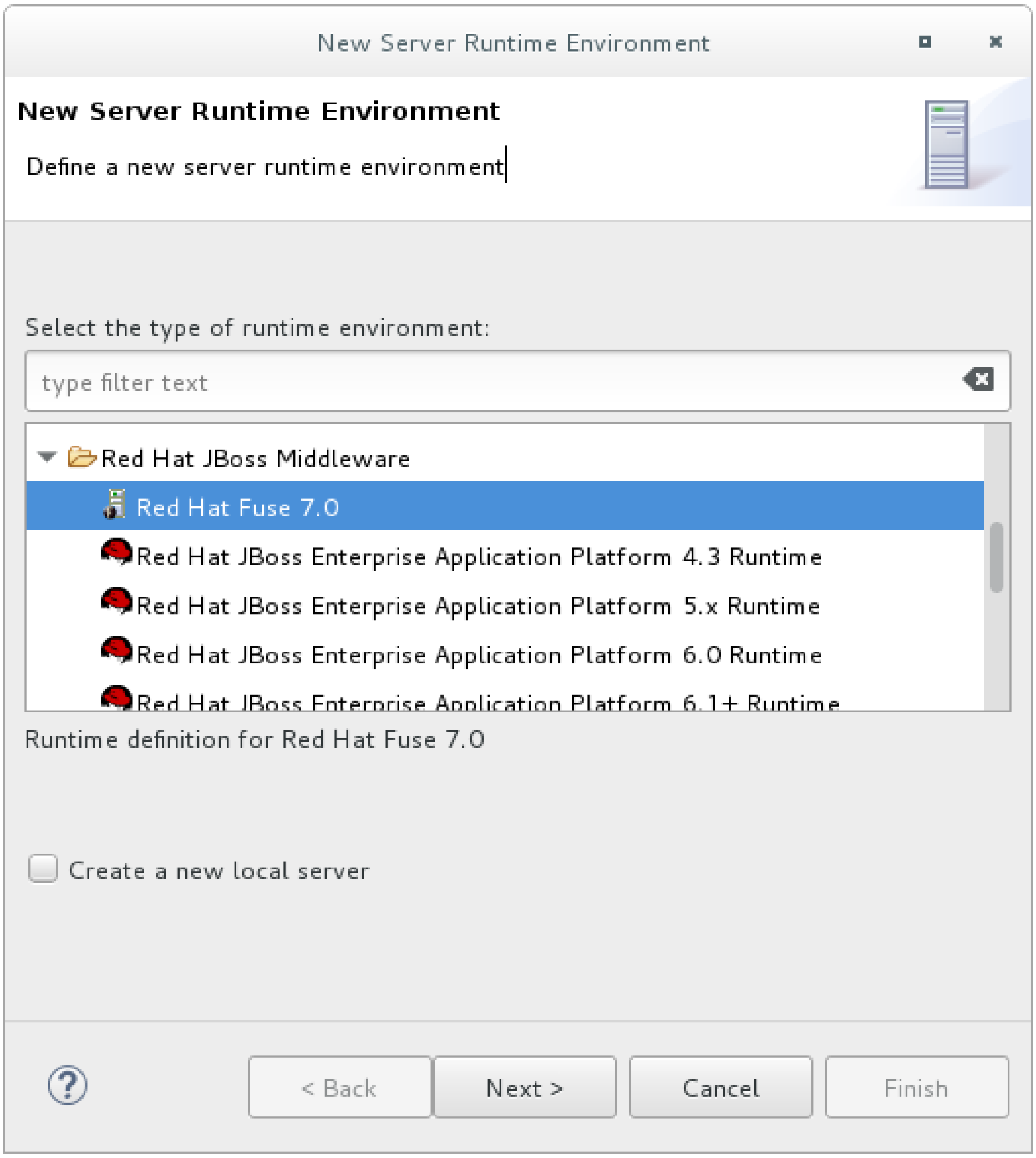
> **NOTE**
>
> After you create, configure, and save a project, you can change the Camel version. See Chapter 12, *Changing the Camel version*.

4. After you choose a runtime environment and a Camel version on which to base your new Fuse Integration project, click **Next** to open the wizard's **Advanced Project Setup** page and then follow the steps in the section called "Selecting a project template".

CREATING A NEW TARGET RUNTIME (OPTIONAL)

For the Karaf and EAP standalone runtime environments, you can optionally create a new runtime configuration from the **New Fuse Integration Project** wizard.

1. From the wizard's the **Select a Target Runtime** page, click **New** to open the **New server runtime environment** page:

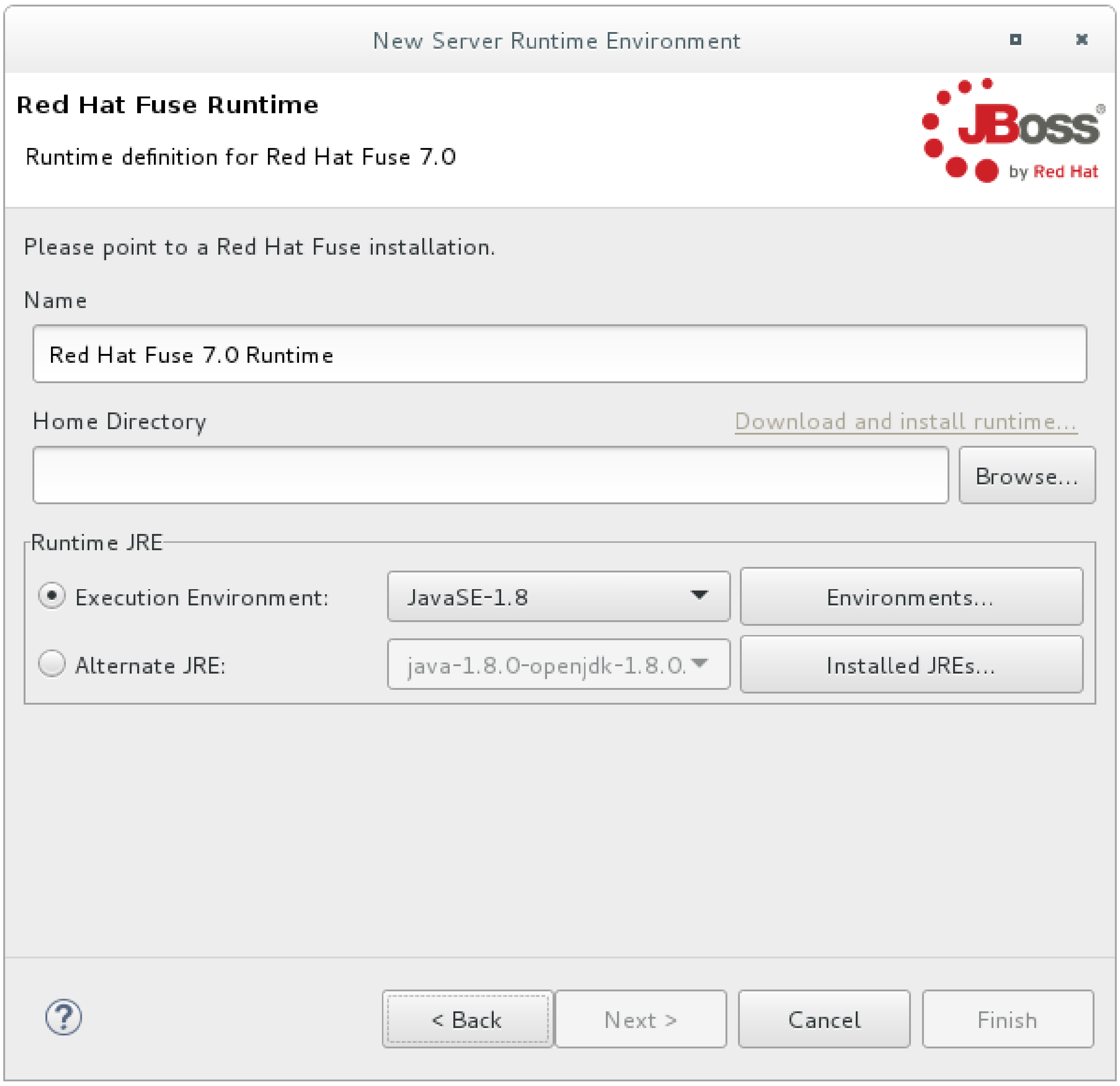

New Server Runtime Environment
New Server Runtime Environment
Define a new server runtime environment
Select the type of runtime environment:
type filter text
Red Hat JBoss Middleware
Red Hat Fuse 7.0
Red Hat JBoss Enterprise Application Platform 4.3 Runtime
Red Hat JBoss Enterprise Application Platform 5.x Runtime
Red Hat JBoss Enterprise Application Platform 6.0 Runtime
Red Hat JBoss Enterprise Application Platform 6.1+ Runtime
Runtime definition for Red Hat Fuse 7.0
Create a new local server
< Back
Next >
Cancel
Finish

2. Expand the **`Red Hat JBoss Middleware`** folder, and then select a Red Hat Fuse runtime environment (for example, **`Red Hat Fuse 7.0`**).
   Leave the **Create a new local server** option unchecked. You can create the local server later when you are ready to publish your project (see Section 28.1, “Adding a Server”).

   NOTE

   If you check the **Create a new local server** option, the **New Fuse Integration Project** wizard walks you through additional steps to define and configure the Fuse server runtime (as described in Section 28.1, “Adding a Server”). Then, when it builds the project, it also adds the server runtime to the **Servers** view in the **Fuse Integration** perspective.

3. Click **Next** to open the server’s **New Server Runtime Environment** page:



Red Hat Fuse Runtime

Runtime definition for Red Hat Fuse 7.0

Please point to a Red Hat Fuse installation.

Name: Red Hat Fuse 7.0 Runtime

Home Directory:

Runtime JRE

(●) Execution Environment: JavaSE-1.8

( ) Alternate JRE: java-1.8.0-openjdk-1.8.0.

4. Specify the **Name**, **Home Directory**, **Execution Environment** of the server runtime:

   - **Name** — Accept the default or enter a new name for the runtime environment.
   - **Home Directory** — Click the **Browse** button to locate and select the server runtime's installation directory.

     NOTE

     If the server is not already installed on your machine, you can install it now by clicking the **Download and install runtime** link and then following the site's download instructions. Depending on the site, you might be required to provide valid credentials before you can continue the download process.

   - **Runtime JRE: Execution Environment** — Accept the default or select another JavaSE version from the drop-down list.
     If the version that you want does not appear on the list, click the **Environments** button and select the version from that list. The JRE version that you select must be installed on your machine.

NOTE

Fuse 7.0 requires JRE version 1.8.

- **Runtime JRE: Alternate JRE** - If your project requires a different version of Java, you can use this option.

5. Click **Finish** to return to the **New Fuse Integration Project** wizard's **Select a Target Runtme** page:
   The newly configured target runtime appears in the **Target Runtime** pane's drop-down menu, and the Camel version supported by the runtime appears in the **Camel Version** pane, grayed out.

   After you create a Fuse Integration project, it is possible to change the Camel version. See Chapter 12, *Changing the Camel version*.

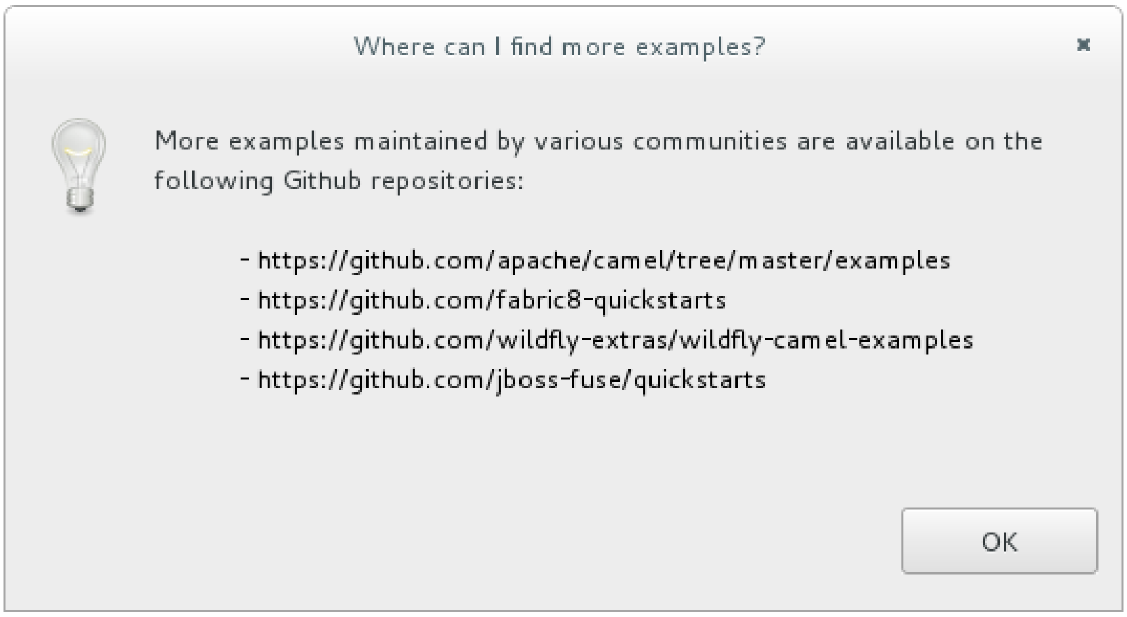
6. Click **Next** to specify a template for the project as described in the section called "Selecting a project template".

SELECTING A PROJECT TEMPLATE

The **Advanced Project Setup** page provides a list of templates that you can use as a starting point for your new project. The templates, based on common use cases, provide sample code and data to get you started quickly. The list of available templates depends on the runtime environment that you selected on the previous page. Select a template to view its description in the right pane.

NOTE

If you do not see a template that meets your requirements, you can click the **Where can I find more examples to use as templates?** link to open an information dialog with a list of URLs with more examples:



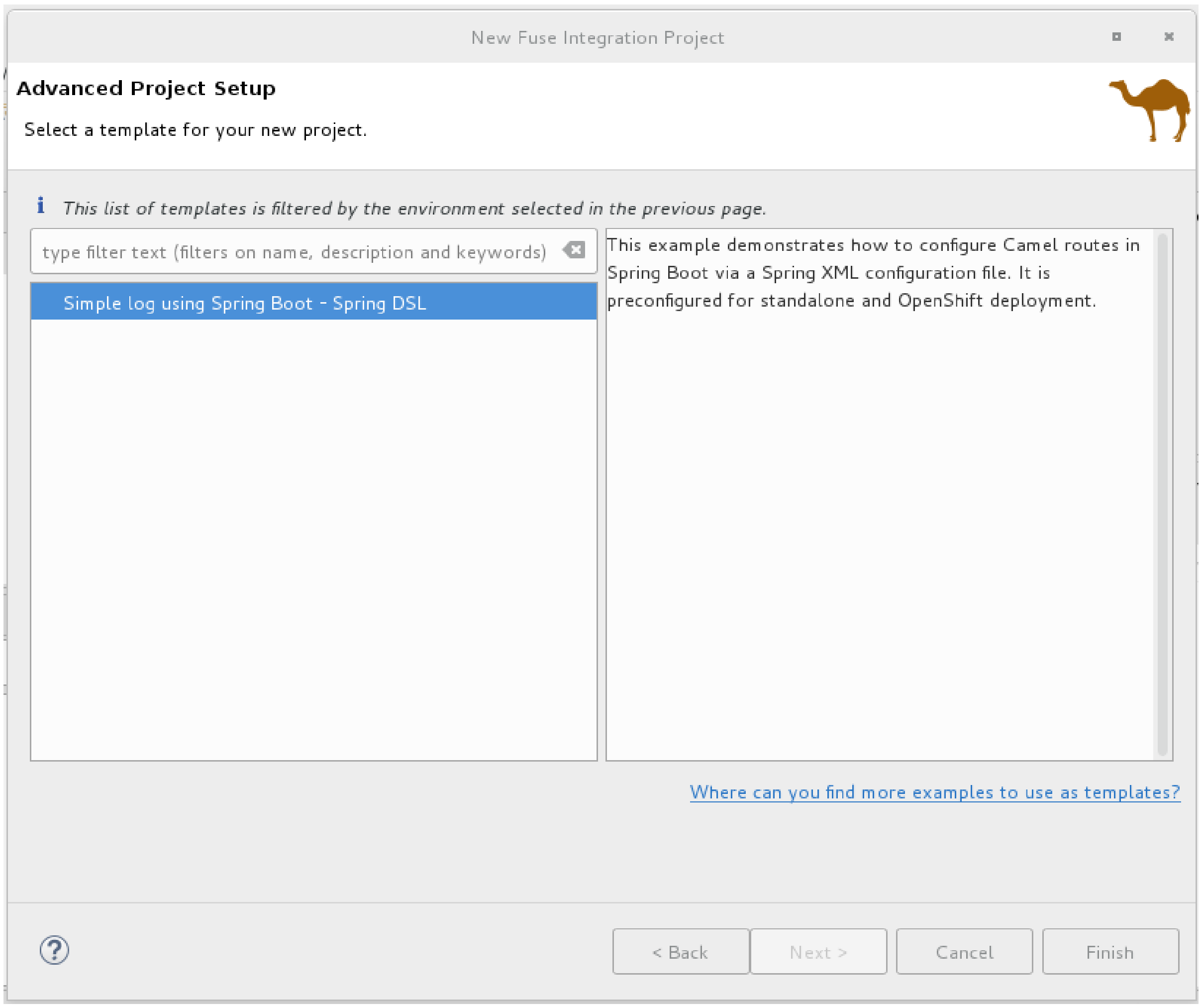
More examples maintained by various communities are available on the following Github repositories:

- https://github.com/apache/camel/tree/master/examples
- https://github.com/fabric8-quickstarts
- https://github.com/wildfly-extras/wildfly-camel-examples
- https://github.com/jboss-fuse/quickstarts

To use one of these examples:

1. Select **Cancel** to exit the **New Fuse Integration Project** wizard.
2. Clone the repository from one of the listed URLs.
3. Import the example project into Developer Studio as described in Chapter 13, *Importing an existing Maven project*.

- For **Fuse on OpenShift** there is a single template that demonstrates how to configure Camel routes in Spring Boot using a Spring XML configuration file. This template creates a Fuse Integration Services project and requires a Camel version newer than *2.18.1.redhat-000012*. This template creates a project that runs on OpenShift servers, and it supports the Spring DSL only. For details on using this template, see Chapter 7, *Getting Started with Fuse Integration Services*.



Advanced Project Setup

Select a template for your new project.

This list of templates is filtered by the environment selected in the previous page.

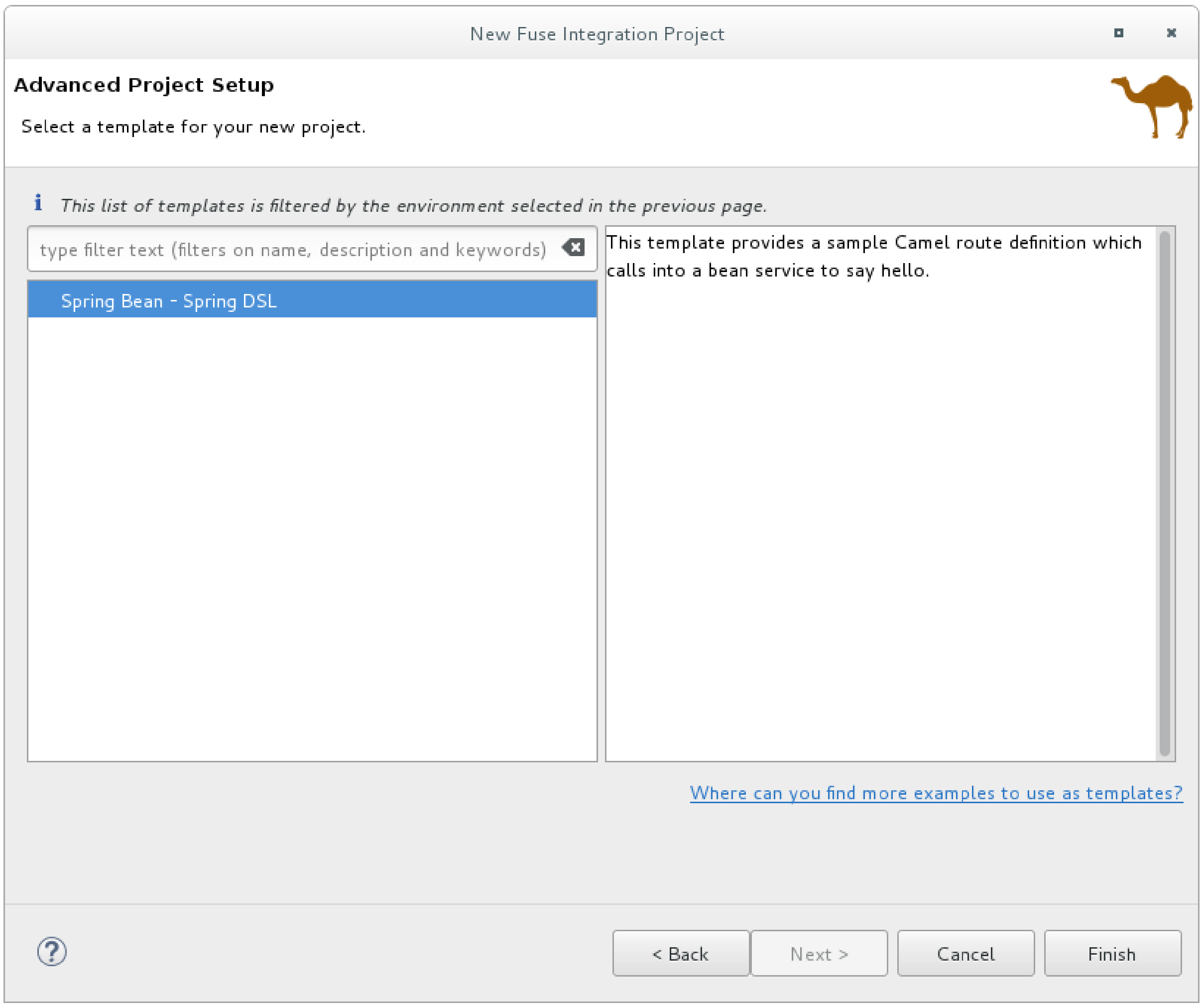
type filter text (filters on name, description and keywords)

Simple log using Spring Boot - Spring DSL

This example demonstrates how to configure Camel routes in Spring Boot via a Spring XML configuration file. It is preconfigured for standalone and OpenShift deployment.

Where can you find more examples to use as templates?

- For **Wildfly** or **Fuse on EAP** there is a single template that provides a sample Camel route that calls into a bean service to say "Hello". This template creates a project that runs on Red Hat EAP servers, and it supports the Spring DSL only.


New Fuse Integration Project

Advanced Project Setup

Select a template for your new project.

This list of templates is filtered by the environment selected in the previous page.

type filter text (filters on name, description and keywords)

Spring Bean - Spring DSL

This template provides a sample Camel route definition which calls into a bean service to say hello.

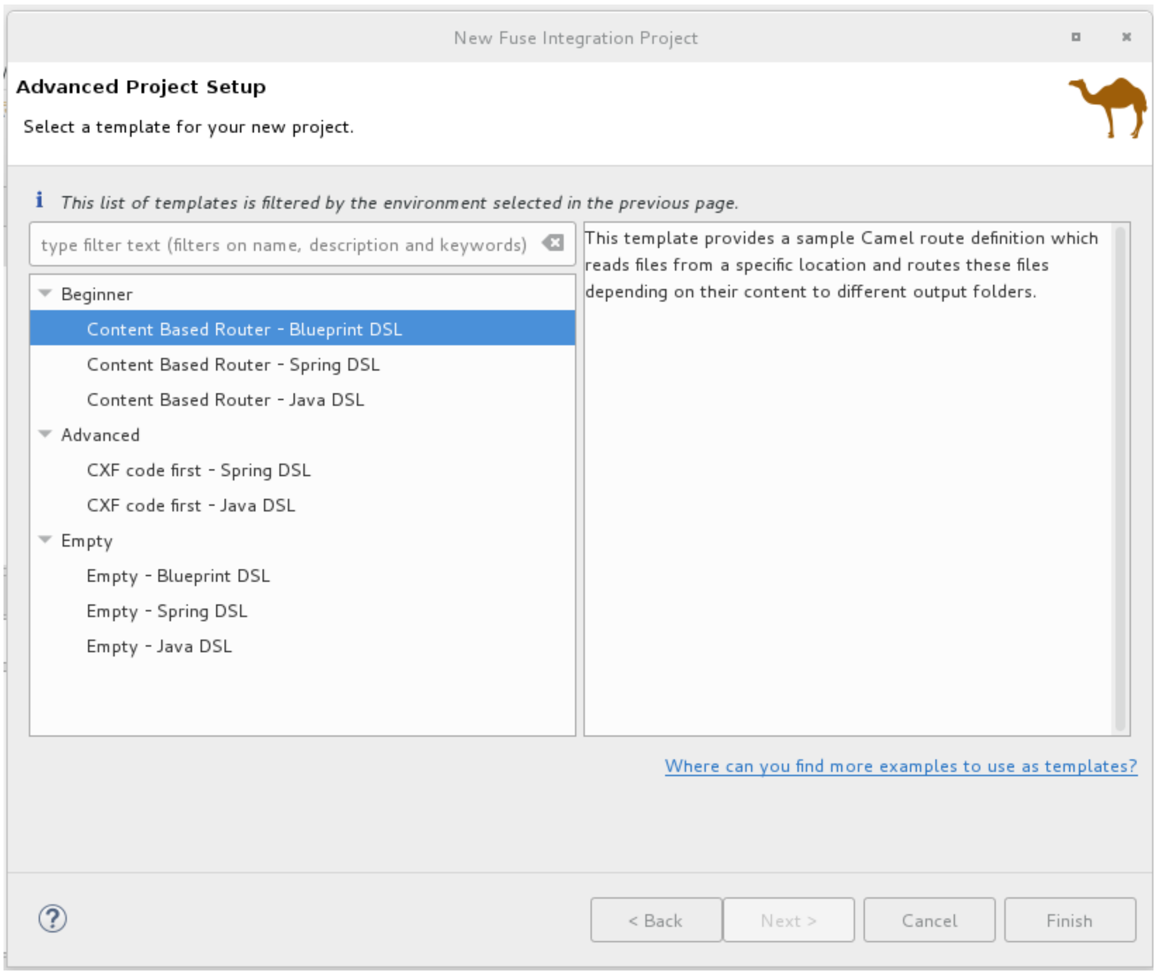
Where can you find more examples to use as templates?

- For **Karaf** or **Fuse on Karaf**, you have a choice of templates. You can create an empty project, which creates a skeleton Camel context routing file based on one of the three supported Domain Specific Languages (DSLs), or you can use a predefined template, each of which is based on a common use case. Individual templates might not support all DSL options.

NOTE

For Java DSL, the tooling generates a `CamelRoute.java` file that you can edit in the tooling's Java editor, but it does not generate a graphical diagram representation of it.

  - **Content Based Router** — Provides a sample Camel route that reads files from a specific location and routes them to different output folders according to message content.
    This template creates a project that runs on Red Hat Fuse servers, and it supports all three of the DSLs.
  - **CXF code first** — Provides a sample Camel route that is started by a CXF web service call.
    This template creates a project that runs on Red Hat Fuse servers, and it supports the Spring and Java DSLs only.



New Fuse Integration Project

Advanced Project Setup

Select a template for your new project.

This list of templates is filtered by the environment selected in the previous page.

type filter text (filters on name, description and keywords)

- Beginner
  - Content Based Router - Blueprint DSL
  - Content Based Router - Spring DSL
  - Content Based Router - Java DSL
- Advanced
  - CXF code first - Spring DSL
  - CXF code first - Java DSL
- Empty
  - Empty - Blueprint DSL
  - Empty - Spring DSL
  - Empty - Java DSL

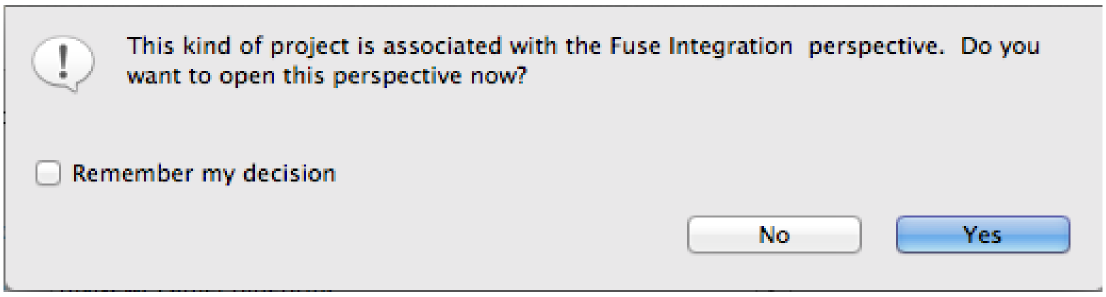
This template provides a sample Camel route definition which reads files from a specific location and routes these files depending on their content to different output folders.

Where can you find more examples to use as templates?

1. Select a template from the list.

2. Click **Finish**.
   The tooling starts building the project and adds it to the **Project Explorer** view.

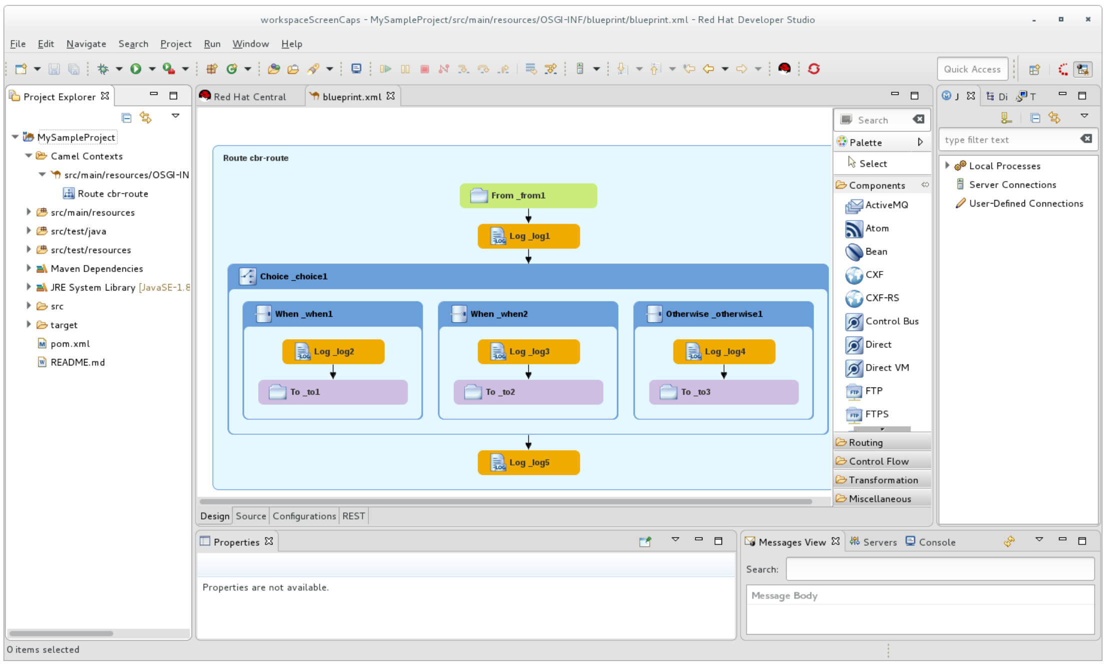
   If the **Fuse Integration** perspective is not already open, the tooling asks whether you want to switch to it now:



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

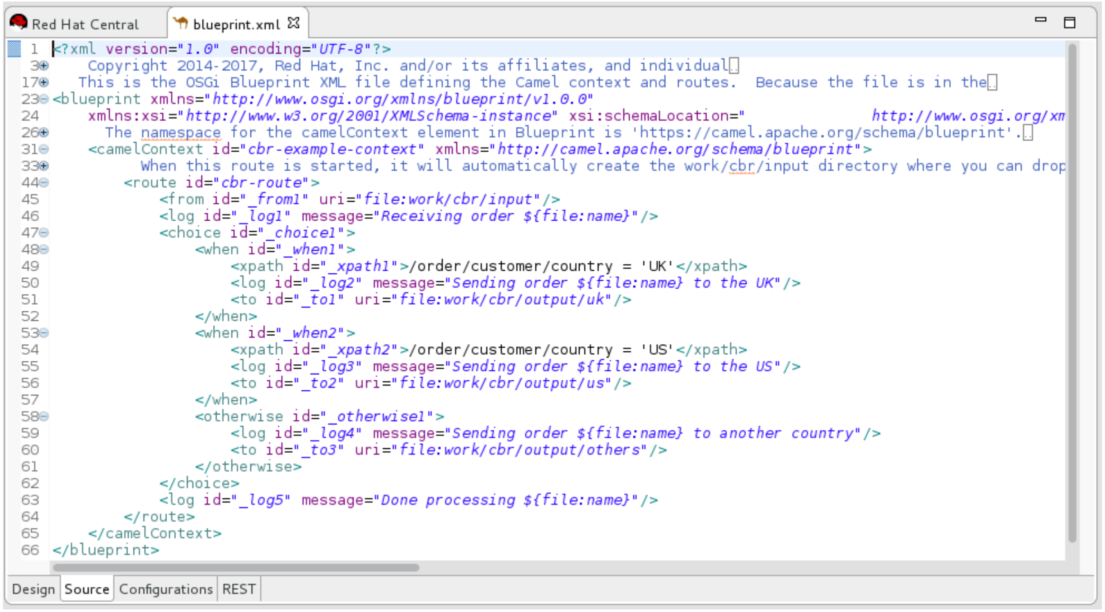
☐ Remember my decision

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



The project appears in the **Project Explorer** view. By default, the project includes an Apache Camel context (XML) file.

4. Click the **Source** tab at the bottom of the canvas to see the generated Camel context file:



```
<?xml version="1.0" encoding="UTF-8"?>
    Copyright 2014-2017, Red Hat, Inc. and/or its affiliates, and individual
   This is the OSGi Blueprint XML file defining the Camel context and routes.  Because the file is in the
<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/xm
      The namespace for the camelContext element in Blueprint is 'https://camel.apache.org/schema/blueprint'.
    <camelContext id="cbr-example-context" xmlns="http://camel.apache.org/schema/blueprint">
          When this route is started, it will automatically create the work/cbr/input directory where you can drop
        <route id="cbr-route">
            <from id="_from1" uri="file:work/cbr/input"/>
            <log id="_log1" message="Receiving order ${file:name}"/>
            <choice id="_choice1">
                <when id="_when1">
                    <xpath id="_xpath1">/order/customer/country = 'UK'</xpath>
                    <log id="_log2" message="Sending order ${file:name} to the UK"/>
                    <to id="_to1" uri="file:work/cbr/output/uk"/>
                </when>
                <when id="_when2">
                    <xpath id="_xpath2">/order/customer/country = 'US'</xpath>
                    <log id="_log3" message="Sending order ${file:name} to the US"/>
                    <to id="_to2" uri="file:work/cbr/output/us"/>
                </when>
                <otherwise id="_otherwise1">
                    <log id="_log4" message="Sending order ${file:name} to another country"/>
                    <to id="_to3" uri="file:work/cbr/output/others"/>
                </otherwise>
            </choice>
            <log id="_log5" message="Done processing ${file:name}"/>
        </route>
    </camelContext>
</blueprint>
```

NOTE

If you want to add another new Camel context file to the project, see Chapter 11, *Creating a New Camel XML file*.

RESOLVING MAVEN DEPENDENCY ERRORS

You might encounter Maven dependency errors after you create a new Fuse Integration project.

Though it can happen at other times, it more typically occurs when you cancel a project build before the process has finished. Interrupting the process in this way often prevents all of the project's dependencies from downloading from the Maven repositories, which can take some time.

You can often resolve these dependency errors by updating Maven dependencies as follows:

1. In the **Project Explorer** view, right-click the root project to open the context menu.
2. Select **Maven** → **Update Project**.
3. In the **Update Maven Project** wizard:
    - Select the project that you want to update, if more than one appears in the wizard’s list.
    - Click the **Force Update of Snapshots**/**Releases** option to enable it.
4. Click **OK**.
   In the bottom, right corner of the workbench, you can view the progress status bar churning as missing dependencies are downloaded from the Maven repositories.

CHAPTER 2. EDITING A ROUTING CONTEXT IN THE ROUTE EDITOR

> The following sections describe how to edit a routing context.

2.1. ADDING PATTERNS TO A ROUTE

Routes consist of a sequence of connected patterns, referred to as **nodes** once they are placed on the canvas inside a **`Route`** container node. A complete route typically consists of a starting endpoint, a string of processing nodes, and one or more destination endpoints.

When you add a pattern into a Route container on the canvas, the pattern takes on a color that indicates its type of node:

- Blue — Route containers, which correspond to route elements in the context file, and other container nodes, such as **`when`** and **`otherwise`** EIPs that contain other EIPs that complete their logic
- Green — Consumer endpoints that input data entering routes
- Orange — EIPs that route, transform, process, or control the flow of data transiting routes
- Purple — Producer endpoints that output the data exiting routes

Procedure

To add a pattern to a route:

1. In the **Palette**, locate the pattern that you want to add to the route.
2. Use one of the following methods:
   - Click the pattern in the **Palette** and then, in the canvas, click the route container.
   - Drag the pattern over the target **`Route`** container and drop it.
     Alternatively, you can add a pattern on an existing node that has no outgoing connection, or on a connection existing between two nodes, to have the tooling automatically wire the connections between all nodes involved.

     The tooling checks whether the resulting connection is valid and then either allows or prevents you from adding the pattern on the target. For valid connections, the tooling behaves differently depending on whether the target is a node or a connection:
     - For an **existing node**, the tooling adds the new node to the target node’s outgoing side (beneath or to the right of it depending on how the editor preferences are set) and automatically wires the connection between them
     - For an **existing connection**, the tooling inserts the new node between the two connected nodes and automatically rewires the connections between the three nodes
3. Optionally, you can manually connect two nodes:
   a. In the **`Route`** container on the canvas, select the source node to display its connector arrow.

   b. Drag the source node's connector arrow ( ) to the target node, and release the mouse button to drop the connector on it.

> **NOTE**
>
> Not all nodes can be connected. When you try to connect a source node to an invalid target node, the tooling displays the symbol attached to the mouse cursor, and the connector fails to stick to the target node.

4. After you add a pattern inside a Route container, you can drag it to different location inside the route container or to another route container on the canvas, as long as it can establish a valid connection. You can also relocate existing nodes that are already connected, as long as the move can establish another valid connection.

5. Select **File** → **Save**. The tooling saves routes in the context file regardless of whether they are complete.

The new pattern appears on the canvas in the `Route` container and becomes the selected node. The **Properties** view displays a list of the new node's properties that you can edit.

Changing the layout direction

When you connect one node to another, the tooling updates the layout according to the route editor's layout preference. The default is **Down**.

To access the route editor 's layout preference:

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

- On OS X, select **Developer Studio** → **Preferences** → **Fuse Tooling** → **Editor** → **Choose the layout direction for the diagram editor**.

Related topics

- Section 2.2, “Configuring a pattern”

- Section 2.3, “Removing patterns from a route”

2.2. CONFIGURING A PATTERN

Overview

Most patterns require some explicit configuration. For example, an endpoint requires an explicitly entered `URI`.

The tooling's **Properties** view provides a form that lists all of the configuration details a particular pattern supports. The **Properties** view also provides the following convenience features:

- validating that all required properties have values

- validating that supplied values are the correct data type for the property

- drop-down lists for properties that have a fixed set of values
- drop-down lists that are populated with the available bean references from the Apache Camel Spring configuration

Procedure

To configure a pattern:

1. On the canvas, select the node you want to configure.
   The **Properties** view lists all of the selected node's properties for you to edit. For EIPs, the **Details** tab lists all of a pattern's properties. For components from the **Components** drawer, the **Details** tab lists the general properties and those that require a value, and the **Advanced** tab lists additional properties grouped according to function.

   The **Documentation** tab describes the pattern and each of its properties.

2. Edit the fields in the **Properties** view to configure the node.

3. When done, save your work by selecting **File** → **Save** from the menu bar.

2.3. REMOVING PATTERNS FROM A ROUTE

Overview

As you develop and update a route, you may need to remove one or more of the route's nodes. The node's icon makes this easy to do. When you delete a node from the canvas, all of its connections with other nodes in the route are also deleted, and the node is removed from the corresponding route element in the context file.

NOTE

You can also remove a node by opening its context menu and selecting **Remove**.

Procedure

To remove a node from a route:

1. Select the node you want to delete.

2. Click its icon.

3. Click **Yes** when asked if you are sure you want to delete this element.

The node and all of its connections are deleted from the canvas, and the node is removed from its corresponding route element in the context file.

Related topics

- Section 2.1, "Adding patterns to a route"

2.4. ADDING ROUTES TO THE ROUTING CONTEXT

Overview

The camelContext element within an XML context file creates a routing context. The camelContext element can contain one or more routes, and each route, displayed on the canvas as a **`Route`** container node, maps to a route element in the generated camelContext element.

Procedure

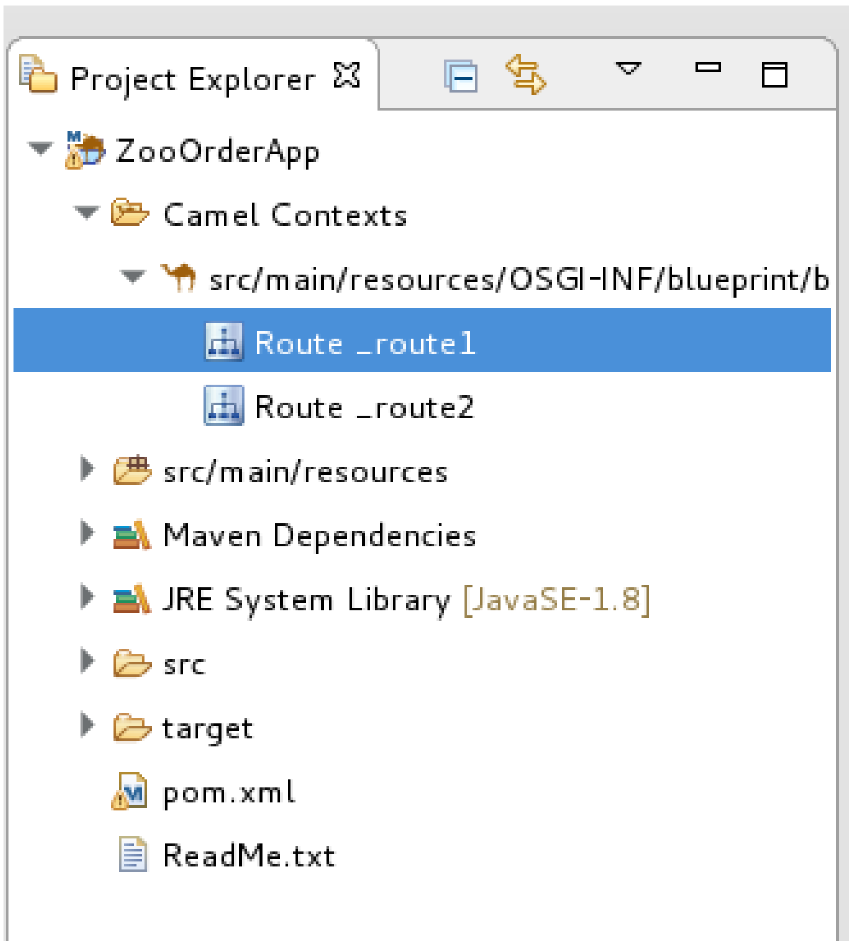
To add another route to the camelContext:

1. In the **Design** tab, do one of the following:
    - Click a **`Route`** pattern in the **Palette**'s **Routing** drawer and then click in the canvas where you want to place the route.
    - Drag a **`Route`** pattern from the **Palette**'s **Routing** drawer and drop it onto the canvas. The **Properties** view displays a list of the new route’s properties that you can edit.
2. In the **Properties** view, enter:
    - An ID (for example, **Route2**) for the new route in the route’s **Id** field

      NOTE

      The tooling automatically assigns an ID to EIP and component patterns dropped on the canvas. You can replace these autogenerated IDs with your own to distinguish the routes in your project.

    - A description of the route in the **Description** field
    - Values for any other properties, as needed. Required properties are indicated by an asterisk (*).
3. On the menu bar, select **File** → **Save** to save the changes you made to the routing context file.
4. To switch between multiple routes, select the route that you want to display on the canvas by clicking its entry under the project’s **`Camel Contexts`** folder in the **Project Explorer** view.



5. To display all routes in the context, as space allows, click the context file entry in the **Project Explorer** view.

6. To view the code generated by the tooling when you add a route to the canvas, click the **Source** tab.

NOTE

You can alternately add a route in the **Source** tab, by adding a <route/> element to the existing list within the camelContext element.

2.5. DELETING A ROUTE

Overview

In some cases you made need to delete an entire route from your routing context. The Route container's 🗑 icon makes this easy to do. When you delete a route, all of the nodes inside the Route container are also deleted, and the corresponding route element in the context file is removed.

NOTE

You can also remove a route using the Route container's context menu and selecting **Remove**.

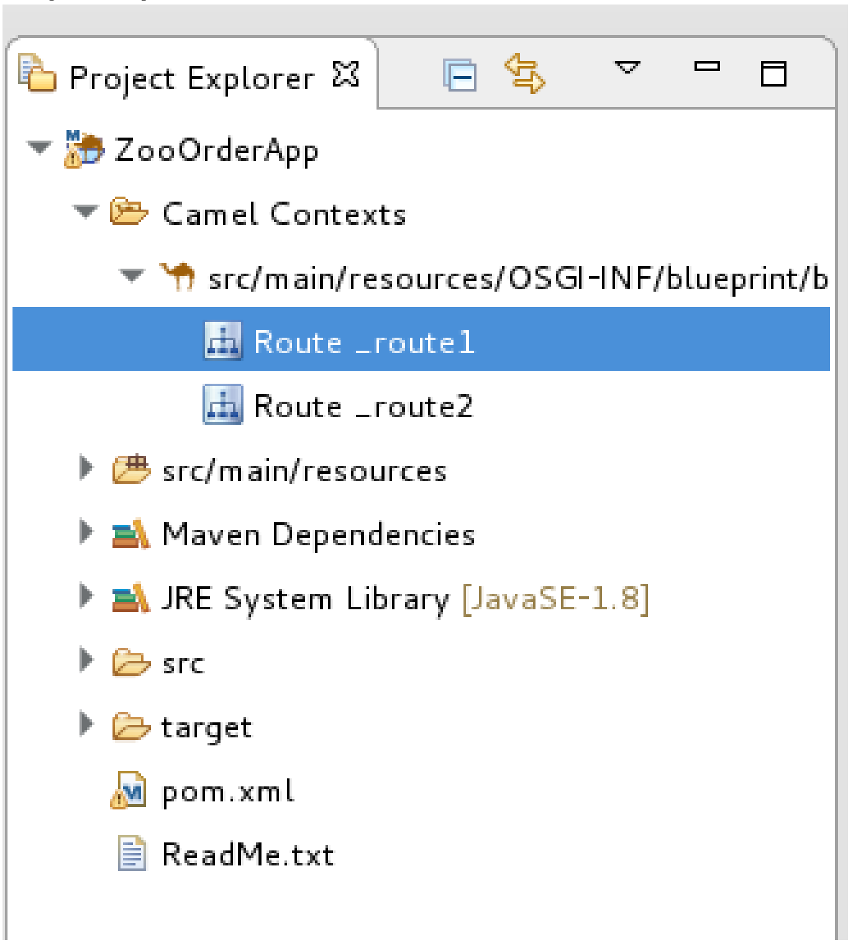
IMPORTANT

You cannot undo this operation.

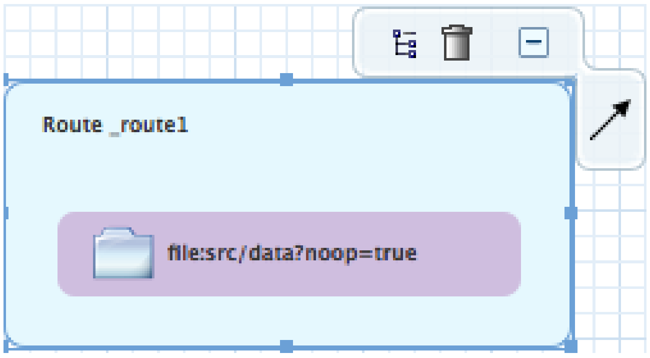
Procedure

To delete a route:

1. If the routing context contains more than one route, first select the route you want to delete in the **Project Explorer** view.



2. On the canvas, click the Route container's 🗑 icon.



3. Click **Yes** when asked if you are sure you want to delete this element.

The route is removed from the canvas, from the context file, and from the **Project Explorer** view.

2.6. ADDING GLOBAL ENDPOINTS, DATA FORMATS, OR BEANS

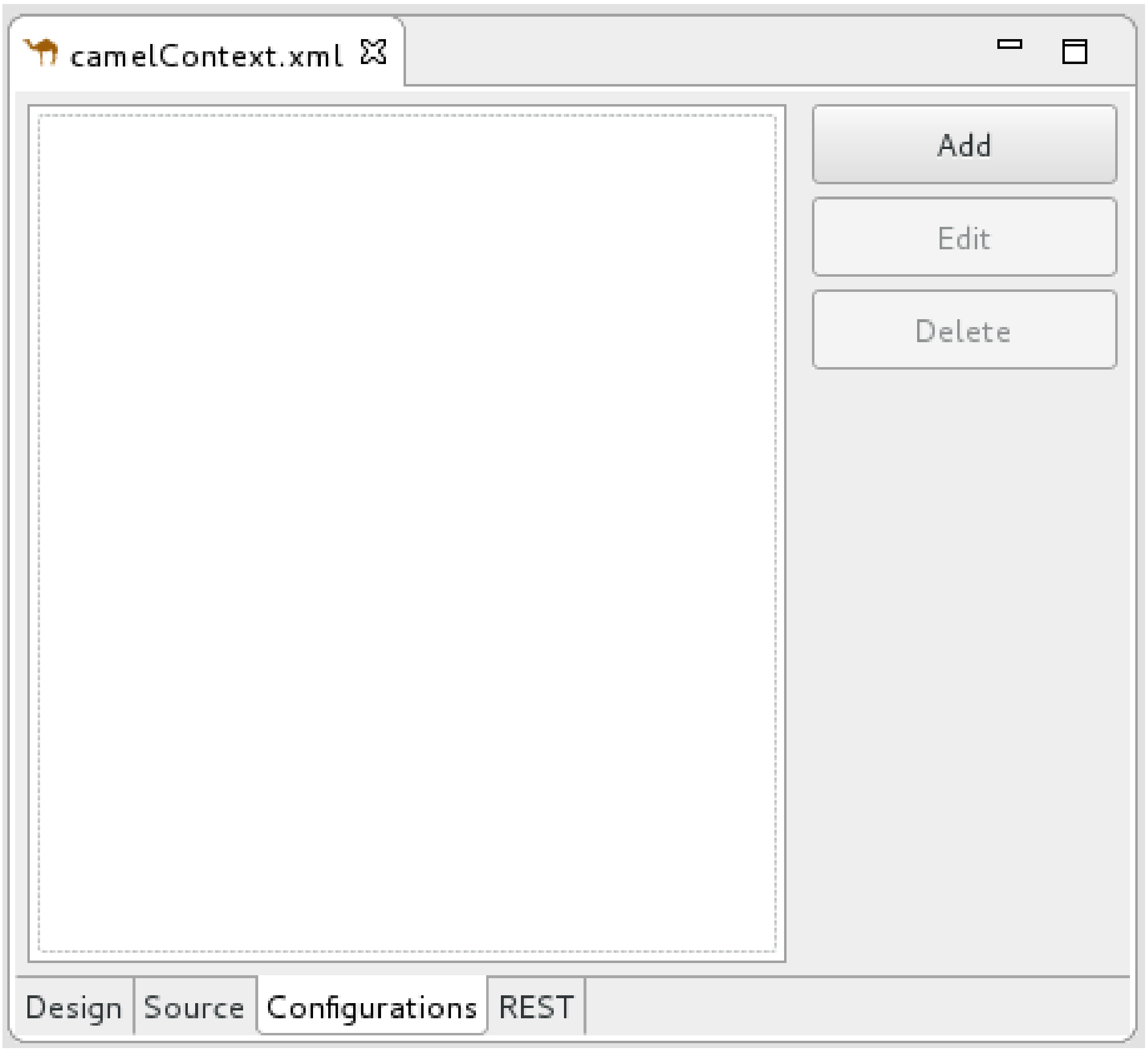
Overview

Some routes rely on shared configuration provided by global endpoints, global data formats, or global beans. You can add global elements to the project's routing context file by using the route editor's **Configurations** tab.

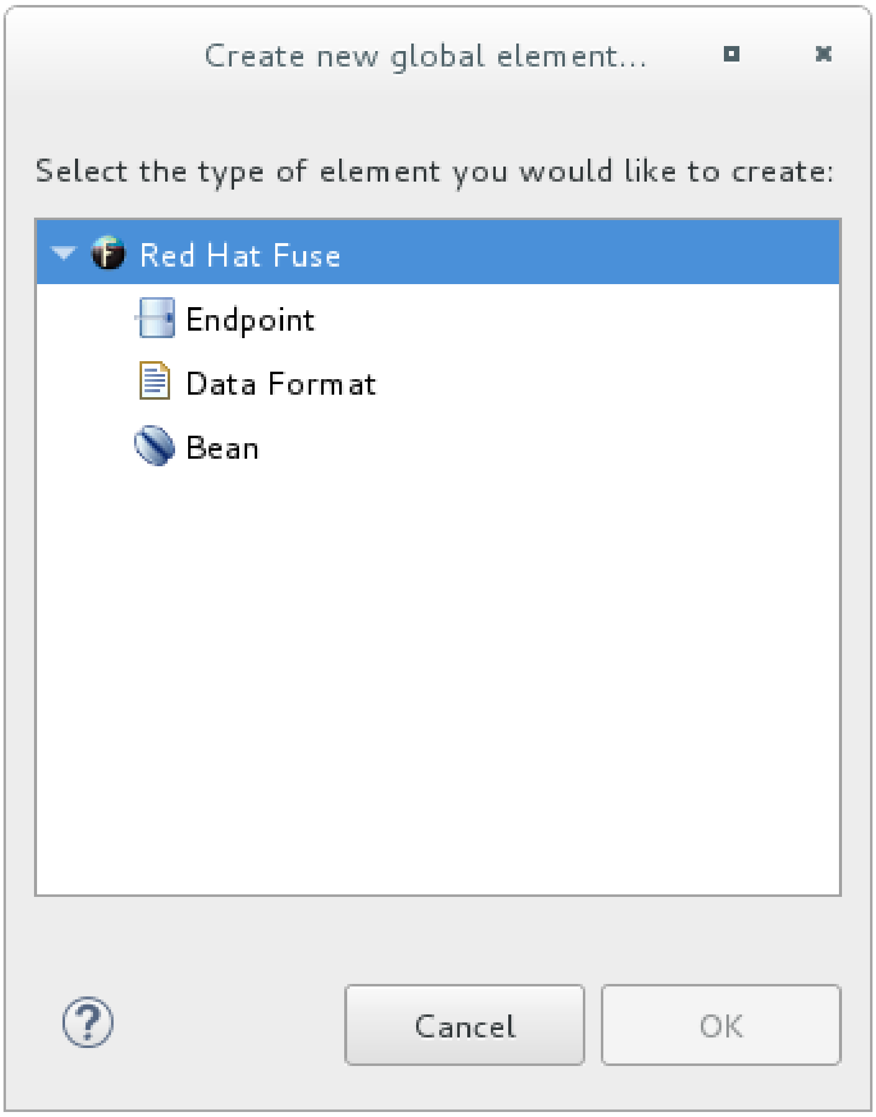
To add global elements to your routing context file:

1. Open your routing context file in the route editor.

2. At the bottom of the route editor, click the **Configurations** tab to display global configurations, if there are any.



3. Click **Add** to open the **Create a new global element** dialog.

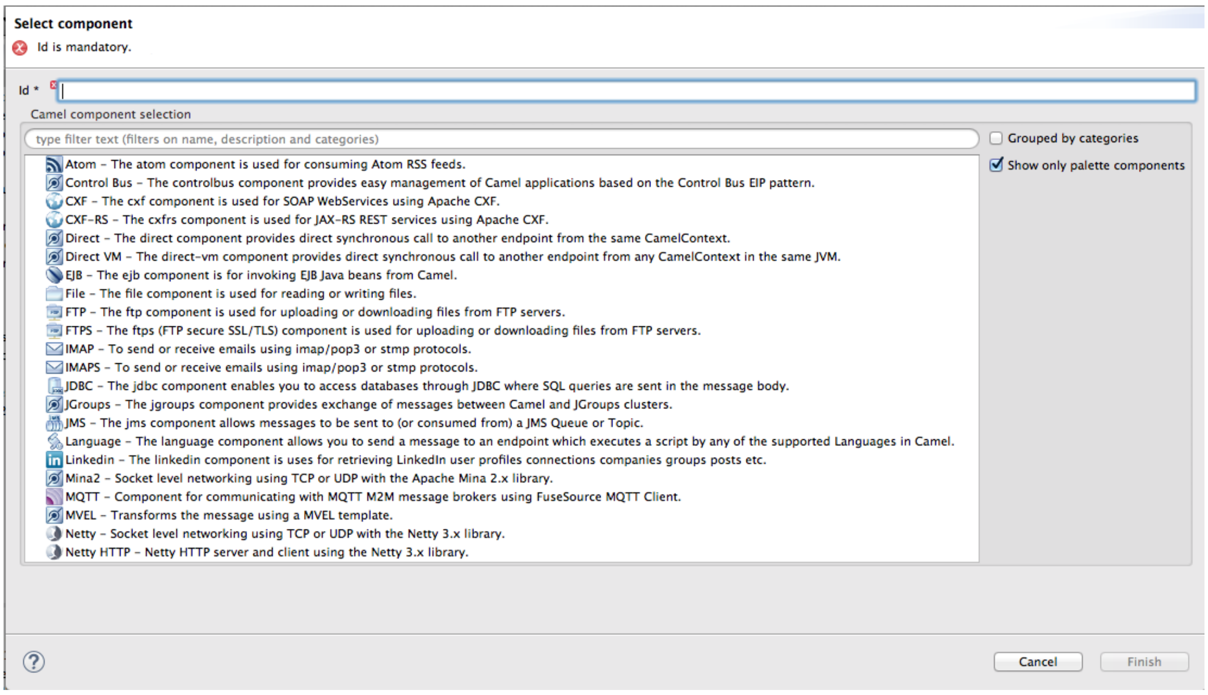

Create new global element...
Select the type of element you would like to create:
Red Hat Fuse
Endpoint
Data Format
Bean
Cancel
OK

The options are:

- Endpoint — see the section called “Adding a global endpoint”.
- Data Format — see the section called “Adding a global data format”.
- Bean — see the section called “Adding a global bean”.

Adding a global endpoint

1. In the **Create a new global element** dialog, select **Endpoint** and click **OK** to open the **Select component** dialog.


Select component

Id is mandatory.

Id *

Camel component selection

type filter text (filters on name, description and categories)

☐ Grouped by categories

☑ Show only palette components

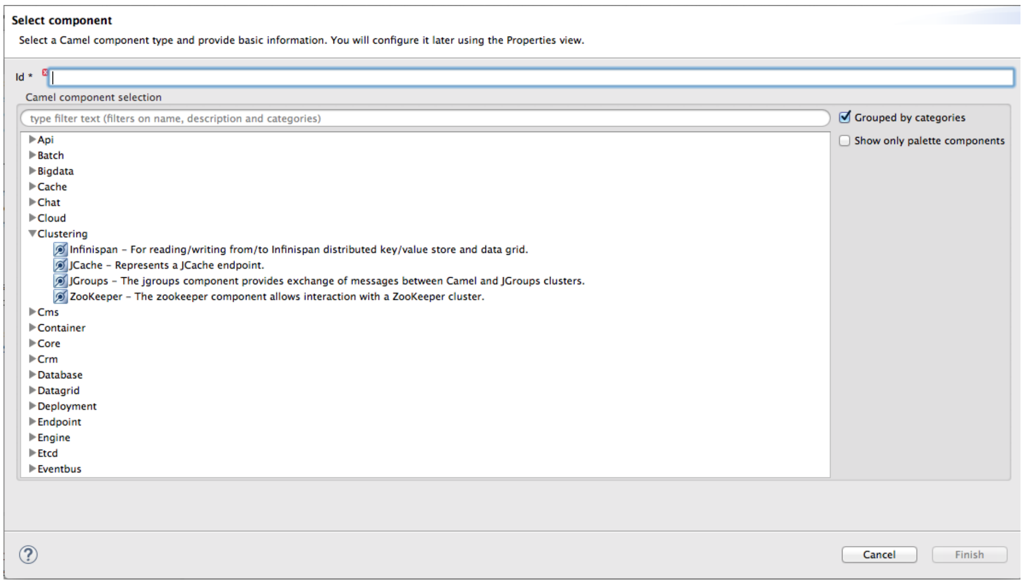
Atom - The atom component is used for consuming Atom RSS feeds.
Control Bus - The controlbus component provides easy management of Camel applications based on the Control Bus EIP pattern.
CXF - The cxf component is used for SOAP WebServices using Apache CXF.
CXF-RS - The cxfrs component is used for JAX-RS REST services using Apache CXF.
Direct - The direct component provides direct synchronous call to another endpoint from the same CamelContext.
Direct VM - The direct-vm component provides direct synchronous call to another endpoint from any CamelContext in the same JVM.
EJB - The ejb component is for invoking EJB Java beans from Camel.
File - The file component is used for reading or writing files.
FTP - The ftp component is used for uploading or downloading files from FTP servers.
FTPS - The ftps (FTP secure SSL/TLS) component is used for uploading or downloading files from FTP servers.
IMAP - To send or receive emails using imap/pop3 or stmp protocols.
IMAPS - To send or receive emails using imap/pop3 or stmp protocols.
JDBC - The jdbc component enables you to access databases through JDBC where SQL queries are sent in the message body.
JGroups - The jgroups component provides exchange of messages between Camel and JGroups clusters.
JMS - The jms component allows messages to be sent to (or consumed from) a JMS Queue or Topic.
Language - The language component allows you to send a message to an endpoint which executes a script by any of the supported Languages in Camel.
Linkedin - The linkedin component is uses for retrieving LinkedIn user profiles connections companies groups posts etc.
Mina2 - Socket level networking using TCP or UDP with the Apache Mina 2.x library.
MQTT - Component for communicating with MQTT M2M message brokers using FuseSource MQTT Client.
MVEL - Transforms the message using a MVEL template.
Netty - Socket level networking using TCP or UDP with the Netty 3.x library.
Netty HTTP - Netty HTTP server and client using the Netty 3.x library.

NOTE

By default, the **Select component** dialog opens with the **Show only palette components** option enabled. To see all available components, uncheck this option.

NOTE

The **Grouped by categories** option groups components by type.


Select component

Select a Camel component type and provide basic information. You will configure it later using the Properties view.

Id *

Camel component selection

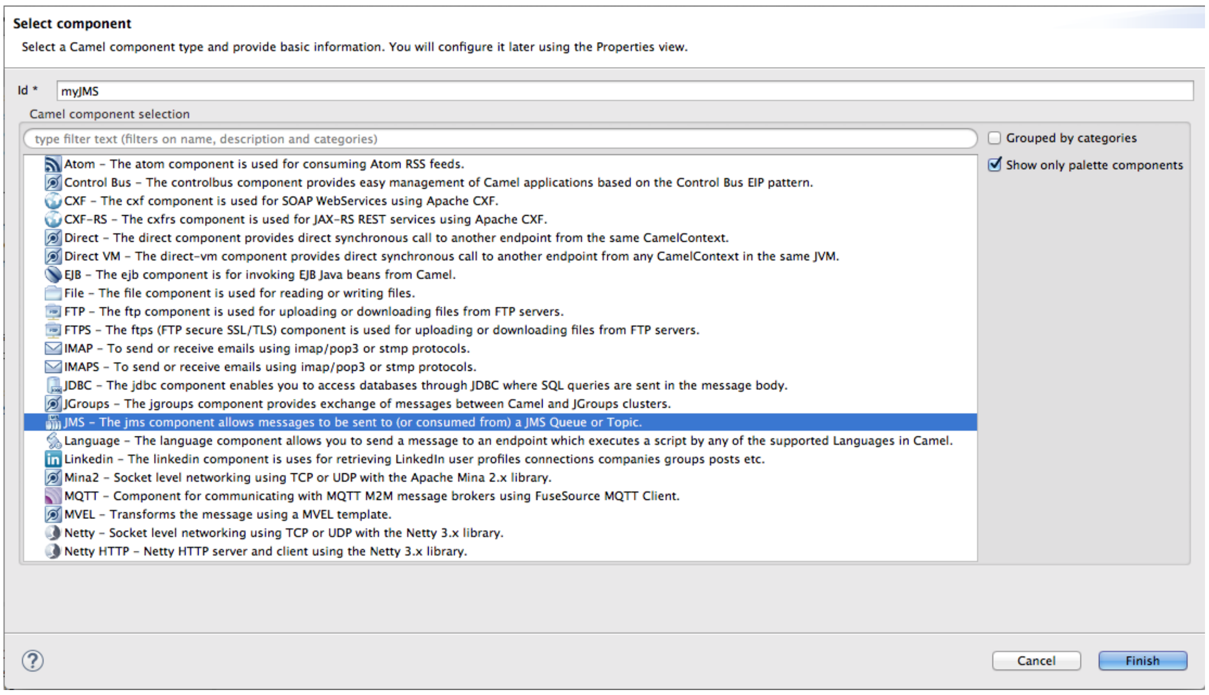
type filter text (filters on name, description and categories)

☑ Grouped by categories

☐ Show only palette components

▶Api
▶Batch
▶Bigdata
▶Cache
▶Chat
▶Cloud
▼Clustering
    Infinispan - For reading/writing from/to Infinispan distributed key/value store and data grid.
    JCache - Represents a JCache endpoint.
    JGroups - The jgroups component provides exchange of messages between Camel and JGroups clusters.
    ZooKeeper - The zookeeper component allows interaction with a ZooKeeper cluster.
▶Cms
▶Container
▶Core
▶Crm
▶Database
▶Datagrid
▶Deployment
▶Endpoint
▶Engine
▶Etcd
▶Eventbus

2. In the **Select component** dialog, scroll through the list of Camel components to find and select the component you want to add to the context file, and then enter an ID for it in the **Id** field.



Select component

Select a Camel component type and provide basic information. You will configure it later using the Properties view.

Id * myJMS

Camel component selection

type filter text (filters on name, description and categories)

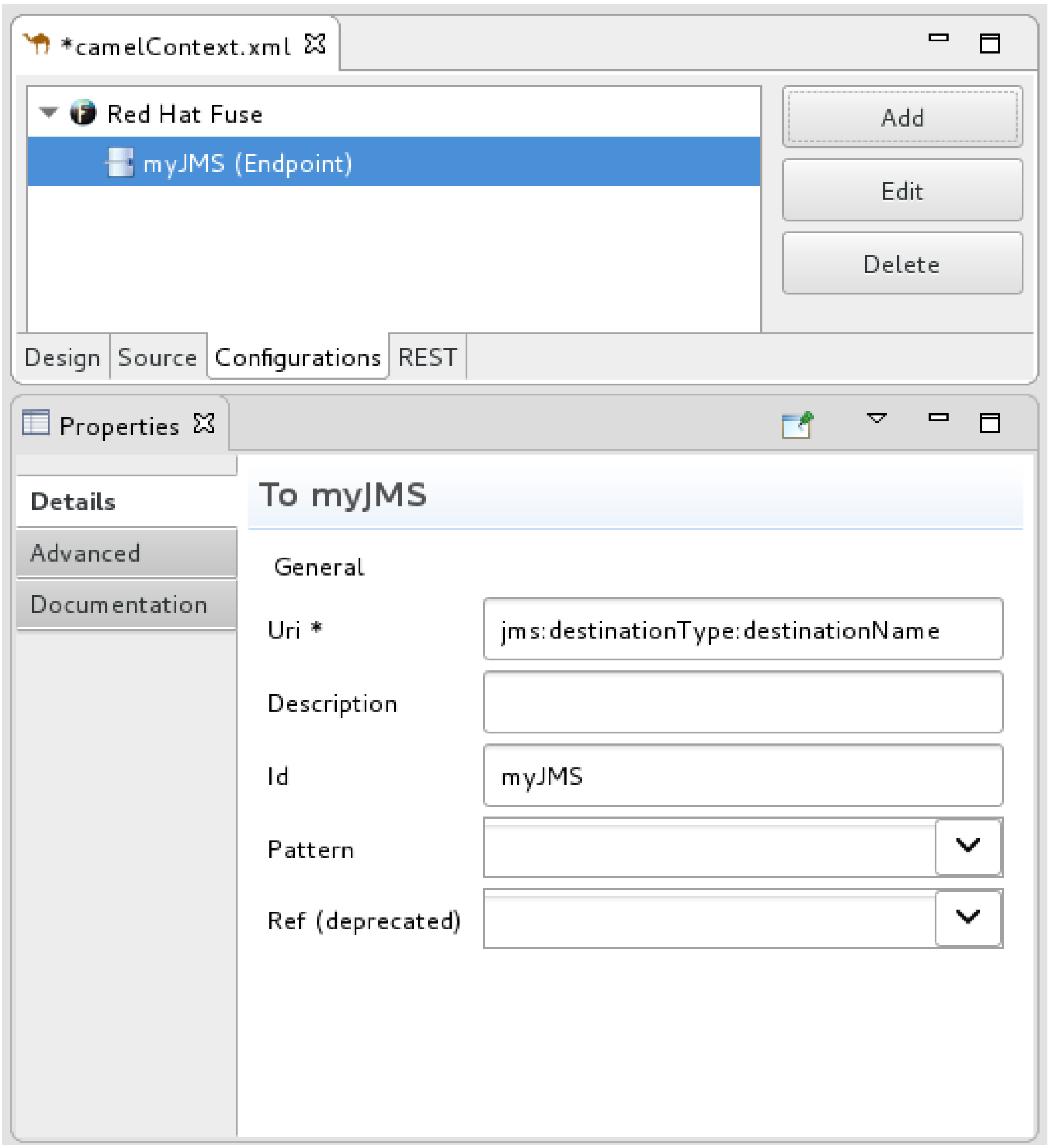
- Atom - The atom component is used for consuming Atom RSS feeds.
- Control Bus - The controlbus component provides easy management of Camel applications based on the Control Bus EIP pattern.
- CXF - The cxf component is used for SOAP WebServices using Apache CXF.
- CXF-RS - The cxfrs component is used for JAX-RS REST services using Apache CXF.
- Direct - The direct component provides direct synchronous call to another endpoint from the same CamelContext.
- Direct VM - The direct-vm component provides direct synchronous call to another endpoint from any CamelContext in the same JVM.
- EJB - The ejb component is for invoking EJB Java beans from Camel.
- File - The file component is used for reading or writing files.
- FTP - The ftp component is used for uploading or downloading files from FTP servers.
- FTPS - The ftps (FTP secure SSL/TLS) component is used for uploading or downloading files from FTP servers.
- IMAP - To send or receive emails using imap/pop3 or stmp protocols.
- IMAPS - To send or receive emails using imap/pop3 or stmp protocols.
- JDBC - The jdbc component enables you to access databases through JDBC where SQL queries are sent in the message body.
- JGroups - The jgroups component provides exchange of messages between Camel and JGroups clusters.
- JMS - The jms component allows messages to be sent to (or consumed from) a JMS Queue or Topic.
- Language - The language component allows you to send a message to an endpoint which executes a script by any of the supported Languages in Camel.
- Linkedin - The linkedin component is uses for retrieving LinkedIn user profiles connections companies groups posts etc.
- Mina2 - Socket level networking using TCP or UDP with the Apache Mina 2.x library.
- MQTT - Component for communicating with MQTT M2M message brokers using FuseSource MQTT Client.
- MVEL - Transforms the message using a MVEL template.
- Netty - Socket level networking using TCP or UDP with the Netty 3.x library.
- Netty HTTP - Netty HTTP server and client using the Netty 3.x library.

☐ Grouped by categories

☑ Show only palette components

In this example, the JMS component is selected and **`myJMS`** is the **Id** value.

3. Click **Finish**.



To myJMS

General

Uri * jms:destinationType:destinationName

Description

Id myJMS

Pattern

Ref (deprecated)

You can now set properties in the **Properties** view as needed.

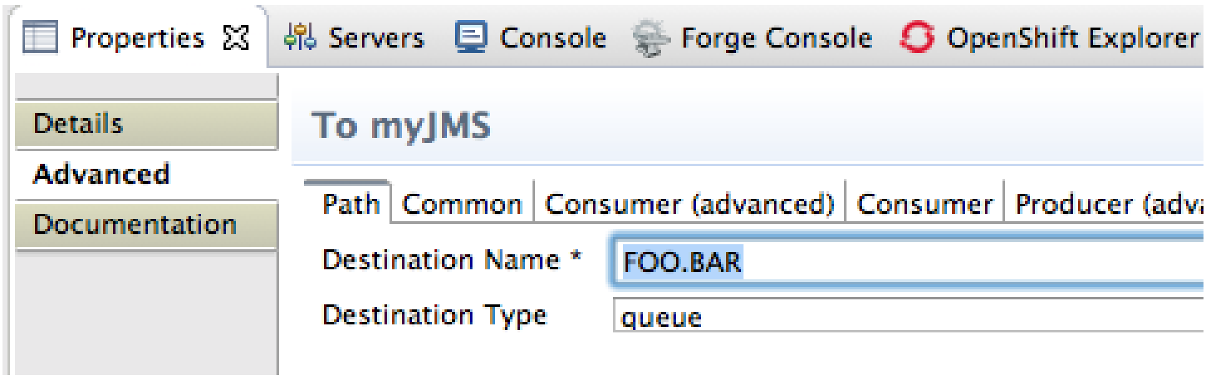
The tooling autofills **Id** with the value you entered in the component's **`Id`** field in [globalEndptSelect]. In this example, Camel builds the **`uri`** (required field) starting with the component's schema (in this case, **`jms:`**), but you must specify the **destinationName** and the **destinationType** to complete the component's **uri**.

NOTE

For the JMS component, the destination type defaults to **`queue`**. This default value does not appear in the **`uri`** field on the **Details** page until you have entered a value in **Destination Name** (required field).

4. To complete the component's uri, click **Advanced**.

5. In the **Destination Name** field, enter the name of the destination endpoint (for example, **`FOO.BAR`**). In the **Destination Type** field, enter the endpoint destination's type (for example, **`queue`**, **`topic`**, **`temp:queue`**, or **`temp:topic`**).

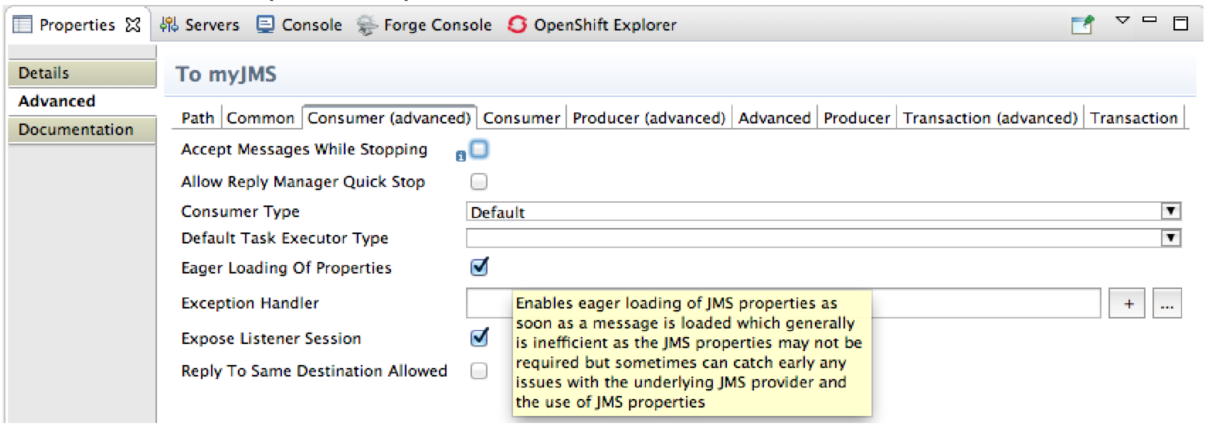


Destination Name * FOO.BAR

Destination Type queue

The **Properties** view's **Details** and **Advanced** tabs provide access to all properties available for configuring a particular component.

6. Click the **Consumer (advanced)** tab.



Accept Messages While Stopping ☐

Allow Reply Manager Quick Stop ☐

Consumer Type Default

Default Task Executor Type

Eager Loading Of Properties ✓

Exception Handler

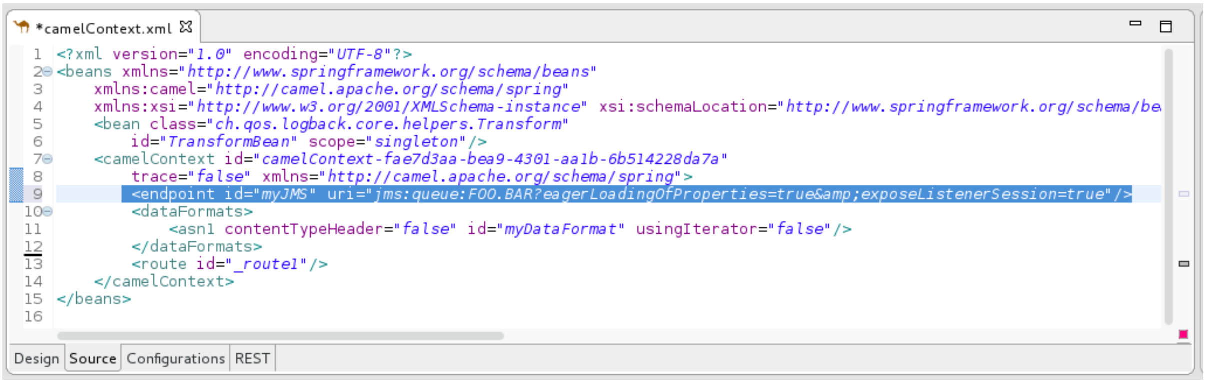
Expose Listener Session ✓

Reply To Same Destination Allowed ☐

Enables eager loading of JMS properties as soon as a message is loaded which generally is inefficient as the JMS properties may not be required but sometimes can catch early any issues with the underlying JMS provider and the use of JMS properties

Enable the properties **Eager Loading Of Properties** and **Expose Listener Session**.

7. In the route editor, switch to the **Source** tab to see the code that the tooling added to the context file (in this example, a configured JMS endpoint), before the first route element.

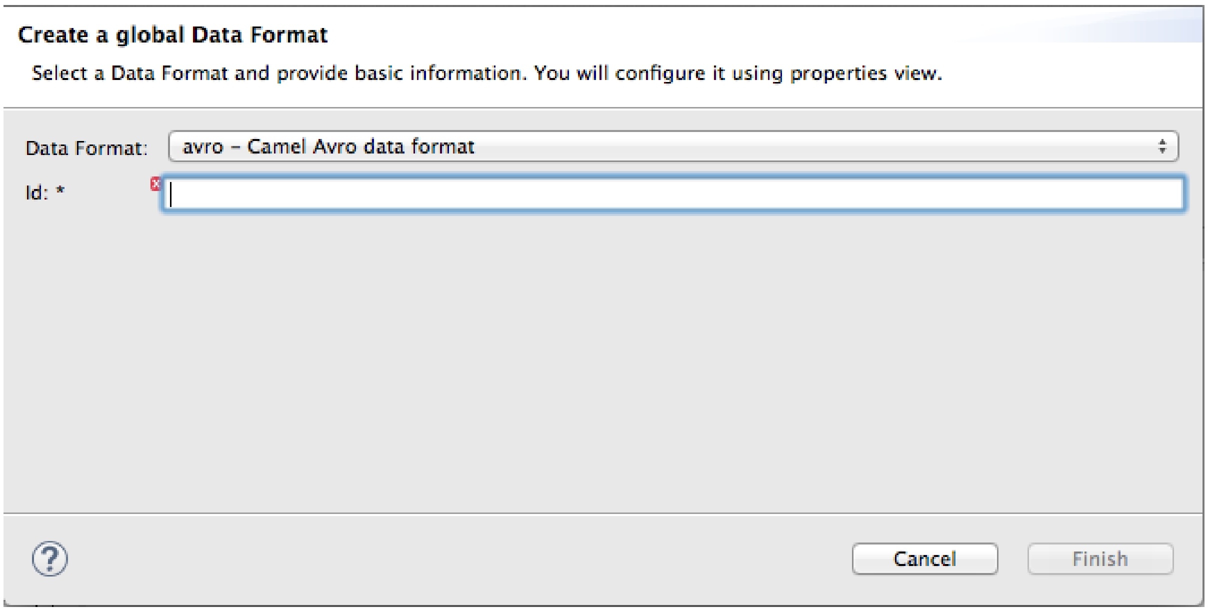


```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:camel="http://camel.apache.org/schema/spring"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.springframework.org/schema/be
    <bean class="ch.qos.logback.core.helpers.Transform"
        id="TransformBean" scope="singleton"/>
    <camelContext id="camelContext-fae7d3aa-bea9-4301-aa1b-6b514228da7a"
        trace="false" xmlns="http://camel.apache.org/schema/spring">
        <endpoint id="myJMS" uri="jms:queue:FOO.BAR?eagerLoadingOfProperties=true&amp;exposeListenerSession=true"/>
        <dataFormats>
            <asn1 contentTypeHeader="false" id="myDataFormat" usingIterator="false"/>
        </dataFormats>
        <route id="_route1"/>
    </camelContext>
</beans>
```

8. When done, save your changes by selecting **File** → **Save** on the menu bar.

Adding a global data format

1. In the **Create a new global element** dialog, select **Data Format** and click **OK** to open the **Create a global Data Format** dialog.


Create a global Data Format

Select a Data Format and provide basic information. You will configure it using properties view.

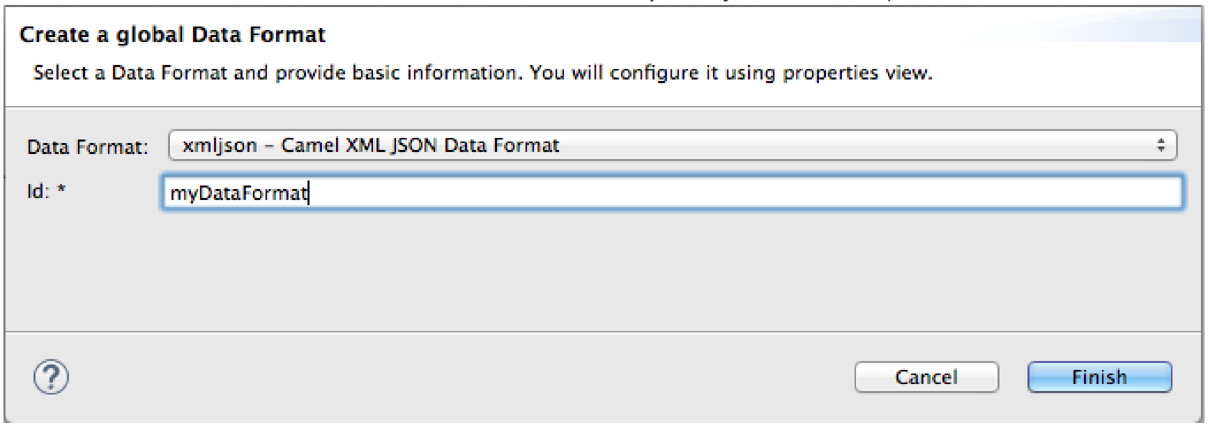
Data Format: avro - Camel Avro data format

Id: *

The data format defaults to **`avro`**, the format at the top of the list of those available.

2. Open the **Data Format** drop-down menu, and select the format you want, for example, **`xmljson`**.

3. In the **Id** field, enter a name for the format, for example, *myDataFormat*).

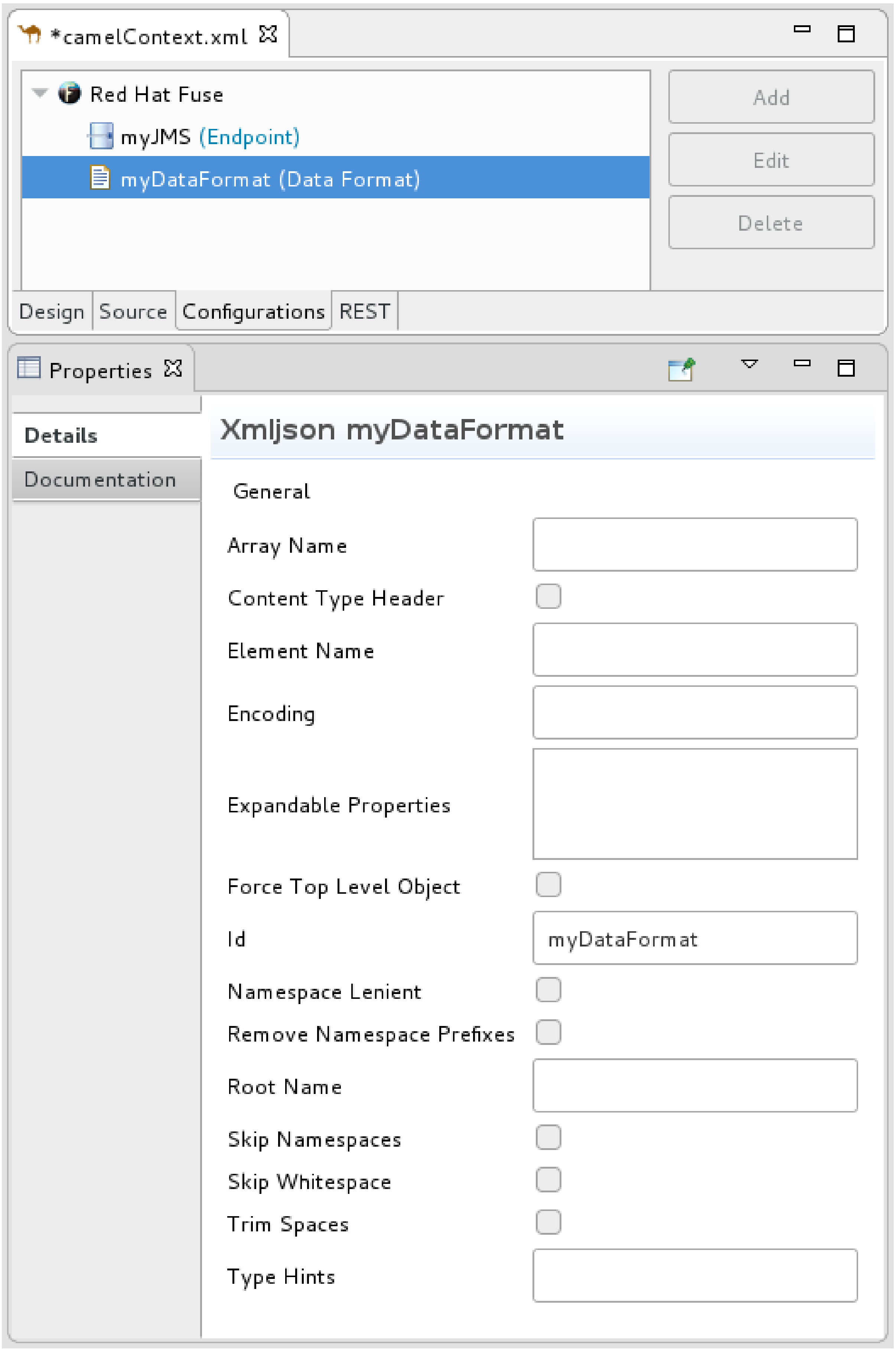

Create a global Data Format

Select a Data Format and provide basic information. You will configure it using properties view.

Data Format: xmljson - Camel XML JSON Data Format

Id: * myDataFormat

4. Click **Finish**.



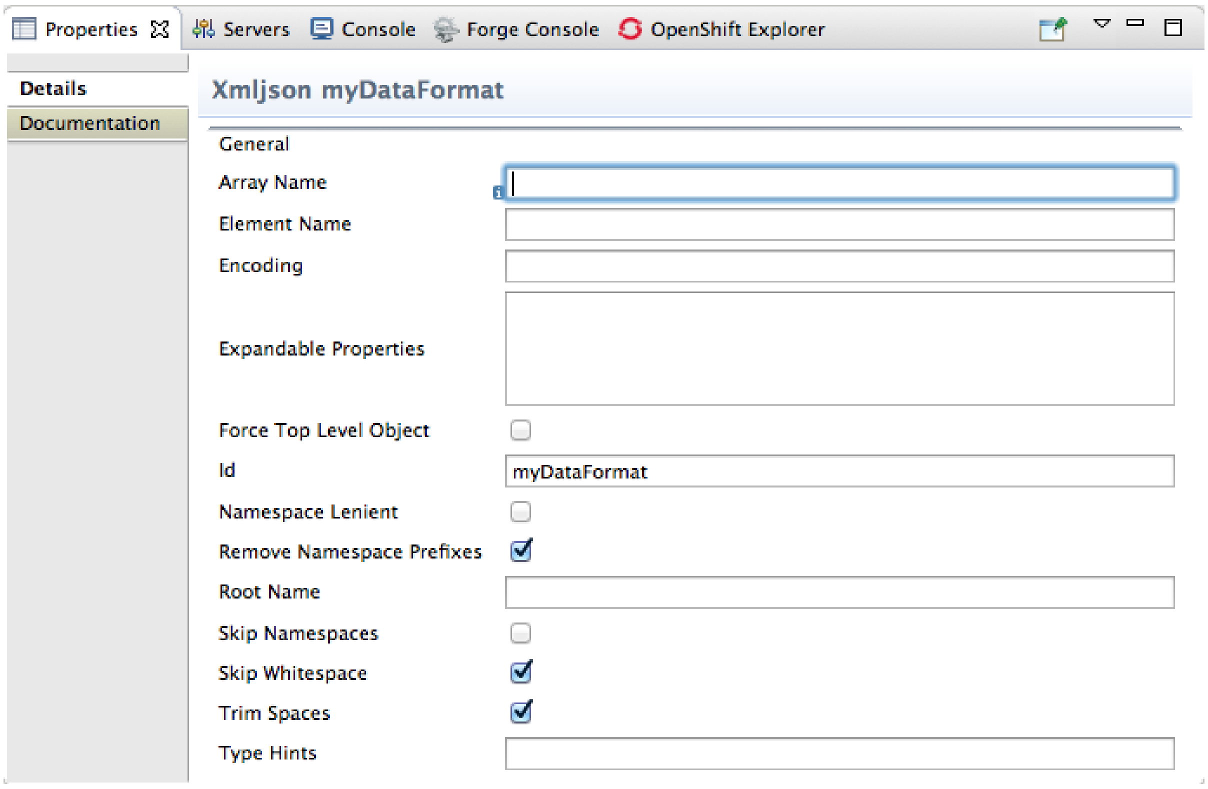
Red Hat Fuse
myJMS (Endpoint)
myDataFormat (Data Format)

Xmljson myDataFormat

General

Array Name	
Content Type Header	☐
Element Name	
Encoding	
Expandable Properties	
Force Top Level Object	☐
Id	myDataFormat
Namespace Lenient	☐
Remove Namespace Prefixes	☐
Root Name	
Skip Namespaces	☐
Skip Whitespace	☐
Trim Spaces	☐
Type Hints	

5. In the **Properties** view, set property values as appropriate for your project, for example:

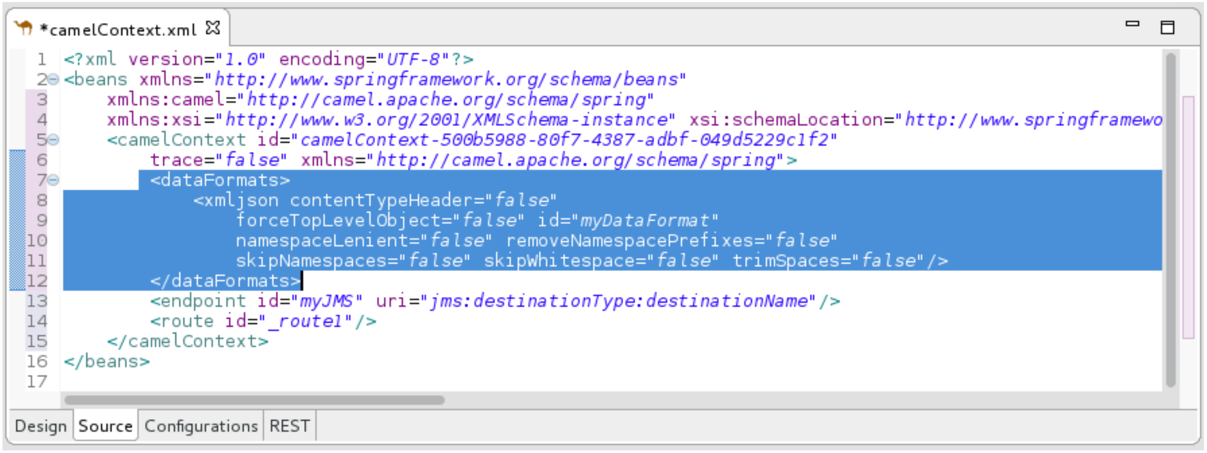


Xmljson myDataFormat

General

Array Name	
Element Name	
Encoding	
Expandable Properties	
Force Top Level Object	☐
Id	myDataFormat
Namespace Lenient	☐
Remove Namespace Prefixes	☑
Root Name	
Skip Namespaces	☐
Skip Whitespace	☑
Trim Spaces	☑
Type Hints	

6. In the route editor, click the **Source** tab to see the code that the tooling added to the context file. In this example, a configured xmljson data format is before the first route element.



```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:camel="http://camel.apache.org/schema/spring"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.springframewo
    <camelContext id="camelContext-500b5988-80f7-4387-adbf-049d5229c1f2"
        trace="false" xmlns="http://camel.apache.org/schema/spring">
        <dataFormats>
            <xmljson contentTypeHeader="false"
                forceTopLevelObject="false" id="myDataFormat"
                namespaceLenient="false" removeNamespacePrefixes="false"
                skipNamespaces="false" skipWhitespace="false" trimSpaces="false"/>
        </dataFormats>
        <endpoint id="myJMS" uri="jms:destinationType:destinationName"/>
        <route id="_route1"/>
    </camelContext>
</beans>
```

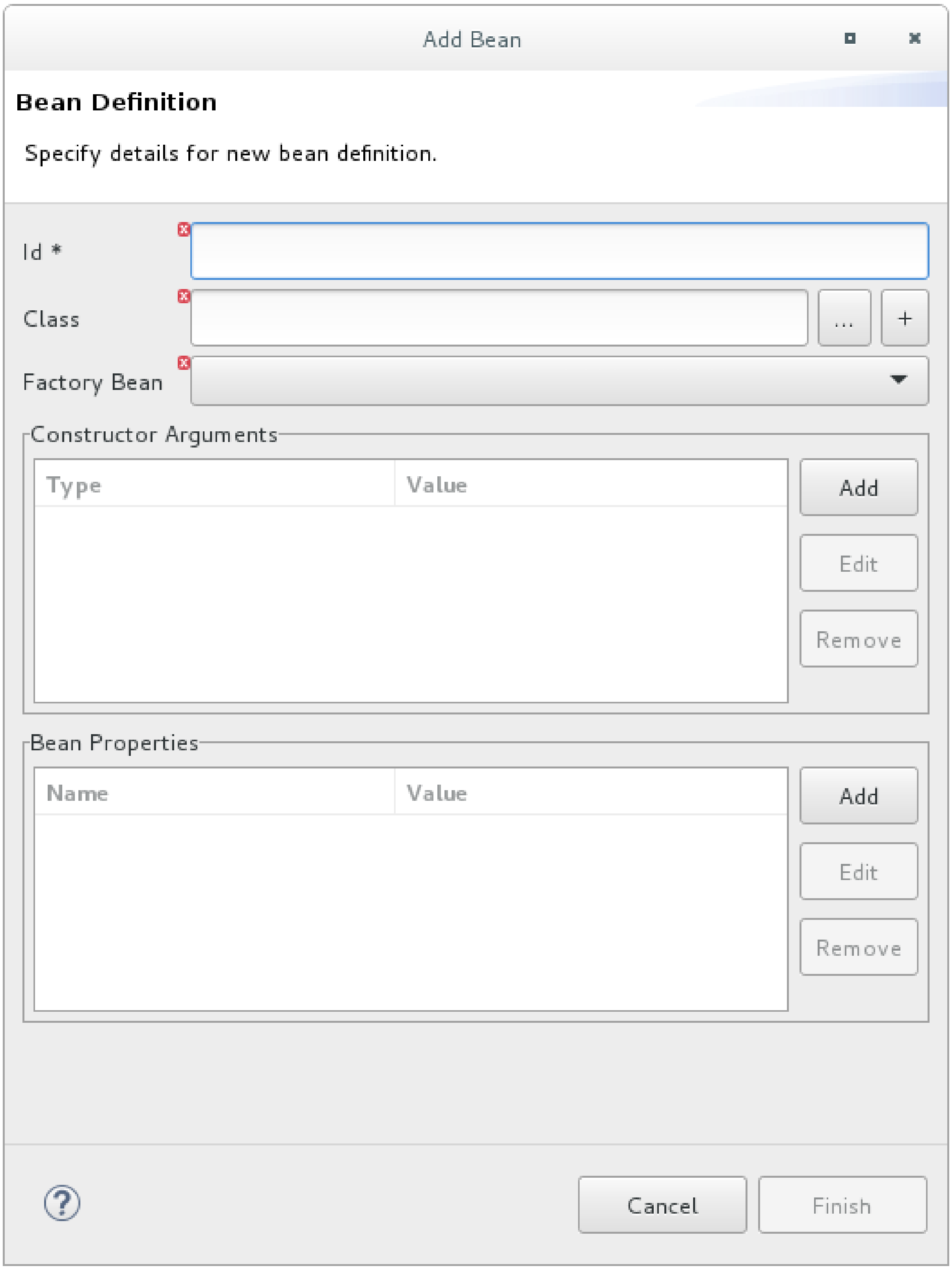
7. When done, save your changes by selecting **File** → **Save** on the menu bar.

Adding a global bean

A global bean enables out-of-route bean definitions that can be referenced from anywhere in the route. When you copy a **Bean** component from the palette to the route, you can find defined global beans in the **Properties** view's **Ref** dropdown. Select the global bean that you want the **Bean** component to reference.

To add a global bean element:

1. In the **Create a new global element** window, select **Bean** and click **OK** to open the **Bean Definition** dialog.



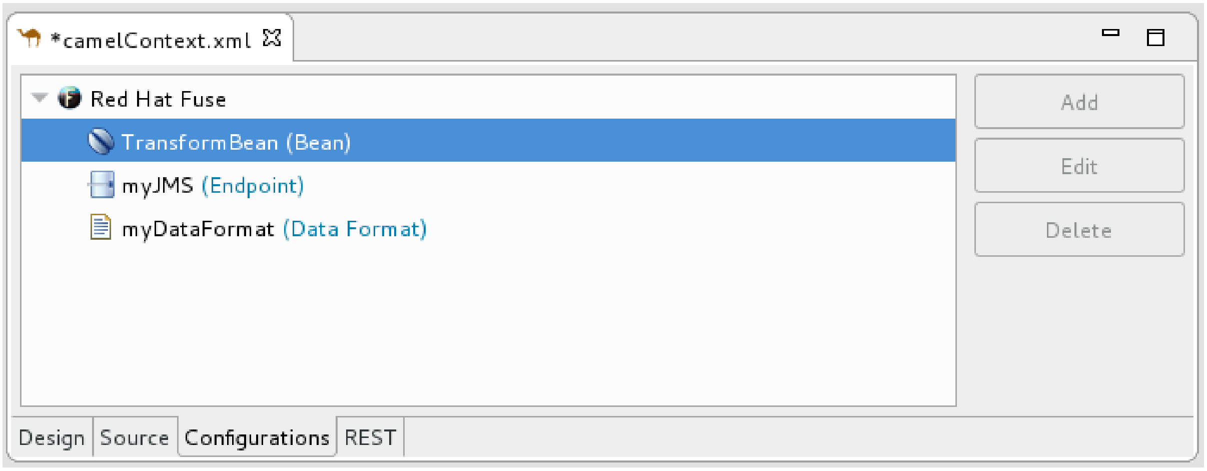
2. In the **Id** field, enter an ID for the global bean, for example, **TransformBean**. The ID must be unique in the configuration.

3. Identify a bean class or a factory bean.
   To specify a factory bean, you must have already added another global bean with a factory class specified. You can then select that global bean to declare it as a global bean factory. One instance of the bean factory class will be in the runtime. Other global beans can call factory methods on that class to create their own instances of other classes.

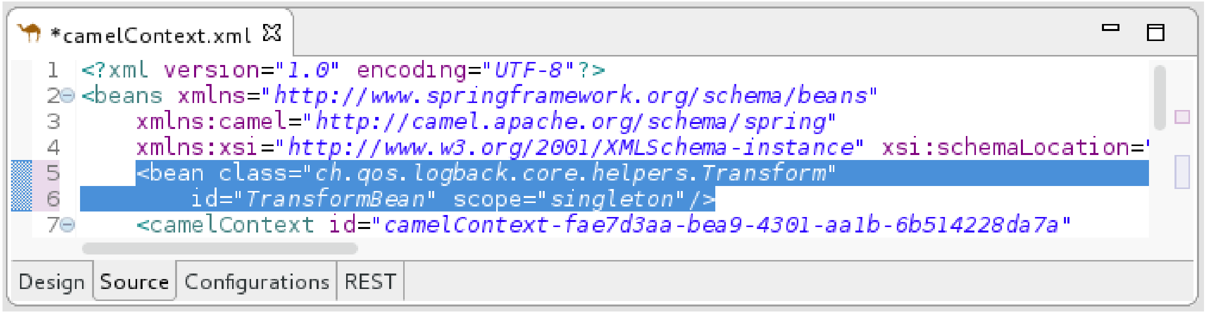
   To fill the **Class** field, do one of the following:

- Enter the name of a class that is in the project or in a referenced project.
- Click **...** to navigate to and select a class that is in the project or in a referenced project.
- Click **+** to define a new bean class and add it as a global bean.

4. If the bean you are adding requires one or more arguments, in the **Constructor Arguments** section, for each argument:
   a. Click **Add**.
   b. Optionally, in the **Type** field, enter the type of the argument. The default is `java.lang.String`.
   c. In the **Value** field, enter the value of the argument.
   d. Click **OK**.
5. Optionally specify one or more properties that are accessible to the global bean. In the **Bean Properties** section, do the following for each property:
   a. Click **Add**.
   b. In the **Name** field, enter the name of the property.
   c. In the **Value** field, enter the value of the property.
   d. Click **OK**.
6. Click **Finish** to add the global bean to the configuration. The global bean ID you specified appears in the **Configurations** tab, for example:

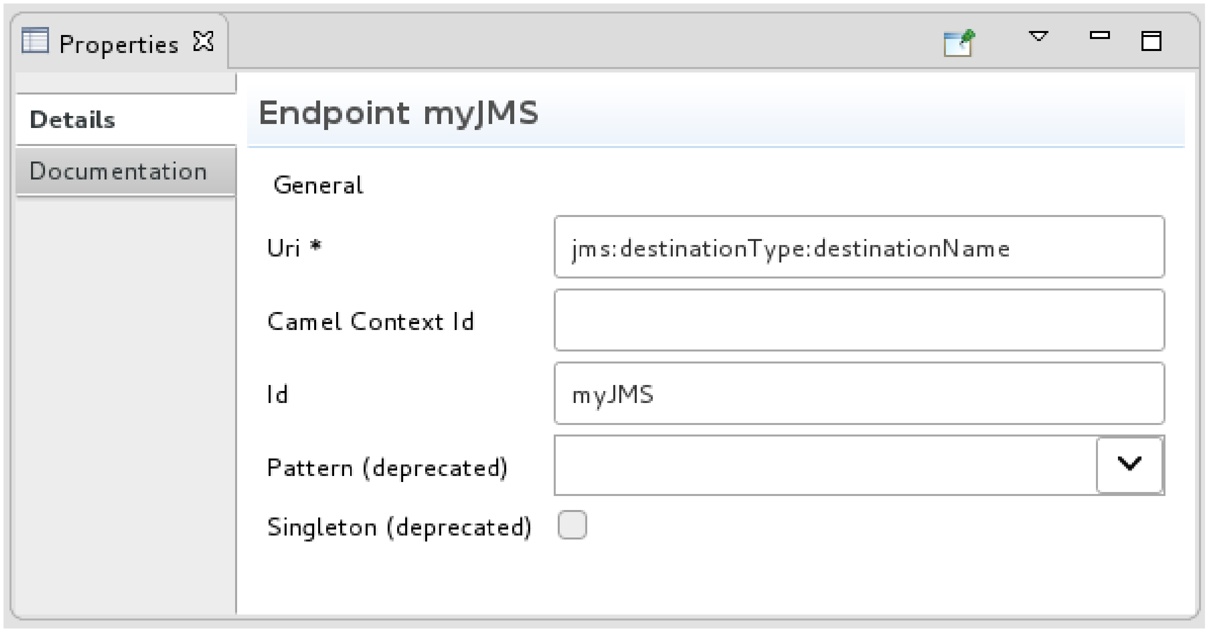


7. Switch to the **Source** tab to see the **bean** element that the tooling added to the context file. For example:



```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:camel="http://camel.apache.org/schema/spring"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation='
    <bean class="ch.qos.logback.core.helpers.Transform"
        id="TransformBean" scope="singleton"/>
    <camelContext id="camelContext-fae7d3aa-bea9-4301-aa1b-6b514228da7a"
```

8. Click the **Configurations** tab to return to the list of global elements and select a global bean to display its standard properties in the **Properties** view, for example:



Endpoint myJMS

General

Uri * jms:destinationType:destinationName

Camel Context Id

Id myJMS

Pattern (deprecated)

Singleton (deprecated)

NOTE

To view or edit a property that you specified when you added a global bean, select the bean in the **Configurations** tab and then click **Edit**.

9. Set global bean properties as needed:

   - **Depends-on** is a string that you can use to identify a bean that must be created before this global bean. Specify the ID (name) of the depended on bean. For example, if you are adding the **`TransformBean`** and you set **Depends-on** to **`ChangeCaseBean`** then **`ChangeCaseBean`** must be created and then **`TransformBean`** can be created. When the beans are being destroyed then **`TransformBean`** is destroyed first.

   - **Factory-method** is useful only when the global bean is a factory class. In this situation, specify or select a static factory method to be called when the bean is referenced.

   - **Scope** is **`singleton`** or **`prototype`**. The default, **`singleton`**, indicates that Camel uses the same instance of the bean each time the bean is called. Specify **`prototype`** when you want Camel to create a new instance of the bean each time the bean is called.

   - **Init-method** lets you specify or select which of the bean's **`init()`** methods to call when the bean is referenced.

   - **Destroy-method** lets you specify or select which of the bean's destory methods to call when the processing performed by the bean is done.

10. When done, save your changes by selecting **File** → **Save** on the menu bar.

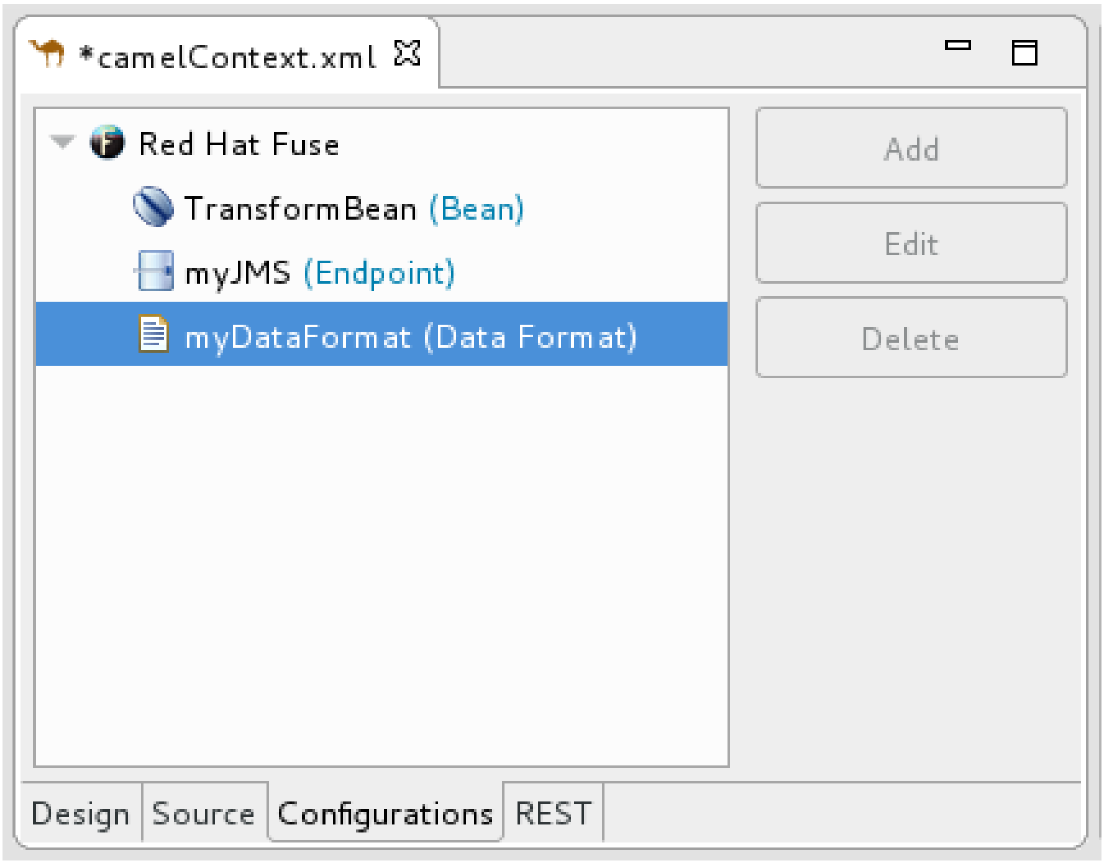
Deleting a global element

The procedure is the same whether removing an endpoint, data format or bean that was previously added to the routing context.

NOTE

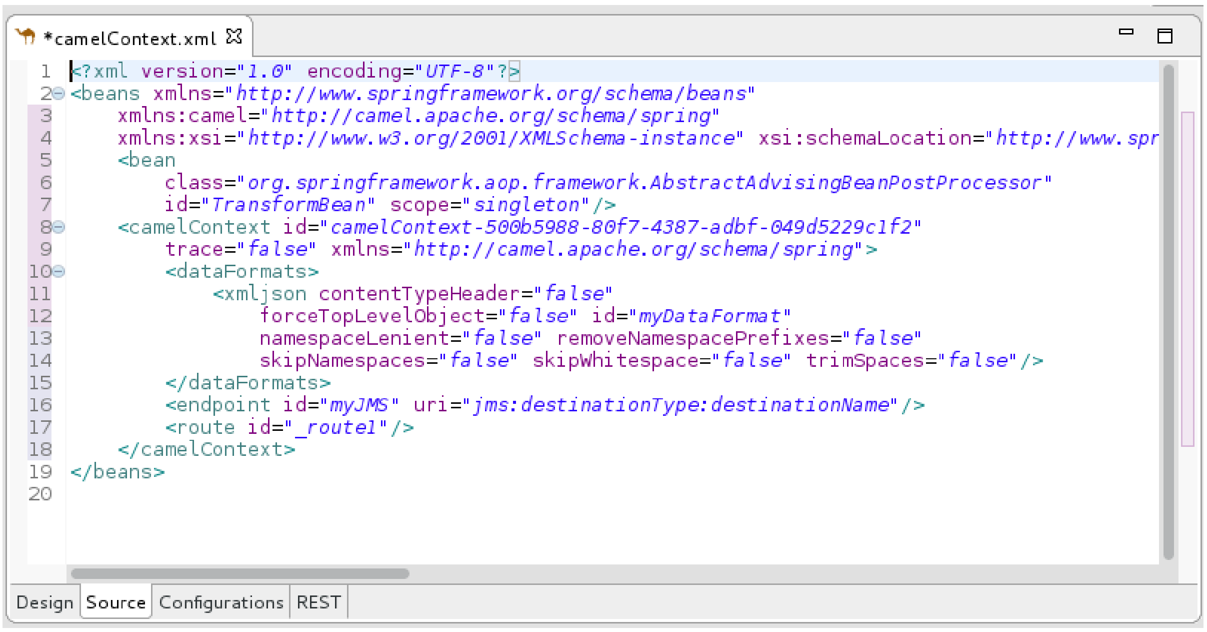
You cannot perform an undo operation for deletion of a global element. If you inadvertently delete a global element that you want to keep in the configuration you might be able to undo the deletion by closing the context file without saving it. If this is not feasible then re-add the inadvertently deleted global element.

1. In the **Configurations** tab, select the global element that you want to delete.
   For example, suppose you want to delete the data format **`myDataFormat`** that was added in the section called "Adding a global data format":



2. Click **Delete**.
   The global element **`myDataFormat`** disappears from the **Configurations** tab.

3. Switch to the **Source** tab to check that the tooling removed the XML code from the routing context.



```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:camel="http://camel.apache.org/schema/spring"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.spr
    <bean
        class="org.springframework.aop.framework.AbstractAdvisingBeanPostProcessor"
        id="TransformBean" scope="singleton"/>
    <camelContext id="camelContext-500b5988-80f7-4387-adbf-049d5229c1f2"
        trace="false" xmlns="http://camel.apache.org/schema/spring">
        <dataFormats>
            <xmljson contentTypeHeader="false"
                forceTopLevelObject="false" id="myDataFormat"
                namespaceLenient="false" removeNamespacePrefixes="false"
                skipNamespaces="false" skipWhitespace="false" trimSpaces="false"/>
        </dataFormats>
        <endpoint id="myJMS" uri="jms:destinationType:destinationName"/>
        <route id="_route1"/>
    </camelContext>
</beans>
```

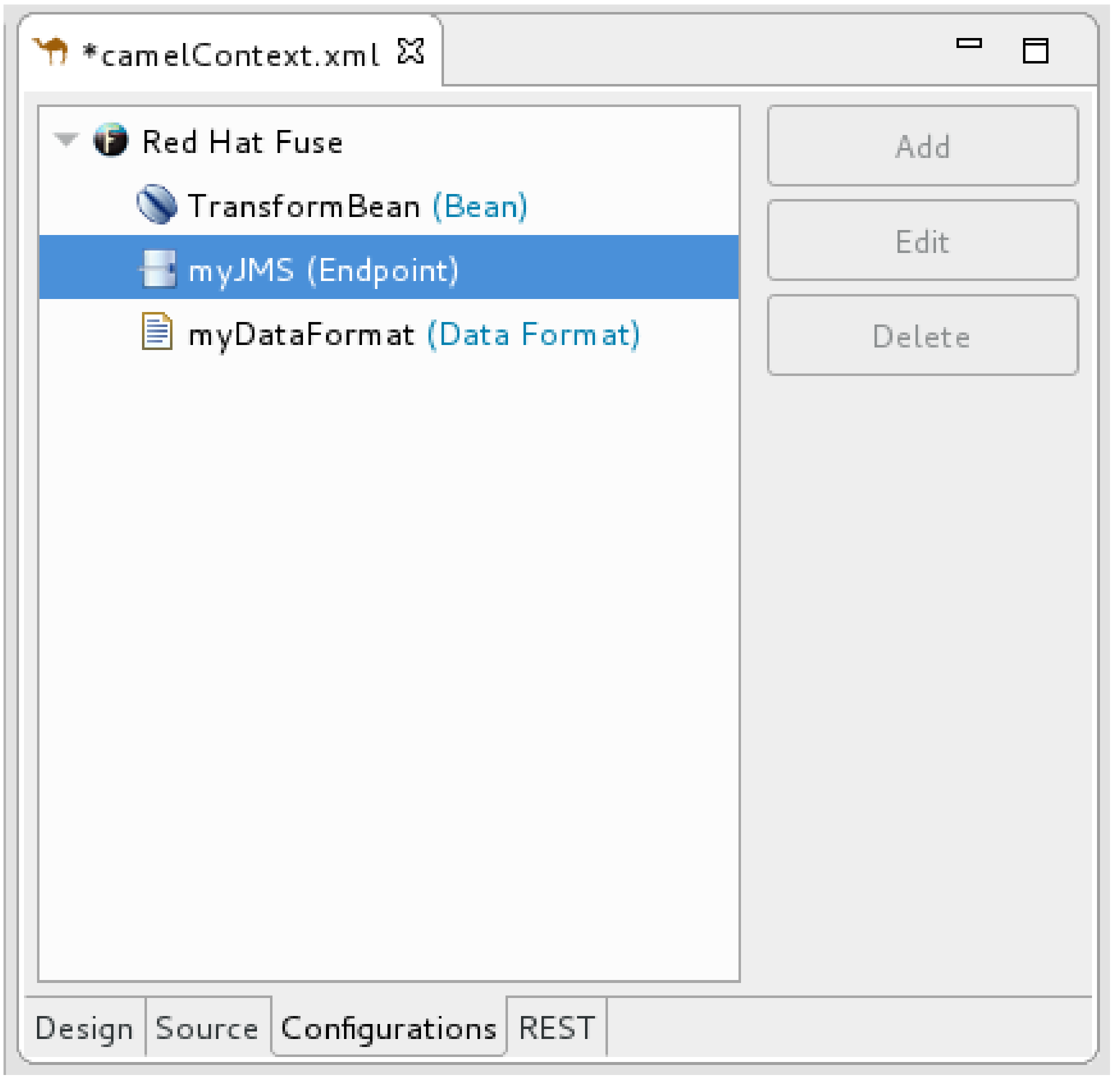
4. When done, save your changes by selecting **File** → **Save** on the menu bar.

Editing a global element

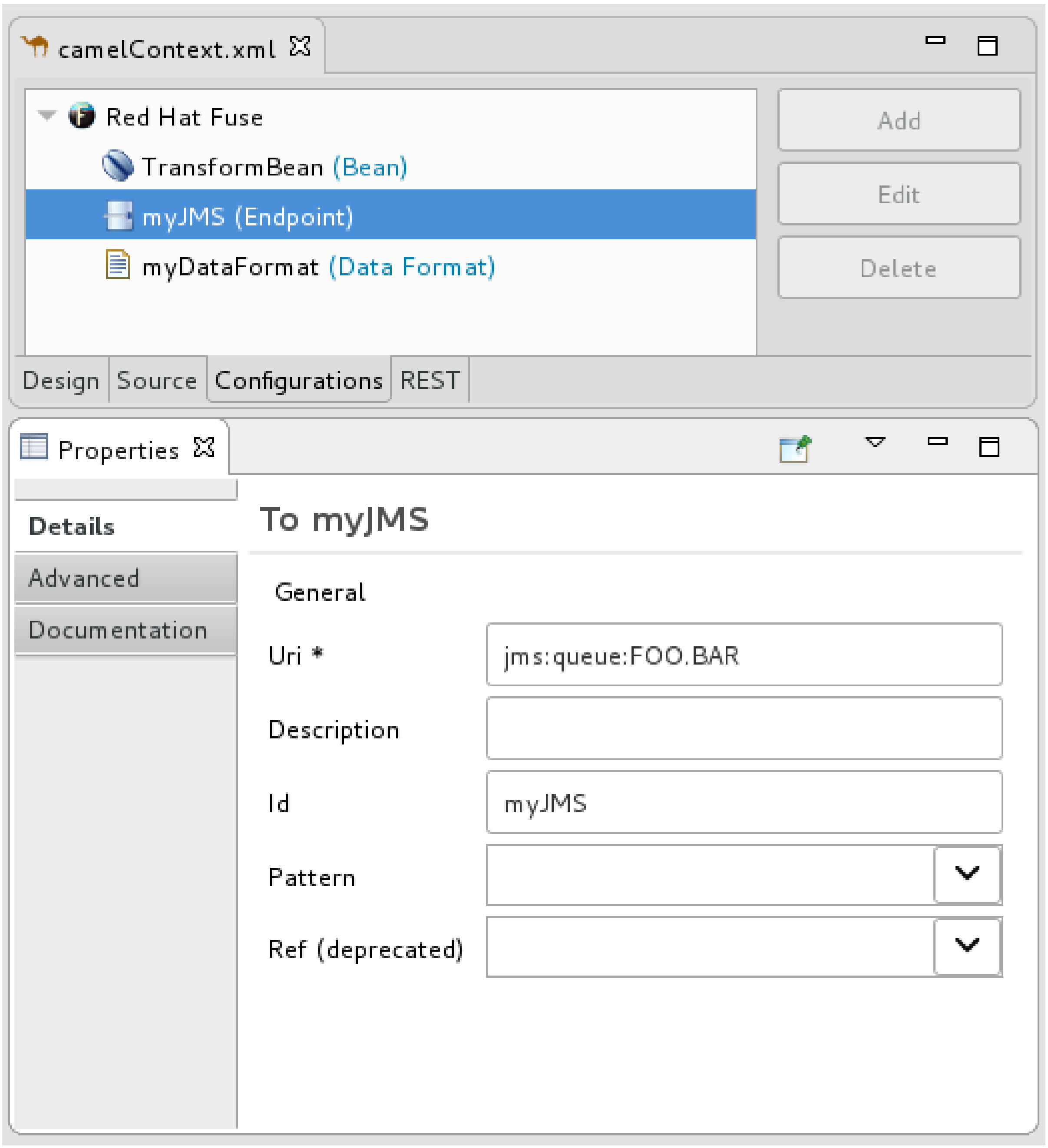
The procedure is the same whether modifying the properties of an endpoint, data format or bean that you added to the routing context.

Typically, you do not want to change the ID of a global element. If the global element is already in use in a running route, changing the ID can break references to the global element.

1. In the **Configurations** tab, select the global element that you want to edit.
   For example, to edit the endpoint **`myJMS`** that was added in the section called “Adding a global endpoint”, select it:


*camelContext.xml
Red Hat Fuse
TransformBean (Bean)
myJMS (Endpoint)
myDataFormat (Data Format)
Add
Edit
Delete
Design
Source
Configurations
REST

2. Click **Edit**.



To myJMS

General

Uri * jms:queue:FOO.BAR

Description

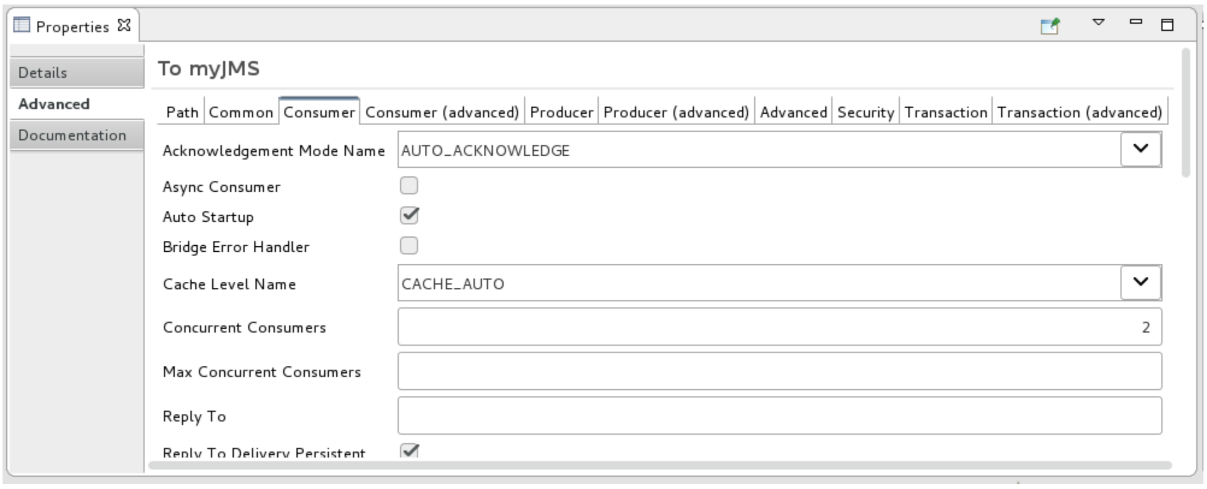
Id myJMS

Pattern

Ref (deprecated)

In the **Properties** view, modify the element’s properties as needed.

3. For example, open the **Advanced** → **Consumer** tab, and change the value of **Concurrent Consumers** to **2**:



To myJMS

Acknowledgement Mode Name AUTO_ACKNOWLEDGE

Async Consumer ☐

Auto Startup ✓

Bridge Error Handler ☐

Cache Level Name CACHE_AUTO

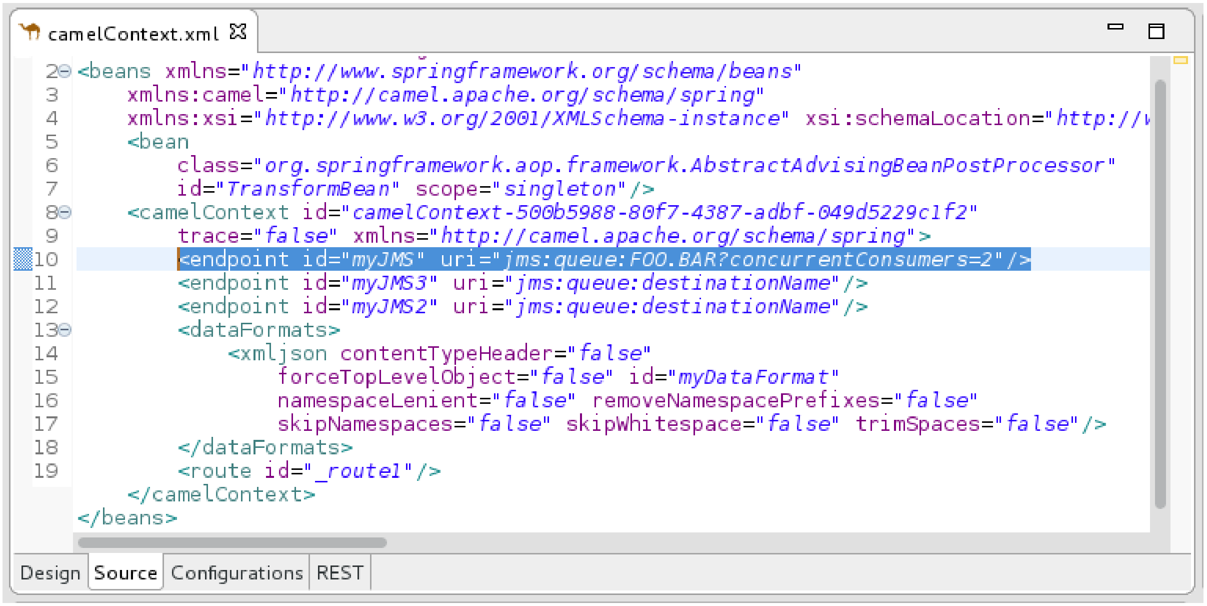
Concurrent Consumers 2

Max Concurrent Consumers

Reply To

Reply To Delivery Persistent ✓

4. In the route editor, click the **Source** tab and check that the tooling added the property **`concurrentConsumers=2`** to the routing context:



```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:camel="http://camel.apache.org/schema/spring"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://w
    <bean
        class="org.springframework.aop.framework.AbstractAdvisingBeanPostProcessor"
        id="TransformBean" scope="singleton"/>
    <camelContext id="camelContext-500b5988-80f7-4387-adbf-049d5229c1f2"
        trace="false" xmlns="http://camel.apache.org/schema/spring">
        <endpoint id="myJMS" uri="jms:queue:FOO.BAR?concurrentConsumers=2"/>
        <endpoint id="myJMS3" uri="jms:queue:destinationName"/>
        <endpoint id="myJMS2" uri="jms:queue:destinationName"/>
        <dataFormats>
            <xmljson contentTypeHeader="false"
                forceTopLevelObject="false" id="myDataFormat"
                namespaceLenient="false" removeNamespacePrefixes="false"
                skipNamespaces="false" skipWhitespace="false" trimSpaces="false"/>
        </dataFormats>
        <route id="_route1"/>
    </camelContext>
</beans>
```

5. When done, save your changes by selecting **File** → **Save** on the menu bar.

2.7. CONFIGURING THE ROUTE EDITOR

Overview

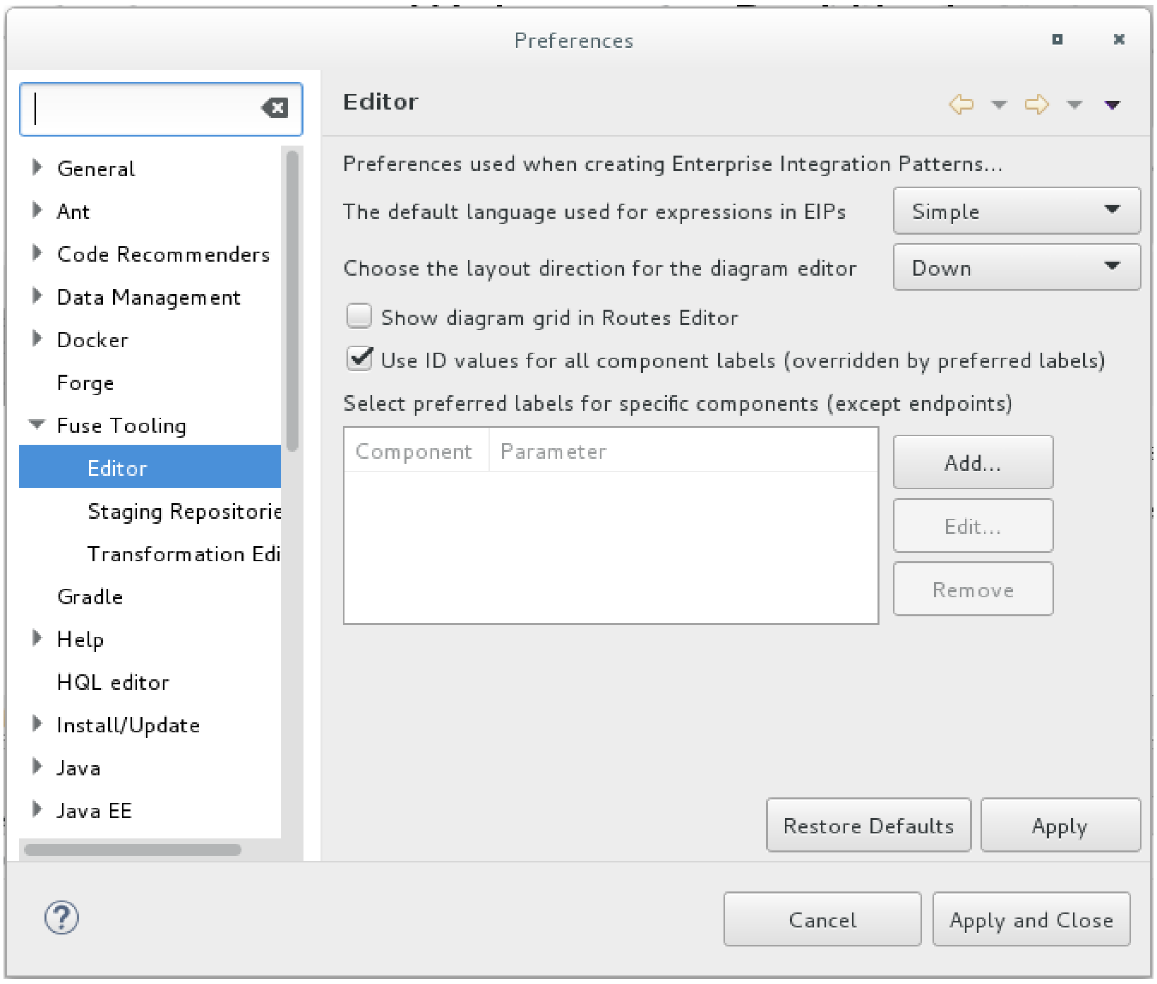
Using Fuse preference settings, you can specify options for the route editor's behavior and user interface:

- The default language to use for expressions in Enterprise Integration Patterns (EIPs)
- The direction (to the right or down) in which patterns flow on the Design canvas when you create routes
- Whether the Design canvas displays a grid overlay in the background of the canvas.
- The method for labeling nodes on the Design canvas

Procedure

To configure the route editor:

1. Open the **Editor** preferences window:
   - On Linux and Windows machines, select **Windows** → **Preferences** → **Fuse Tooling** → **Editor**.
   - On OS X, select **Developer Studio** → **Preferences** → **Fuse Tooling** → **Editor**.



Preferences used when creating Enterprise Integration Patterns...

The default language used for expressions in EIPs: Simple

Choose the layout direction for the diagram editor: Down

[ ] Show diagram grid in Routes Editor

[✓] Use ID values for all component labels (overridden by preferred labels)

Select preferred labels for specific components (except endpoints)

2. To select the default language that you want to use for expressions in Enterprise Integration Pattern (EIP) components, select a language from the drop-down list. The default is *Simple*.

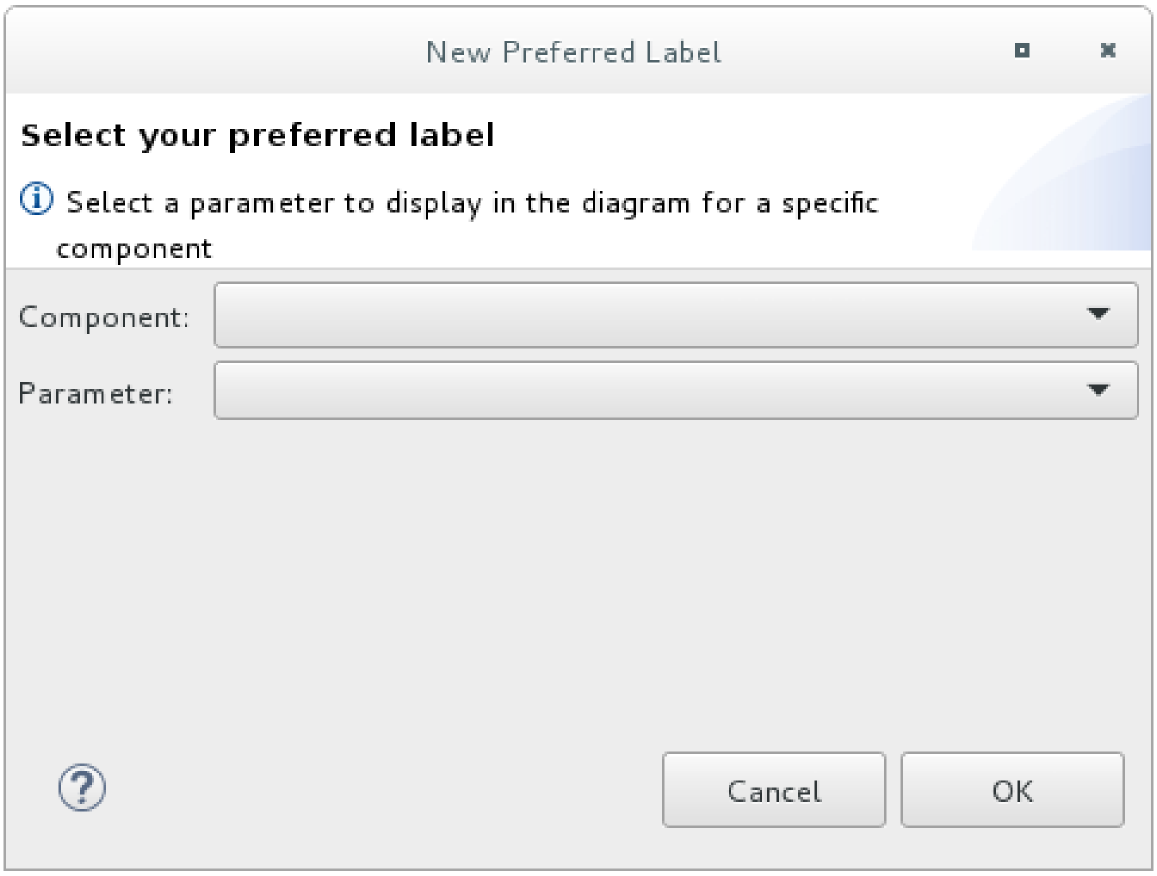
3. To specify the direction in which you want the route editor to align the patterns in a route, select **Down** or **Right**. The default is *Down*.

4. To enable or disable displaying a grid overlay on the background of the canvas, check the box next to **Show diagram grid in Routes Editor**. The default is *enabled*.

5. To enable or disable using component IDs as labels in the route editor's **Design** tab, check the box next to **Use ID values for component labels**. The default is *disabled*.
   If you check this option and also specify a preferred label for a component (see Step 6), then the preferred label is used for that component instead of the ID value.

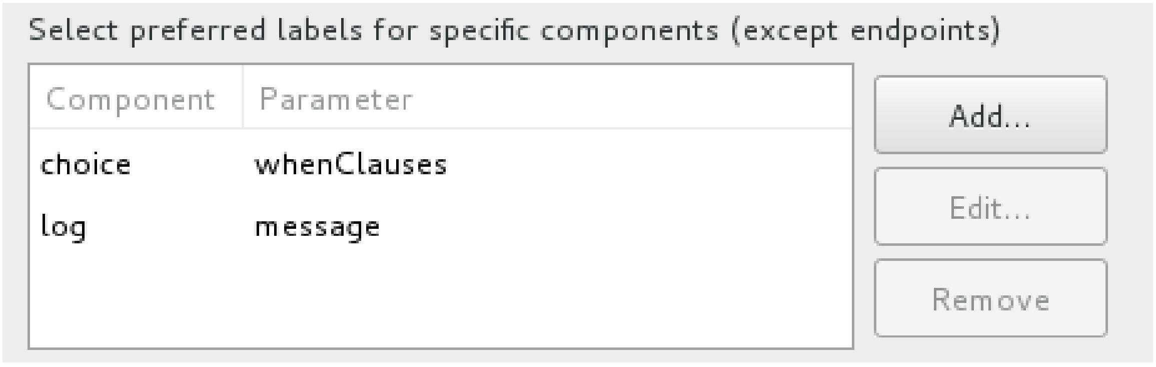
6. To use a parameter as the label for a component (except for endpoints, such as **File** nodes) in the route editor's **Design** tab:

   a. In the **Preferred labels** section, click **Add**. The **New Preferred Label** dialog opens.


New Preferred Label
Select your preferred label
Select a parameter to display in the diagram for a specific component
Component:
Parameter:
Cancel
OK

b. Select a **Component** and then select the **Parameter** to use as the label for the component.

c. Click **OK**. The component and parameter pairs are listed in the Editor Preferences window.


Select preferred labels for specific components (except endpoints)

Component	Parameter
choice	whenClauses
log	message

You can optionally **Edit** and **Remove** component labels.

NOTE

If you check the **Use ID values for component labels** option, it applies to all components except for the components listed in the **Preferred labels** section.

7. Click **Apply and Close** to apply the changes to the **Editor** preferences and close the **Preferences** window.

NOTE

You can restore the route editor's original defaults at any time by returning to the **Editor** preferences dialog and clicking **Restore Defaults**.

CHAPTER 3. VIEWING AND EDITING REST DSL COMPONENTS

If your project’s Camel Context file contains Camel Rest DSL components, you can view a graphical representation of the components in the route editor’s **REST** tab and edit the components in the **Source** tab.

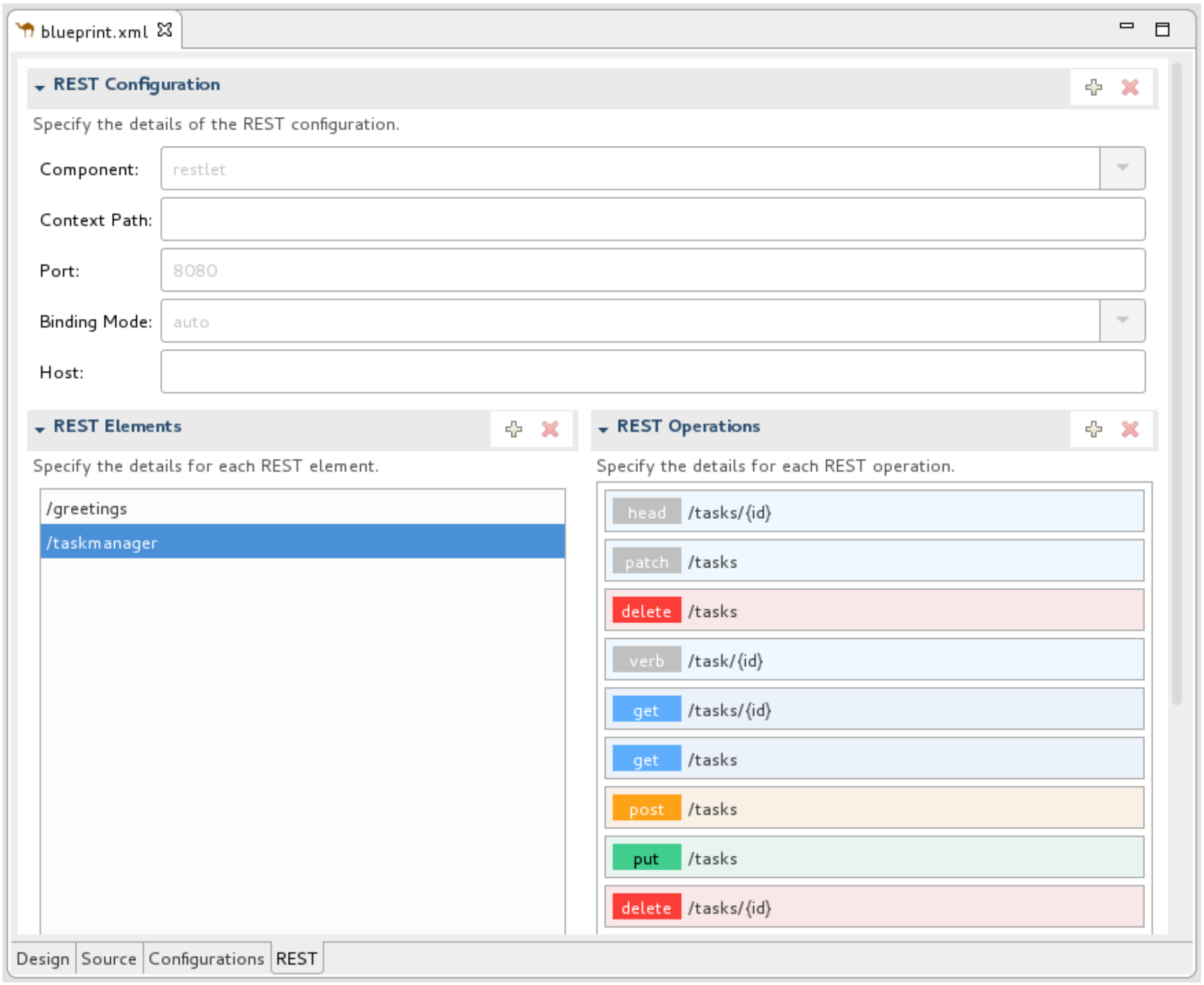
3.1. VIEWING REST DSL COMPONENTS

If your project’s Camel Context file contains Camel Rest DSL components, follow these steps to view the REST components:

1. Open the Camel Context file in the route editor.
2. Click the **REST** tab to the view the Rest DSL components.

NOTE

For this release, the **REST** tab provides a read-only graphical view of Camel Rest DSL components. All fields are greyed out and the ✚ and ✖ icons are disabled.



blueprint.xml

REST Configuration

Specify the details of the REST configuration.

Component:	restlet
Context Path:	
Port:	8080
Binding Mode:	auto
Host:	

REST Elements

Specify the details for each REST element.

/greetings
/taskmanager

REST Operations

Specify the details for each REST operation.

head	/tasks/{id}
patch	/tasks
delete	/tasks
verb	/task/{id}
get	/tasks/{id}
get	/tasks
post	/tasks
put	/tasks
delete	/tasks/{id}

Design | Source | Configurations | REST

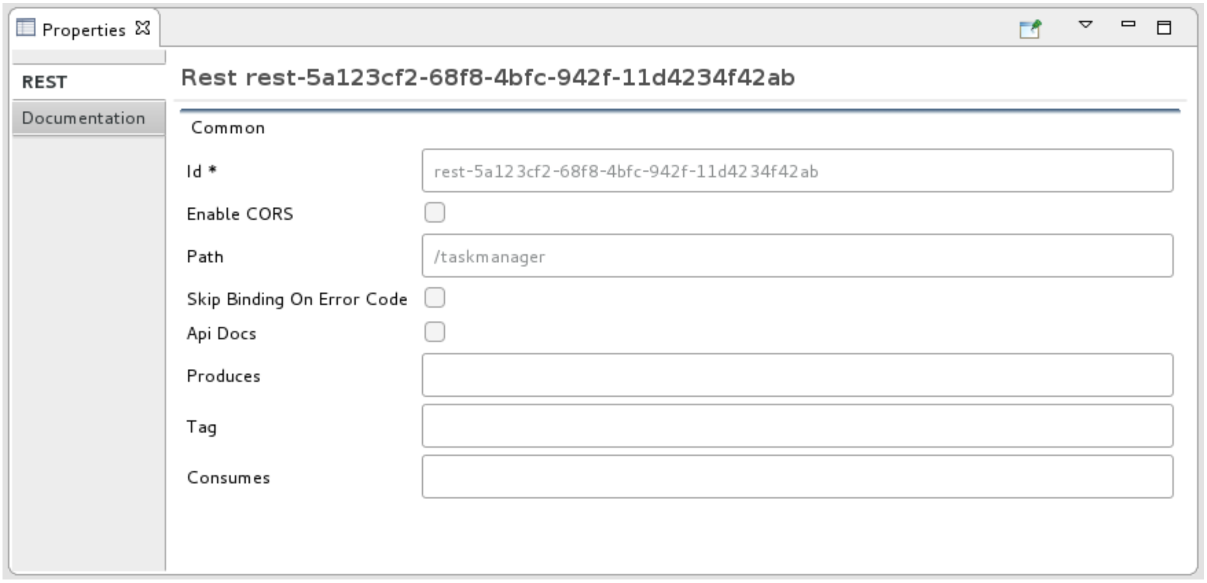
The **REST Configuration** section displays these configuration options:

- **Component** — The Camel component to use for the REST transport.

- **Context Path** — The leading context-path for the REST services. You can use this option for components, such as Servlet, where the web application is deployed using a context-path.
- **Port** — The port number that exposes the REST service.
- **Binding Mode** — The binding mode for JSON or XML format messages. Possible values are: **off** (the default), **auto**, **json**, **xml**, or **json_xml**.
- **Host** — The hostname to use for exposing the REST service.

3. Click a REST element to view its associated operations (for example, **`GET`**, **`POST`**, **`PUT`**, and **`DELETE`**) in the **REST Operations** section.

4. Click a REST element or a REST operation to view its properties in the **Properties** view.



Rest rest-5a123cf2-68f8-4bfc-942f-11d4234f42ab

Common

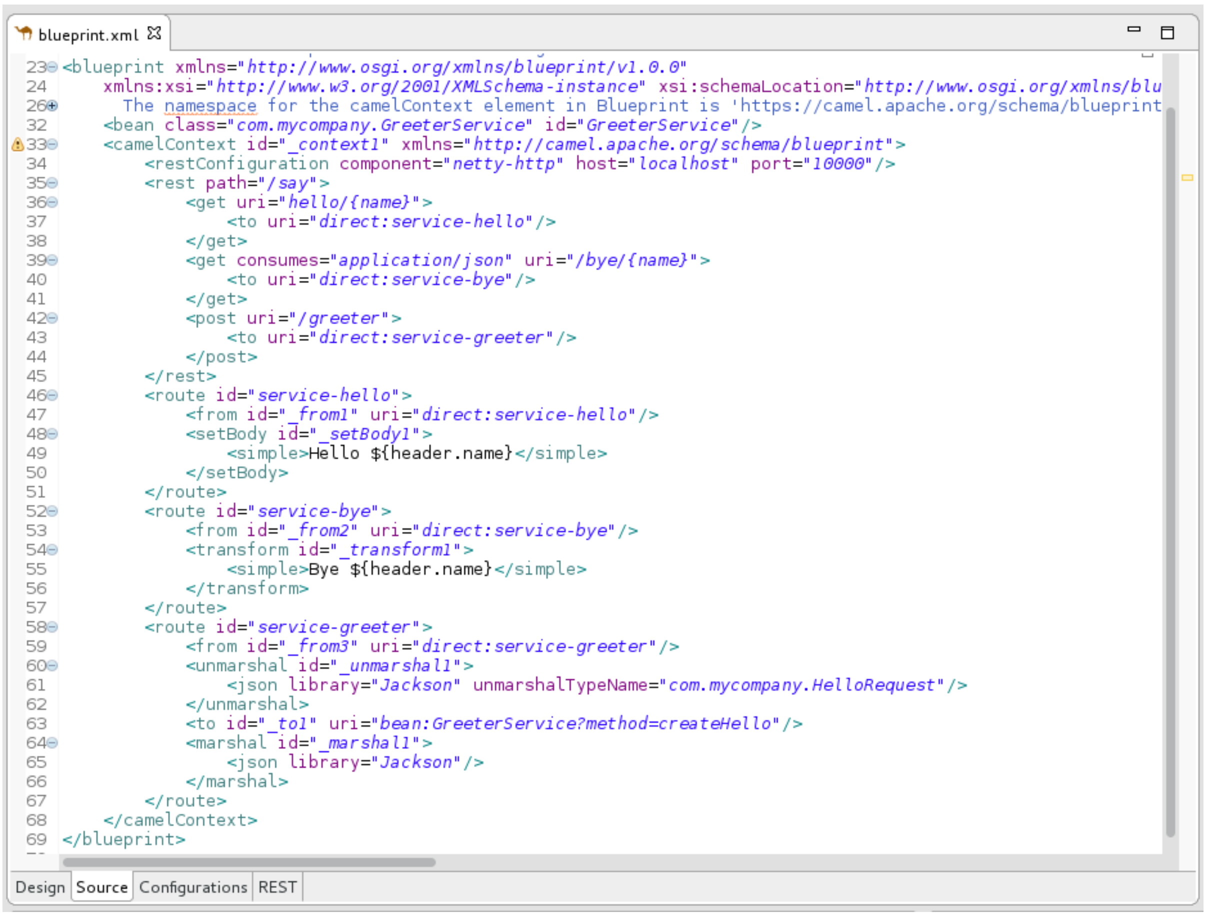
Id *	rest-5a123cf2-68f8-4bfc-942f-11d4234f42ab
Enable CORS	☐
Path	/taskmanager
Skip Binding On Error Code	☐
Api Docs	☐
Produces	
Tag	
Consumes	

For information about the Camel Rest DSL, see the “Defining REST services” section of the Apache Camel Development Guide.

3.2. EDITING REST DSL COMPONENTS

To add, edit, or delete REST elements in your project’s Camel Context file:

1. Open the Camel Context file in the route editor.
2. Click the route editor’s **Source** tab and then edit the code.



```
<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/blu
      The namespace for the camelContext element in Blueprint is 'https://camel.apache.org/schema/blueprint
    <bean class="com.mycompany.GreeterService" id="GreeterService"/>
    <camelContext id="_context1" xmlns="http://camel.apache.org/schema/blueprint">
        <restConfiguration component="netty-http" host="localhost" port="10000"/>
        <rest path="/say">
            <get uri="hello/{name}">
                <to uri="direct:service-hello"/>
            </get>
            <get consumes="application/json" uri="/bye/{name}">
                <to uri="direct:service-bye"/>
            </get>
            <post uri="/greeter">
                <to uri="direct:service-greeter"/>
            </post>
        </rest>
        <route id="service-hello">
            <from id="_from1" uri="direct:service-hello"/>
            <setBody id="_setBody1">
                <simple>Hello ${header.name}</simple>
            </setBody>
        </route>
        <route id="service-bye">
            <from id="_from2" uri="direct:service-bye"/>
            <transform id="_transform1">
                <simple>Bye ${header.name}</simple>
            </transform>
        </route>
        <route id="service-greeter">
            <from id="_from3" uri="direct:service-greeter"/>
            <unmarshal id="_unmarshal1">
                <json library="Jackson" unmarshalTypeName="com.mycompany.HelloRequest"/>
            </unmarshal>
            <to id="_to1" uri="bean:GreeterService?method=createHello"/>
            <marshal id="_marshal1">
                <json library="Jackson"/>
            </marshal>
        </route>
    </camelContext>
</blueprint>
```

3. Optionally, click the **REST** tab to see the changes in the graphical view.

4. To save your changes, select **File** → **Save**.

CHAPTER 4. MIGRATING A SOAP APPLICATION TO RESTFUL WEB SERVICES

You can use the **WSDL-to-Camel Rest** wizard to migrate from an existing JAX-WS SOAP web services implementation to a RESTful web services implementation by using an existing WSDL. The wizard is available as part of Fuse tooling and it is based on the `wsdl2rest` utility, available in Github wsdl2rest project.

It supports the following specifications/configurations:

- Document/literal (doc/lit)
- Document/literal Wrapped (doc/lit wrap)
- RPC/literal (rpclit)

4.1. WIZARD WORKFLOW

The wizard uses a WSDL path which can be a valid URL (in form of a file:// URL) or a remote URL (using `http` or `https`) and the selected Fuse project. In the wizard, you provide the paths for the generated Java code, the generated Camel configuration file, the target address for the generated camel endpoint, and the bean implementation class.

The wizard generates the destination Java and Camel paths based on the project. The generated Java code is available in `src/main/java` and the Camel file in an appropriate location for Spring or Blueprint, such as `src/main/resources/META-INF/spring`.

As a user you can modify the generated classes and configure them to a certain point.

4.2. PREREQUISITES

- A pre-existing Fuse Integration Project. You can create a new project using the **File→ New→ Fuse Integration Project** wizard.
- A WSDL file accessible through URL. It can be local, for example, `file:// url` or remote using `http` or `https`.

4.3. RUN THE WIZARD USING A SAMPLE PROJECT

1. In Developer Studio, select your Fuse project in the **Project Explorer** view.
2. Right-click on the Fuse project and then select **New → Camel Rest DSL from WSDL**.
3. Click **Next**. The **Select Incoming WSDL and Project for Generated Output** page opens. The **Destination Project** field is automatically pre-populated with the Fuse project selected in the Project Explorer. This project is the destination for the artifacts that the wizard generates.
4. For **WSDL File**, specify the original SOAP service for processing.
5. Click **Next**. The **Specify Advanced Options for WSDL-to-REST Processing** page appears.
    a. For **Destination Java Folder**, specify the location of the CXF-generated Java classes.

b. For **Destination Camel Folder**, specify the location of the generated Camel Rest DSL configuration file.

> **NOTE**
>
> The path and name of the generated Camel file varies based on the type of Camel project: * For Spring projects: **`src/main/resources/META-INF/rest-camel-context.xml`** * For Spring Boot projects: **`src/main/resources/spring/rest-springboot-context.xml`** * For Blueprint projects: **`src/main/resources/OSGI-INF/rest-blueprint-context.xml`**

c. Optionally, for **Target Service Address** provide the SOAP address specified in the WSDL binding. You can change this option to match the actual address of the SOAP service referenced by the WSDL.

d. Optionally, for **Target REST Service Address** provide the URL for accessing the REST service. This URL is decomposed into settings for the **`<restConfiguration>`** and **`<rest>`** tags in the Rest DSL to specify the complete URL for REST operations mapped to the SOAP service.

6. Click **Finish** to create two files as the output:

   - The Camel configuration with Rest DSL.
   - Java classes from CXF to wrap the SOAP service.

CHAPTER 5. CREATING A NEW APACHE CAMEL JUNIT TEST CASE

OVERVIEW

A common way of testing routes is to use JUnit. The design time tooling includes a wizard that simplifies creating a JUnit test case for your routes. The wizard uses the endpoints you specify to generate the starting point code and configuration for the test.

NOTE

After you create the boilerplate JUnit test case, you need to modify it to add expectations and assertions specific to the route that you've created or modified, so the test is valid for the route.

PREREQUISITES

Before you create a new JUnit test case, you need to perform a preliminary task:

- If you are replacing an existing JUnit test case, you need to delete it before you create a new one. See the section called "Deleting and existing JUnit test case".
- If you are creating a new JUnit test case in a project that hasn't one, you need to first create the *project_root*/`src/test/java` folder for the test case that is included in the build path. See the section called "Creating and adding the `src/test/java` folder to the build path".

DELETING AND EXISTING JUNIT TEST CASE

1. In the **Project Explorer** view, expand the project's root node to expose the *<root_project>*`/src/test/java` folder.
2. Locate the JUnit test case file in the `/src/test/java` folder.
   Depending on which DSL the project is based on, the JUnit test case file is named `BlueprintXmlTest.java` or `CamelContextXmlTest.java`.
3. Right-click the JUnit test case `.java` file to open the context menu, and then select **Delete**.
   The JUnit test case `.java` file disappears from the **Project Explorer** view.

   You can now create a new JUnit test case.

CREATING AND ADDING THE `SRC/TEST/JAVA` FOLDER TO THE BUILD PATH

1. In the **Project Explorer** view, right-click the project's root to open the context menu.
2. Select **New** → **Folder** to open the **Create a new folder resource** wizard.
3. In the wizard's project tree pane, expand the project's root node and select the `src` folder.
   Make sure *<project_root>*`/src` appears in the **Enter or select the parent folder** field.
4. In **Folder name**, enter `/test/java`. This folder will store the new JUnit test case you create.

5. Click **Finish**.
   In the **Project Explorer** view, the new **`src/test/java`** folder appears under the **`src/main/resources`** folder. You can verify that this folder is on the class path by opening its context menu and selecting **Build Path**. If **Remove from Build Path** is a menu option, you know the **`src/test/java`** folder is on the class path.

   You can now create a new JUnit test case.

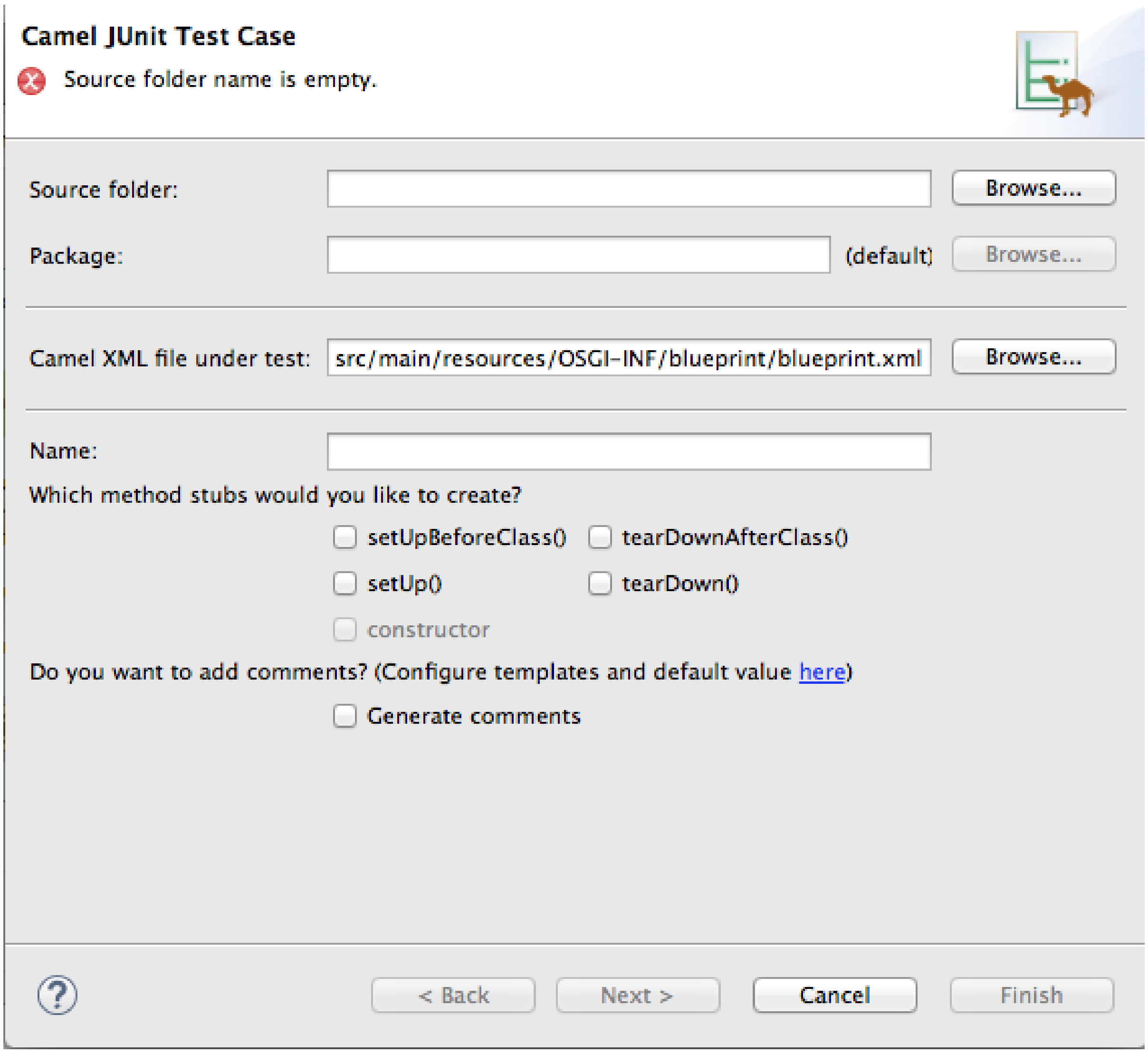
CREATING A JUNIT TEST CASE

To create a new JUnit test case for your route:

1. In the **Project Explorer** view, select the routing context **`.xml`** file in your project.

2. Right-click it to open the context menu, and then select **New** → **Camel Test Case** to open the **New Camel JUnit Test Case** wizard, as shown in Figure 5.1, "New Camel JUnit Test Case wizard".

   Figure 5.1. New Camel JUnit Test Case wizard



Camel JUnit Test Case

Source folder name is empty.

Source folder:

Package: (default)

Camel XML file under test: src/main/resources/OSGI-INF/blueprint/blueprint.xml

Name:

Which method stubs would you like to create?

setUpBeforeClass() tearDownAfterClass()

setUp() tearDown()

constructor

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

Generate comments

   Alternatively, you can open the wizard by selecting **File** → **New** → **Other** > **Fuse** > **Camel Test Case** from the menu bar.

3. In **Source folder**, accept the default location of the source code for the test case, or enter another location.

   You can click Browse... to search for a location.

4. In **Package**, accept the default package name for the generated test code, or enter another package name.

   You can click Browse... to search for a package.

5. In **Camel XML file under test**, accept the default pathname of the routing context file that contains the route you want to test, or enter another pathname.

   You can click Browse... to search for a context file.

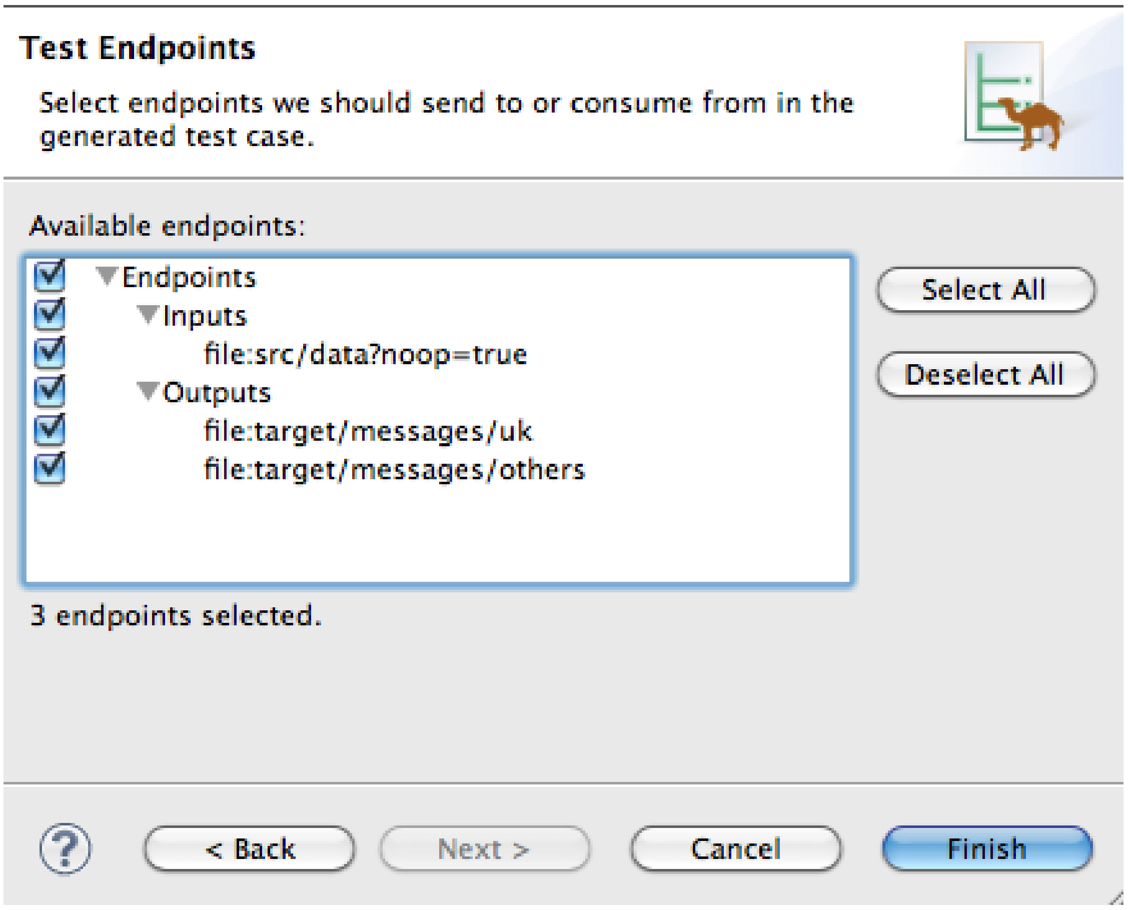
6. In **Name**, accept the default name for the generated test class, or enter another name.

7. Select the method stubs you want to include in the generated code.

8. If you want to include the default generated comments in the generated code, check the **Generate comments** box.

9. Click Next > to open the **Test Endpoints** page. For example, Figure 5.2, "New Camel JUnit Test Case page" shows a route's input and output file endpoints selected.

Figure 5.2. New Camel JUnit Test Case page



Test Endpoints

Select endpoints we should send to or consume from in the generated test case.

Available endpoints:

- [✓] Endpoints
  - [✓] Inputs
    - [✓] file:src/data?noop=true
  - [✓] Outputs
    - [✓] file:target/messages/uk
    - [✓] file:target/messages/others

3 endpoints selected.

10. Under **Available endpoints**, select the endpoints you want to test. Click the checkbox next to any selected endpoint to deselect it.

11. Click Finish .

NOTE

If prompted, add JUnit to the build path.

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

CHAPTER 6. RUNNING ROUTES INSIDE RED HAT FUSE TOOLING

There are two ways to run your routes using the tooling:

- Section 6.1, “Running routes as a local Camel context”
- Section 6.2, “Running routes using Maven”

6.1. RUNNING ROUTES AS A LOCAL CAMEL CONTEXT

Overview

The simplest way to run an Apache Camel route is as a **Local Camel Context**. This method enables you to launch the route directly from the **Project Explorer** view’s context menu. When you run a route from the context menu, the tooling automatically creates a runtime profile for you. You can also create a custom runtime profile for running your route.

Your route runs as if it were invoked directly from the command line and uses Apache Camel’s embedded Spring container. You can configure a number of the runtime parameters by editing the runtime profile.

Procedure

To run a route as a local Camel context:

1. In the **Project Explorer** view, select a routing context file.
2. Right-click it to open the context menu, and then select **Run As** → **Local Camel Context**.

NOTE

Selecting **Local Camel Context (without tests)** directs the tooling to run the project without performing validation tests, which may be faster.

Result

The **Console** view displays the output generated from running the route.

Related topics

- Section 6.3.1, “Editing a Local Camel Context runtime profile”

6.2. RUNNING ROUTES USING MAVEN

Overview

If the project containing your route is a Maven project, you can use the m2e plug-in to run your route. Using this option, you can execute any Maven goals, before the route runs.

Procedure

To run a route using Maven:

1. In the **Project Explorer** view, select the root of the project .
2. Right-click it to open the context menu, and then select **Run As → Maven build**.
    a. The first time you run the project using Maven, the **Edit Configuration and launch** editor opens, so you can create a Maven runtime profile.
    To create the runtime profile, on the **Maven** tab:
        i. Make sure the route directory of your Apache Camel project appears in the **Base directory:** field.
        For example, on Linux the root of your project is similar to **`~/workspace/simple-router`**.
        ii. In the **Goals:** field, enter **`camel:run`**.

        IMPORTANT

        If you created your project using the Java DSL, enter **`exec:java`** in the **Goals:** field.

        iii. Click **Apply** and then **Run**.
    b. Subsequent Maven runs use this profile, unless you modify it between runs.

Results

The **Console** view displays the output from the Maven run.

Related topics

- Section 6.3.2, “Editing a Maven runtime profile”

6.3. WORKING WITH RUNTIME PROFILES

Red Hat Fuse Tooling stores information about the runtime environments for each project in *runtime profiles*. The runtime profiles keep track of such information as which Maven goals to call, the Java runtime environment to use, any system variables that need to be set, and so on. A project can have more than one runtime profile.

6.3.1. Editing a Local Camel Context runtime profile

Overview

A **Local Camel Context** runtime profile configures how Apache Camel is invoked to execute a route. A **Local Camel Context** runtime profile stores the name of the context file in which your routes are defined, the name of the **`main`** to invoke, the command line options passed into the JVM, the JRE to use, the classpath to use, any environment variables that need to be set, and a few other pieces of information.

The runtime configuration editor for a **Local Camel Context** runtime profile contains the following tabs:

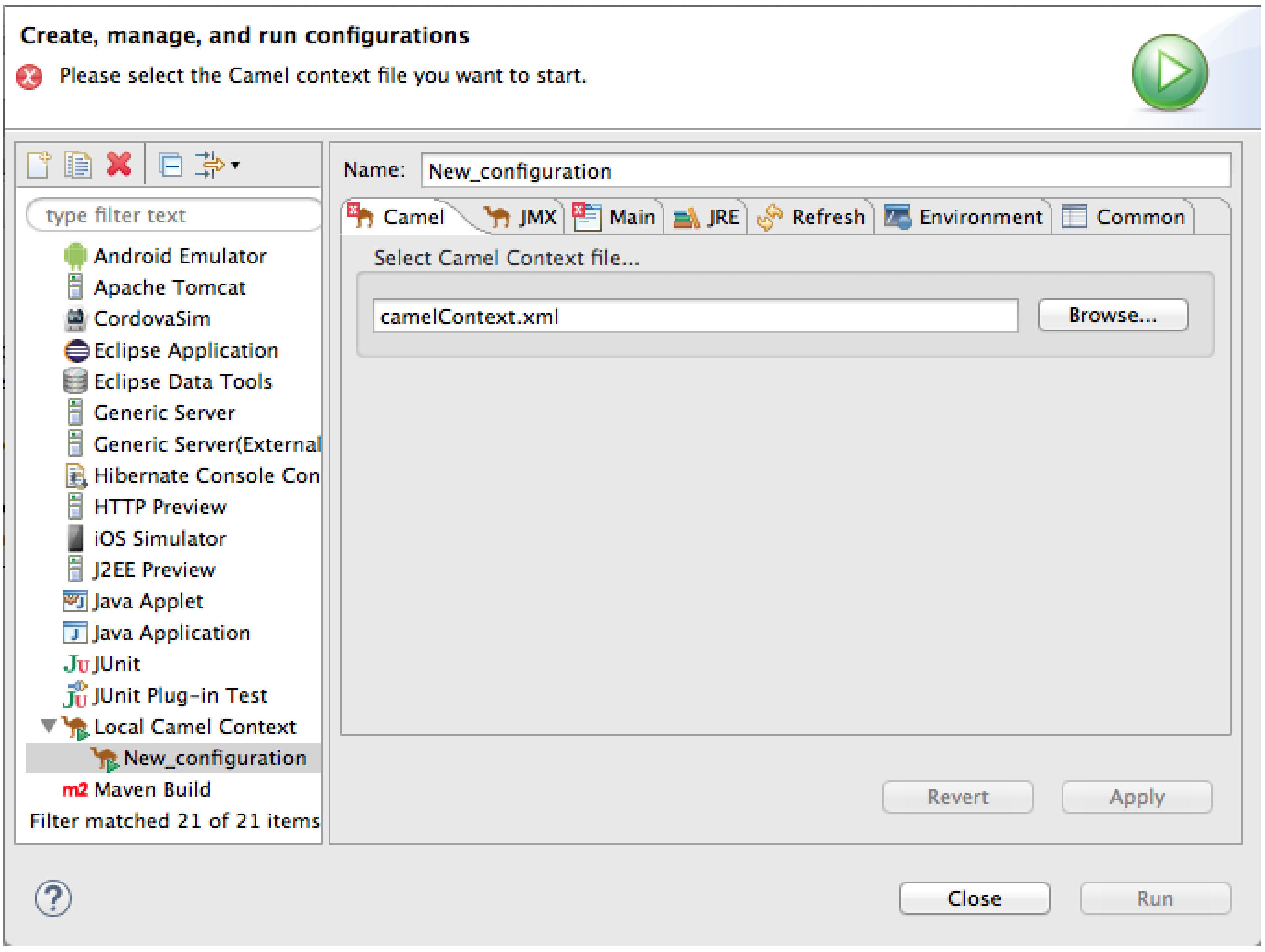
- **Camel Context File** — specifies the name of the new configuration and the full path of the routing context file that contains your routes.
- **JMX** — specifies JMX connection details, including the JMX URI and the user name and password (optional) to use to access it.
- **Main** — specifies the fully qualified name of the project's base directory, a few options for locating the base directory, any goals required to execute before running the route, and the version of the Maven runtime to use.
- **JRE** — specifies the JRE and command line arguments to use when starting the JVM.
- **Refresh** — specifies how Maven refreshes the project's resource files after a run terminates.
- **Environment** — specifies any environment variables that need to be set.
- **Common** — specifies how the profile is stored and the output displayed.

The first time an Apache Camel route is run as a **Local Camel Context**, Red Hat Fuse Tooling creates for the routing context file a default runtime profile, which should not require editing.

Accessing the Local Camel Context's runtime configuration editor

1. In the **Project Explorer** view, select the Camel context file for which you want to edit or create a custom runtime profile.
2. Right-click it to open the context menu, and then select **Run As → Run Configurations** to open the **Run Configurations** dialog.
3. In the **context selection** pane, select **Local Camel Context**, and then click at the top, left of the **context selection** pane.
4. In the **Name** field, enter a new name for your runtime profile.

Figure 6.1. Runtime configuration editor for Local Camel Context



Create, manage, and run configurations

Please select the Camel context file you want to start.

type filter text

- Android Emulator
- Apache Tomcat
- CordovaSim
- Eclipse Application
- Eclipse Data Tools
- Generic Server
- Generic Server(External
- Hibernate Console Con
- HTTP Preview
- iOS Simulator
- J2EE Preview
- Java Applet
- Java Application
- JUnit
- JUnit Plug-in Test
- Local Camel Context
  - New_configuration
- Maven Build

Filter matched 21 of 21 items

Name: New_configuration

Select Camel Context file...

camelContext.xml

Setting the camel context file

The **Camel Context File** tab has one field, **Select Camel Context file…**. Enter the full path to the routing context file that contains your route definitions.

The **Browse** button accesses the **Open Resource** dialog, which facilitates locating the target routing context file. This dialog is preconfigured to search for files that contain Apache Camel routes.

Changing the command line options

By default the only command line option passed to the JVM is:

```
-fa context-file
```

If you are using a custom main class you may need to pass in different options. To do so, on the **Main** tab, click the **Add** button to enter a parameter’s name and value. You can click the **Add Parameter** dialog’s **Variables…** button to display a list of variables that you can select.

To add or modify JVM-specific arguments, edit the **VM arguments** field on the **JRE** tab.

Changing where output is sent

By default, the output generated from running the route is sent to the **Console** view. But you can redirect it to a file instead.

To redirect output to a file:

1. Select the **Common** tab.
2. In the **Standard Input and Output** pane, click the checkbox next to the **Output File:** field, and then enter the path to the file where you want to send the output.
   The **Workspace**, **File System**, and **Variables** buttons facilitate building the path to the output file.

Related topics

- Section 6.1, “Running routes as a local Camel context”

6.3.2. Editing a Maven runtime profile

Overview

A Maven runtime profile configures how Maven invokes Apache Camel. A Maven runtime profile stores the Maven goals to execute, any Maven profiles to use, the version of Maven to use, the JRE to use, the classpath to use, any environment variables that need to be set, and a few other pieces of information.

IMPORTANT

The first time an Apache Camel route is run using Maven, you must create a default runtime profile for it.

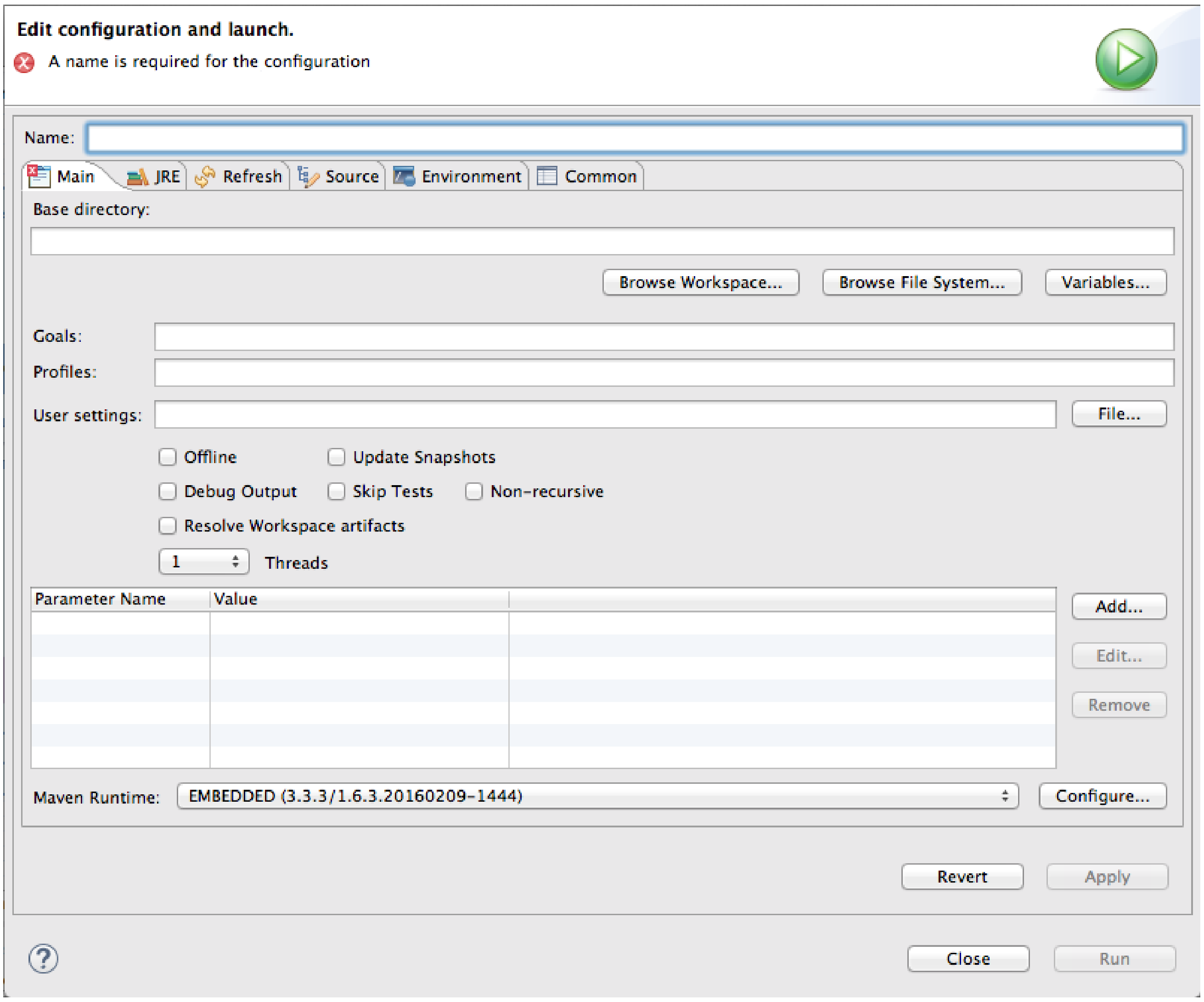
The runtime configuration editor for a Fuse runtime profile contains the following tabs:

- **Main** — specifies the name of the new configuration, the fully qualified name of the project’s base directory, a few options for locating the base directory, any goals required to execute before running the route, and the version of the Maven runtime to use.
- **JRE** — specifies the JRE and command line arguments to use when starting the JVM.
- **Refresh** — specifies how Maven refreshes the project’s resource files after a run terminates.
- **Source** — specifies the location of any additional sources that the project requires.
- **Environment** — specifies any environment variables that need to be set.
- **Common** — specifies how the profile is stored and the output displayed.

Accessing the Maven runtime configuration editor

1. In the **Project Explorer** view, select the root of the project for which you want to edit or create a custom runtime profile.
2. Right-click it to open the context menu, and then select **Run As** → **Run Configurations** to open the **Run Configurations** dialog.
3. In the **context selection** pane, select **Maven Build**, and then click at the top, left of the **context selection** pane.

Figure 6.2. Runtime configuration editor for Maven



Edit configuration and launch.

A name is required for the configuration

Name:

Main | JRE | Refresh | Source | Environment | Common

Base directory:

Browse Workspace... | Browse File System... | Variables...

Goals:

Profiles:

User settings: File...

- [ ] Offline
- [ ] Update Snapshots
- [ ] Debug Output
- [ ] Skip Tests
- [ ] Non-recursive
- [ ] Resolve Workspace artifacts

1 Threads

Parameter Name	Value

Add... | Edit... | Remove

Maven Runtime: EMBEDDED (3.3.3/1.6.3.20160209-1444) Configure...

Revert | Apply

Close | Run

Changing the Maven goal

The most commonly used goal when running a route is **camel:run**. It loads the routes into a Spring container running in its own JVM.

The Apache Camel plug-in also supports a **camel:embedded** goal that loads the Spring container into the same JVM used by Maven. The advantage of this is that the routes should bootstrap faster.

Projects based on Java DSL use the **exec:java** goal.

If your POM contains other goals, you can change the Maven goal used by clicking the **Configure…** button next to the **Maven Runtime** field on the **Main** tab. On the **Installations** dialog, you edit the **Global settings for <selected_runtime> installation** field.

Changing the version of Maven

By default, Red Hat Fuse Tooling for Eclipse uses m2e, which is embedded in Eclipse. If you want to use a different version of Maven or have a newer version installed on your development machine, you can select it from the **Maven Runtime** drop-down menu on the **Main** tab.

Changing where the output is sent

By default, the output from the route execution is sent to the **Console** view. But you can redirect it to a file instead.

To redirect output to a file:

1. Select the **Common** tab.
2. Click the checkbox next to the **Output File:** field, and then enter the path to the file where you want to send the output.
   The **Workspace**, **File System**, and **Variables** buttons facilitate building the path to the output file.

Related topics

- Section 6.2, “Running routes using Maven”

CHAPTER 7. GETTING STARTED WITH FUSE INTEGRATION SERVICES

OVERVIEW

Fuse Integration Services (FIS) provides a set of tools and containerized xPaaS images for developing, deploying, and managing microservices on OpenShift.

IMPORTANT

For FIS projects, Fuse Tooling requires installation of the Red Hat Container Development Kit (CDK) v3.*x*. See Getting Started Guide for instructions. In addition to the prerequisites specified in this guide, you need to establish a Red Hat account if you do not have one. Your Red Hat user name and password are required to start the virtual OpenShift instance provided in the Red Hat Container Development Kit.

You can easily get an account by registering on the Red Hat Customer Portal. Click in the upper right corner of the white banner, and then click REGISTER on the **Login to Your Red Hat Account** page.

Fuse Tooling enables you to develop and deploy FIS 2.0 projects using the s2i binary workflow. In this workflow, the tooling builds your project locally, assembles it into an image stream, then pushes the image stream to OpenShift, where it is used to build the Docker container. Once the Docker container is built, OpenShift deploys it in a pod.

IMPORTANT

Fuse Tooling works only with the S2I binary workflow and only with projects based on the Spring Boot framework.

NOTE

Although Fuse Tooling can deploy FIS projects created using the tooling to remote OpenShift servers, this chapter describes creating and deploying FIS projects to a virtual OpenShift instance, installed locally using the Red Hat Container Development Kit (CDK) v3.*x*.

Creating and deploying your first Fuse FIS project involves:

- the section called “Adding the Red Hat Container Development Kit server”
- the section called “Starting the Container Development Environment (CDE) and virtual OpenShift server”
- the section called “Creating a new OpenShift project”
- the section called “Creating a new Fuse FIS project”
- the section called “Deploying the Fuse FIS project to OpenShift”

NOTE

You can also run a Fuse FIS project as a local Camel context, see Section 6.1, “Running routes as a local Camel context”, and then connect to it in the **JMX Navigator** view, where you can monitor and test the routing context. You can also run the Camel debugger on a Fuse FIS project (Part II, “Debugging Routing Contexts”) to expose and fix any logic errors in the routing context.

ADDING THE RED HAT CONTAINER DEVELOPMENT KIT SERVER

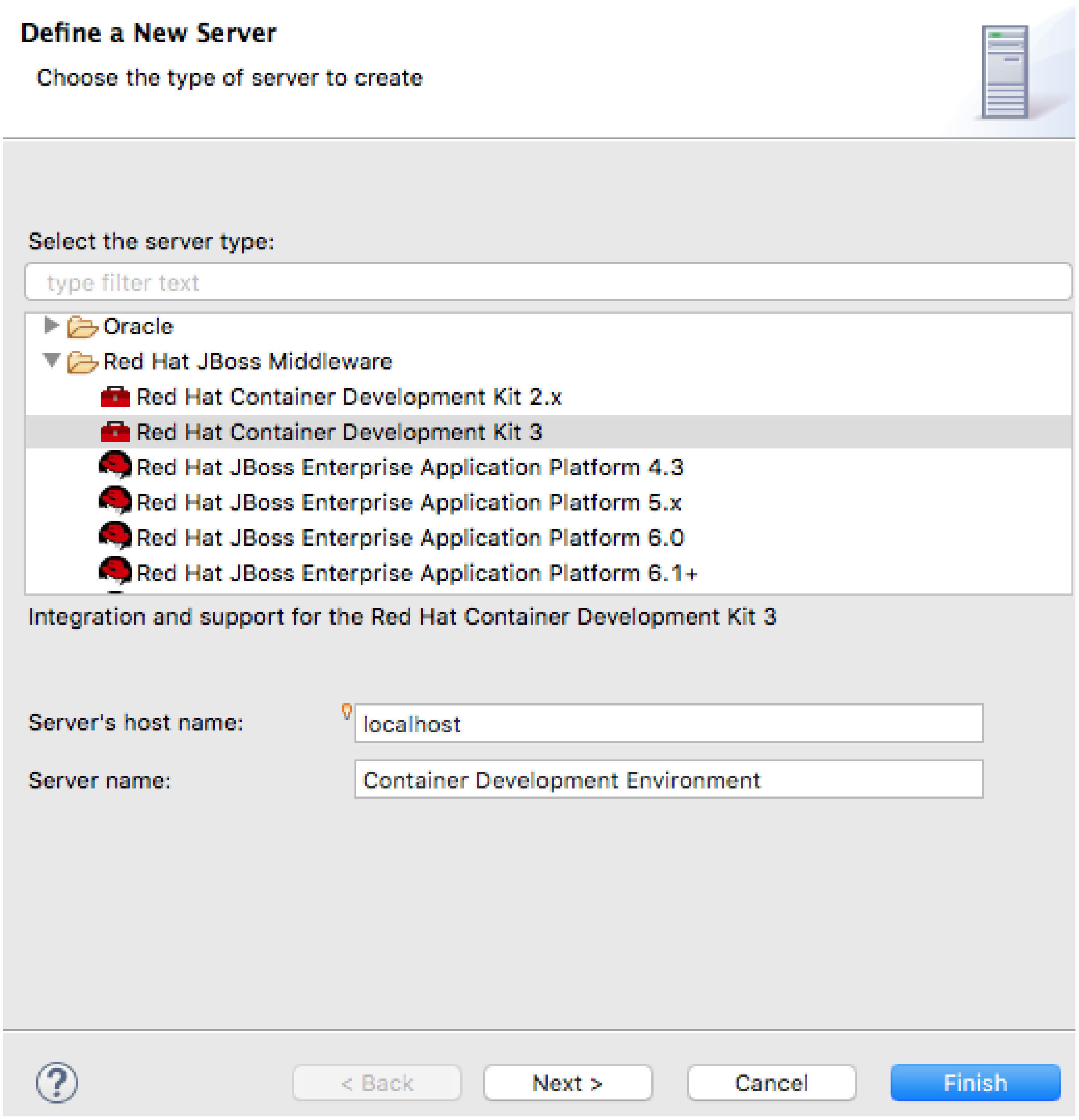
To add the Red Hat Container Development Kit to the **Servers** view:

1. If necessary, switch to the **Fuse Integration** perspective.

   NOTE

   If, in this or any other section in this chapter, a view described in a procedure is not open, you can open it by clicking **Window → Show View → *Other → *view_name**.

2. In the **Servers** view, click the link **No servers are available. Click this link to create a new server…** to open the **Define a New Server** wizard. This link appears only when the **Servers** view contains no server entry.
   Otherwise, right-click in the view to open the context menu, and then select **New → Server** to open the **Define a New Server** wizard.


Define a New Server

Choose the type of server to create

Select the server type:

type filter text

- Oracle
- Red Hat JBoss Middleware
  - Red Hat Container Development Kit 2.x
  - Red Hat Container Development Kit 3
  - Red Hat JBoss Enterprise Application Platform 4.3
  - Red Hat JBoss Enterprise Application Platform 5.x
  - Red Hat JBoss Enterprise Application Platform 6.0
  - Red Hat JBoss Enterprise Application Platform 6.1+

Integration and support for the Red Hat Container Development Kit 3

Server's host name: localhost

Server name: Container Development Environment

3. Select **Red Hat JBoss Middleware** → **Red Hat Container Development Kit 3**.
   Accept the defaults for:

   - **Server's host name** — `localhost`
   - **Server name** — `Container Development Environment`

4. Click **Next** to open the **Red Hat Container Development Environment** page.

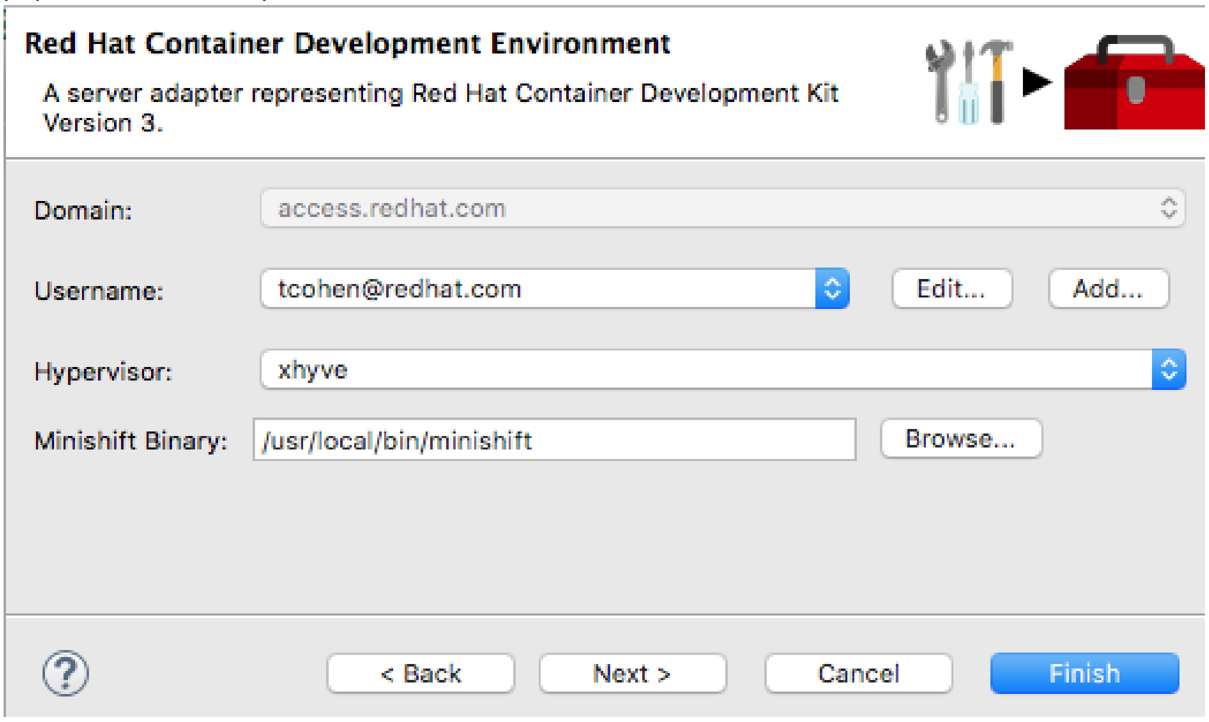
5. Next to **Folder**, click **Browse**, navigate to the location where you installed the Red Hat Container Development Kit 3.*x* and click **Open**.

6. Next to **Username**, click **Add** to open the **Add a Credential** page.

7. Set the credentials this way:

   - **Username** — Enter the name you use to log into your Red Hat account.
   - **Always prompt for password** — Leave as is (disabled).
   - **Password** — Enter the password you use to log into your Red Hat account.

8. Click **OK** to return to the **Red Hat Container Development Environment** page, which is now populated. For example:



Red Hat Container Development Environment

A server adapter representing Red Hat Container Development Kit Version 3.

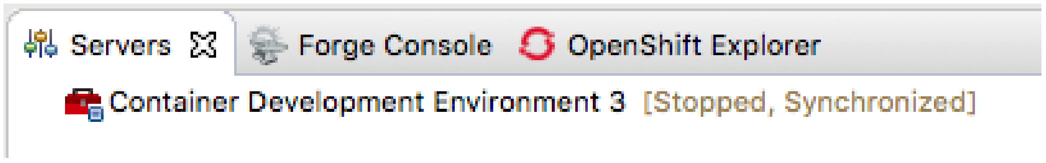
Domain: access.redhat.com

Username: tcohen@redhat.com

Hypervisor: xhyve

Minishift Binary: /usr/local/bin/minishift

9. Click **Finish**. **Container Development Environment 3 [Stopped, Synchronized]** appears in the **Servers** view. **Container Development Environment 3** is the default server name when you add a CDK 3.*x* server.

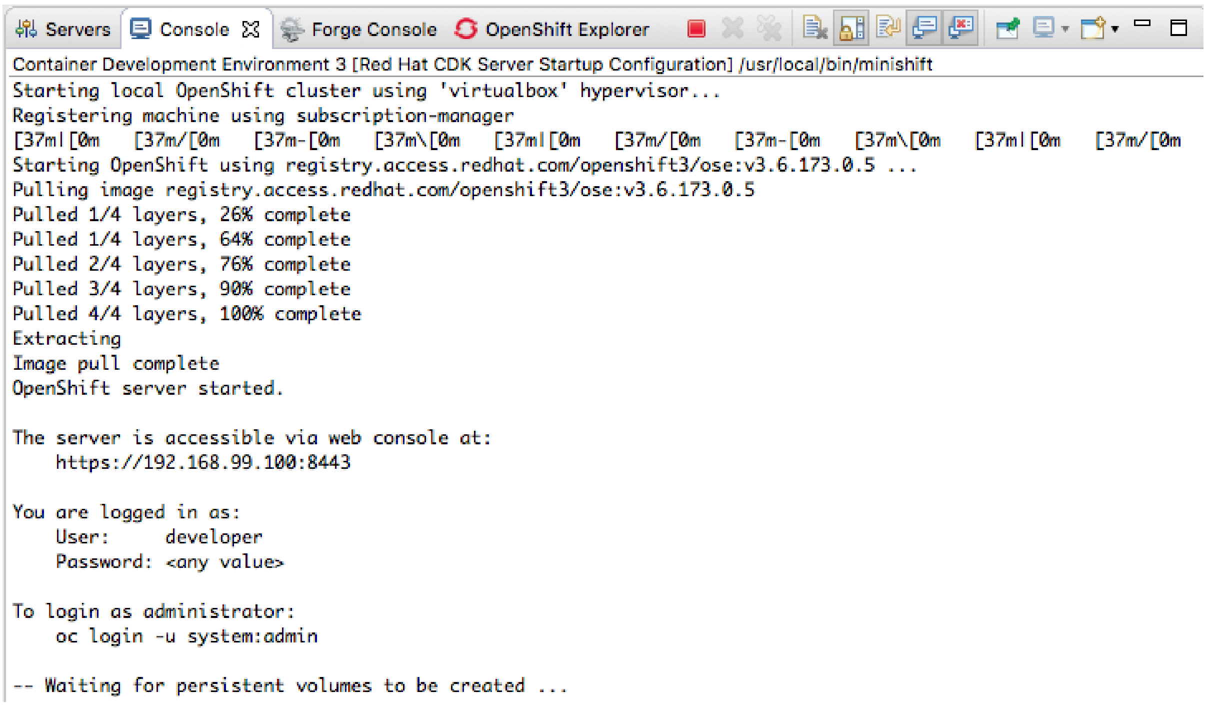


Container Development Environment 3 [Stopped, Synchronized]

STARTING THE CONTAINER DEVELOPMENT ENVIRONMENT (CDE) AND VIRTUAL OPENSHIFT SERVER

Starting the Container Development Environment (CDE) also starts the virtual OpenShift server. Stopping the CDE also stops the virtual OpenShift server.

1. In the **Servers** view, select **Container Development Environment 3 [stopped, Synchronized]**, and then click ▶ on the **Servers** menu bar.
   Console view opens and displays the status of the startup process:

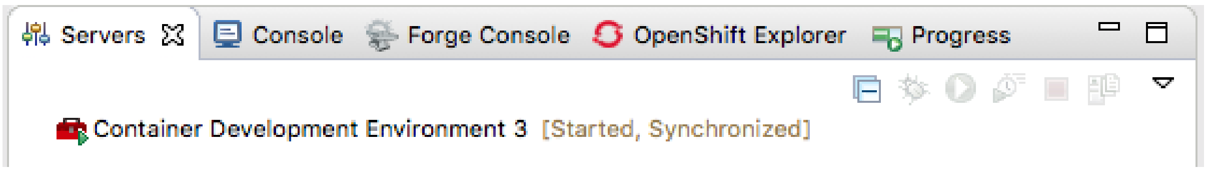


```
Container Development Environment 3 [Red Hat CDK Server Startup Configuration] /usr/local/bin/minishift
Starting local OpenShift cluster using 'virtualbox' hypervisor...
Registering machine using subscription-manager
[37m|[0m   [37m/[0m   [37m-[0m   [37m\[0m   [37m|[0m   [37m/[0m   [37m-[0m   [37m\[0m   [37m|[0m   [37m/[0m
Starting OpenShift using registry.access.redhat.com/openshift3/ose:v3.6.173.0.5 ...
Pulling image registry.access.redhat.com/openshift3/ose:v3.6.173.0.5
Pulled 1/4 layers, 26% complete
Pulled 1/4 layers, 64% complete
Pulled 2/4 layers, 76% complete
Pulled 3/4 layers, 90% complete
Pulled 4/4 layers, 100% complete
Extracting
Image pull complete
OpenShift server started.

The server is accessible via web console at:
    https://192.168.99.100:8443

You are logged in as:
    User:     developer
    Password: <any value>

To login as administrator:
    oc login -u system:admin

-- Waiting for persistent volumes to be created ...
```

NOTE

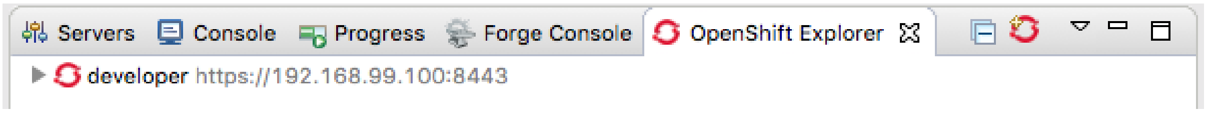
On initial startup, the CDE asks whether you accept the untrusted SSL certificate. Click **Yes**.

When the startup process has finished, the **Servers** view displays:



Container Development Environment 3 [Started, Synchronized]

2. Switch to the **OpenShift Explorer** view.
   The virtual OpenShift server instance, **`developer`**, is also running:



developer https://192.168.99.100:8443

`https://192.168.99.100:8443` is an example of a URL for the OpenShift developer web console. Your installation displays the URL for your instance. For more details, see the section called “Accessing the OpenShift Web Console”.

CREATING A NEW OPENSHIFT PROJECT

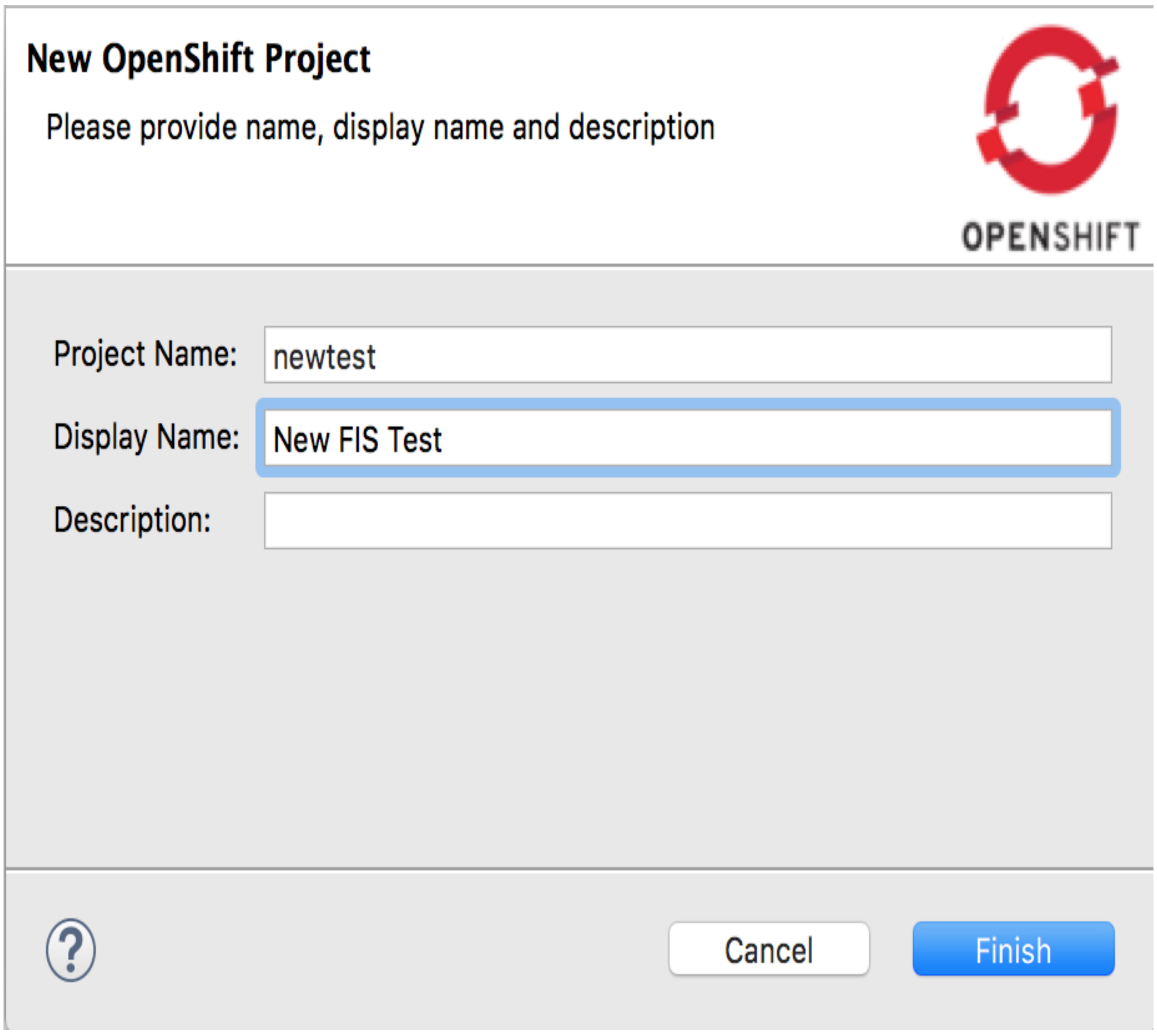
When you deploy your Fuse FIS project to OpenShift, it is published to the OpenShift project you create here.

1. In the **OpenShift Explorer** view, right-click the **developer** entry, to open the context menu.
2. Select **New** → **Project** to open the **New OpenShift Project** wizard.
3. Set the new project’s properties this way:

- In the **Project Name** field, enter the name for the project's namespace on the virtual OpenShift server.
  Only lower case letters, numbers, and dashes are valid.

- In the **Display Name** field, enter the name to display on the virtual OpenShift web console's **Overview** page.

- Leave the **Description** field as is.
  For example:



New OpenShift Project

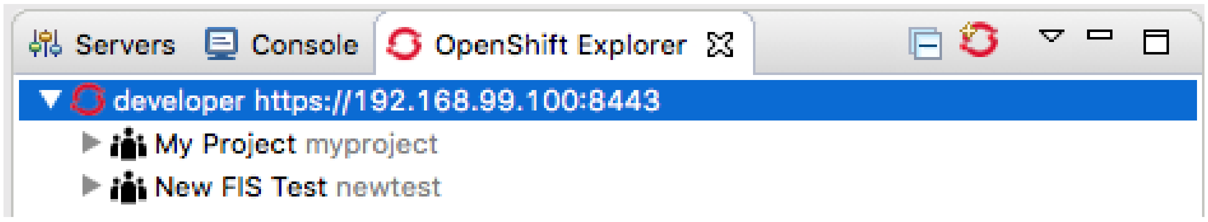
Please provide name, display name and description

Project Name: newtest

Display Name: New FIS Test

Description:

4. Click **Finish**.
   The new OpenShift project (in this example, **New FIS Test newtest**) appears in the **OpenShift Explorer** tab, under, in this example, **developer** https://192.168.99.100:8443:

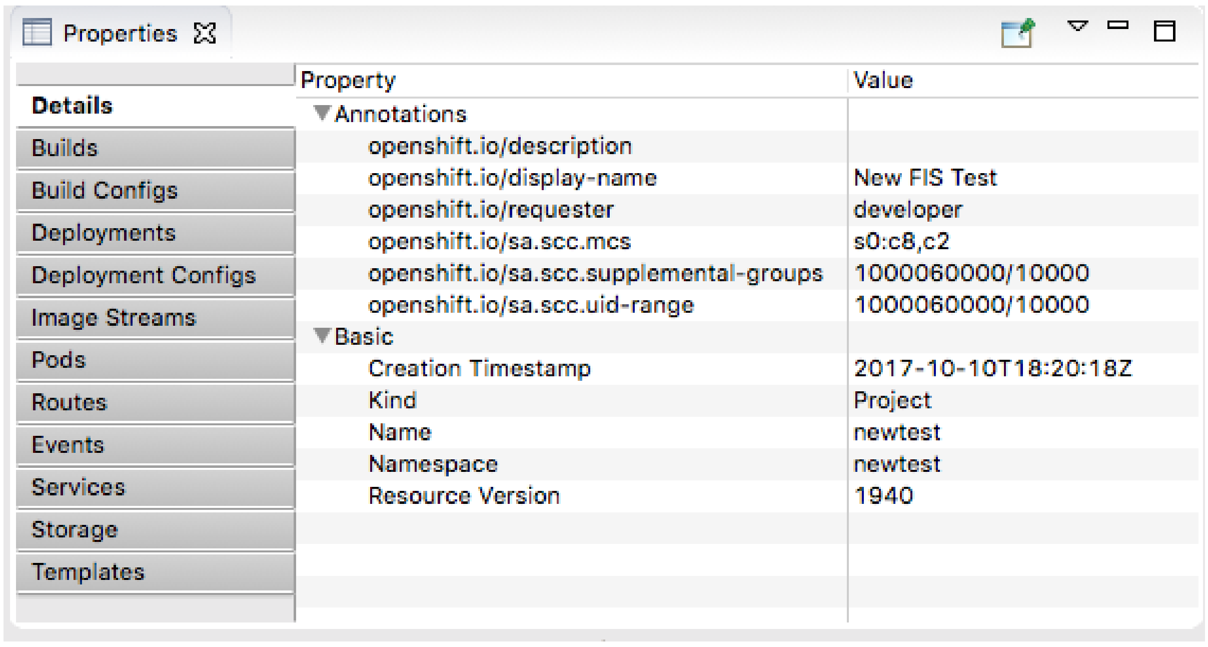


developer https://192.168.99.100:8443
My Project myproject
New FIS Test newtest

NOTE

MyProject myproject is an initial example project included with OpenShift.

With **New FIS Test newtest** selected in the **OpenShift Explorer** view, the **Properties** view displays the project's details. For example:



Property	Value
▼Annotations	
openshift.io/description	
openshift.io/display-name	New FIS Test
openshift.io/requester	developer
openshift.io/sa.scc.mcs	s0:c8,c2
openshift.io/sa.scc.supplemental-groups	1000060000/10000
openshift.io/sa.scc.uid-range	1000060000/10000
▼Basic	
Creation Timestamp	2017-10-10T18:20:18Z
Kind	Project
Name	newtest
Namespace	newtest
Resource Version	1940

NOTE

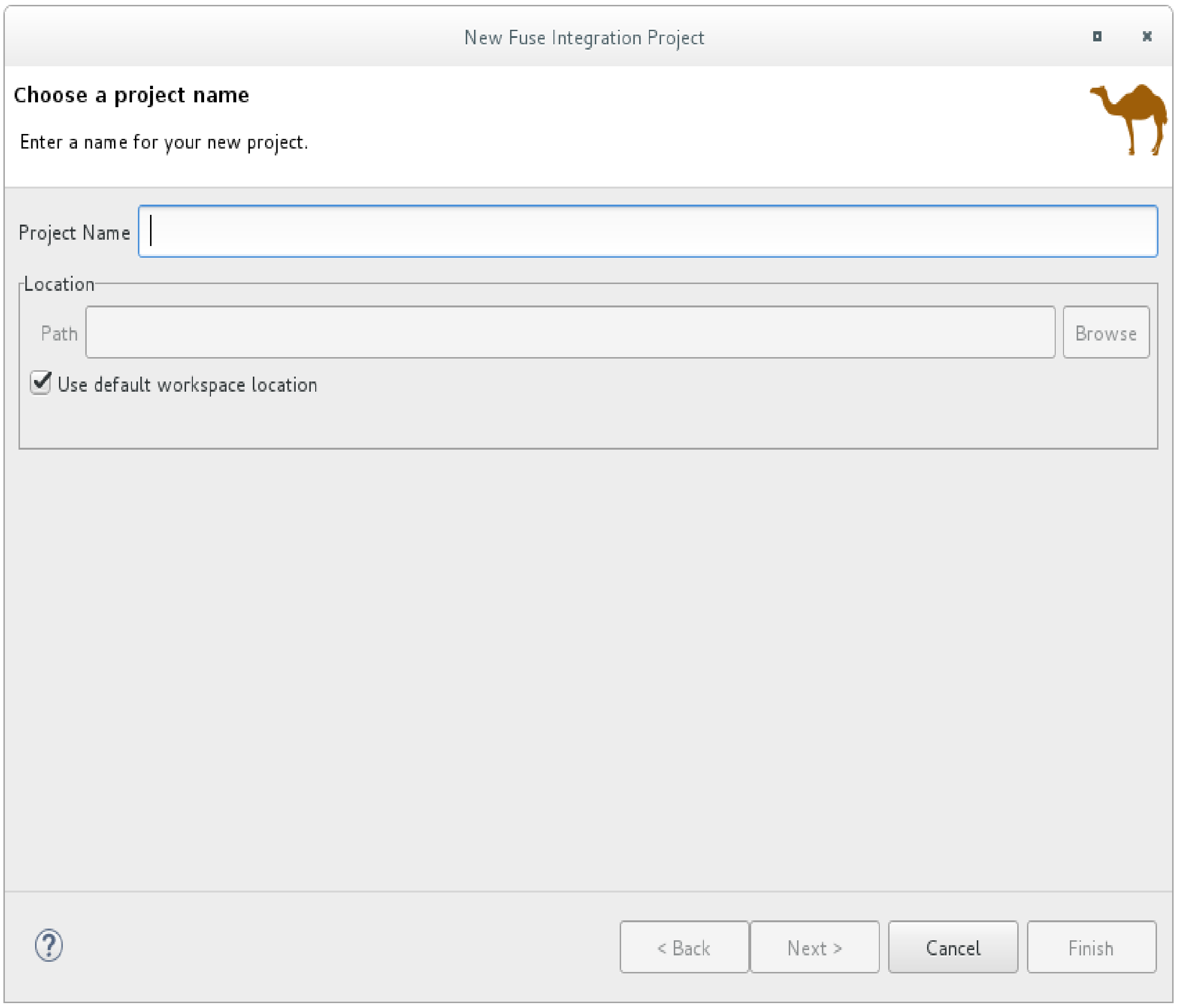
When you deploy the project to OpenShift, the **Properties** view gathers and displays the same information about the project that the OpenShift web console does.

CREATING A NEW FUSE FIS PROJECT

Before you create a new Fuse FIS project, you should enable staging repositories. This is needed because some Maven artifacts are not in default Maven repositories. To enable staging repositories, select **Window** → **Preferences** → **Fuse Tooling** → **Staging Repositories**.

To create a FIS project, use the **Spring Boot on OpenShift** template:

1. In the **Project Explorer** view, right-click to open the context menu and then select**New** → **Fuse Integration Project** to open the wizard's **Choose a project name** page:


New Fuse Integration Project

Choose a project name

Enter a name for your new project.

Project Name

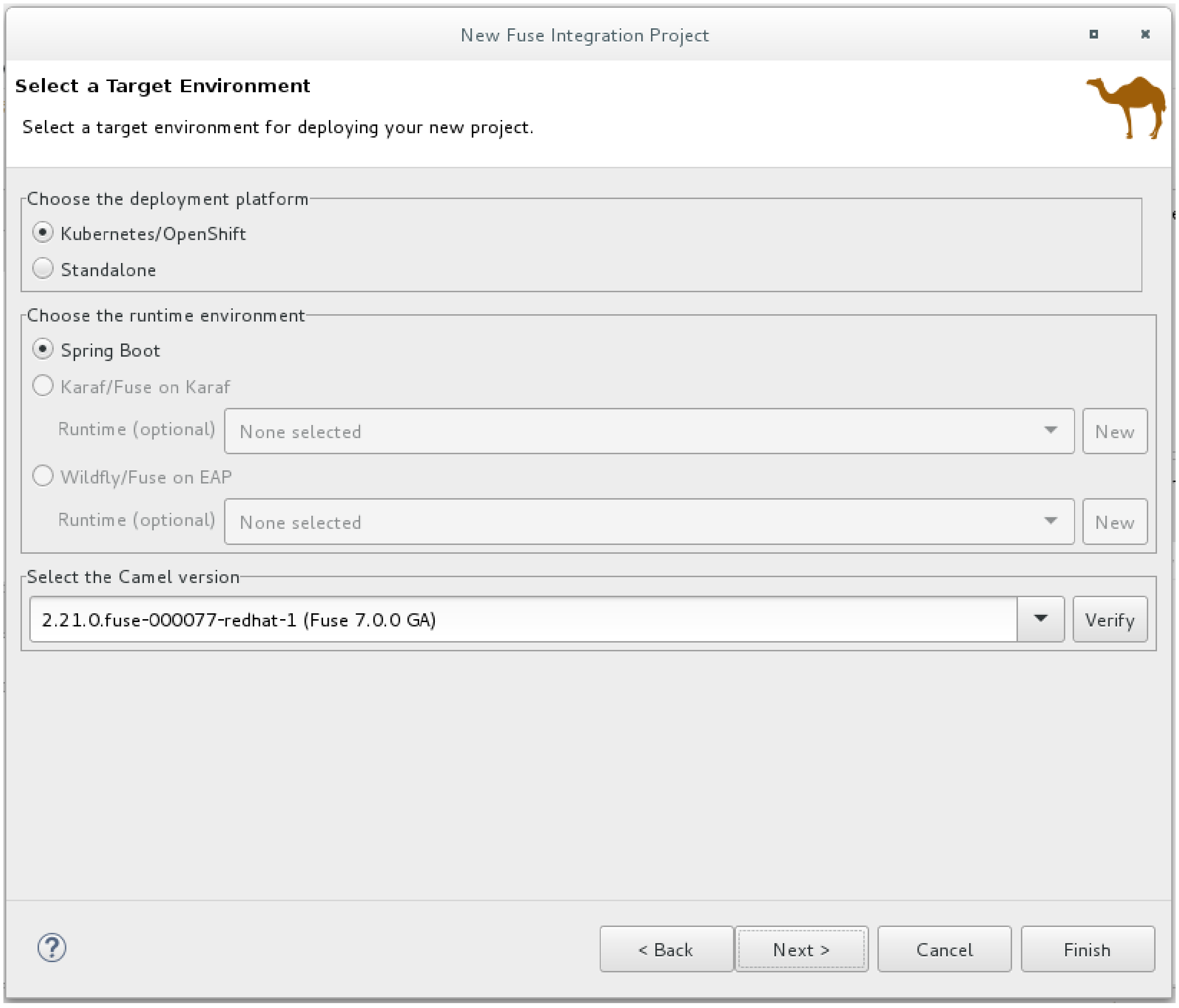
Location

Path

Use default workspace location [✓]

2. In the **Project Name** field, enter a name that is unique to the workspace you are using, for example, **`myFISproject`**.
   Accept the defaults for the other options.

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


New Fuse Integration Project

Select a Target Environment

Select a target environment for deploying your new project.

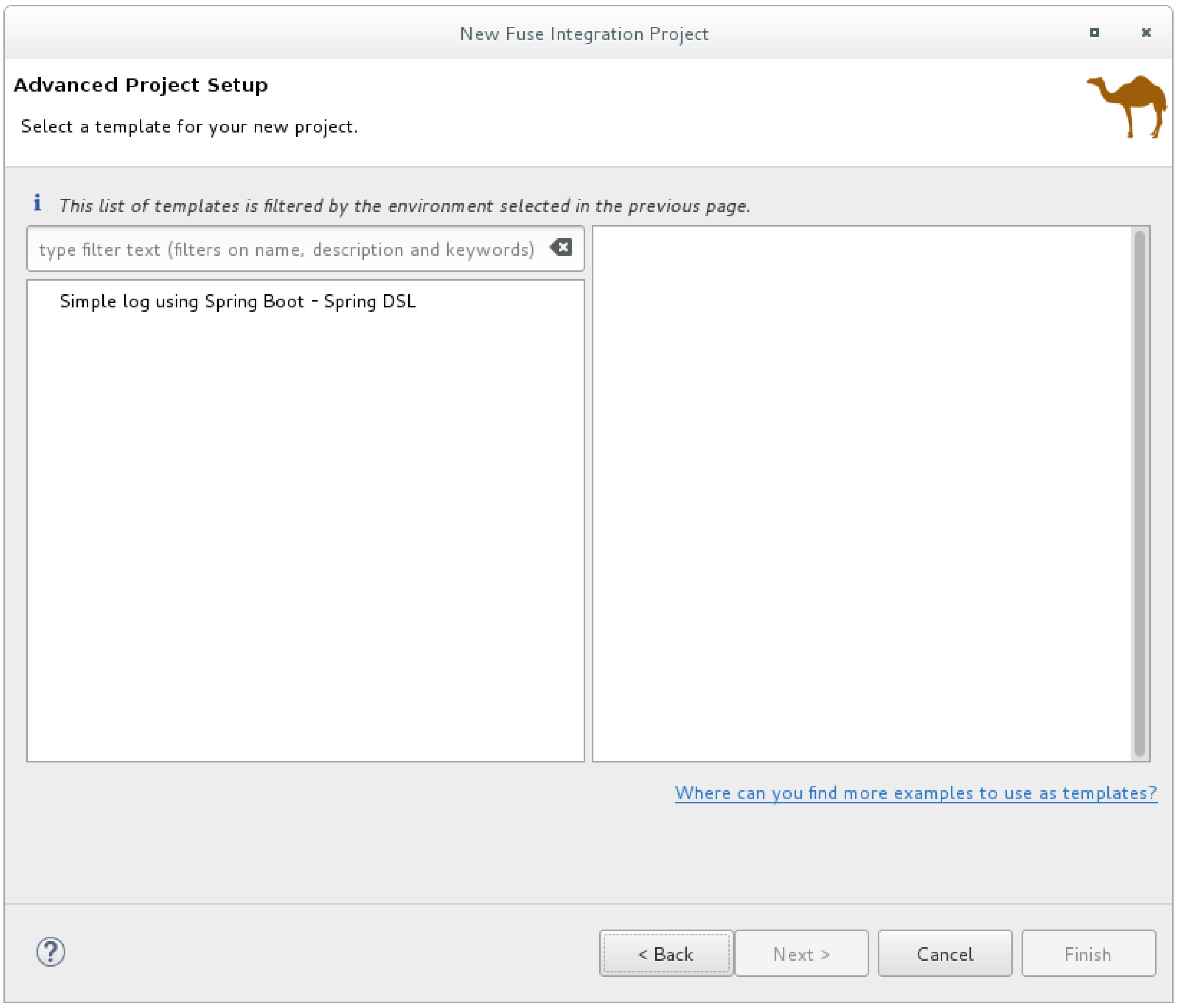
Choose the deployment platform
- [● Kubernetes/OpenShift]
- Standalone

Choose the runtime environment
- [● Spring Boot]
- Karaf/Fuse on Karaf
  - Runtime (optional): None selected
- Wildfly/Fuse on EAP
  - Runtime (optional): None selected

Select the Camel version: 2.21.0.fuse-000077-redhat-1 (Fuse 7.0.0 GA)

Leave the defaults for **Target Runtime** (*No Runtime selected*) and **Camel Version** (*2.18.1.redhat-000021 (FIS 2.0 R3)*).

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



Advanced Project Setup

Select a template for your new project.

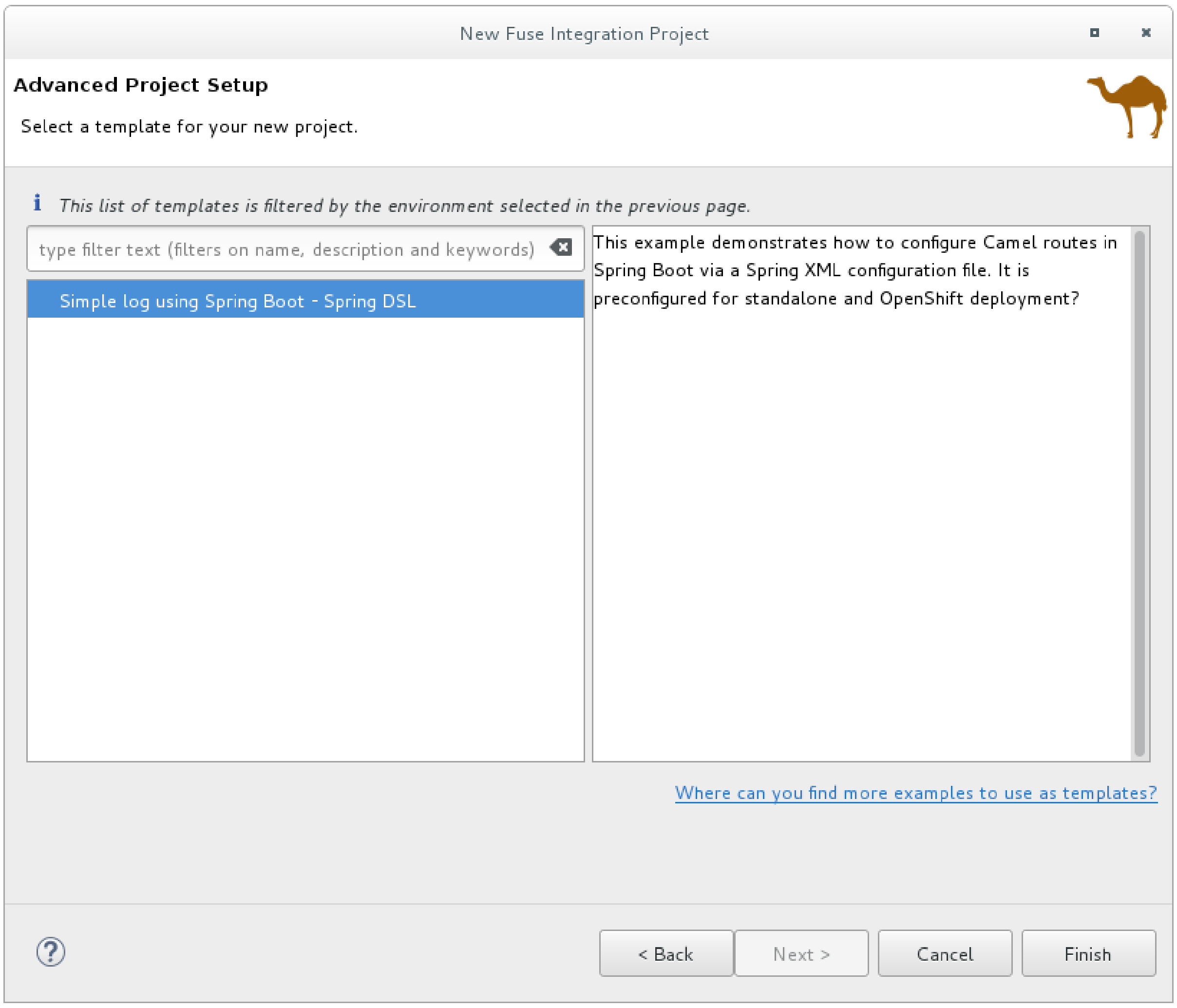
This list of templates is filtered by the environment selected in the previous page.

type filter text (filters on name, description and keywords)

Simple log using Spring Boot - Spring DSL

Where can you find more examples to use as templates?

5. Select the **Simple log using Spring Boot - Spring DSL** template.



Advanced Project Setup

Select a template for your new project.

This list of templates is filtered by the environment selected in the previous page.

type filter text (filters on name, description and keywords)

Simple log using Spring Boot - Spring DSL

This example demonstrates how to configure Camel routes in Spring Boot via a Spring XML configuration file. It is preconfigured for standalone and OpenShift deployment?

Where can you find more examples to use as templates?

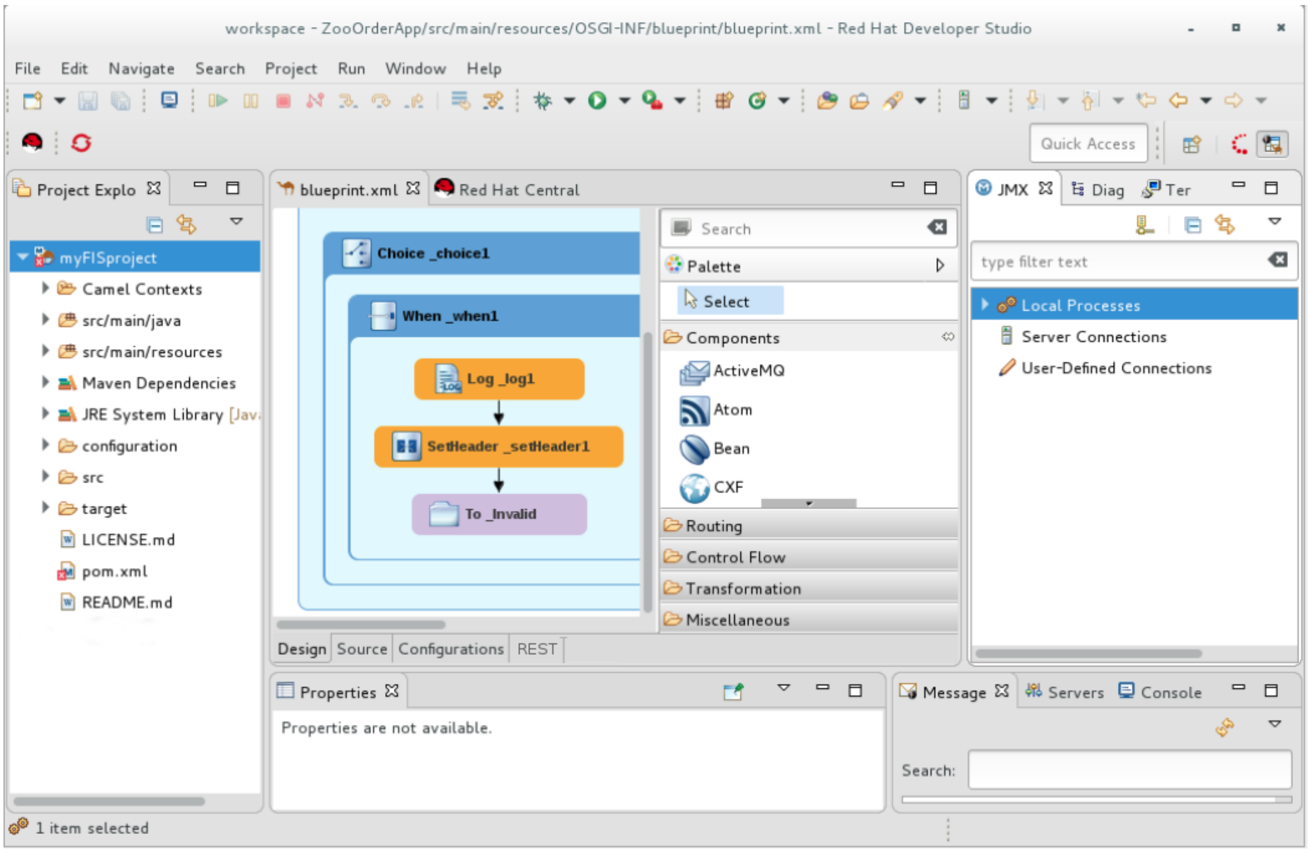
6. Click **Finish**.

NOTE

Because of the number of dependencies that are downloaded for a first-time Fuse FIS project, building it can take some time.

If the **Fuse Integration** perspective is not already open, Developer Studio prompts you to indicate whether you want to open it now. Click **Yes**.

When the build is done the **Fuse Integration** perspective displays the project, for example:

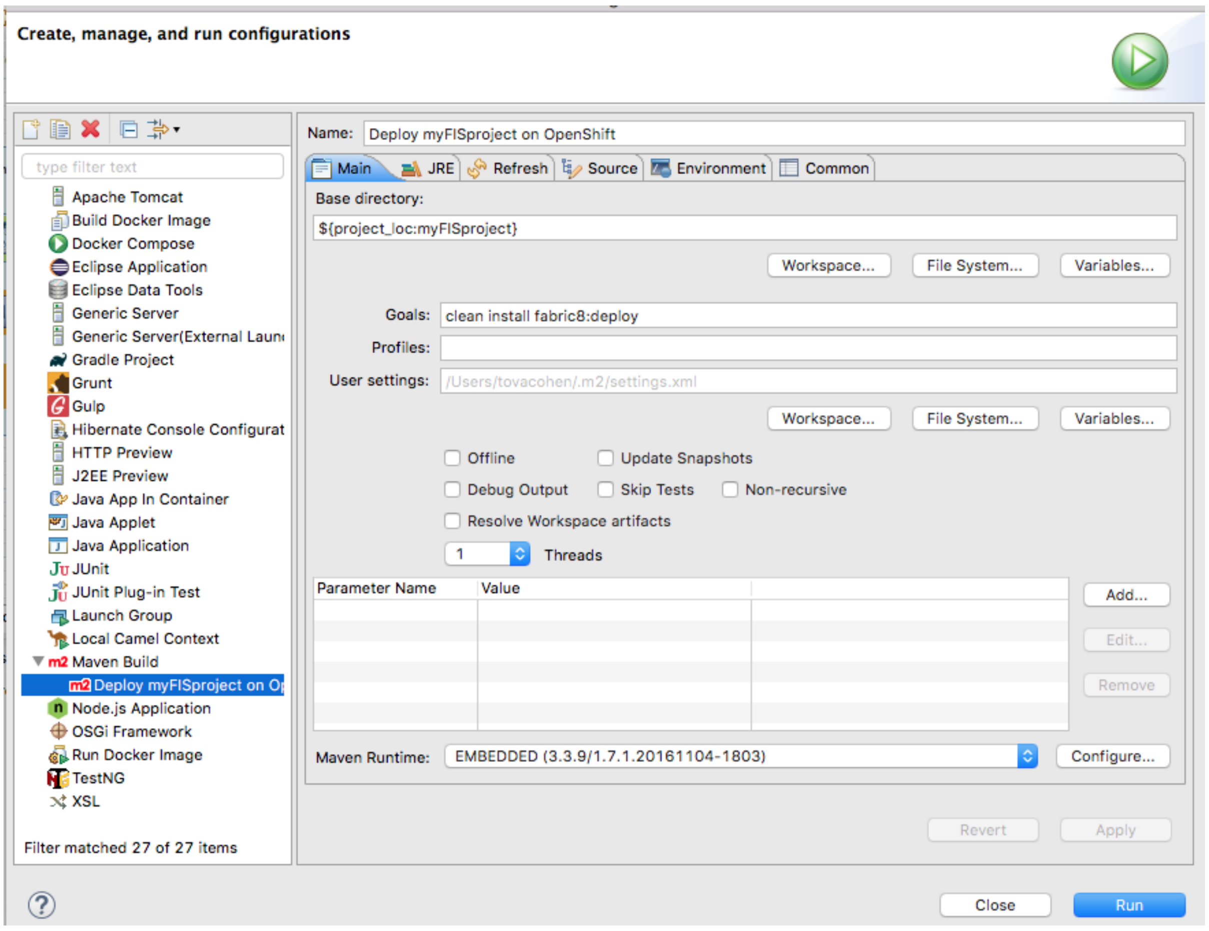


At this point, you can:

- Deploy the project on OpenShift
- Section 6.1, “Running routes as a local Camel context” to verify that the routing context runs successfully on your local machine
  Connecting to the running context in the **JMX Navigator** view (see the section called “Viewing processes in a local JMX server”), you can monitor route components and test whether the route performs as expected:
  - View a route component’s JMX statistics — see Chapter 21, *Viewing a component’s JMX statistics*.
  - Edit the running route — see Chapter 25, *Managing routing endpoints*.
  - Suspend/resume the running route — see Chapter 27, *Managing routing contexts*
  - Start/stop tracing on the running route — see Chapter 23, *Tracing Routes*
- Run the Camel debugger on the project’s **`camel-context.xml`** file to discover and fix logic errors — see Part II, “Debugging Routing Contexts”

DEPLOYING THE FUSE FIS PROJECT TO OPENSHIFT

1. In the **Project Explorer** view, right-click the project’s root (in this example, **`myFISproject`**) to open the context menu.
2. Select **Run As** → **Run Configurations** to open the **Run Configurations** wizard.
3. In the sidebar menu, select **Maven Build** → **Deploy <projectname> on OpenShift** (in this example, **Deploy myFISproject on OpenShift**) to open the project’s default run configuration:


Create, manage, and run configurations

type filter text

- Apache Tomcat
- Build Docker Image
- Docker Compose
- Eclipse Application
- Eclipse Data Tools
- Generic Server
- Generic Server(External Laun
- Gradle Project
- Grunt
- Gulp
- Hibernate Console Configurat
- HTTP Preview
- J2EE Preview
- Java App In Container
- Java Applet
- Java Application
- JUnit
- JUnit Plug-in Test
- Launch Group
- Local Camel Context
- m2 Maven Build
  - Deploy myFISproject on O
- Node.js Application
- OSGi Framework
- Run Docker Image
- TestNG
- XSL

Filter matched 27 of 27 items

Name: Deploy myFISproject on OpenShift

Main | JRE | Refresh | Source | Environment | Common

Base directory: ${project_loc:myFISproject}

Workspace... File System... Variables...

Goals: clean install fabric8:deploy

Profiles:

User settings: /Users/tovacohen/.m2/settings.xml

Workspace... File System... Variables...

- [ ] Offline
- [ ] Update Snapshots
- [ ] Debug Output
- [ ] Skip Tests
- [ ] Non-recursive
- [ ] Resolve Workspace artifacts

1 Threads

Parameter Name	Value

Add... Edit... Remove

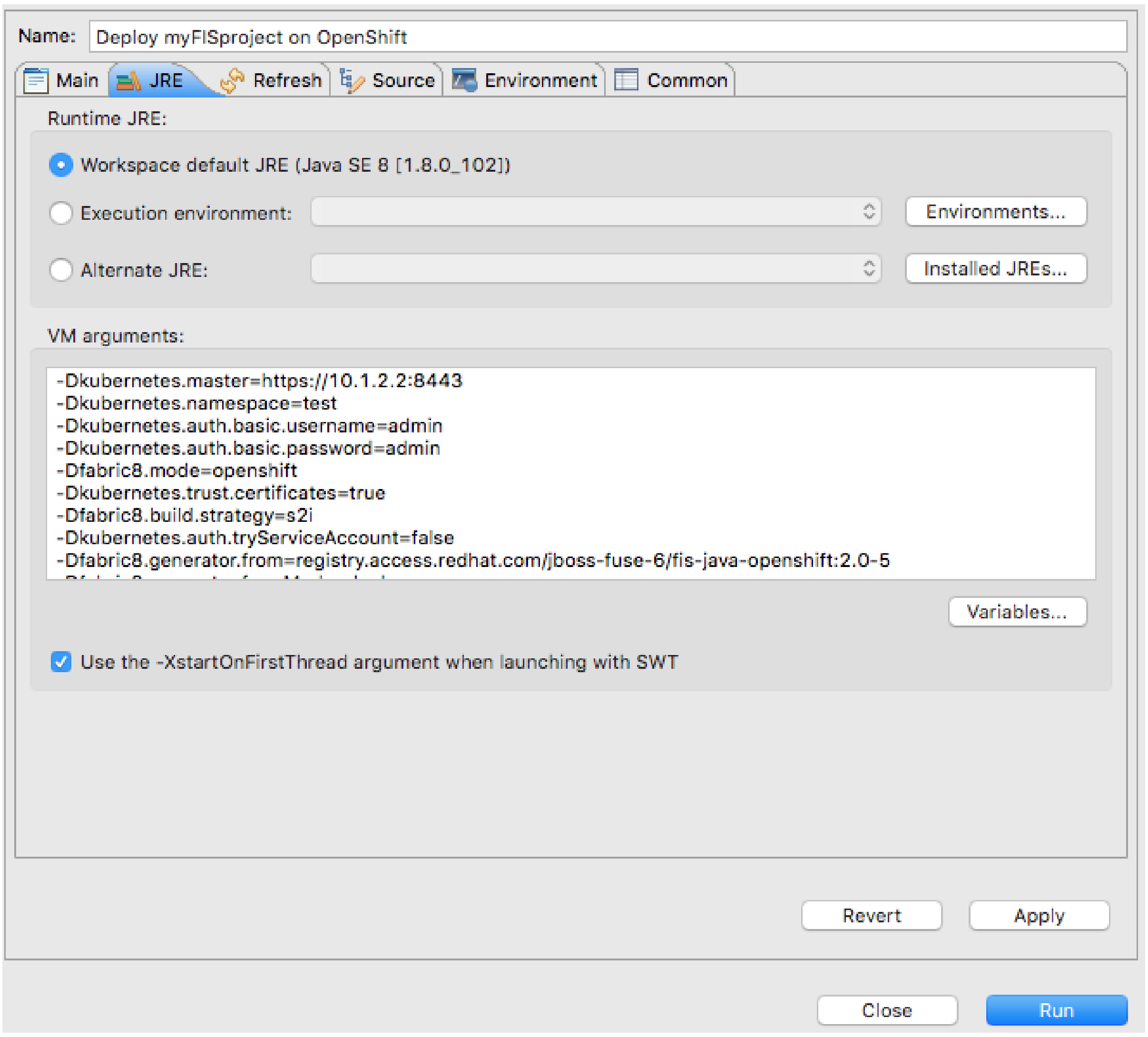
Maven Runtime: EMBEDDED (3.3.9/1.7.1.20161104-1803) Configure...

Revert Apply

Close Run

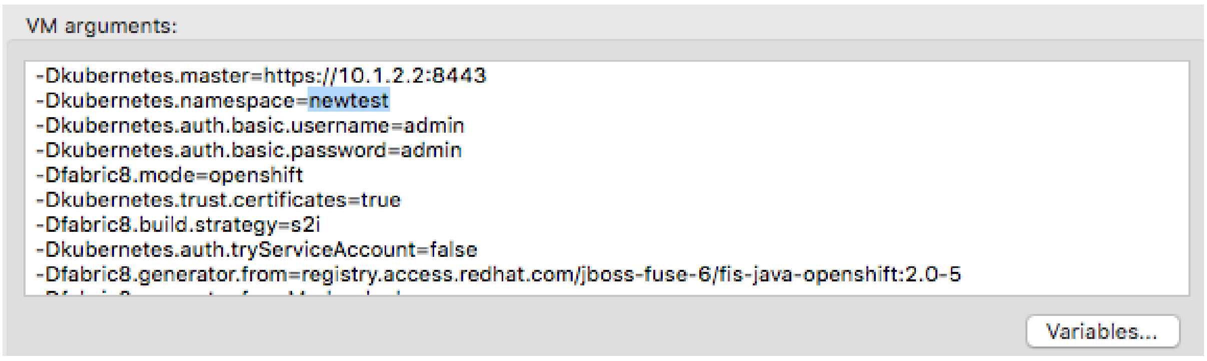
Leave the default settings as they are on the **Main** tab.

4. Open the **JRE** tab to access the VM arguments:



```
-Dkubernetes.master=https://10.1.2.2:8443
-Dkubernetes.namespace=test
-Dkubernetes.auth.basic.username=admin
-Dkubernetes.auth.basic.password=admin
-Dfabric8.mode=openshift
-Dkubernetes.trust.certificates=true
-Dfabric8.build.strategy=s2i
-Dkubernetes.auth.tryServiceAccount=false
-Dfabric8.generator.from=registry.access.redhat.com/jboss-fuse-6/fis-java-openshift:2.0-5
```

5. In the **VM arguments** pane, change the value of the **`-Dkubernetes.namespace=test`** argument to match the **`Project name`** you used for the OpenShift project when you created it (OpenShift project name in the section called “Creating a new OpenShift project”.
In this example, change the default value **`test`** to **`newtest`**:



```
-Dkubernetes.master=https://10.1.2.2:8443
-Dkubernetes.namespace=newtest
-Dkubernetes.auth.basic.username=admin
-Dkubernetes.auth.basic.password=admin
-Dfabric8.mode=openshift
-Dkubernetes.trust.certificates=true
-Dfabric8.build.strategy=s2i
-Dkubernetes.auth.tryServiceAccount=false
-Dfabric8.generator.from=registry.access.redhat.com/jboss-fuse-6/fis-java-openshift:2.0-5
```

Depending on your OpenShift configuration, you may need to modify other`VM arguments to support it:

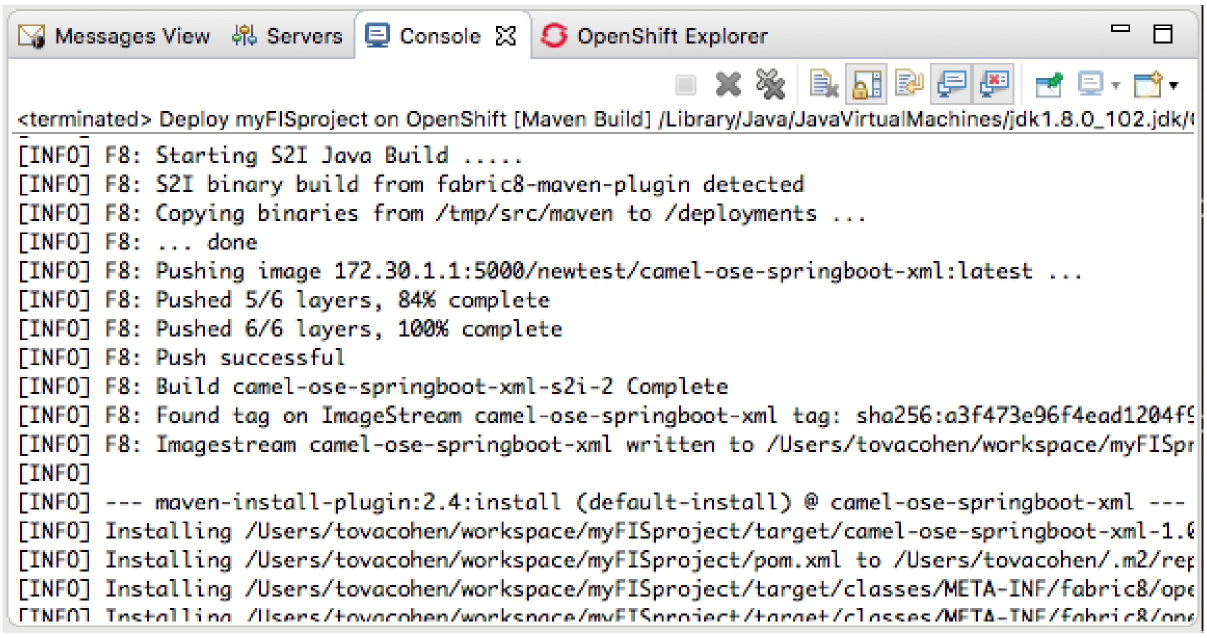
- **`-Dkubernetes.master=https://192.168.99.1:8443`**
When running multiple OpenShift instances or using a remote instance, you need to specify the URL of the OpenShift instance targeted for the deployment. The URL above is an example.

- **`-Dkubernetes.trust.certificates=true`**

- When using the CDK, this argument is required. Leave it set to **`true`**.
- If you are using an OpenShift instance that has a valid SSL certificate, change the value of this argument to **`false`**.

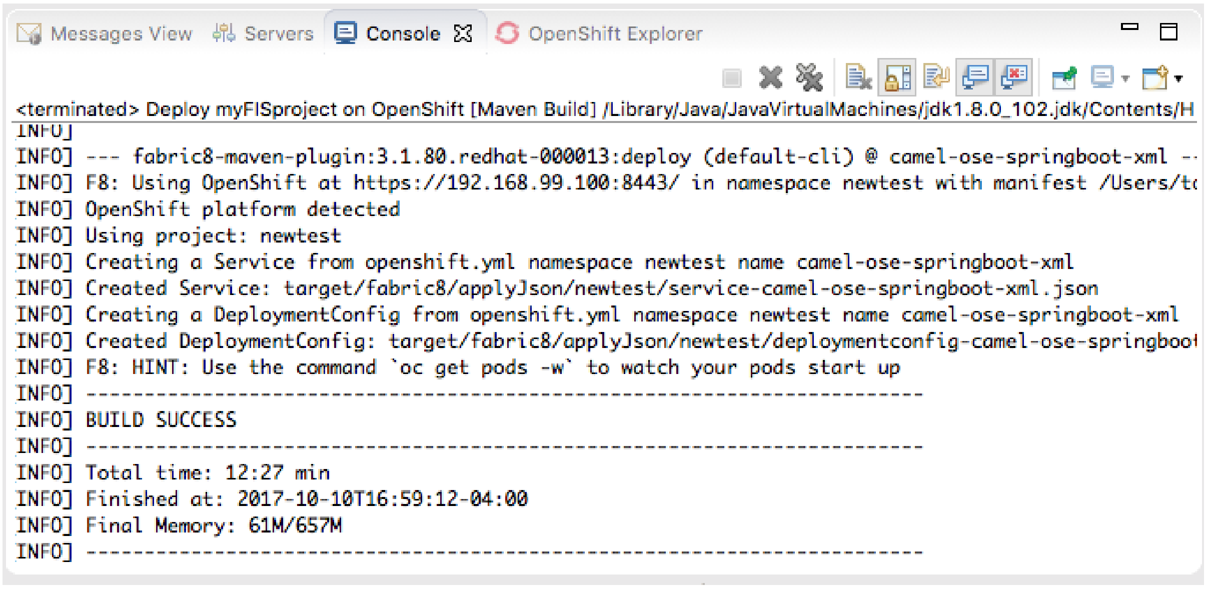
6. Click **Apply** and then click **Run**.
   Because of the number of dependencies to download, first-time deployment can take some time. The speed of your computer and your internet connection are contributing factors. Typically, it takes 25 to 35 minutes to complete a first-time deployment.

   In the **Console** view, you can track the progress of the deploy process. In the following output, the entry *Pushing image 172.30.1 ….. * indicates that the project built successfully and the application images are being pushed to OpenShift, where they will be used to build the Docker container.



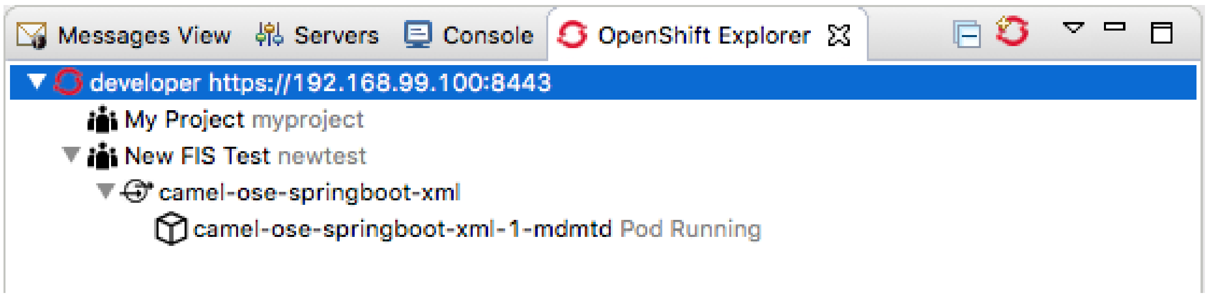
```
<terminated> Deploy myFISproject on OpenShift [Maven Build] /Library/Java/JavaVirtualMachines/jdk1.8.0_102.jdk/
[INFO] F8: Starting S2I Java Build .....
[INFO] F8: S2I binary build from fabric8-maven-plugin detected
[INFO] F8: Copying binaries from /tmp/src/maven to /deployments ...
[INFO] F8: ... done
[INFO] F8: Pushing image 172.30.1.1:5000/newtest/camel-ose-springboot-xml:latest ...
[INFO] F8: Pushed 5/6 layers, 84% complete
[INFO] F8: Pushed 6/6 layers, 100% complete
[INFO] F8: Push successful
[INFO] F8: Build camel-ose-springboot-xml-s2i-2 Complete
[INFO] F8: Found tag on ImageStream camel-ose-springboot-xml tag: sha256:a3f473e96f4ead1204f
[INFO] F8: Imagestream camel-ose-springboot-xml written to /Users/tovacohen/workspace/myFISpr
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ camel-ose-springboot-xml ---
[INFO] Installing /Users/tovacohen/workspace/myFISproject/target/camel-ose-springboot-xml-1.
[INFO] Installing /Users/tovacohen/workspace/myFISproject/pom.xml to /Users/tovacohen/.m2/rep
[INFO] Installing /Users/tovacohen/workspace/myFISproject/target/classes/META-INF/fabric8/ope
[INFO] Installing /Users/tovacohen/workspace/myFISproject/target/classes/META-INF/fabric8/ope
```

The **Console** view displays **BUILD SUCCESS** when deployment completes successfully:



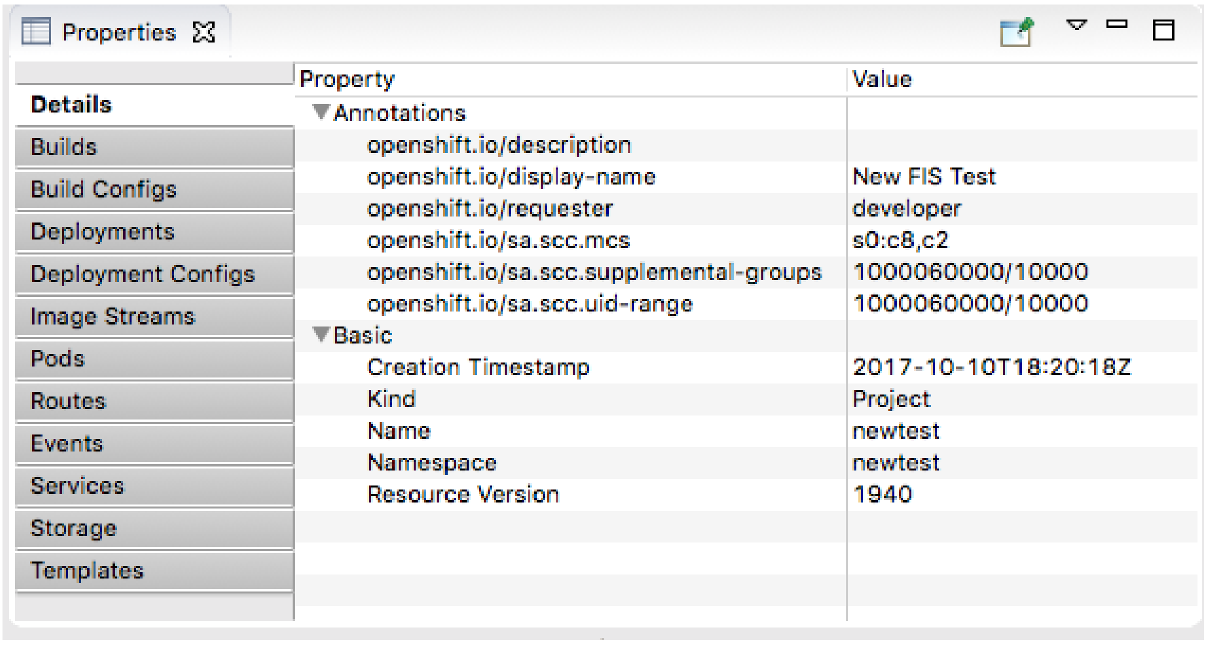
```
<terminated> Deploy myFISproject on OpenShift [Maven Build] /Library/Java/JavaVirtualMachines/jdk1.8.0_102.jdk/Contents/H
INFO]
INFO] --- fabric8-maven-plugin:3.1.80.redhat-000013:deploy (default-cli) @ camel-ose-springboot-xml -
INFO] F8: Using OpenShift at https://192.168.99.100:8443/ in namespace newtest with manifest /Users/t
INFO] OpenShift platform detected
INFO] Using project: newtest
INFO] Creating a Service from openshift.yml namespace newtest name camel-ose-springboot-xml
INFO] Created Service: target/fabric8/applyJson/newtest/service-camel-ose-springboot-xml.json
INFO] Creating a DeploymentConfig from openshift.yml namespace newtest name camel-ose-springboot-xml
INFO] Created DeploymentConfig: target/fabric8/applyJson/newtest/deploymentconfig-camel-ose-springboot
INFO] F8: HINT: Use the command `oc get pods -w` to watch your pods start up
INFO] ------------------------------------------------------------------------
INFO] BUILD SUCCESS
INFO] ------------------------------------------------------------------------
INFO] Total time: 12:27 min
INFO] Finished at: 2017-10-10T16:59:12-04:00
INFO] Final Memory: 61M/657M
INFO] ------------------------------------------------------------------------
```

7. Switch to the **OpenShift Explorer** view and select **New FIS Test newtest**:



```
▼ developer https://192.168.99.100:8443
      My Project myproject
    ▼ New FIS Test newtest
      ▼ camel-ose-springboot-xml
          camel-ose-springboot-xml-1-mdmtd Pod Running
```

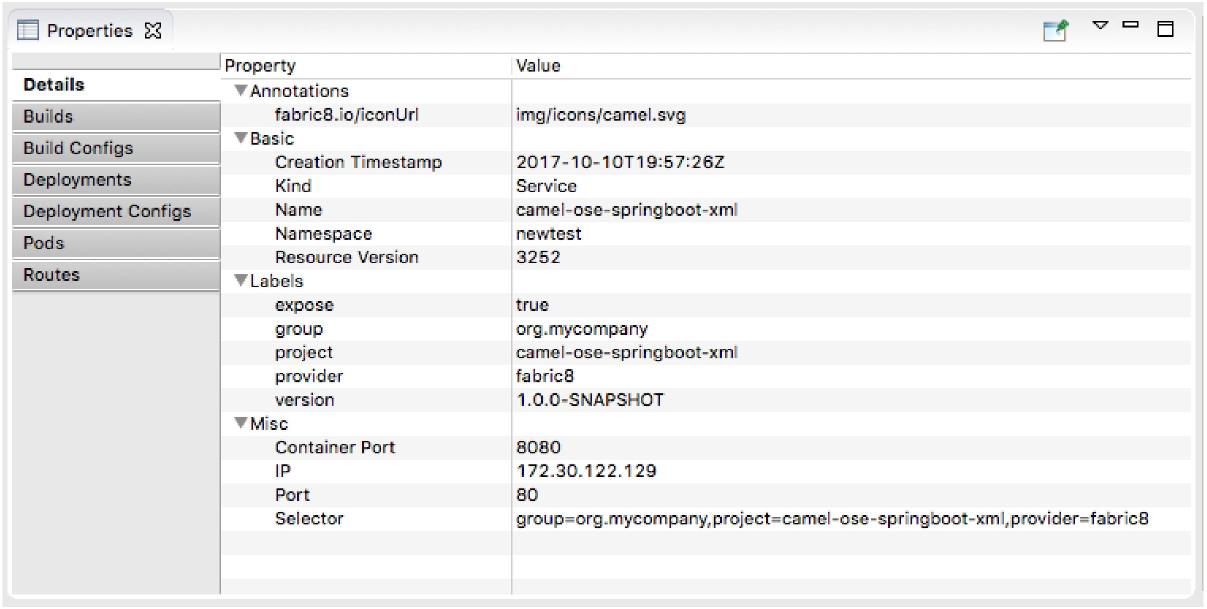
In the **Properties** view, the **Details** page displays all of the project's property values.



Property	Value
▼Annotations	
openshift.io/description	
openshift.io/display-name	New FIS Test
openshift.io/requester	developer
openshift.io/sa.scc.mcs	s0:c8,c2
openshift.io/sa.scc.supplemental-groups	1000060000/10000
openshift.io/sa.scc.uid-range	1000060000/10000
▼Basic	
Creation Timestamp	2017-10-10T18:20:18Z
Kind	Project
Name	newtest
Namespace	newtest
Resource Version	1940

Open the other tabs (**Builds**, **Build Configs**, **Deployments**,…) to view other properties of the project. The **Properties** view provides the same information as the OpenShift Web Console.

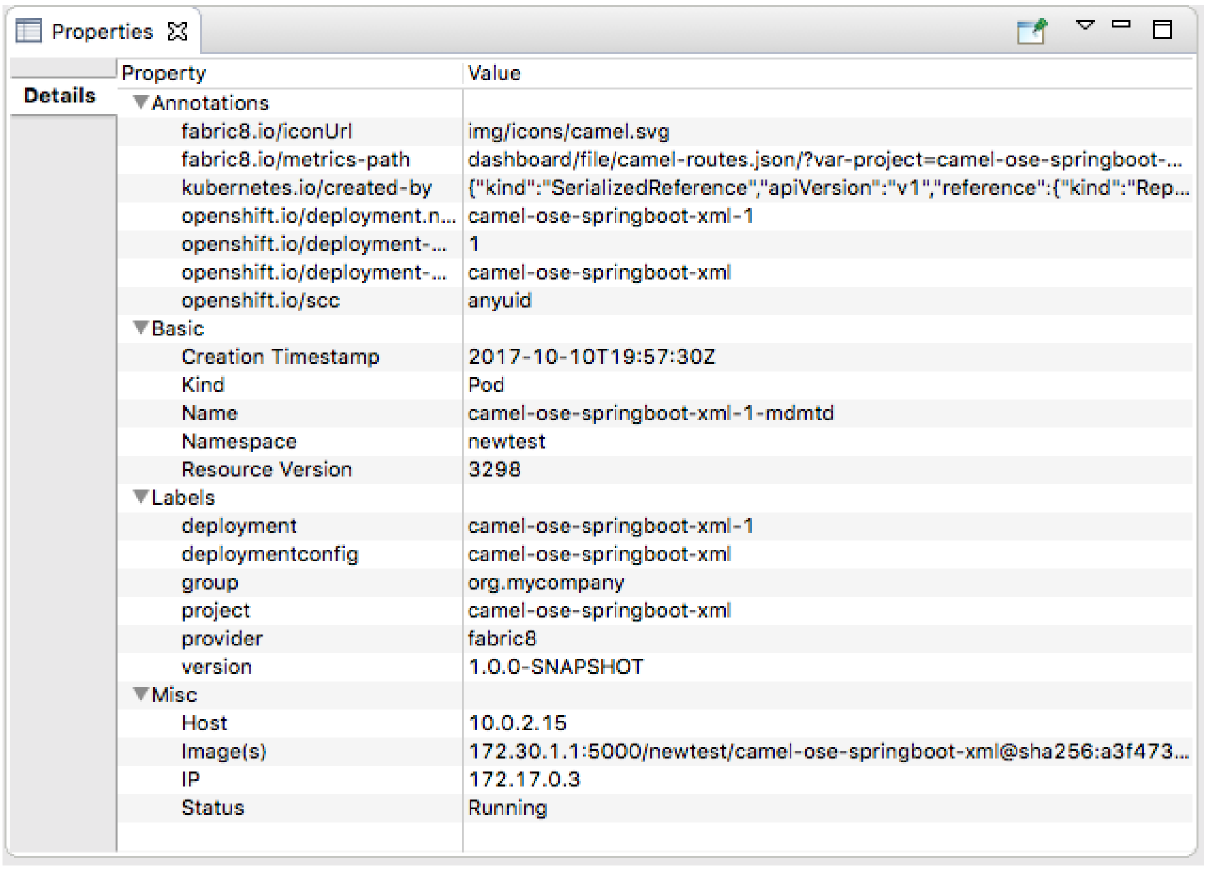
8. In the **OpenShift Explorer** view, select **`camel-ose-springboot-xml`** to view its details in the **Properties** view:



Property	Value
▼Annotations	
fabric8.io/iconUrl	img/icons/camel.svg
▼Basic	
Creation Timestamp	2017-10-10T19:57:26Z
Kind	Service
Name	camel-ose-springboot-xml
Namespace	newtest
Resource Version	3252
▼Labels	
expose	true
group	org.mycompany
project	camel-ose-springboot-xml
provider	fabric8
version	1.0.0-SNAPSHOT
▼Misc	
Container Port	8080
IP	172.30.122.129
Port	80
Selector	group=org.mycompany,project=camel-ose-springboot-xml,provider=fabric8

Scroll through the other tabs to view other properties of the deployment configuration.

9. In the **OpenShift Explorer** view, select **`camel-ose-springboot-xml-1-mdmtd Pod Running`**, and then view the details of the running instance in the **Properties** view:



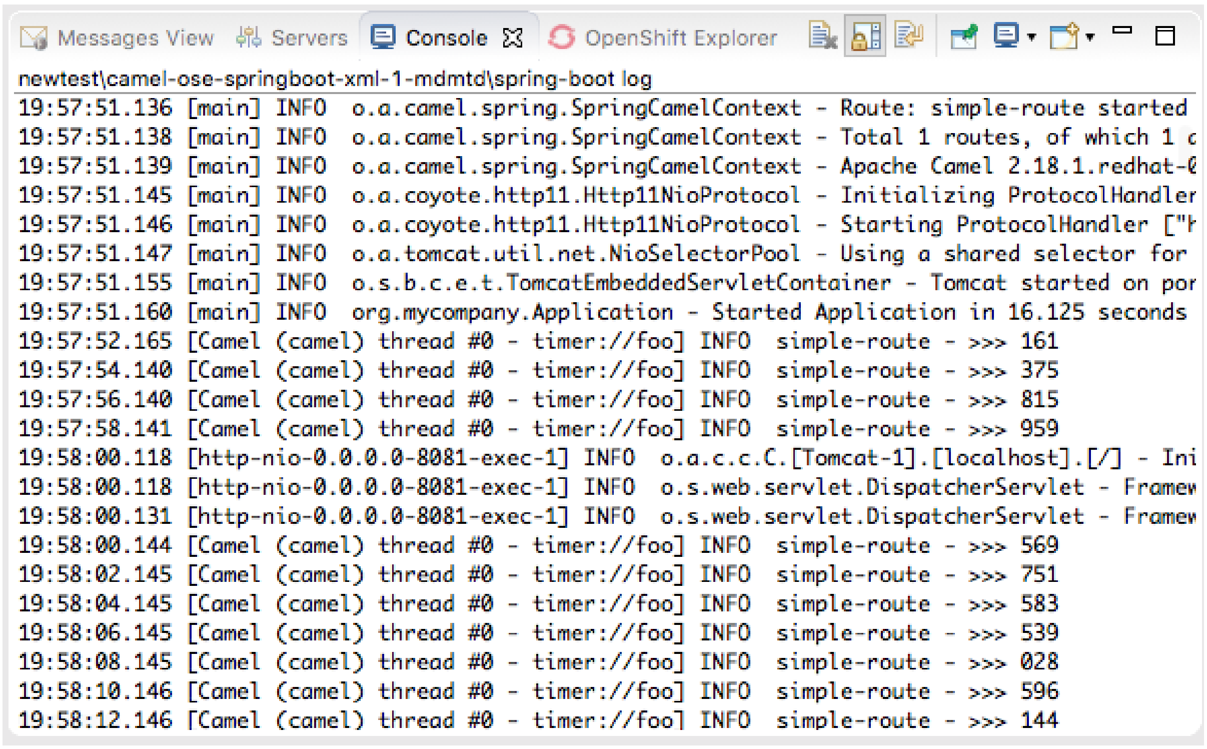
Property	Value
▼Annotations	
fabric8.io/iconUrl	img/icons/camel.svg
fabric8.io/metrics-path	dashboard/file/camel-routes.json/?var-project=camel-ose-springboot-...
kubernetes.io/created-by	{"kind":"SerializedReference","apiVersion":"v1","reference":{"kind":"Rep...
openshift.io/deployment.n...	camel-ose-springboot-xml-1
openshift.io/deployment-...	1
openshift.io/deployment-...	camel-ose-springboot-xml
openshift.io/scc	anyuid
▼Basic	
Creation Timestamp	2017-10-10T19:57:30Z
Kind	Pod
Name	camel-ose-springboot-xml-1-mdmtd
Namespace	newtest
Resource Version	3298
▼Labels	
deployment	camel-ose-springboot-xml-1
deploymentconfig	camel-ose-springboot-xml
group	org.mycompany
project	camel-ose-springboot-xml
provider	fabric8
version	1.0.0-SNAPSHOT
▼Misc	
Host	10.0.2.15
Image(s)	172.30.1.1:5000/newtest/camel-ose-springboot-xml@sha256:a3f473...
IP	172.17.0.3
Status	Running

10. In the **OpenShift Explorer** view, right-click **`camel-ose-springboot-xml-1-mdmtd Pod Running`**, and then select **Pod Logs...**.

NOTE

If prompted, enter the path to the installed **oc** executable. It is required to retrieve pod logs.

The **Console** view automatically opens, displaying the logs from the running pod:



```
newtest\camel-ose-springboot-xml-1-mdmtd\spring-boot log
19:57:51.136 [main] INFO  o.a.camel.spring.SpringCamelContext - Route: simple-route started
19:57:51.138 [main] INFO  o.a.camel.spring.SpringCamelContext - Total 1 routes, of which 1 c
19:57:51.139 [main] INFO  o.a.camel.spring.SpringCamelContext - Apache Camel 2.18.1.redhat-0
19:57:51.145 [main] INFO  o.a.coyote.http11.Http11NioProtocol - Initializing ProtocolHandler
19:57:51.146 [main] INFO  o.a.coyote.http11.Http11NioProtocol - Starting ProtocolHandler ["h
19:57:51.147 [main] INFO  o.a.tomcat.util.net.NioSelectorPool - Using a shared selector for
19:57:51.155 [main] INFO  o.s.b.c.e.t.TomcatEmbeddedServletContainer - Tomcat started on por
19:57:51.160 [main] INFO  org.mycompany.Application - Started Application in 16.125 seconds
19:57:52.165 [Camel (camel) thread #0 - timer://foo] INFO  simple-route - >>> 161
19:57:54.140 [Camel (camel) thread #0 - timer://foo] INFO  simple-route - >>> 375
19:57:56.140 [Camel (camel) thread #0 - timer://foo] INFO  simple-route - >>> 815
19:57:58.141 [Camel (camel) thread #0 - timer://foo] INFO  simple-route - >>> 959
19:58:00.118 [http-nio-0.0.0.0-8081-exec-1] INFO  o.a.c.c.C.[Tomcat-1].[localhost].[/] - Ini
19:58:00.118 [http-nio-0.0.0.0-8081-exec-1] INFO  o.s.web.servlet.DispatcherServlet - Framew
19:58:00.131 [http-nio-0.0.0.0-8081-exec-1] INFO  o.s.web.servlet.DispatcherServlet - Framew
19:58:00.144 [Camel (camel) thread #0 - timer://foo] INFO  simple-route - >>> 569
19:58:02.145 [Camel (camel) thread #0 - timer://foo] INFO  simple-route - >>> 751
19:58:04.145 [Camel (camel) thread #0 - timer://foo] INFO  simple-route - >>> 583
19:58:06.145 [Camel (camel) thread #0 - timer://foo] INFO  simple-route - >>> 539
19:58:08.145 [Camel (camel) thread #0 - timer://foo] INFO  simple-route - >>> 028
19:58:10.146 [Camel (camel) thread #0 - timer://foo] INFO  simple-route - >>> 596
19:58:12.146 [Camel (camel) thread #0 - timer://foo] INFO  simple-route - >>> 144
```

Click ✖ in the **Console** view's menu bar to terminate the session and clear console output.

ACCESSING THE OPENSHIFT WEB CONSOLE

NOTE

This information applies to Red Hat Container Development Kit installations only.

To access the OpenShift Web Console, open a browser and enter the OpenShift server's URL, which is specific to your instance and your machine. For example, enter `https://192.168.99.100:8443`, in the browser's address field.

You can log into the web console either as a developer or as an administrator, using the default credentials:

- Default developer role
  Developer users can view only their own projects and the supplied OpenShift sample project, which demonstrates OpenShift v3 features. Developer users can create, edit and delete any project that they own that is deployed on OpenShift.
  - **Username`developer`**
  - **Password`developer`**
- Default administrator role
  An administrator user can view and access all projects on OpenShift (CDK). Administrator users can create, edit and delete, any project deployed on OpenShift.
  - **Username`admin`**
  - **Password`admin`**

For more information on using the OpenShift web console, see NameOfCDKGettingStarted}.

CHAPTER 8. USING THE RED HAT FUSE SAP TOOL SUITE

The Red Hat Fuse SAP Tool Suite makes it possible to integrate your Camel routes with a remote SAP Application Server. A variety of SAP components are provided to support Remote Function Calls (RFC) and the sending and receiving of Intermediate Documents (IDocs). The SAP Tool Suite depends on the JCo and IDoc client libraries from SAP. To install and use these libraries, you must have an SAP Service Marketplace Account.

8.1. INSTALLING THE RED HAT FUSE SAP TOOL SUITE

Overview

The Red Hat Fuse SAP Tool Suite provides the Edit SAP Connection Configuration dialog, which helps you to create and manage the SAP Application Server and Destination connections. The suite is not installed by default, because it requires third-party JCo and IDoc client libraries, which are licensed separately by SAP.

Platform restrictions for SAP tooling

Because the SAP tool suite depends on the third-party JCo 3.0 and IDoc 3.0 libraries, it can only be installed on the platforms that these libraries support. For more details about the platform restrictions for SAP tooling, see Red Hat Fuse Supported Configurations.

Prerequisites

Before you can install the Fuse SAP Tool Suite, you must download the JCo and IDoc libraries from the following location:

- http://service.sap.com/connectors

To download these libraries, you must have an SAP Service Marketplace Account. Be sure to choose the appropriate JCo and IDoc libraries for your operationg system. Also:

- Only version 3.0.11 or greater of the JCo library is supported.
- Only version 3.0.10 or greater of the IDoc library is supported.

For this installation procedure, you can leave the downloaded files in archive format. There is no need to extract the contents.

Procedure

To install the Fuse SAP Tool Suite into Red Hat Developer Studio, perform the following steps:

1. In Red Hat Developer Studio, select **File** → **Import** to open the **Import** wizard.
2. In the **Select** screen of the **Import** wizard, select **Fuse** → **Install Fuse SAP Tool Suite**, and then click **Next**.
3. The **Install the Red Hat Fuse SAP Tool Suite** screen opens, which displays the instructions for downloading the JCo and IDoc libraries from the SAP Service Marketplace. Click **Next**.
4. The **Select JCo3 and IDoc3 Archive to Import** screen opens. Next to the **JCo Archive File** field, use the **Browse** button to select the JCo archive that you downloaded from the SAP

Service Marketplace. After selecting the JCo archive, the **Archive Version** and **Archive OS Platform** fields are automatically filled in, so that you can check whether the library you are installing has the correct version and OS platform.
Next to the IDoc3 Archive File field, use the **Browse** button to select the IDoc archive that you downloaded from the SAP Service Marketplace.

After selecting both archive files, click **Finish**.

5. A new **Install** wizard (for installing Eclipse plug-ins) opens automatically. This wizard displays the following to plug-ins to be installed:
   - Fuse SAP Tool Suite
   - SAP JCo3 and IDoc3 Libraries
     Make sure that both of these plug-ins are selected. Click **Next**.

   NOTE

   The **`SAP JCo3 and IDoc3 Libraries`** plug-in is dynamically constructed from the selected JCo and IDoc libraries.

6. The **Install Details** screen allows you to review the plug-ins to be installed. Click **Next**.
7. The **Review Licenses** dialog opens. Select the **I accept** radiobutton option, and then click **Finish**.
8. If you encounter a **Security Warning** dialog (warning of unsigned content), click **OK** to ignore the warning and continue installing.
9. The **Restart Eclipse** dialog opens. Click **OK** to restart Eclipse.

8.2. CREATE AND TEST SAP DESTINATION CONNECTION

Overview

In Fuse SAP Tool suite, the Edit SAP Connection Configuration dialog helps you to create and manage the SAP Application Destination connections. This section describes how to create and test the SAP destination connection.

Procedure

To create and test an SAP destination connection, perform the following steps:

1. Navigate to the global **Configurations** tab of the route editor and click **Add**.
   The **Create new global element** view appears.
2. Under **SAP**, select the type of connection you would like to create. Choose the **SAP Connection** and click **Ok**.
   The **Edit SAP Connection Configuration** dialog appears. It allows you to create, edit and delete the Destination and Server Connection Configurations.
3. To create a new **Destination Data Store**, click the **Add Destination** tab.
   The **Create Destination** dialog appears.

4. Enter a name for the destination in the **Destination Name** field and click **Ok**.

5. In the **Properties** dialog,

    a. Click the **Basic** tab to configure the basic properties required to connect to an SAP destination. In this tab, fill in the following property fields to configure the connection:

    - **SAP Application Server**
    - **SAP System Number**
    - **SAP Client**
    - **Logon User**
    - **Logon Password**
    - **Logon Language**

    b. Click the **Connection** tab to add values required to connect to an SAP destination. Fill in the following property fields to configure the connection:

    - **SAP System Number**
    - **SAP Router String**
    - **SAP Application Server**
    - **SAP Message Server**
    - **SAP Message Server Port**
    - **Gateway Host**
    - **Gateway Port**
    - **SAP System ID**
    - **SAP Application Server Group**

    c. Click the **Authenticate** tab to add values required to validate an SAP destination. Fill in the following property fields to configure the connection.

    - **SAP Authentication type**
    - **SAP Client**
    - **Logon User**
    - **Logon User Alias**
    - **Logon Password**
    - **SAP SSO Logon Ticket**
    - **SAP X509 Login Ticket**
    - **Logon Language**

d. Click the **Special** tab. In this tab, fill in the following property fields to configure the connection:

   - **Select CPIC Trace**
   - **Initial Codepage**

e. Click the **Pool** tab and fill in the following property fields to configure the connection:

   - **Connection Pool Peak Limit**
   - **Connection Pool Capacity**
   - **Connection Pool Expiration Time**
   - **Connection Pool Expire Check Period**
   - **Connection Pool Max Get Client Time**

f. Click the **SNC** tab and fill in the following property fields to configure the connection:

   - **SNC Partner Name**
   - **SNC Level of Security**
   - **SNC Name**
   - **SNC Library Path**

g. Click the **Repository** tab and fill in the following property fields to configure the connection:

   - **Repository Destination**
   - **Repository Logon User**
   - **Repository Logon Password**

   NOTE

   If you need more information about these settings, refer the SAP documentation.

6. You are now ready to test the destination connection. In the **Edit SAP Connection Configuration** dialog, right-click on the destination name and select**Test**.
   The **Test Destination Connection** dialog opens.

7. The dialog uses the current destination configuration settings to connect to the **SAP Destination Data Store**. If the test is successful, you will see the following message in the status area:

```
Connection test for destination 'YourDestination' succeeded.
```

   Otherwise, an error report appears in the status area.

8. Click **Close** to close the **Test Destination Connection** dialog.

9. Click **Finish**. The newly created **SAP Destination Connection** appears under SAP.

8.3. CREATE AND TEST SAP SERVER CONNECTION

Overview

In Fuse SAP Tool suite, the Edit SAP Connection Configuration dialog helps you to create and manage the SAP Application Server connections. This section describes how to create and test the SAP Server connection.

Procedure

To create and test the SAP Server connection, perform the following steps:

1. Navigate to the global **Configurations** tab of route editor and click **Add**.
   The **Create new global element** view appears.

2. Under **SAP**, select the type of connection you would like to create. Choose the **SAP Connection** and click **Ok**.
   The **Edit SAP Connection Configuration** dialog appears. It allows you to create, edit and delete the Destination and Server Connection Configurations.

3. To create a new **Server Data Store**, click the **Add Server** tab.
   The **Create Server** dialog appears.

4. Enter a name for the Server in the **Server Name** field and click **Ok**.

5. In the **Properties** dialog,

   a. Click the **Mandatory** tab to configure the basic properties required to connect to an SAP server. In this tab, fill in the following property fields to configure the connection:

      - **Gateway Host**
      - **Gateway Port**
      - **Program ID**
      - **Repository Destination**
      - **Connection Count**

   b. Click the **Optional** tab and fill in the following property fields to configure the connection:

      - **SAP Router String**
      - **Worker Thread Count**
      - **Minimum Worker Thread Count**
      - **Maximum Startup Delay**
      - **Repository Map**

   c. Click the **SNC** tab and fill in the following property fields to configure the connection.

      - **SNC Level of Security**
      - **SNC Name**

- **SNC Library Path**

> **NOTE**
>
> For more information about the settings, refer the SAP documentation.

6. You are now ready to test the server connection. In the **Edit SAP Connection Configuration** dialog, right-click on the server name and select **Test**.
   The **Test Server Connection** dialog opens.

7. The dialog uses the current server configuration settings to connect to the **SAP Server Data Store**. If the test is successful, you will see the following message in the status area:

   ```
   Server state: STARTED
   Server state: ALIVE
   ```

   If the test fails, the server status is reported as **DEAD**.

8. Click **Stop** to shut down the Test Sever.

9. Click **Close** to close the **Test Server Connection** dialog.

10. Click **Finish**. The newly created **SAP Server Connection** appears under SAP.

8.4. DELETING DESTINATION AND SERVER CONNECTIONS

Overview

This following section describes how to delete the SAP Destination and Server connections in the Edit SAP Connection Configuration dialog.

Procedure

If you want to delete the Destination and Server connections, perform the following steps:

1. Navigate to the global **Configurations** tab of route editor and click **Add**.
   The **Create new global element** view appears.

2. Under **SAP**, select the **SAP Connection** and click **Ok**.
   The **Edit SAP Connection Configuration** dialog appears. It allows you to create, edit and delete the Destination and Server Connection Configurations.

3. In the **Edit SAP Connection Configuration** dialog, select the Destination and Server Data Stores which you want to delete.

4. Click **Delete**. It will delete the selected connections.
   Atlast, click **Finish**. It will save all the changes.

8.5. CREATE A NEW SAP ENDPOINT

Overview

You can use the Components palette in the route editor to add SAP components to a route with the help of the Edit SAP Connection Configuration dialog.

NOTE

If you are using the SAP Connection view, remember to paste the requisite SAP connection configuration data into your Blueprint XML or Spring XML code.

Prerequisites

You must already have created some SAP destination connections and/or server connections with the help of the Edit SAP Connection Configuration dialog.

NOTE

If you are using the SAP Connection view, export this configuration to a file of the appropriate type (Blueprint XML or Spring XML).

Procedure

To create a new SAP endpoint, perform the following steps:

1. It is assumed that you already have a Fuse project and a Camel XML file to work with (which could either be in Blueprint XML or Spring XML format).
2. Open your Camel XML file in the route editor. If you have already installed the Red Hat Fuse SAP Tool Suite, you should be able to see the SAP components under the **Components** palette in the route editor. The following SAP components are provided by the tool suite:
   - **SAP IDoc Destination**
   - **SAP IDoc List Destination**
   - **SAP IDoc List Server**
   - **SAP qRFC Destination**
   - **SAP Queued IDoc Destination**
   - **SAP Queued IDoc List Destination**
   - **SAP sRFC Destination**
   - **SAP sRFC Server**
   - **SAP tRFC Destination**
   - **SAP tRFC Server**
     In the **Design** tab of the route editor, drag one of these components onto the canvas to create a new SAP endpoint in the current `camelContext`.

     NOTE

     The SAP Netweaver component does not belong to the Red Hat Fuse SAP Tool Suite. It is hosted in the Apache Camel project.

3. Switch to the **Source** tab of the route editor, by clicking the **Source** tab at the bottom of the canvas. You can see the XML source of the routes.

4. When specifying an SAP endpoint URI, you must embed either a destination name or a server connection name in the URI format. For example, the **`sap-srfc-destination`** component has the following URI format:

   ```
   sap-srfc-destination:destinationName:rfcName
   ```

   To reference a particular destination, use the value of the relevant **`entry`** element’s **`key`** attribute as the **`destinationName`** in this URI.

CHAPTER 9. GETTING STARTED WITH DATA TRANSFORMATION

9.1. FUSE TRANSFORMATION TOOLING

One of the challenges that comes with system and data integration is that the component systems often work with different data formats. You cannot simply send messages from one system to another without translating it into a format (or language) recognized by the receiving system. Data transformation is the term given to this translation.

9.2. DATA TRANSFORMATION TUTORIAL

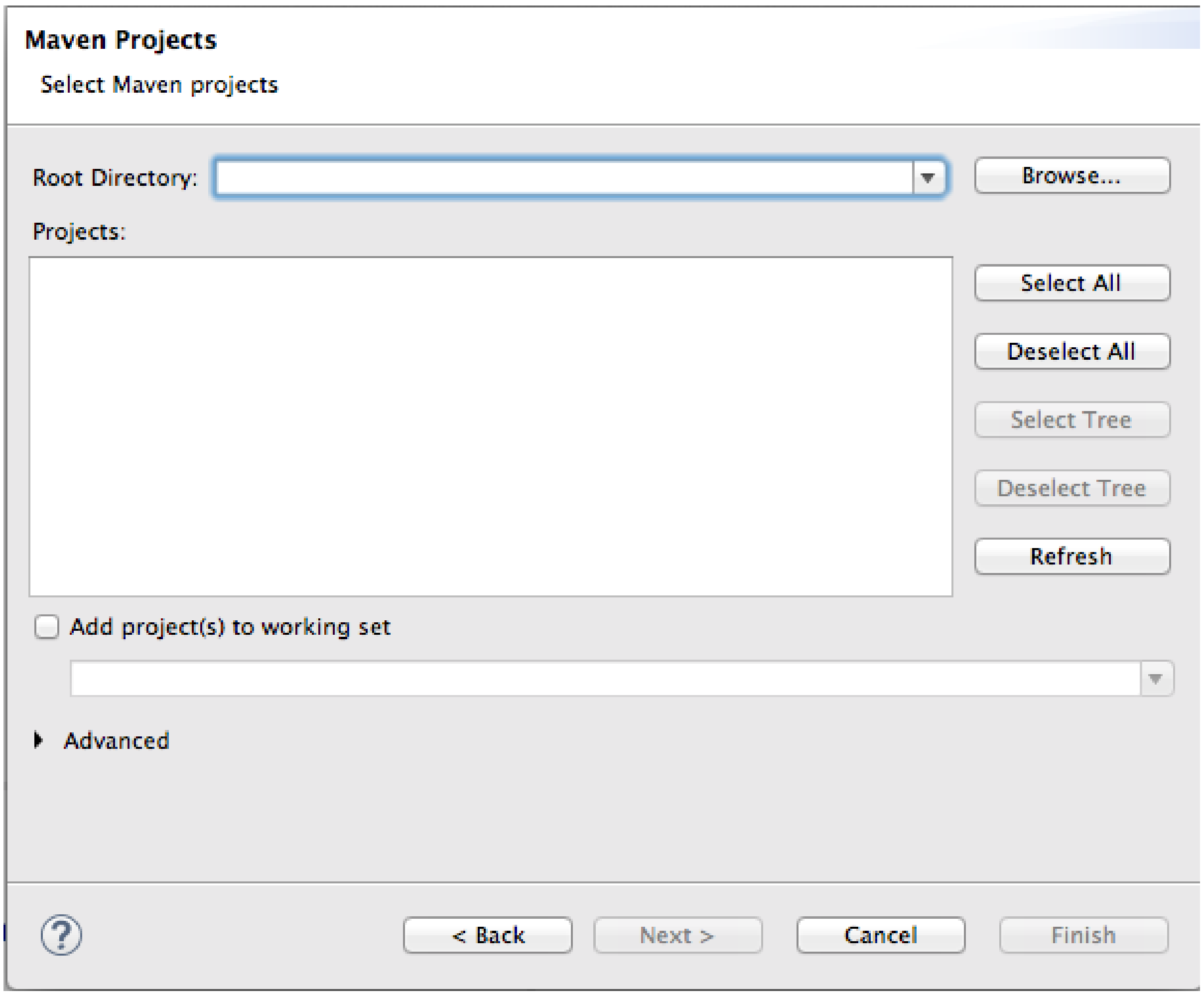
In this tutorial you will learn how to use the data transformation tooling to include data transformation in a predefined Camel route. The Camel route directs messages from a source endpoint that produces XML data to a target endpoint that consumes JSON data. You will add and define a data transformation component that maps the source's XML data format to the target's JSON data format.

Prerequisites

- Install Fuse tooling in Red Hat Developer Studio. See the Developer Studio documentation page, select the Developer Studio version and click the link for the installation guide.
- Install and configure Maven. See Deploying into Apache Karaf.
- Download and install the transformation quickstart applications.

Importing the `starter` quickstart application

1. Right-click in the **Project Explorer** view to open the context menu.
2. Select **Import** → **Import**.
3. Expand the **Maven** folder, and select **Existing Maven Projects**.
4. Click **Next** to open the **Maven Projects** wizard.

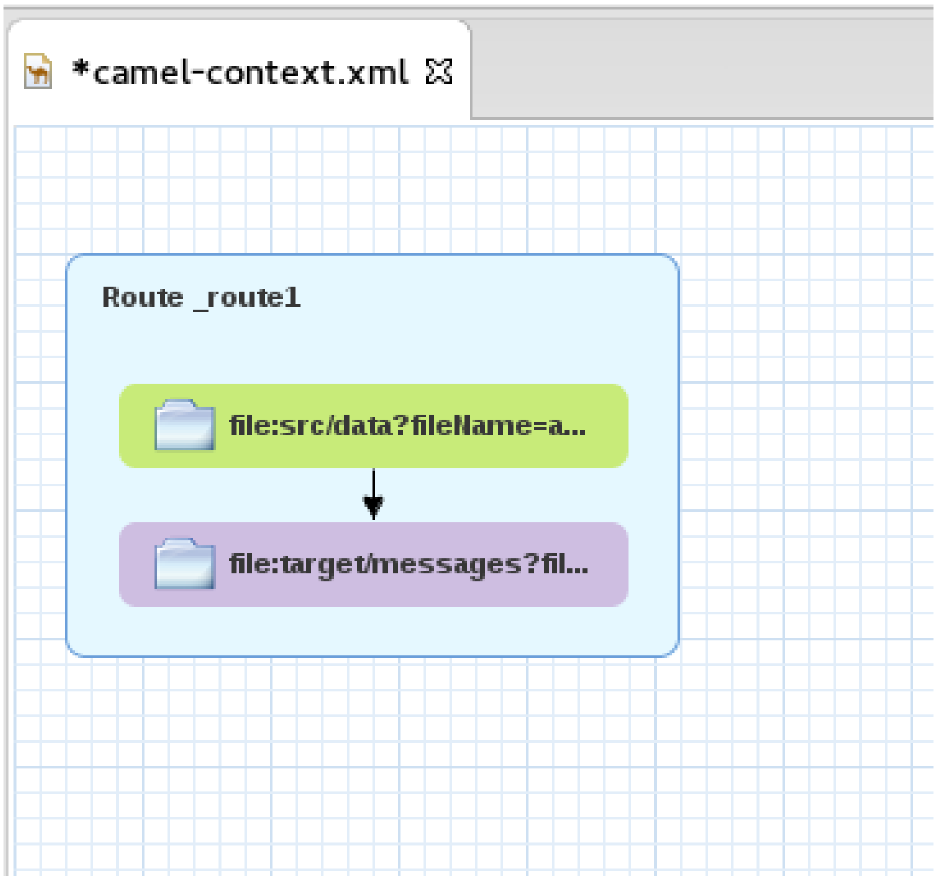


5. Click **Browse** to find and select the root directory of the **`starter`** quickstart application.
   If the browse operation finds multiple projects, make sure you select the **`starter`** quickstart application. The full path to the **`starter`** quickstart application appears in the **Projects** pane.

6. Click **Finish**.
   After the import operation finishes, the **`starter`** project appears in the **Project Explorer** view.

7. In the **Project Explorer** view, expand the **`starter`** project.

8. Double-click **`starter/src/main/resources/META-INF/spring/camel-context.xml`** to open the route in the route editor's **Design** tab.



9. Click the **Source** tab to view the underlying XML.
   You can see that an XML file is produced from a source endpoint and a JSON file is consumed by a target endpoint.

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

    <camelContext id="camelContext-272d04f8-e498-466d-b34b-
3c24d01a4e10" xmlns="http://camel.apache.org/schema/spring"
        <route id="_route1">
            <from id="_from1" uri="file:src/data?fileName=abc-
order.xml&noop=true"/>
            <to id="_to1" uri="file:target/messages?fileName=xyz-
order.json"/>
        </route>
    </camelContext>
</beans>
```

10. Click the **Design** tab to return to the graphical display of the route.

11. Remove the arrow that connects the source and target endpoints.

12. If the **Console** view is not already open, open it now by selecting **Window** → **Show View** → **Console**.

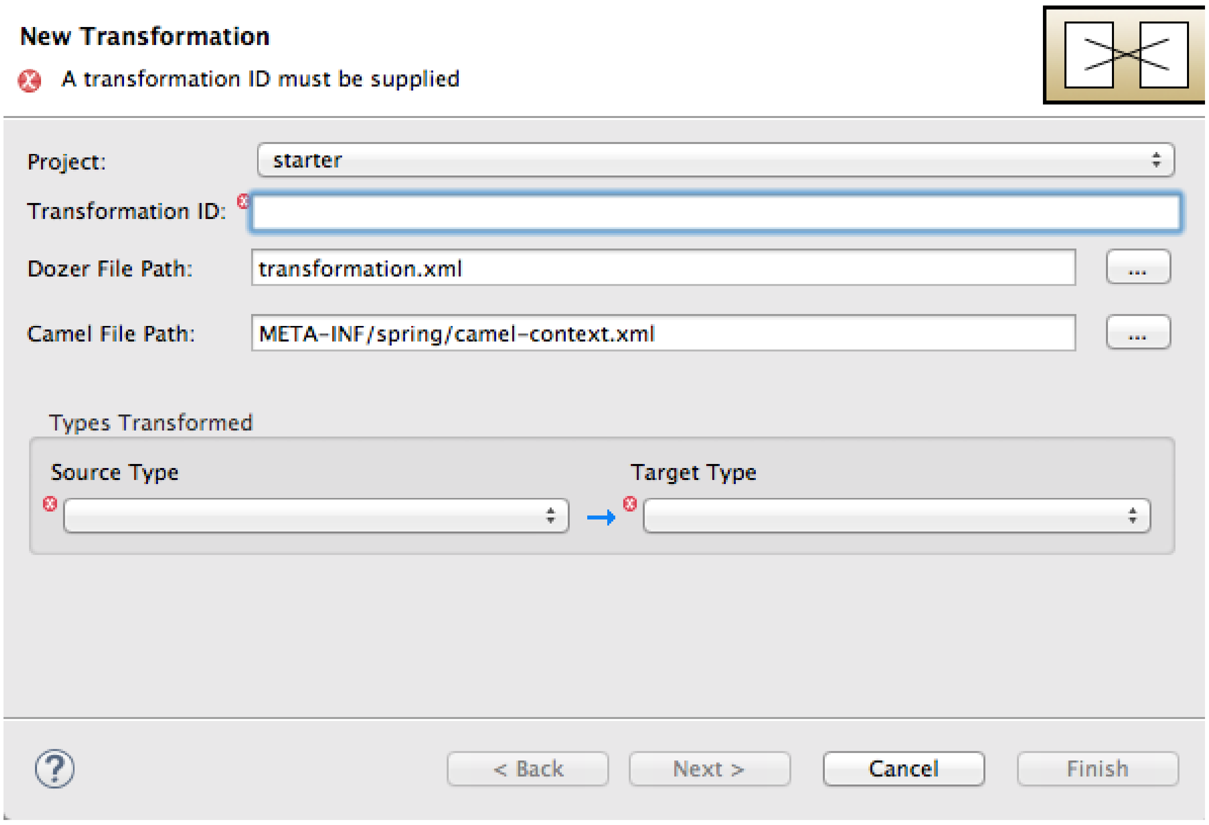
NOTE

Although the following mini tutorials are written to be run in consecutive order, you can run them in any order. If you change the order then the console output for a tutorial will differ from that shown here.

Adding a data transformation node to the Camel route

1. In the **Palette**, expand the **Transformation** *drawer*.

2. Drag a **Data Transformation** pattern over the canvas and drop it on the `Route_route1` container.
The **New Transformation** wizard opens with the **Project**, **Dozer File Path**, and **Camel File Path** fields auto filled.



New Transformation

A transformation ID must be supplied

Project: starter

Transformation ID:

Dozer File Path: transformation.xml

Camel File Path: META-INF/spring/camel-context.xml

Types Transformed

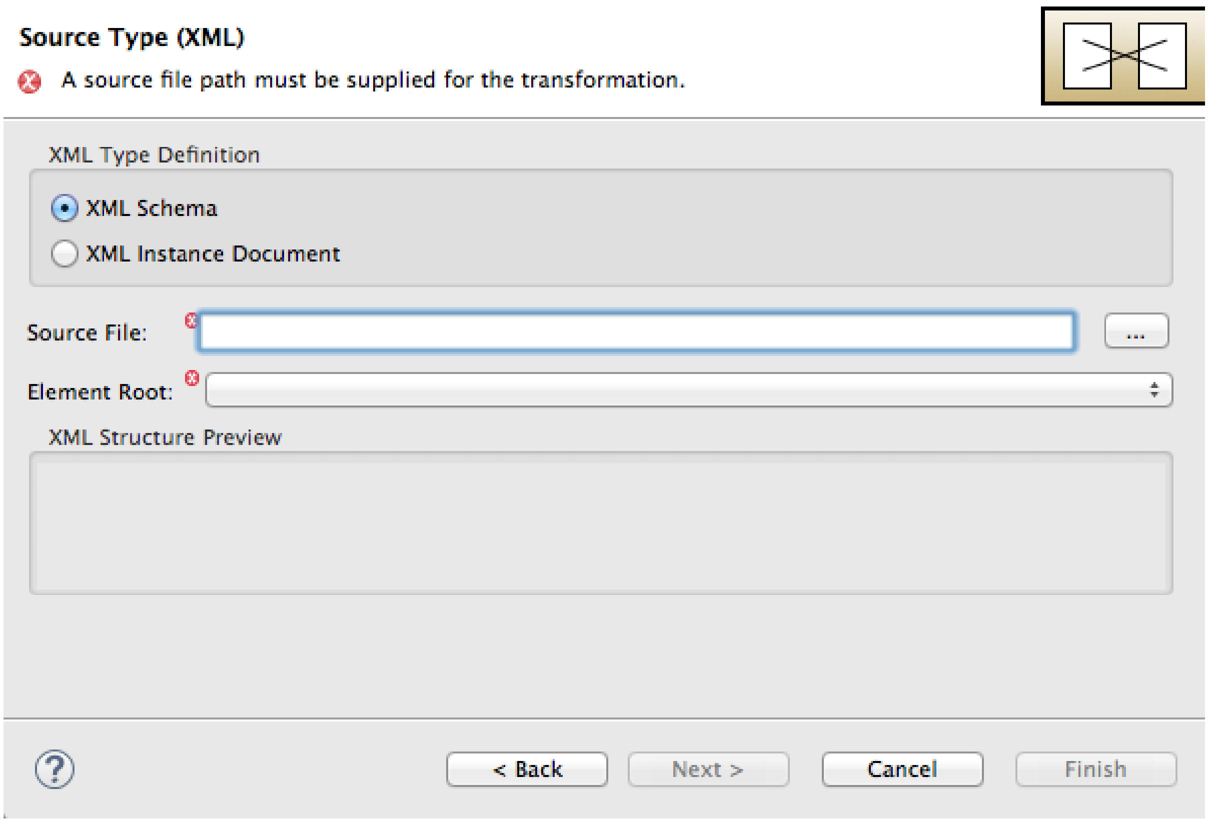
Source Type

Target Type

3. Fill in the remaining fields:

   - In the **Transformation ID** field, enter `xml2json`.
   - For **Source Type**, select **XML** from the drop-down menu.
   - For **Target Type**, select **JSON** from the drop-down menu.

4. Click **Next**.
The **Source Type (XML)** definition page opens, where you specify either an **XML Schema** (default) or an example **XML Instance Document** to provide the type definition of the source data:


Source Type (XML)

A source file path must be supplied for the transformation.

XML Type Definition

(●) XML Schema

( ) XML Instance Document

Source File:

Element Root:

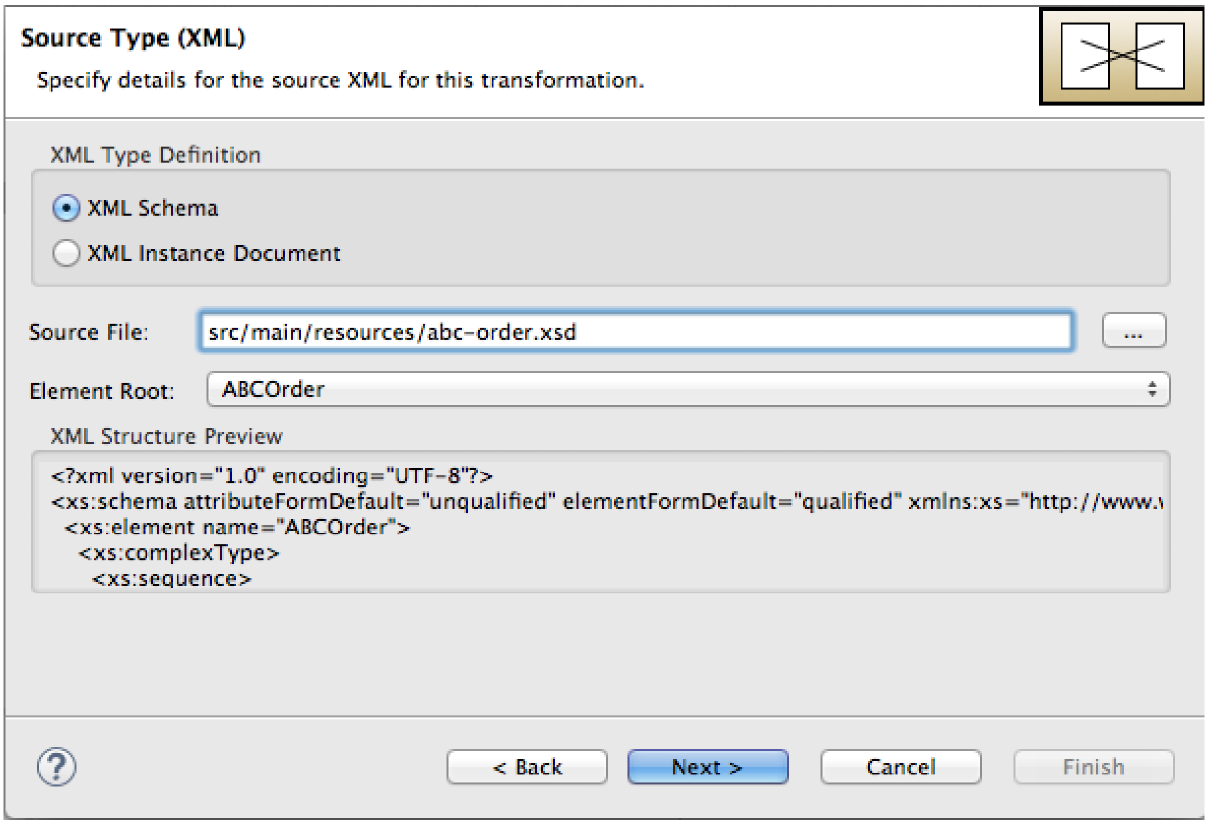
XML Structure Preview

5. Leave **XML Schema** enabled.

6. For **Source file**, browse to the location of the XML schema file or the XML instance file to use for the type definition of the source data, and select it (in this case, **`abc-order.xsd`**).
   The **XML Structure Preview** pane displays a preview of the XML structure.

7. In the **Element root** field, enter **`ABCOrder`**.
   The tooling uses this text to label the pane that displays the source data items to map.

   The **Source Type (XML)** definition page should now look like this:



[● XML Schema] XML Instance Document

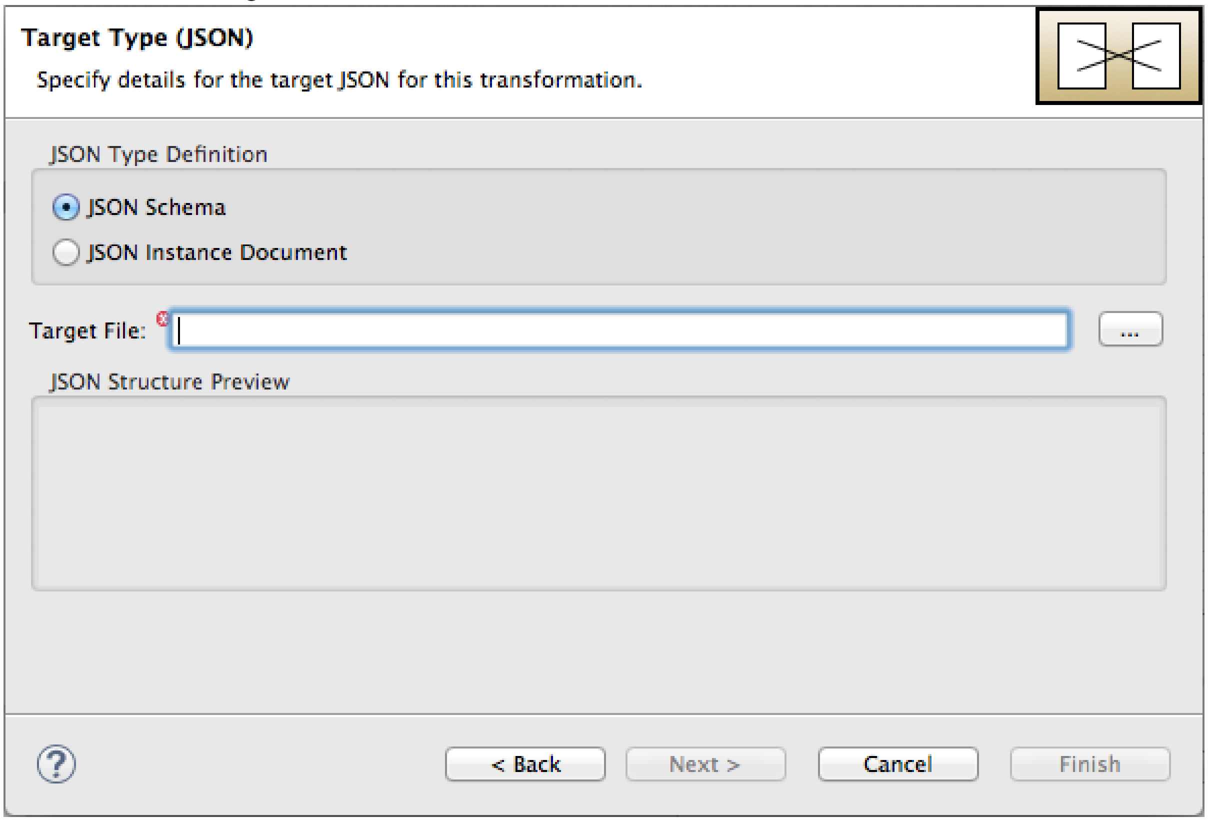
Source File: src/main/resources/abc-order.xsd

Element Root: ABCOrder

XML Structure Preview

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.\
  <xs:element name="ABCOrder">
    <xs:complexType>
      <xs:sequence>
```

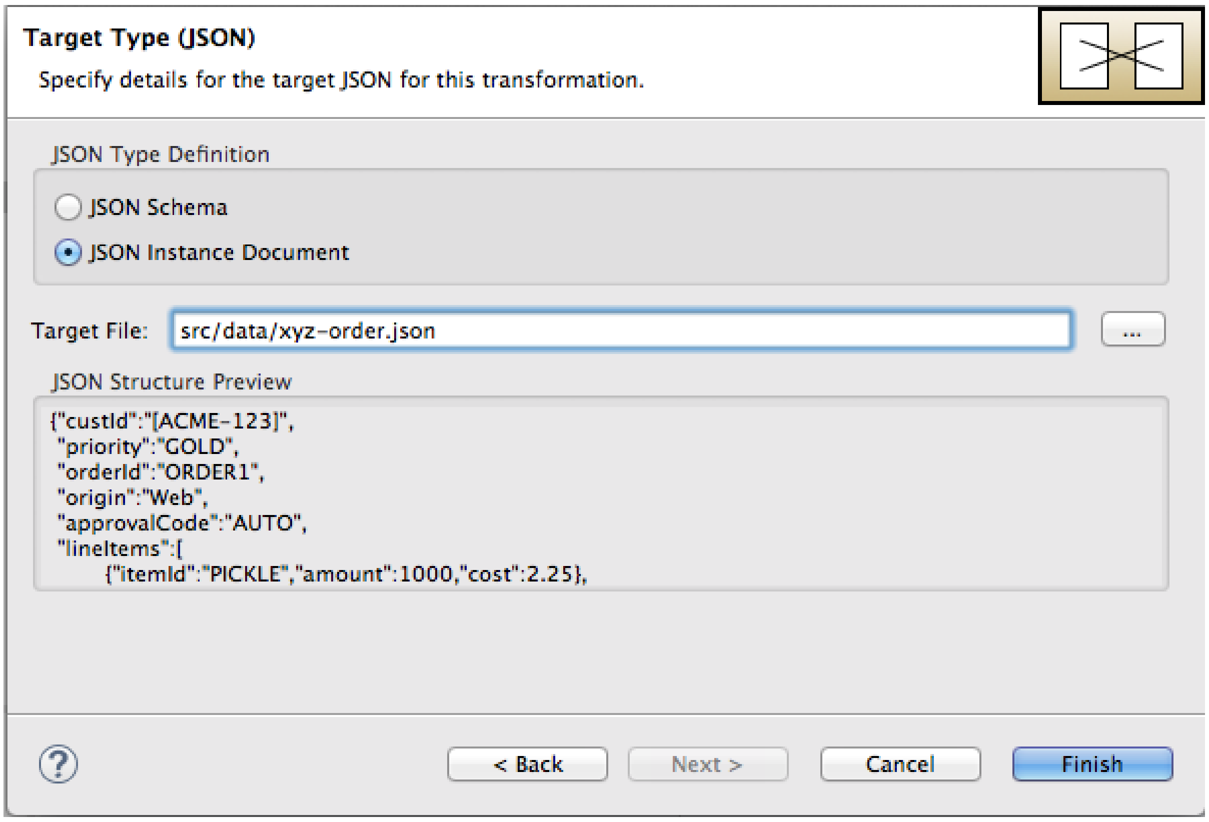
8. Click **Next** to open the **Target Type (JSON)** definition page. This is where you specify the type definition for the target data.


Target Type (JSON)
Specify details for the target JSON for this transformation.
JSON Type Definition

[● JSON Schema] JSON Instance Document

Target File:

9. Click **JSON Instance Document**.
   In the **Target File** field, enter the path to the **`xyz-order.json`** instance document, or browse to it. The **JSON Structure Preview** pane displays a preview of the JSON data structure:


Target Type (JSON)

Specify details for the target JSON for this transformation.

JSON Type Definition

( ) JSON Schema

(•) JSON Instance Document

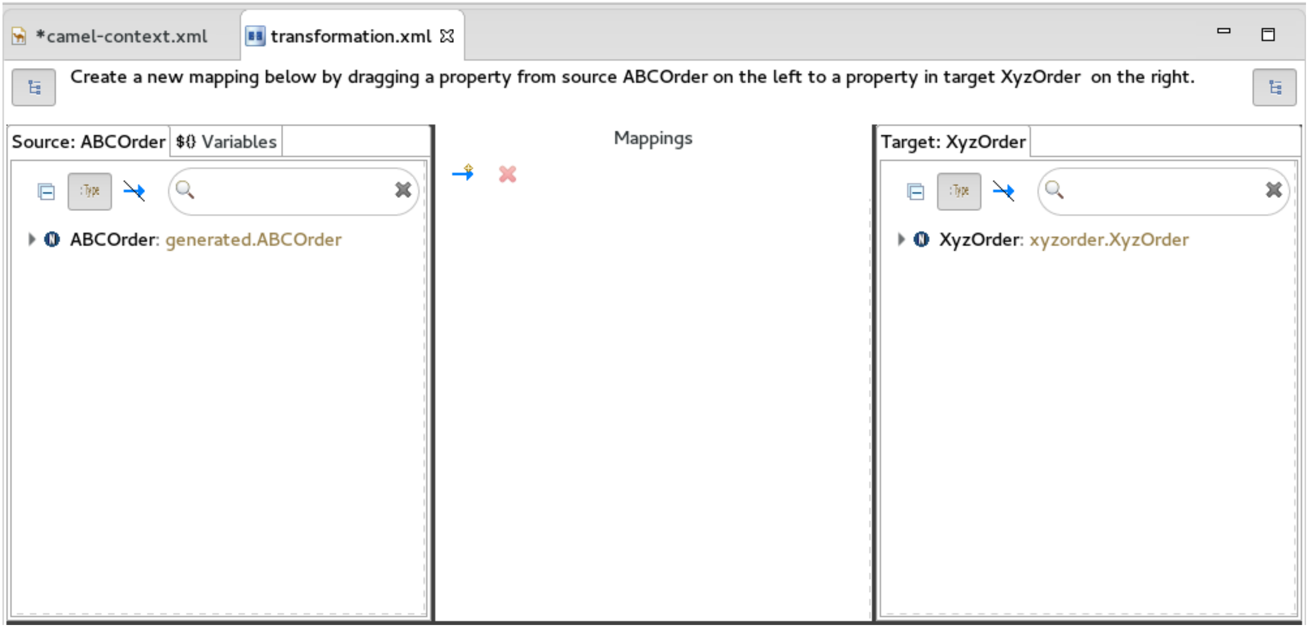
Target File: src/data/xyz-order.json

JSON Structure Preview

```
{"custId":"[ACME-123]",
 "priority":"GOLD",
 "orderId":"ORDER1",
 "origin":"Web",
 "approvalCode":"AUTO",
 "lineItems":[
       {"itemId":"PICKLE","amount":1000,"cost":2.25},
```

10. Click **Finish**.

The transformation editor opens. This is where you can map data items in your XML source to data items in your JSON target.



Create a new mapping below by dragging a property from source ABCOrder on the left to a property in target XyzOrder on the right.

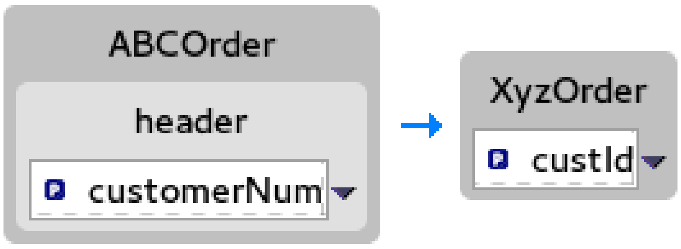
ABCOrder: generated.ABCOrder

XyzOrder: xyzorder.XyzOrder

The transformation editor is composed of three panels:

- **Source** — lists the available data items of the source
- **Mappings** — displays the mappings between the source and target data items
- **Target** — lists the available data items of the target

In addition, the editor's details pane, located just below the editor's three panels (once the first mapping has been made), graphically displays the hierarchical ancestors for both the mapped source and target data items currently selected. For example:

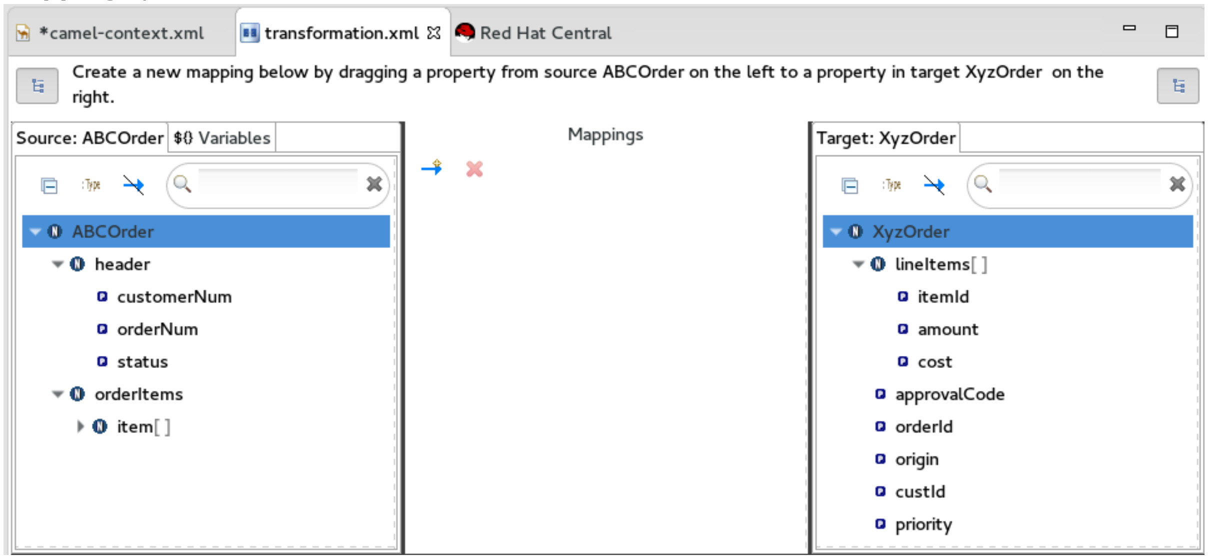


Using the details pane, you can customize the mapping for the selected source and target data items:

- **Set property** — Modify an existing mapping or map a simple data item to one in a collection (see the section called “Mapping a simple data item to a data item in a collection”).
- **Set variable** — Specify a constant value for a data item (see the section called “Mapping a constant variable to a data item”).
- **Set expression** — Map a data item to the dynamic evaluation of a specified expression (seethe section called “Mapping an expression to a data item”).
- **Add transformation** — Modify the value of a mapped data item using a built-in function (seethe section called “Adding a built-in function to a mapped data item”).
- **Add custom transformation** — Modify the value of a mapped data item using the Java method you create or one you previously created (see the section called “Adding a custom transformation to a mapped data item”).

Mapping source data items to target data items

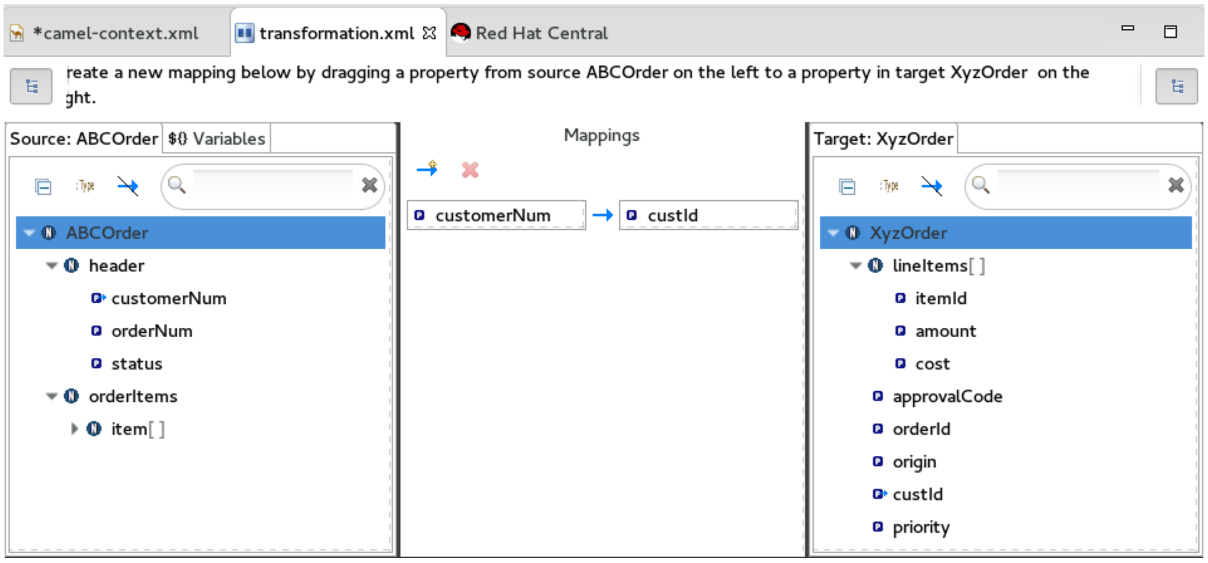
1. Expand all items in the **Source** and **Target** panels located on left and right sides of the **Mappings** panel.



Create a new mapping below by dragging a property from source ABCOrder on the left to a property in target XyzOrder on the right.

2. Drag a data item from the **Source** panel and drop it on its corresponding data item in the **Target** panel.

For example, drag the **`customerNum`** data item from the **Source** panel and drop it on the **`custId`** data item in the **Target** panel.



The mapping appears in the **Mappings** panel, and the details of both the **Source** and **Target** data items appear below in the details pane.

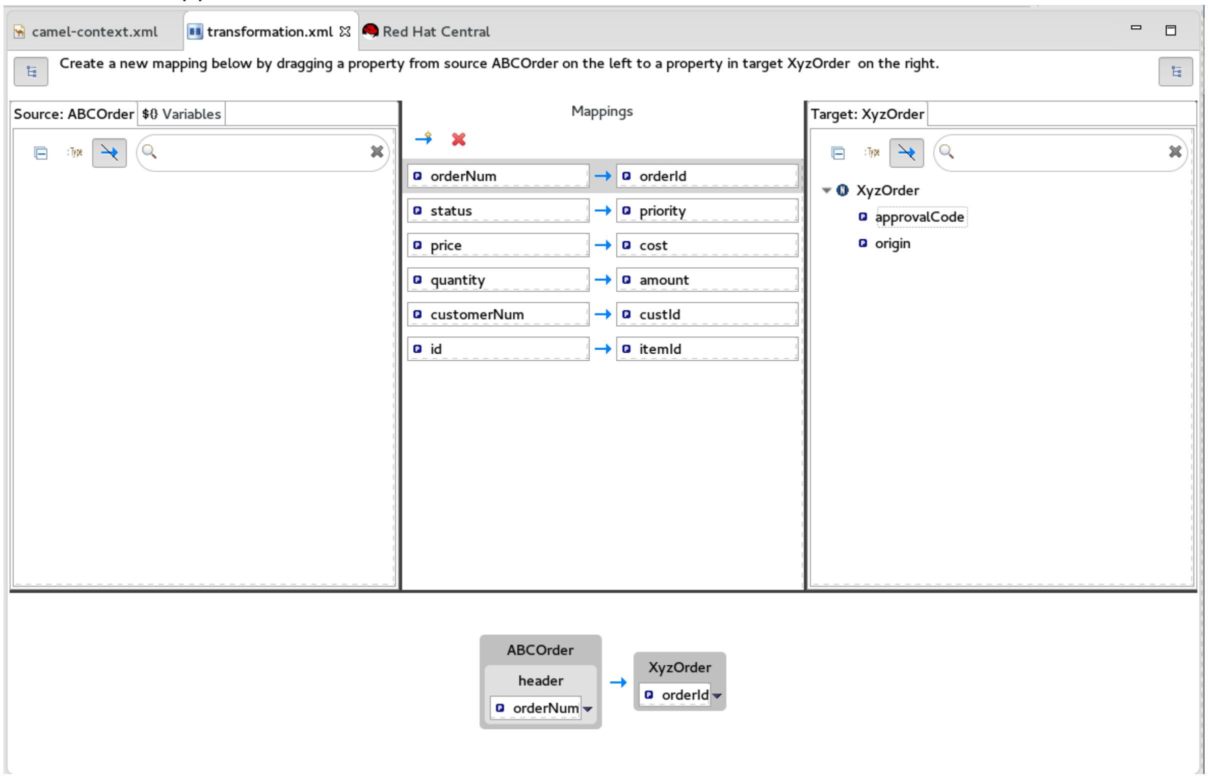
3. Continue dragging and dropping source data items onto their corresponding target data items until you have completed all basic mappings.
   In the **`starter`** example, the remaining data items to map are:

Source	Target +
`orderNum`	`orderId` +
`status`	`priority` +
`id`	`itemId` +
`price`	`cost` +
`quantity`	`amount` +

NOTE

You can map collections (data items containing lists or sets) to noncollection data items and vice versa, but you cannot map collections to other collections.

4. Click on both the **Source** and **Target** panels to quickly determine whether all data items have been mapped.

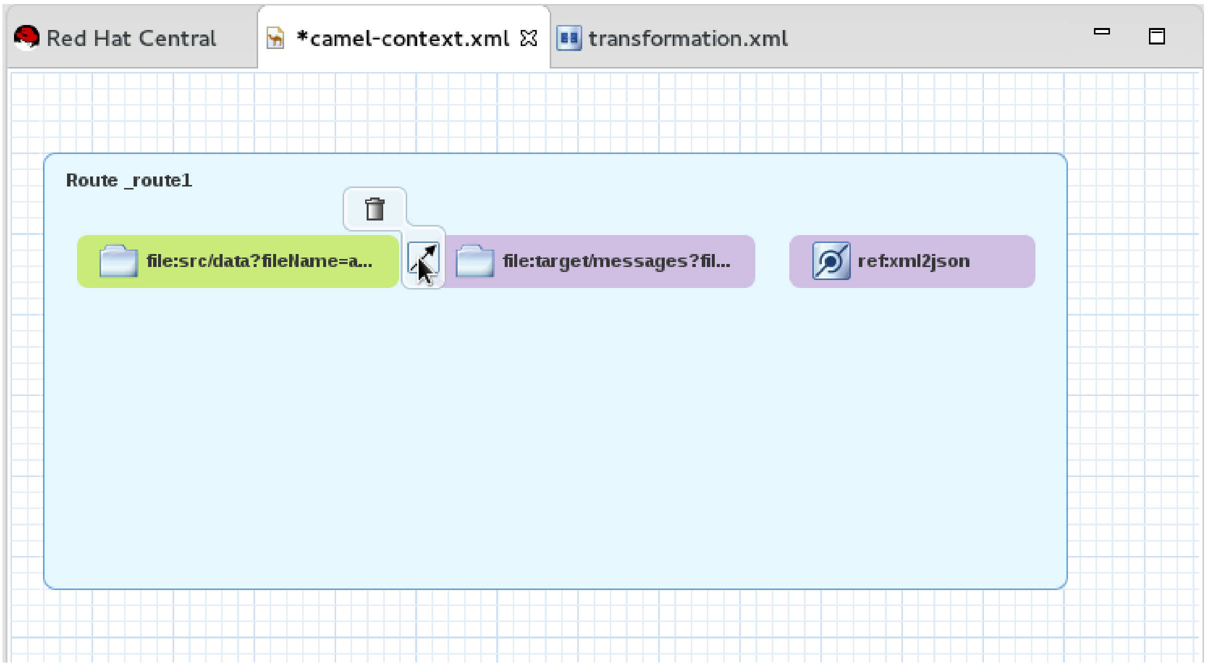


Only data items that have not been mapped are listed in the **Source** and **Target** panels.

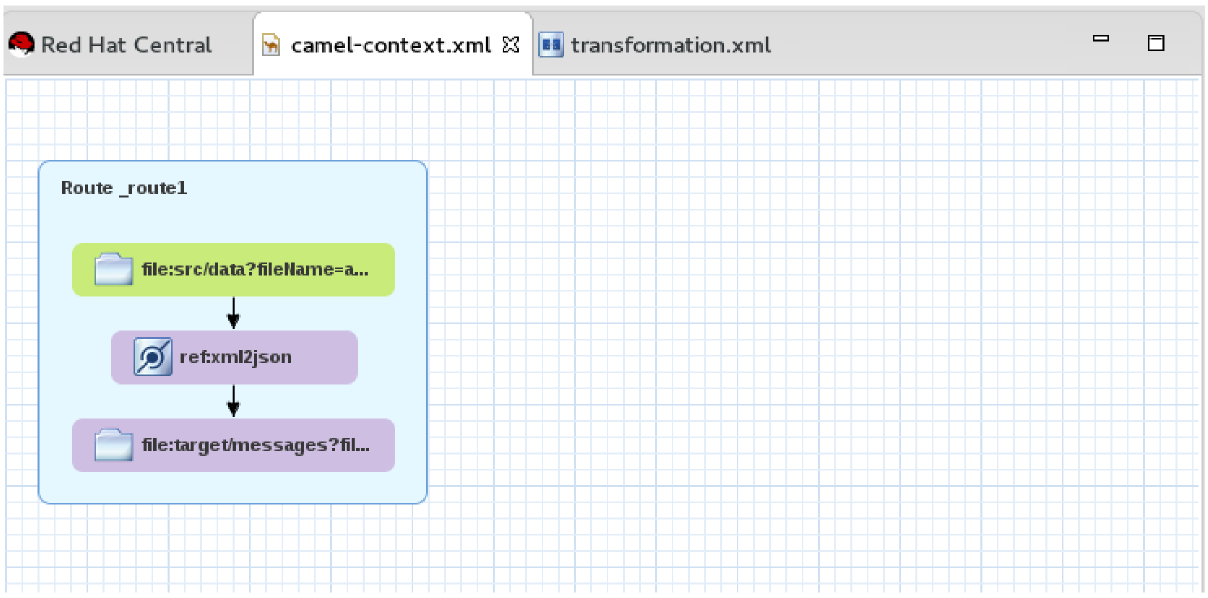
In the **`starter`** example, the remaining unmapped **Target** attributes are **`approvalCode`** and **`origin`**.

5. Click the **camel-context.xml** tab to return to the graphical display of the route.

6. Hover your cursor over each endpoint to reveal its connecter arrow.



7. Selecting the **`file:src/data?fil…`** node, drag and drop its connector arrow onto the **`ref:xml2json`** node. Likewise drag and drop the connector arrow from the**`ref:xml2json`** node onto the **`file:target/messa…`** node.



Connecting the nodes on the canvas creates a valid Camel route, which you can now save.

8. Click **File** → **Save**.

You can run a JUnit test on your transformation file after you create the transformation test. For details, see the section called “Creating the transformation test file and running the JUnit test”. If you do so at this point, you will see this output in the **Console** view:

```
<?xml version="1.0" encoding="UTF-8"?>
<ABCOrder xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:java="http://java.sun.com">
    <header>
        <status>GOLD</status>
        <customer-num>ACME-123</customer-num>
        <order-num>ORDER1</order-num>
    </header>
    <order-items>
        <item id="PICKLE">
            <price>2.25</price>
            <quantity>1000</quantity>
        </item>
        <item id="BANANA">
            <price>1.25</price>
            <quantity>400</quantity>
        </item>
    </order-items>
</ABCOrder>
```

for the source XML data, and

```
{"custId":"ACME-123","priority":"GOLD","orderId":"ORDER1","lineItems":
[{"itemId":"PICKLE",
"amount":1000,"cost":2.25},{"itemId":"BANANA","amount":400,"cost":1.25
```

for the target JSON data.

Creating the transformation test file and running the JUnit test

1. Right-click the **`starter`** project in the **Project Explorer** view, and select **New** → **Other** → **Fuse Tooling** → **Fuse Transformation Test**.

2. Select **Next** to open the **New Transformation Test** wizard.

3. In the **New Transformation Test** wizard, set the following values:

Field	Value +
Package	`example` +
Transformation ID	`xml2json` +

4. Click **Finish**.

5. In the **Project Explorer** view, navigate to **`starter/src/test/java/example`**, and open the **`TransformationTest.java`** file.

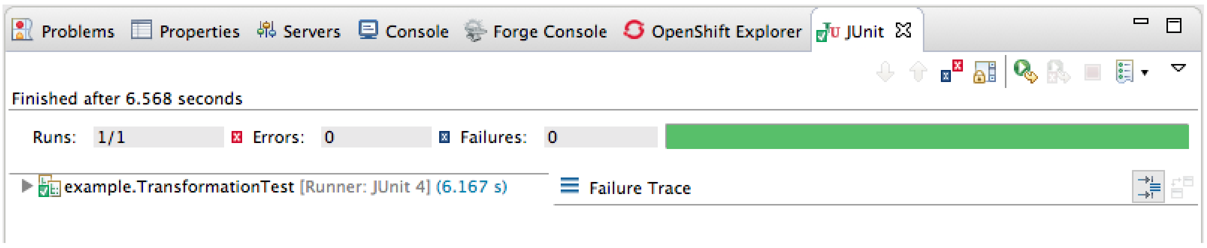
6. Add the following code to the **`transform`** method:

```
startEndpoint.sendBodyAndHeader(readFile("src/data/abc-order.xml"),
"approvalID", "AUTO_OK");
```

7. Click **File** → **Save**.
   You can now run a JUnit test on your transformation file at any point in these tutorials.

8. In the **Project Explorer** view, expand the **`starter`** project to expose the **`/src/test/java/example/TransformationTest.java`** file.

9. Right click it to open the context menu, and select **Run as JUnit Test**.
   The JUnit Test pane opens to display the status of the test. To avoid cluttering your workspace, drag and drop the pane in the bottom panel near the **Console** view.



Finished after 6.568 seconds

Runs: 1/1 Errors: 0 Failures: 0

example.TransformationTest [Runner: JUnit 4] (6.167 s)

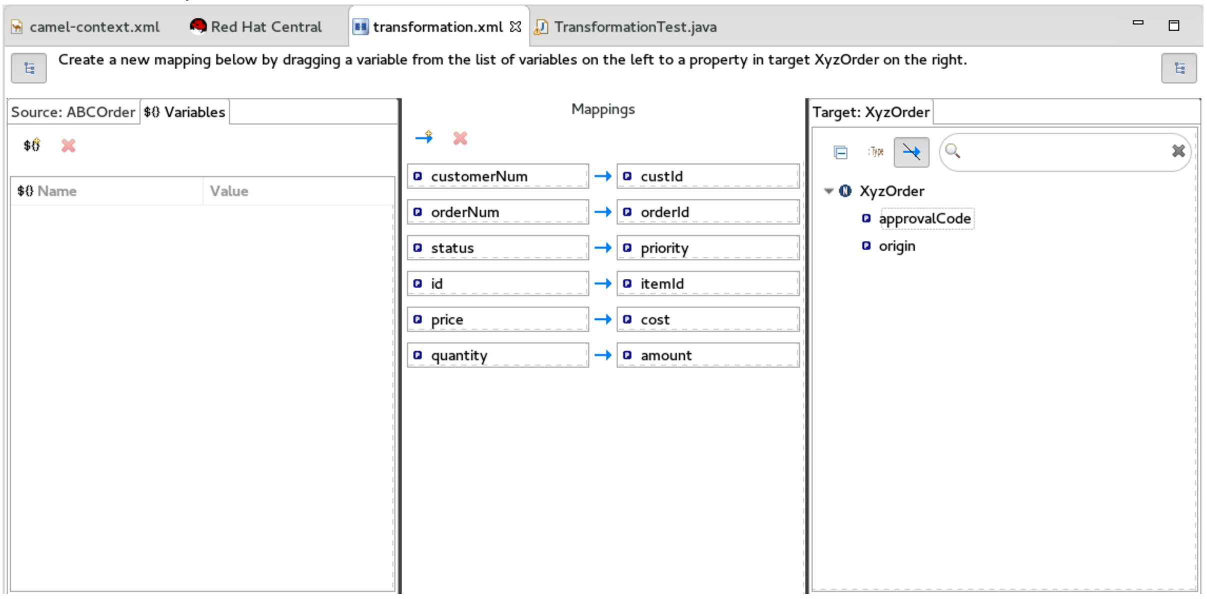
10. Open the **Console** view to see the log output.

Mapping a constant variable to a data item

When a source/target data item has no corresponding target/source data item, you can map a constant variable to the existing data item.

In the **`starter`** example, the target data item **`origin`** does not have a corresponding source data item. To map the **`origin`** attribute to a constant variable:

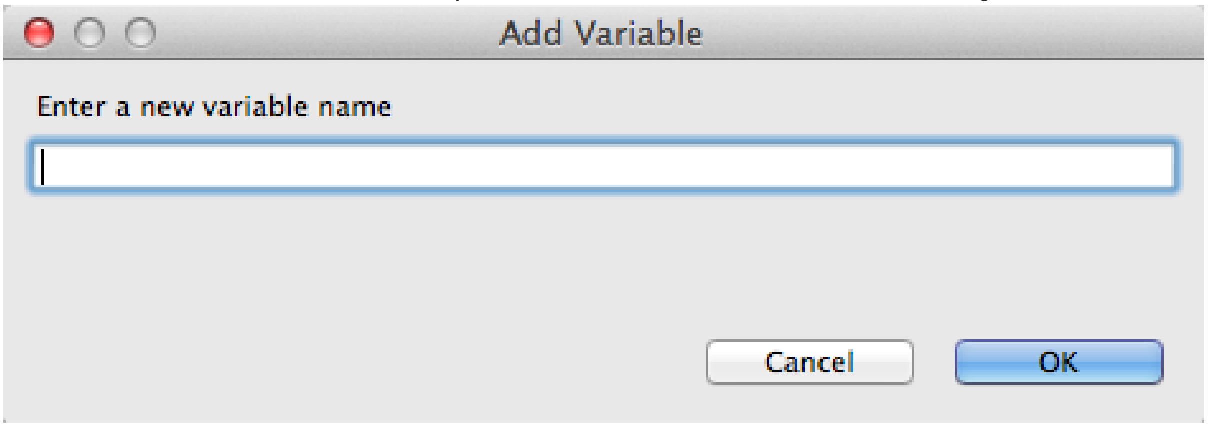
1. In the **Source** panel, click the **Variables** view.



Create a new mapping below by dragging a variable from the list of variables on the left to a property in target XyzOrder on the right.

customerNum	custId
orderNum	orderId
status	priority
id	itemId
price	cost
quantity	amount

2. In the **Variables** view, click to open the **Enter a new variable name** dialog.



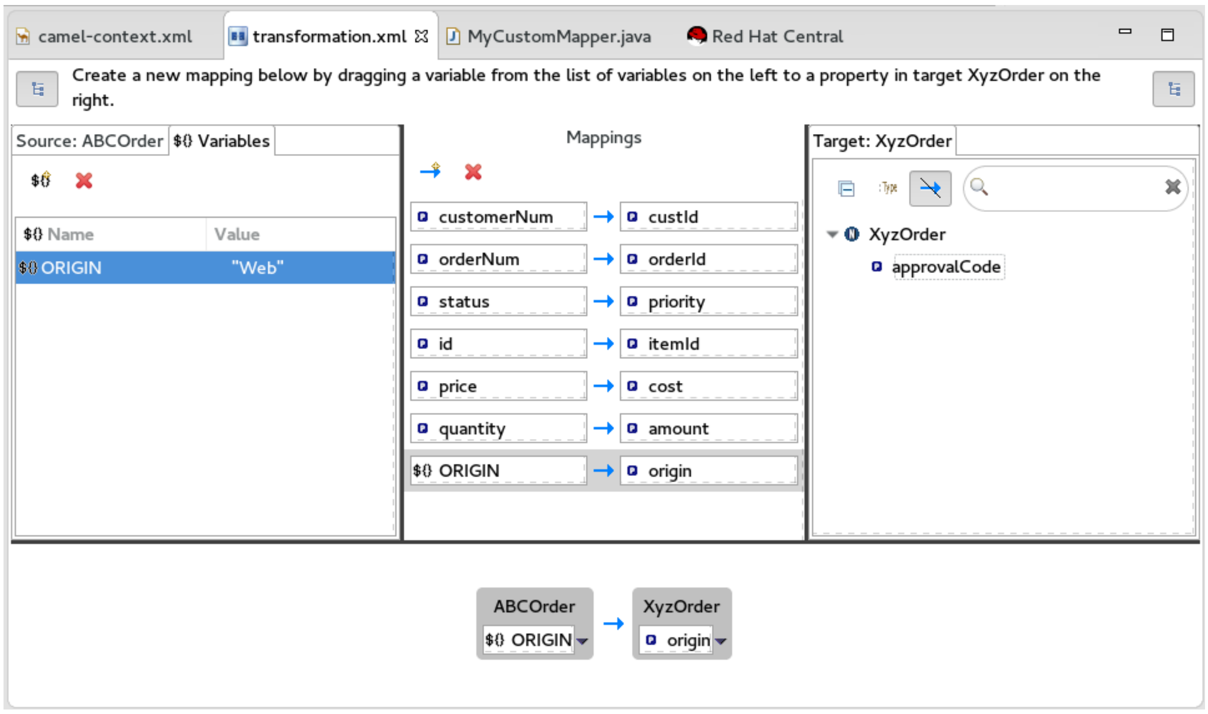
3. Enter a name for the variable you want to create.
   For the `starter` example, enter `ORIGIN`.

4. Click **OK**.
   The newly created variable `ORIGIN` appears in the **Variables** view in the **Name** column and the default value `ORIGIN` in the **Value** column.

5. Click the default value to edit it, and change the value to `Web`.

6. Press **Enter**.

7. Drag and drop the new variable `ORIGIN` onto the `origin` data item in the **Target** panel.



Create a new mapping below by dragging a variable from the list of variables on the left to a property in target XyzOrder on the right.

$() Name	Value
$() ORIGIN	"Web"

customerNum	custId
orderNum	orderId
status	priority
id	itemId
price	cost
quantity	amount
$() ORIGIN	origin

ABCOrder $() ORIGIN → XyzOrder origin

The new mapping of the variable **`$(ORIGIN)`** appears in the **Mappings** panel and in the details pane.

8. Run a JUnit test on your **`TransformationTest.java`** file. For details, see the section called “Creating the transformation test file and running the JUnit test”.
   The **Console** view displays the JSON-formatted output data:

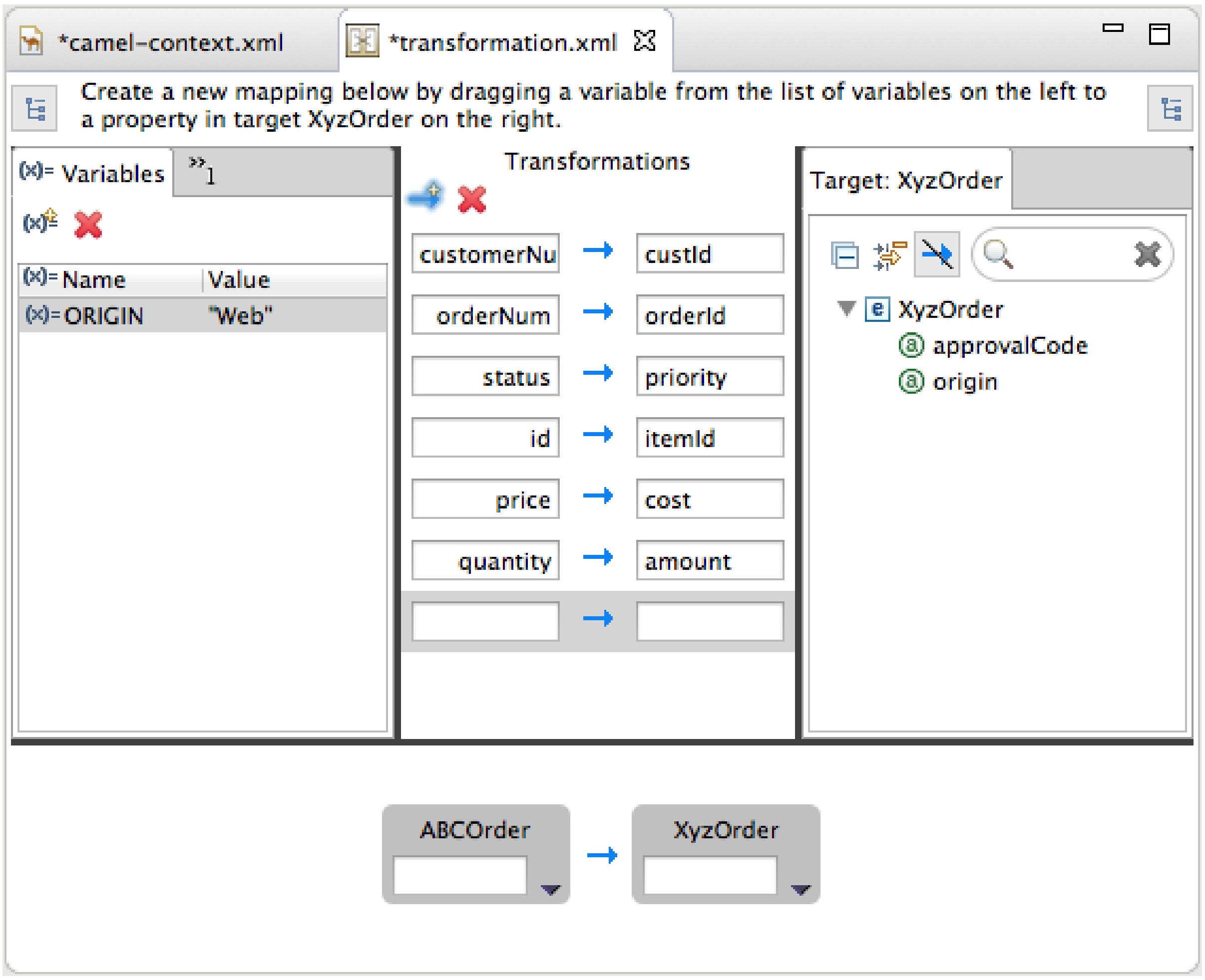
```
{"custId":"ACME-
123","priority":"GOLD","orderId":"ORDER1","origin":"Web",
"approvalCode":"AUTO_OK","lineItems":
[{"itemId":"PICKLE","amount":1000,"cost":2.25},
{"itemId":"BANANA","amount":400,"cost":1.25}]}
```

Mapping an expression to a data item

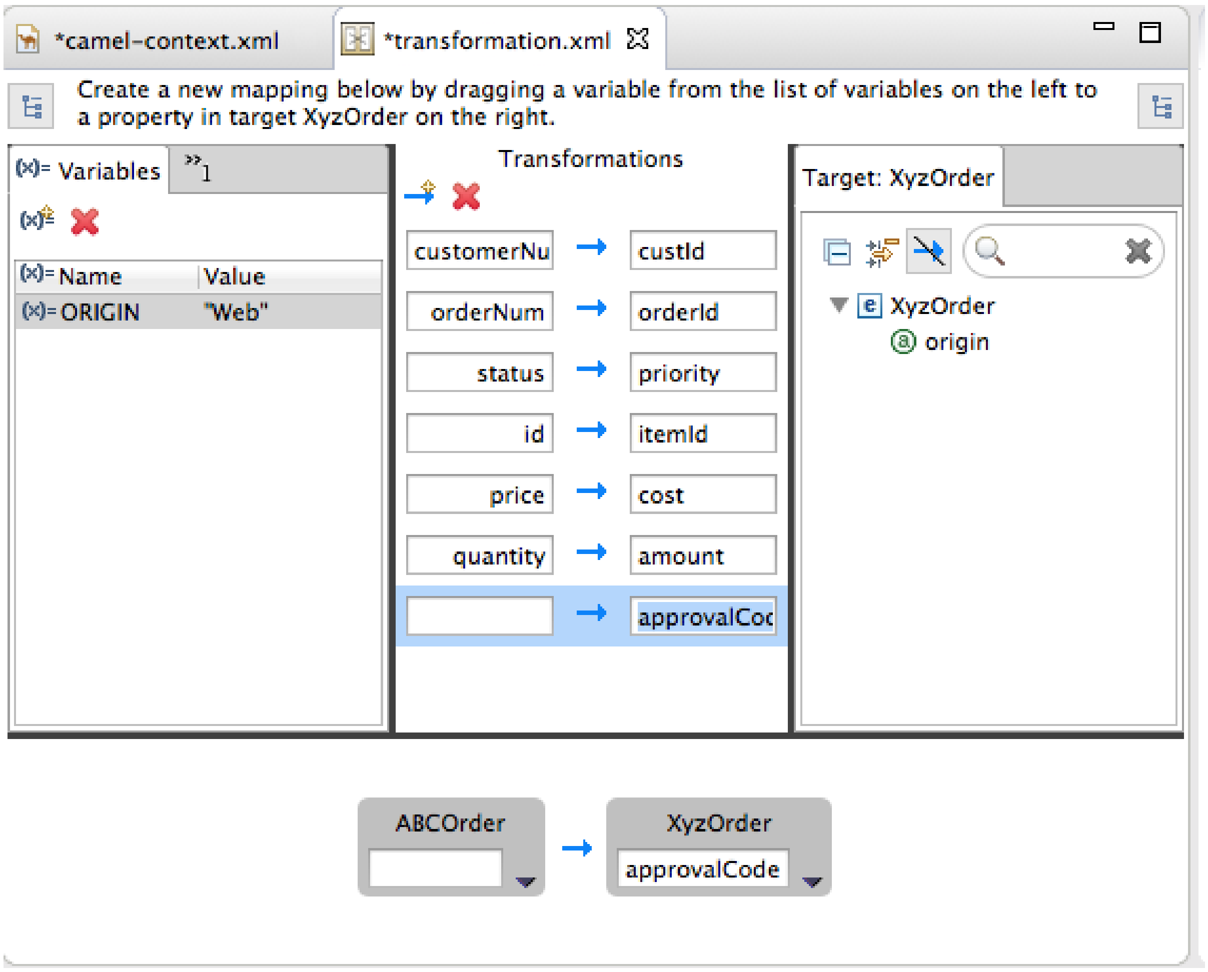
This feature enables you, for example, to map a target data item to the dynamic evaluation of a Camel language expression.

Use the target **`approvalCode`** data item, which lacks a corresponding source data item:

1. Click ➕ to add an empty transformation map to the **Mappings** panel.

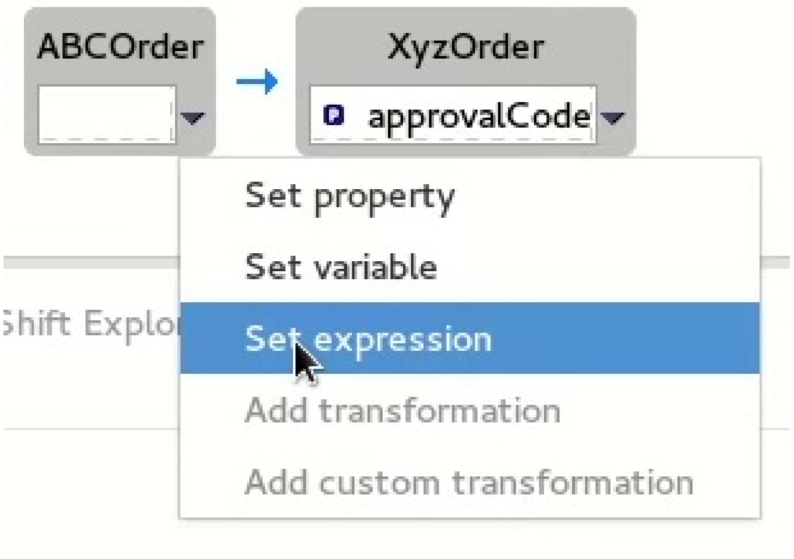


2. From the **Target** panel, drag and drop the **`approvalCode`** data item to the target field of the newly created mapping in the **Mappings** panel.



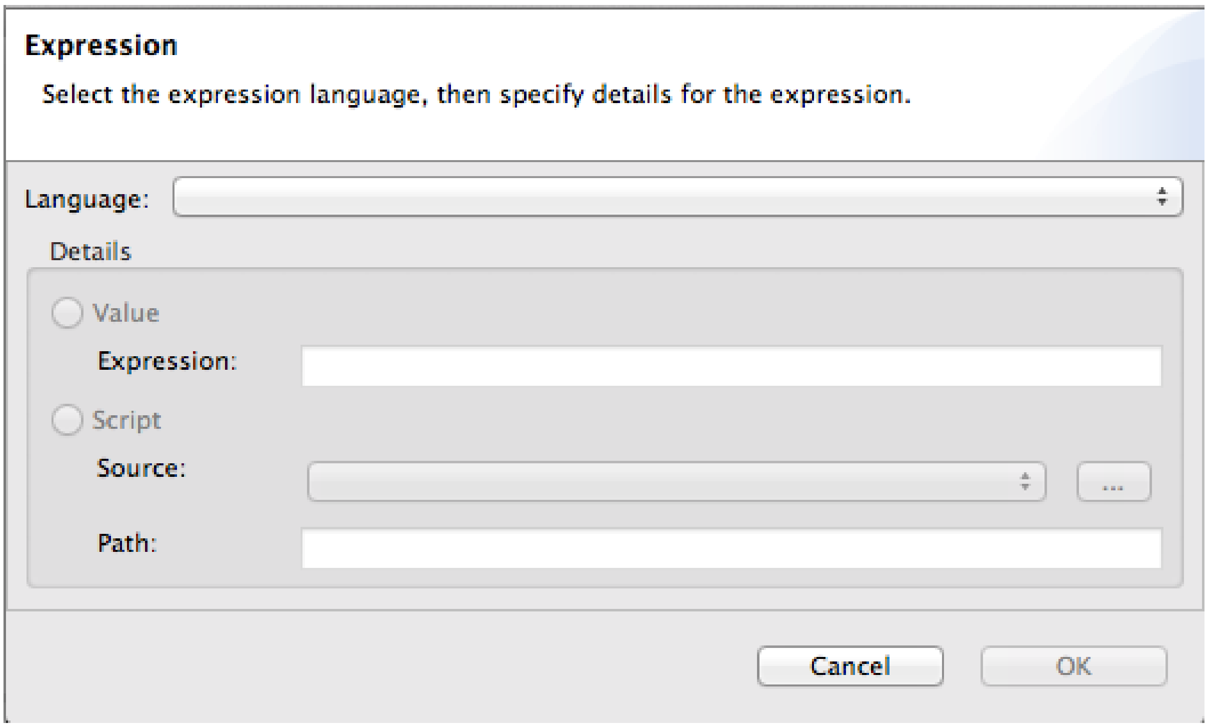
The **approvalCode** data item also appears in the details pane's target box.

3. In the details pane, click ▾ on the **ABCOrder** source box to open the drop-down menu.



Menu options depend on the selected data item's data type. The available options are bolded.

4. Select **Set expression** to open the **Expression** dialog.


Expression

Select the expression language, then specify details for the expression.

Language:

Details

Value

Expression:

Script

Source:

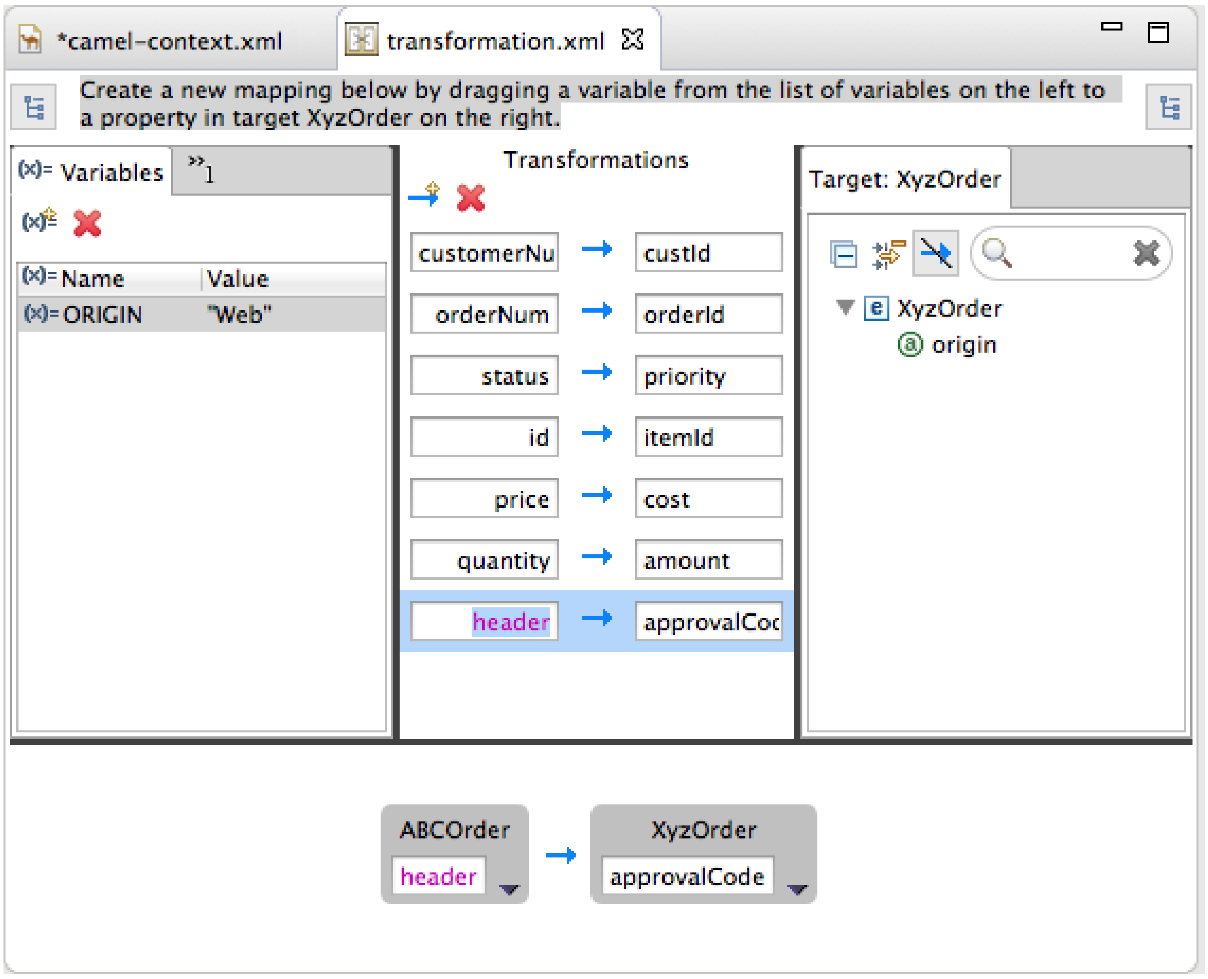
Path:

5. In **Language**, select the expression language to use from the list of those available. Available options depend on the data item's data type.
   For the **`starter`** example, select **Header**.

6. In the details pane, select the source of the expression to use.
   The options are **Value** and **Script**.

   For the **`starter`** example, click **Value**, and then enter **`ApprovalID`**.

7. Click **OK**.



Both the **Mappings** panel and the details pane display the new mapping for the target data item **`approvalCode`**.

8. Run a JUnit test on your **`TransformationTest.java`** file. For details, see the section called "Creating the transformation test file and running the JUnit test".
   The **Console** view displays the JSON-formatted output data:

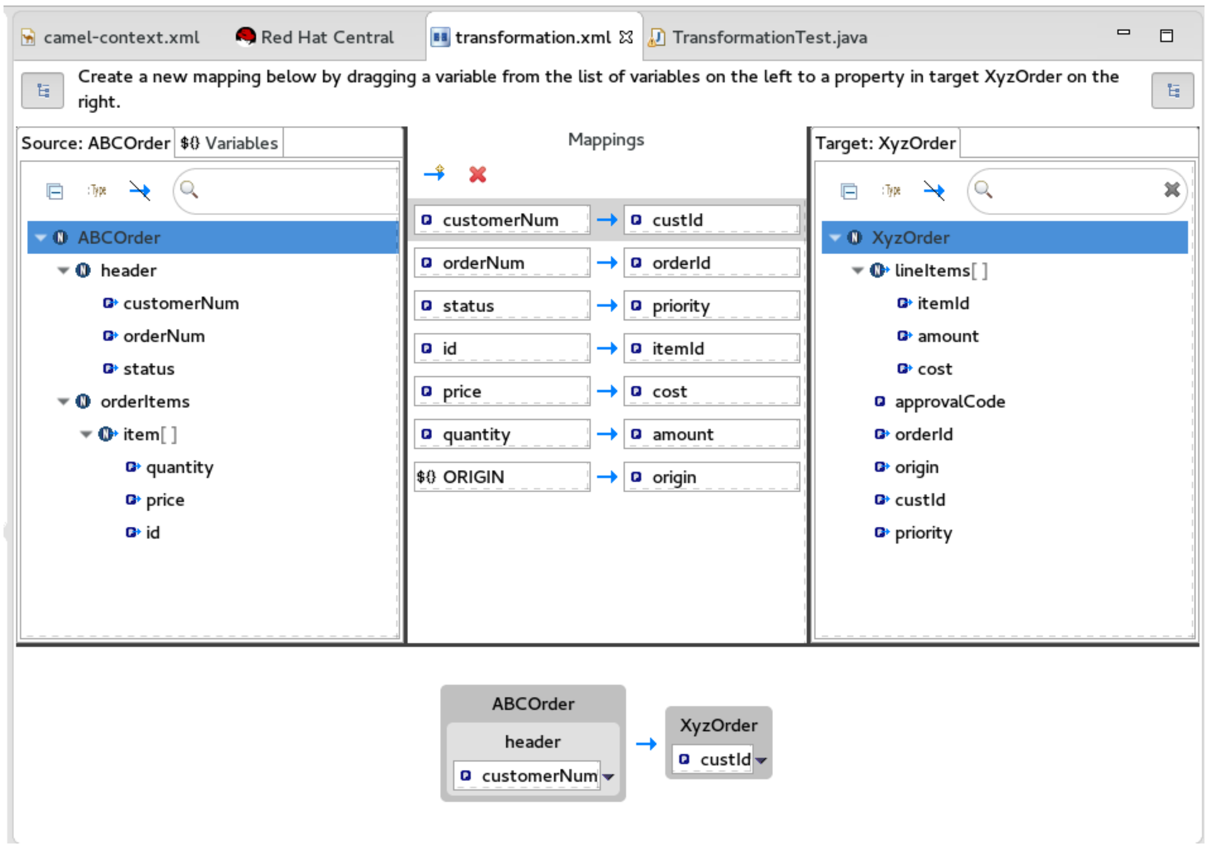
```
{"custId":"ACME-
123","priority":"GOLD","orderId":"ORDER1","origin":"Web",
"approvalCode":"AUTO_OK","lineItems":
[{"itemId":"PICKLE","amount":1000,"cost":2.25},
{"itemId":"BANANA","amount":400,"cost":1.25}]}
```

Adding a custom transformation to a mapped data item

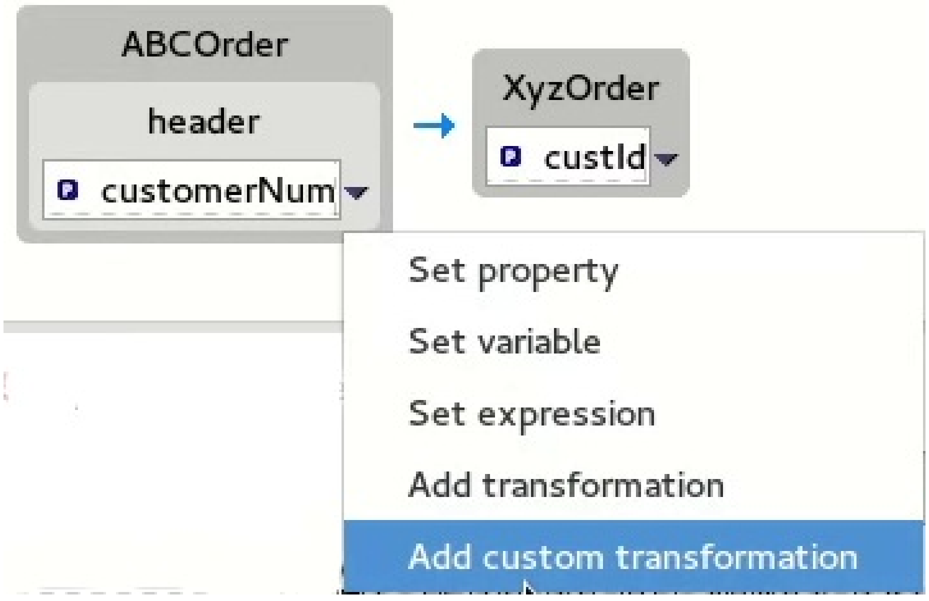
You may need to modify the formatting of source data items when they do not satisfy the requirements of the target system.

For example, to satisfy the target system's requirement that all customer IDs be enclosed in brackets:

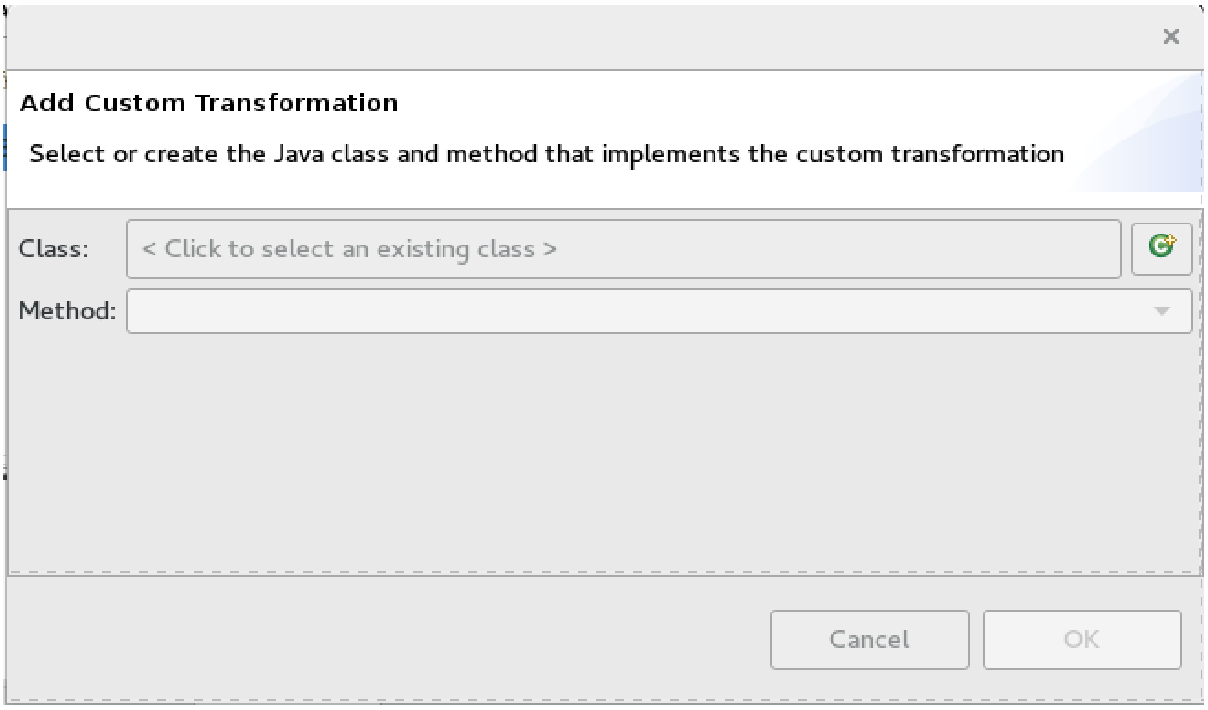
1. In the **Mappings** panel, select the **`customerNum`** mapping to populate the details pane.



2. In the details pane, click ▾ on the **ABCOrder** source box to open the drop-down menu.



3. Select **Add custom transformation** to open the **Add Custom Transformation** page.



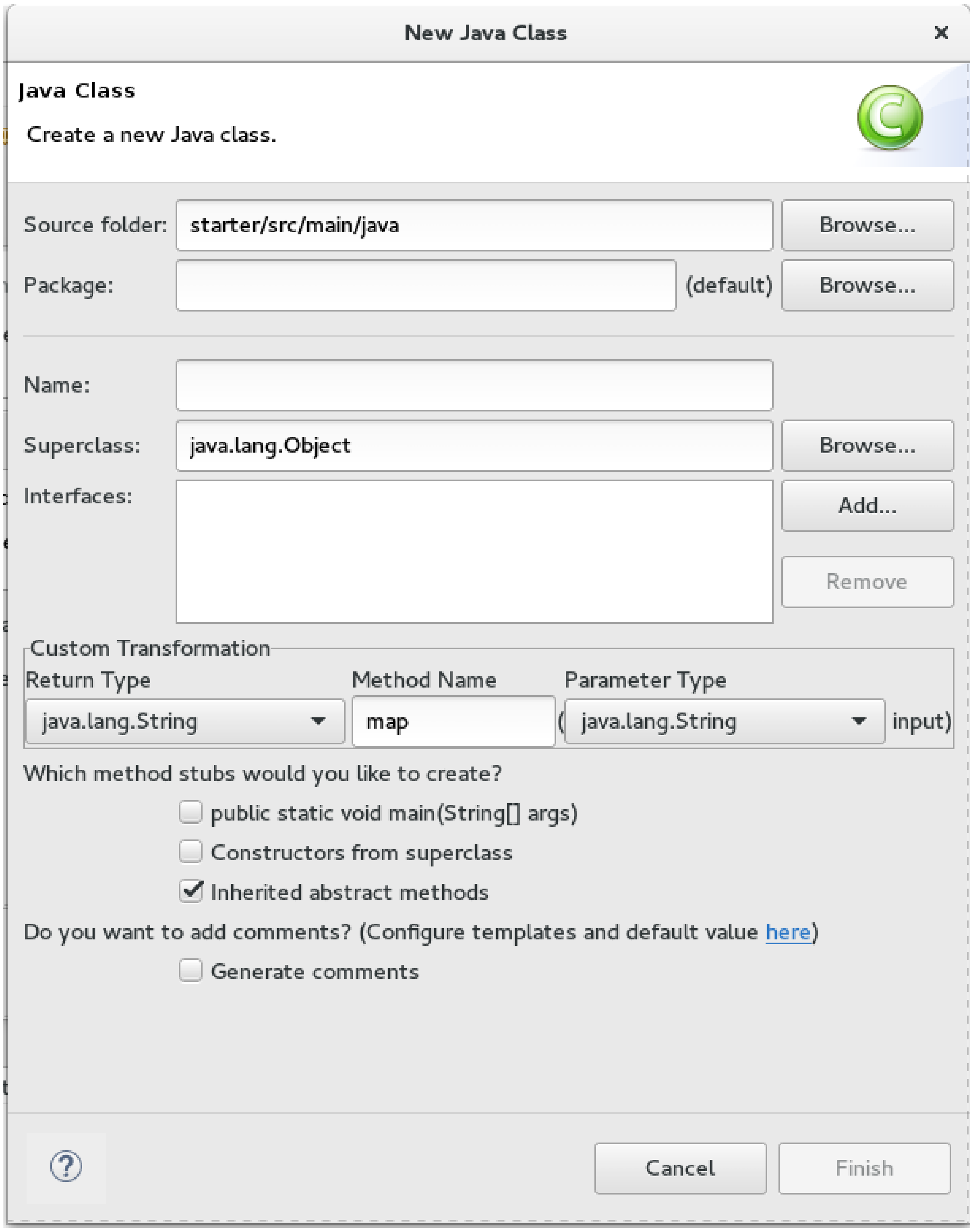
Add Custom Transformation

Select or create the Java class and method that implements the custom transformation

Class: < Click to select an existing class >

Method:

4. Click next to the **Class** field to open the **Create a New Java Class** wizard.


New Java Class

Java Class

Create a new Java class.

Source folder: starter/src/main/java

Package: (default)

Name:

Superclass: java.lang.Object

Interfaces:

Custom Transformation

Return Type: java.lang.String

Method Name: map

Parameter Type: (java.lang.String input)

Which method stubs would you like to create?

- [ ] public static void main(String[] args)
- [ ] Constructors from superclass
- [✓] Inherited abstract methods

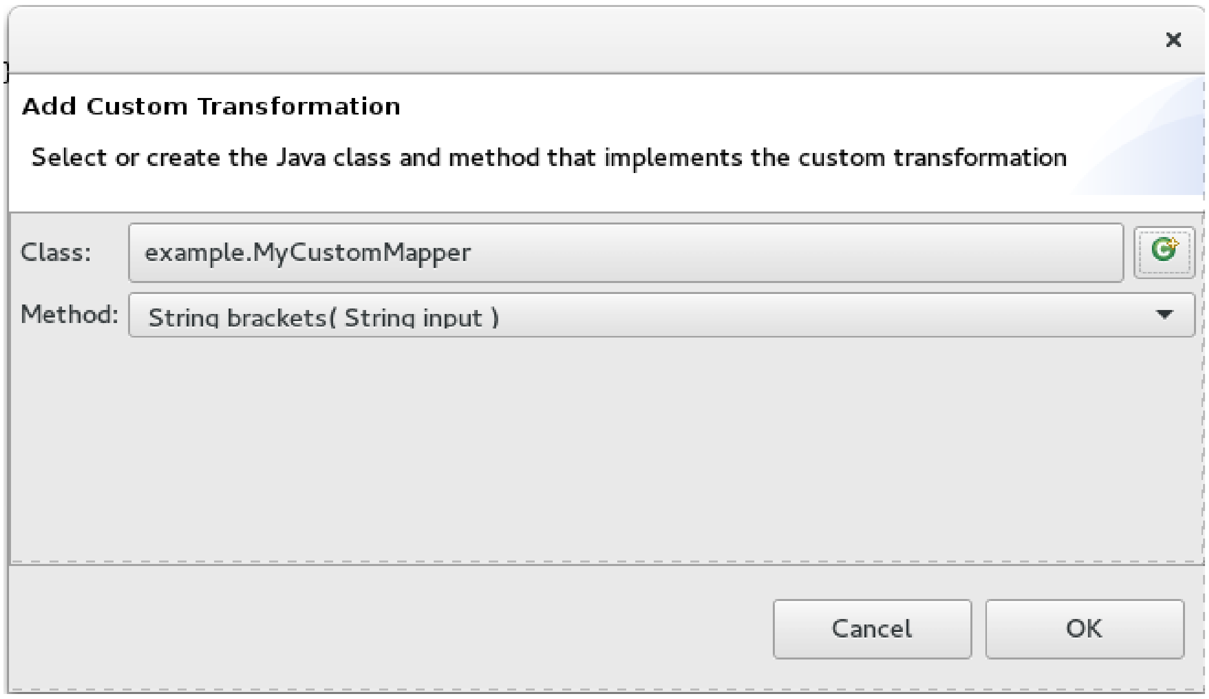
Do you want to add comments? (Configure templates and default value here)

- [ ] Generate comments

5. Modify the following fields:

   - **Package** — Enter **`example`**.
   - **Name** — Enter **`MyCustomMapper`**.
   - **Method Name** — Change **`map`** to **`brackets`**.
     Leave all other fields as is.

6. Click **Finish**.
   The **Add Custom Transformation** page opens with the **Class** and **Method** fields auto filled:

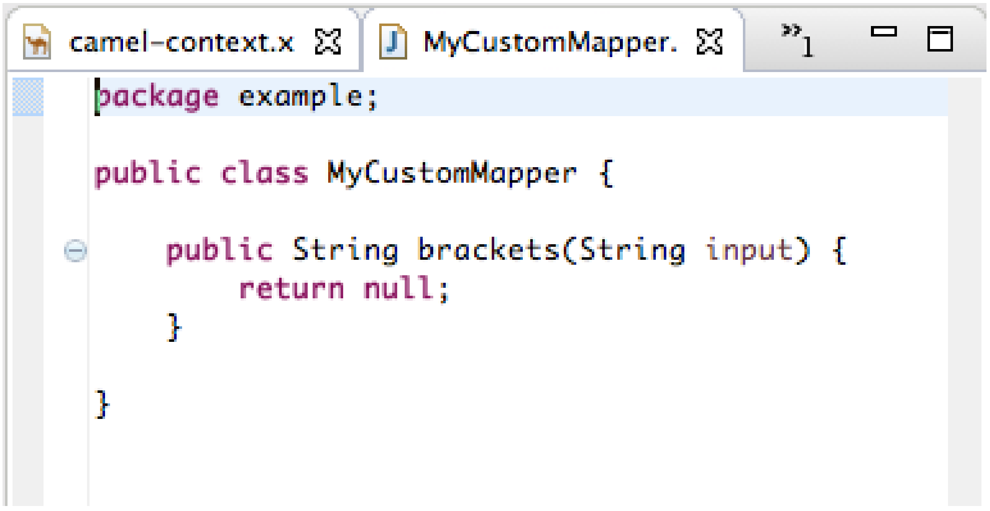

Add Custom Transformation

Select or create the Java class and method that implements the custom transformation

Class: example.MyCustomMapper

Method: String brackets(String input)

7. Click **OK** to open the **`MyCustomMapper.java`** file in the Java editor:

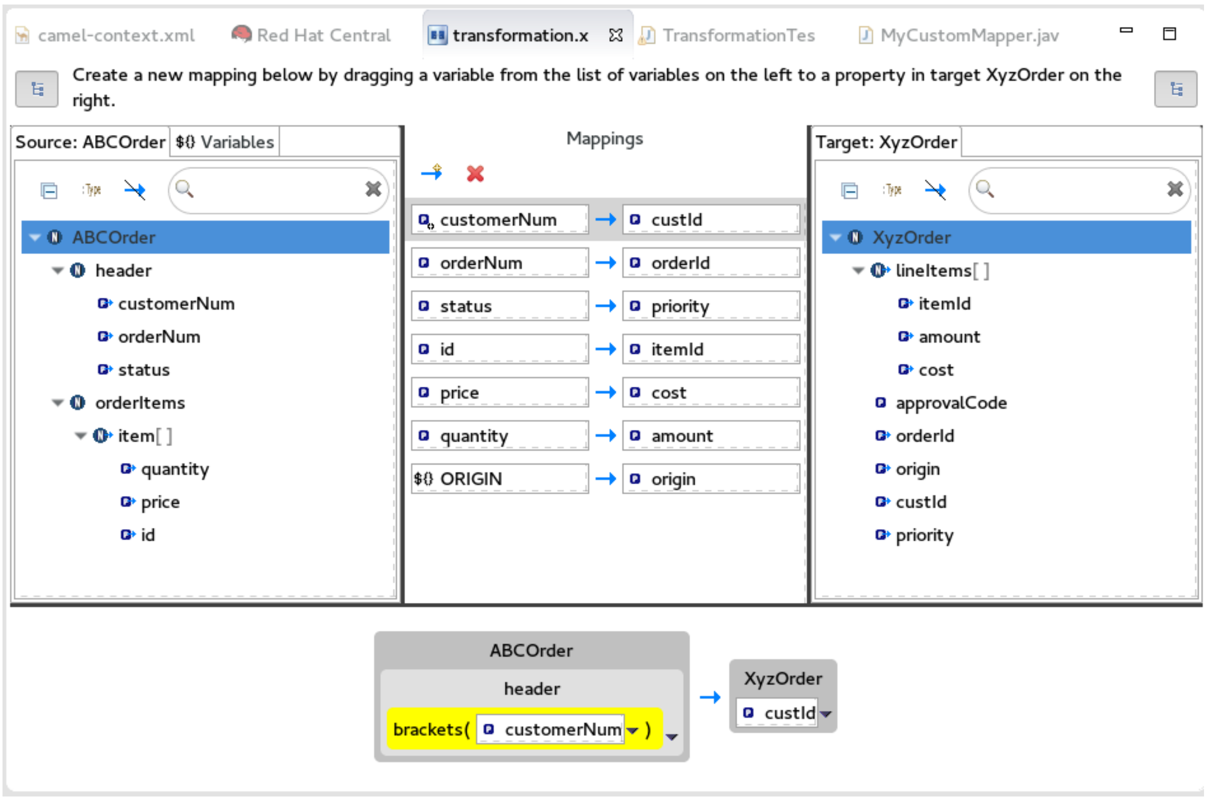


```
package example;

public class MyCustomMapper {

    public String brackets(String input) {
        return null;
    }

}
```

8. Edit the **`brackets`** method to change the last line **`return null;`** to this:

```
return "[" + input + "]";
```

9. Click the **transformation.xml** tab to switch back to the transformation editor.



The details pane shows that the **brackets** method has been associated with the **customerNum** data item.

The **brackets** method is executed on the source input before it is sent to the target system.

10. Run a JUnit test on your **TransformationTest.java** file. For details, see the section called “Creating the transformation test file and running the JUnit test”.
The **Console** view displays the JSON-formatted output data:

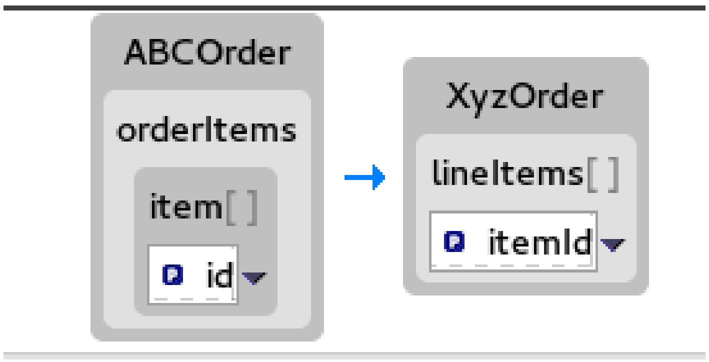
```
{"custId":"[ACME-
123]","priority":"GOLD","orderId":"ORDER1","origin":"Web",
"approvalCode":"AUTO_OK","lineItems":
[{"itemId":"PICKLE","amount":1000,"cost":2.25},
{"itemId":"BANANA","amount":400,"cost":1.25}]}
```

Mapping a simple data item to a data item in a collection

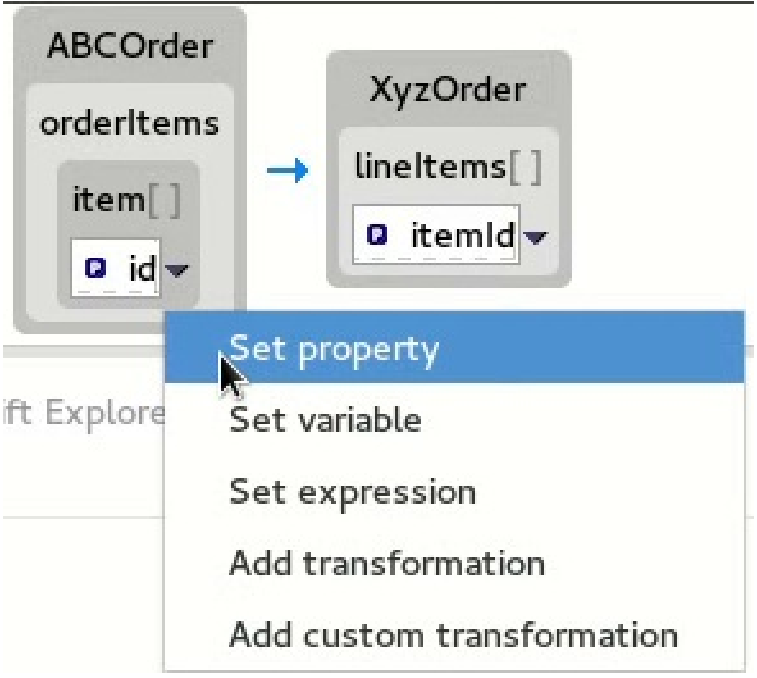
In this tutorial, you will modify an existing mapping that maps all **id**s in the Source to the **itemId**s in the Target. The new mapping will map the **customerNum** data item in the Source to the **itemId** of the second item in the **lineItems** collection in the Target.

With this change, no **id**s in the Source will be mapped to **itemId**s in the Target.

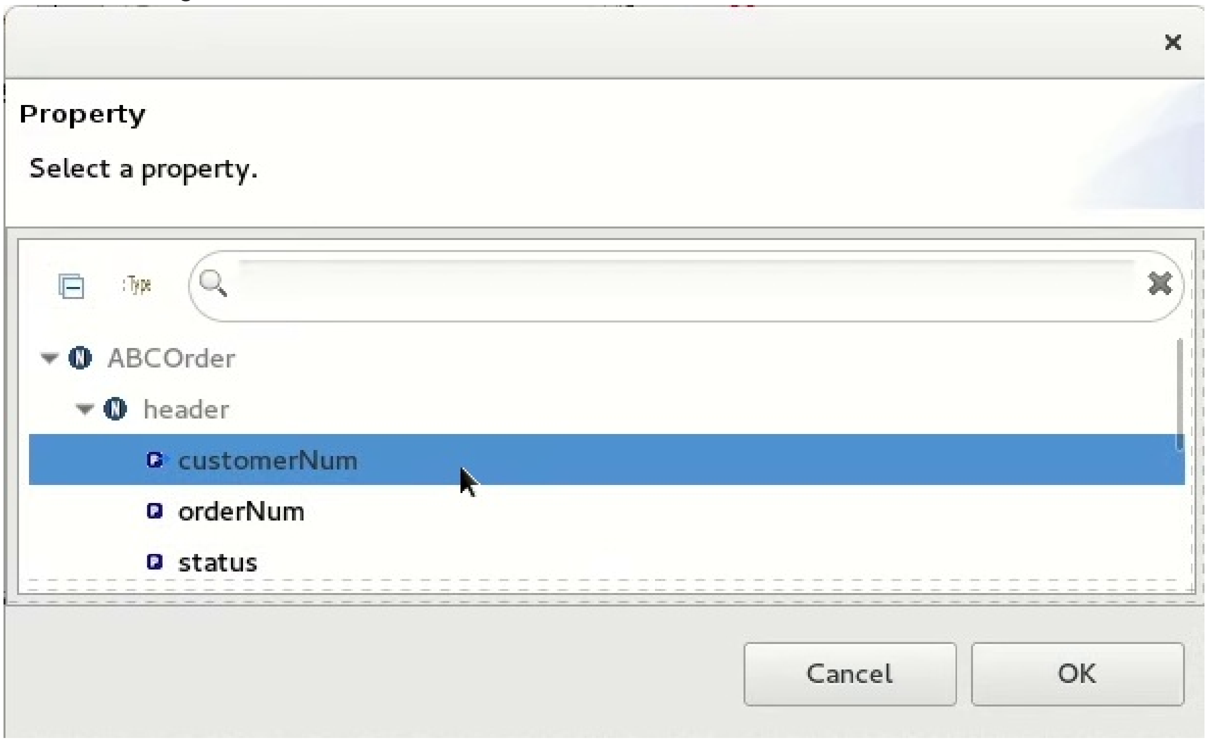
1. In the **Mappings** panel, select the mapping **id** —> **itemId** to display the mapping in the details pane.



2. On the Source box, click ▾ to open the drop-down menu, and select **Set property**.



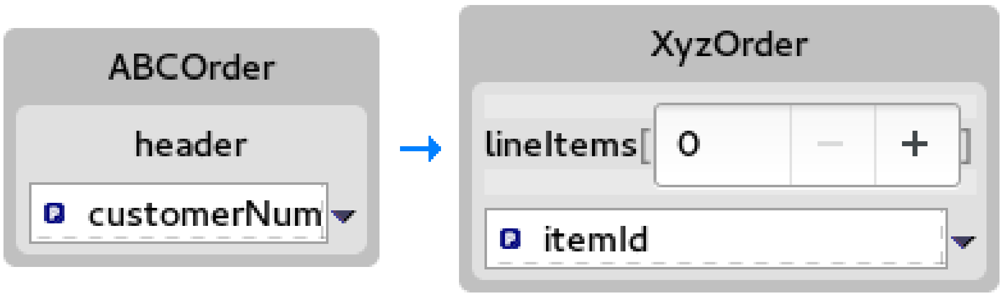
3. In the **Select a property** page, expand the **`header`** node and select **`customerNum`**. Click **OK** to save the changes.



4. The details pane now shows that **`XyzOrder`** has a **`lineItems`** field. Click the toggle button next

to **`lineItems`** to increase its value to **`1`**.

> **NOTE**
>
> Indexes are zero-based, so a value of **`1`** selects the second instance of **`itemId`** in the collection.



Notice that the details pane shows **`customerNum`** mapped to the **`itemId`** of the second item in the **`lineItems`** collection.

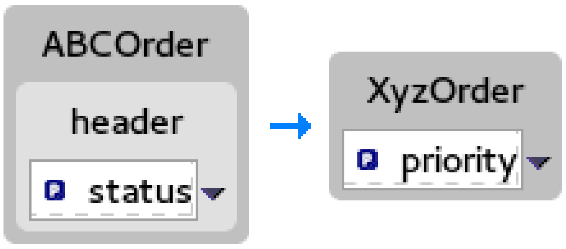
5. Run a JUnit test on your **`TransformationTest.java`** file. For details, see the section called “Creating the transformation test file and running the JUnit test”.
   The **Console** view displays the JSON-formatted output data:

```
{"custId":"[ACME-
123]","priority":"GOLD","orderId":"ORDER1","origin":"Web",
"approvalCode":"AUTO_OK","lineItems":[{"amount":1000,"cost":2.25},
{"itemId":"ACME-123","amount":400,"cost":1.25}]}
```

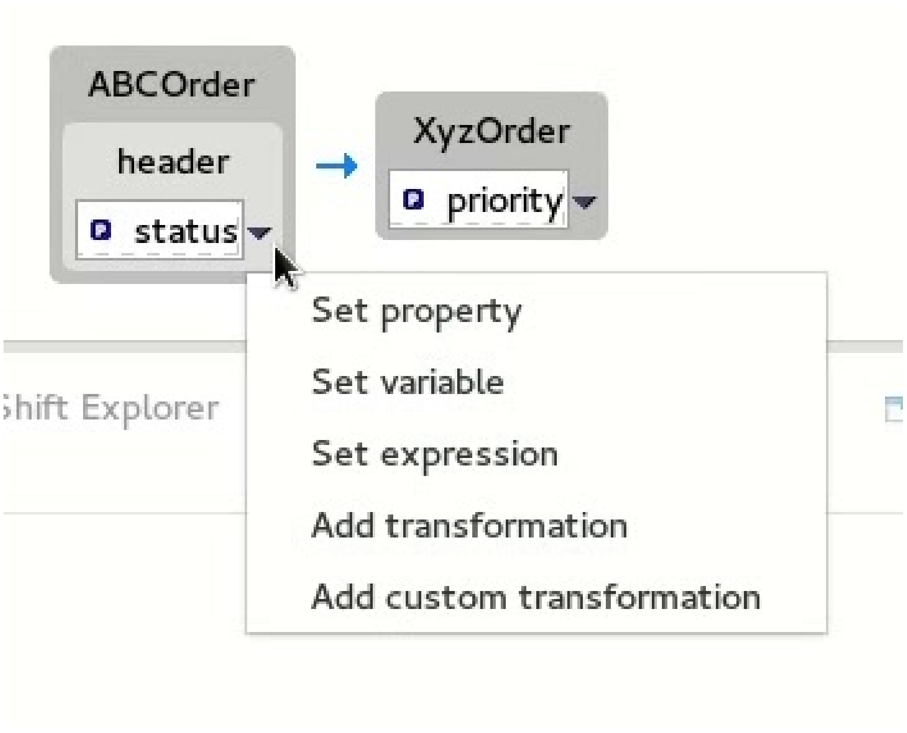
Adding a built-in function to a mapped data item

You can use the built-in string-related functions to apply transformations to mapped data items.

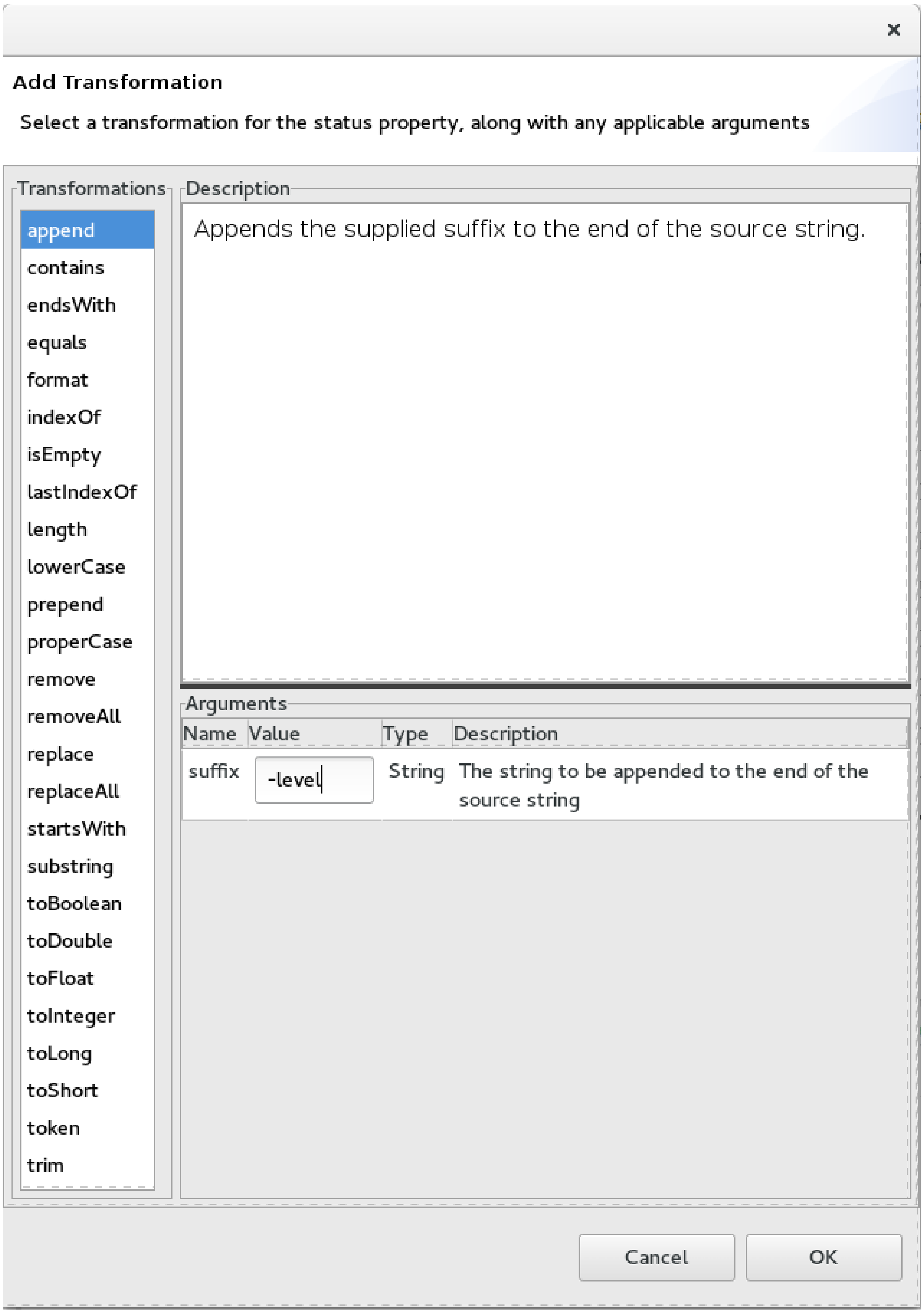
1. In the **Transformations** panel, select the **`status`** to **`priority`** mapping to populate the details pane.



2. In the Source box, click ▾ to open the drop-down menu, and select **Add transformation**.



3. In the **Transformations** pane, select **append**, and in the **Arguments** pane, enter **`-level`** for the value of **suffix**.
   This **`append`** function adds the specified suffix to the end of the**`status`** string before mapping it to the target **`priority`** data item.



Add Transformation

Select a transformation for the status property, along with any applicable arguments

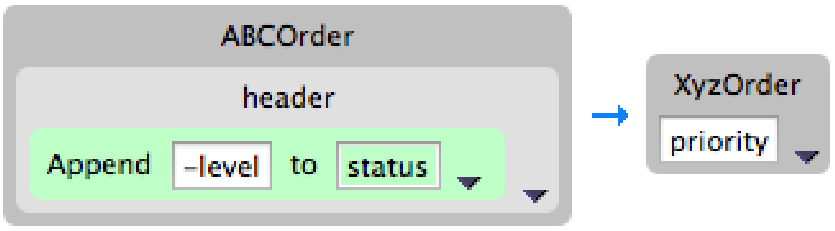
Description

Appends the supplied suffix to the end of the source string.

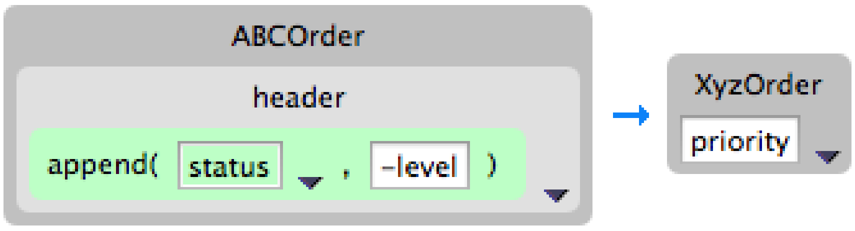
Arguments

Name	Value	Type	Description
suffix	-level	String	The string to be appended to the end of the source string

4. Click **OK**.



By default, the details pane displays the results of adding the **append** function to the **`status`** data item in a user-friendly format. You can change this formatting by clicking the right-most ▾ on the Source box, and selecting **Show standard formatting**.



5. Run a JUnit test on your **`TransformationTest.java`** file. For details, see the section called “Creating the transformation test file and running the JUnit test”.
   The **Console** view displays the JSON-formatted output data:

   ```
   {"custId":"[ACME-123]","priority":"GOLD-
   level","orderId":"ORDER1","origin":"Web",
   "approvalCode":"AUTO_OK","lineItems":[{"amount":1000,"cost":2.25},
   {"itemId":"ACME-123",
   "amount":400,"cost":1.25}]}
   ```

Publishing a Fuse Integration project with data transformation to a Red Hat Fuse

```
server
```

Before you publish your data transformation project to a Fuse server (see Chapter 29, *Publishing Fuse Integration Projects to a Server*), you need to install the following features in the Fuse runtime:

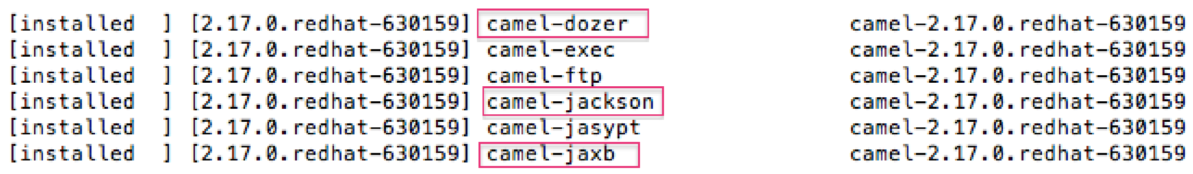
- **`camel-dozer`**
- **`camel-jackson`**
- **`camel-jaxb`**

To install the required features on the Fuse runtime:

1. If not already there, switch to the **Fuse Integration** perspective.
2. If necessary, add the Fuse server to the **Servers** list (see Section 28.1, “Adding a Server”).
3. Start the Fuse Server (see Section 28.2, “Starting a Server”), and wait for the JBoss Fuse shell to appear in the **Terminal** view.
4. For each of the required **`camel-`** features, at the **`JBossFuse:admin@root>`** prompt type:
   `features:install camel-<featureName>`

   Where *featureName* is one of **`dozer`**, **`jackson`**, or **`jaxb`**.
5. To verify that each of the features was successfully installed, at the **`JBossFuse:admin@root>`** prompt type:
   `features:list --ordered --installed`

   You should see the camel features you just installed in the output listing:



```
[installed  ] [2.17.0.redhat-630159] camel-dozer                camel-2.17.0.redhat-630159
[installed  ] [2.17.0.redhat-630159] camel-exec                 camel-2.17.0.redhat-630159
[installed  ] [2.17.0.redhat-630159] camel-ftp                  camel-2.17.0.redhat-630159
[installed  ] [2.17.0.redhat-630159] camel-jackson              camel-2.17.0.redhat-630159
[installed  ] [2.17.0.redhat-630159] camel-jasypt               camel-2.17.0.redhat-630159
[installed  ] [2.17.0.redhat-630159] camel-jaxb                 camel-2.17.0.redhat-630159
```

CHAPTER 10. DEVELOPING EXTENSIONS FOR IGNITE INTEGRATIONS

Ignite is a Red Hat Fuse feature that provides a web interface for integrating applications. Without writing code, a business expert can use Ignite to connect to applications and optionally operate on data between connections to different applications. If Ignite does not provide a feature that an integrator needs, then a developer can create an extension that defines the needed behavior.

You can use Fuse Tooling to develop extensions that provide features for use in Ignite. An extension defines:

- One or more custom **steps** that operate on data between connections in an integration or
- One custom **connector**

In Ignite, a **connector** represents a specific application to obtain data from or send data to. Each connector is a template for creating a connection to that specific application. For example, the Salesforce connector is the template for creating a connection to Salesforce. If Ignite does not provide a connector that the Ignite user needs, you can develop an extension that defines a custom connector.

In Ignite, a data operation that happens between connections in an integration is referred to as a **step**. Ignite provides steps for operations such as filtering and mapping data. To operate on data between connections in ways that are not provided by Ignite built-in steps, you can develop an Ignite extension that defines one or more custom steps.

10.1. OVERVIEW OF TASKS

Here is an overview of the tasks for developing an Ignite extension:

1. In Red Hat Developer Studio, create an Ignite extension project and select **Custom Connector** or **Custom Step** as the extension type.
2. Depending on the extension type, write the code for the extension:
   - For a **Custom Connector**: Define the base Camel component, the connector icon, global connector properties, and the connector actions.
   - For a **Custom Step**: Add routes, define actions, and specify any dependencies.
3. Build a **`.jar`** file.
4. Provide the **`.jar`** file to the Ignite user.

The Ignite user uploads the **`.jar`** file to Ignite and can then add the custom connector or custom step defined in the extension to an integration. The Ignite user uploads the .jar file to Ignite, which makes the custom connector or custom step(s) available for use. For information about Ignite and how to create integrations, see Integrating Applications with Ignite.

10.2. PREREQUISITES

Before you begin, you need the following information and knowledge:

- A description of the required functionality for the Ignite custom connector or step (from the Ignite user).

- The Ignite version number for the extension.
- For a custom connector, an icon image file in PNG or SVG format. Ignite uses this icon when it displays the flow of an integration. If you do not provide an icon, then Ignite generates one when the .jar that contains the extension is uploaded.
- You should be familiar with:
  - Ignite
  - Spring Boot XML or Java
  - Apache Camel routes (if you want to create a route-based step extension)
  - JSON
  - Maven

10.3. CREATING A CUSTOM CONNECTOR

In Ignite, a custom connector consists of one or more connection configuration parameters, one or more connection actions, and optional configuration parameters for each action.

Here is an overview of the tasks for developing a custom connector:

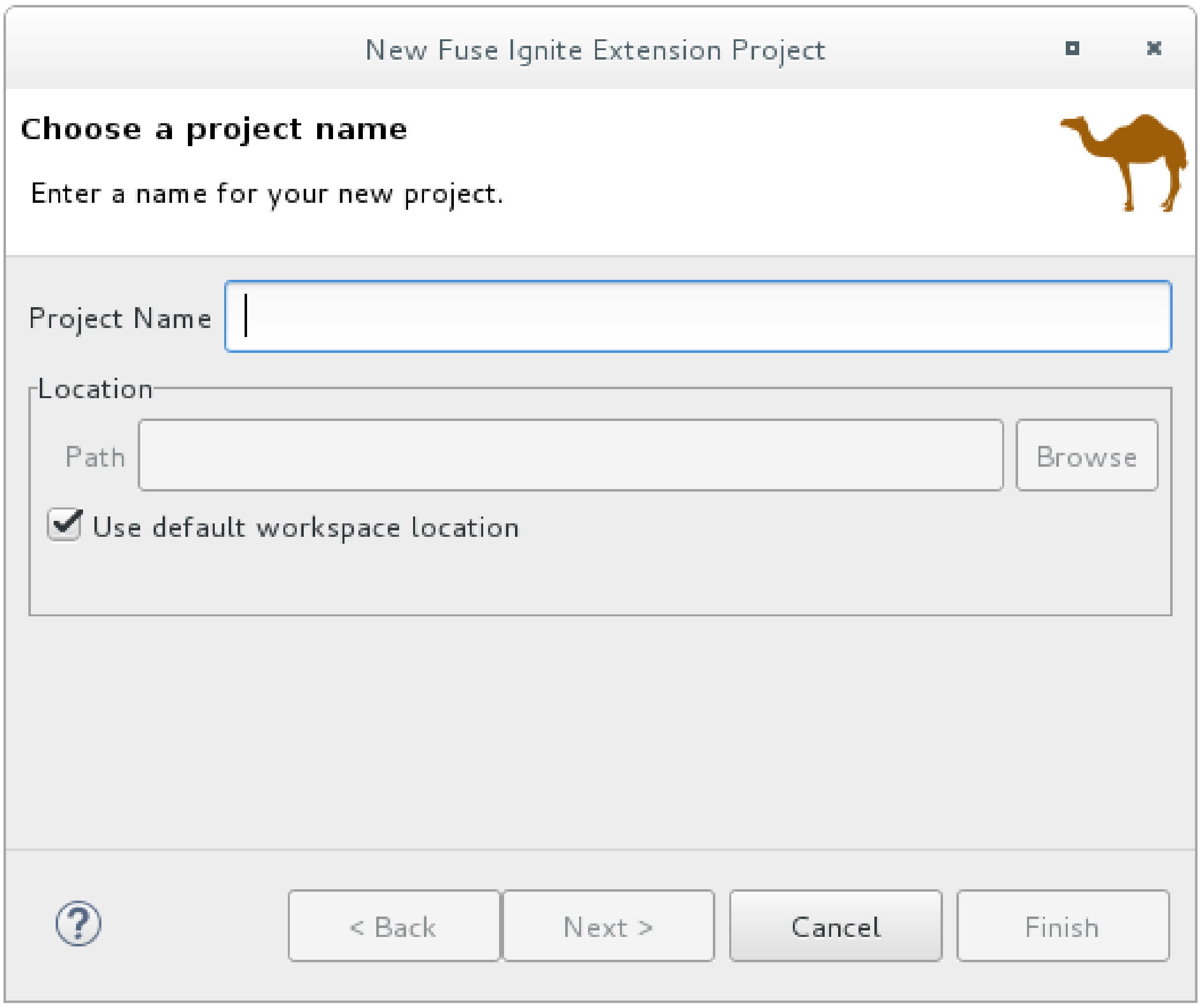
1. In Red Hat Developer Studio, create an Ignite extension project and select **Custom Connector** as the extension type.
2. Write the code for the extension. Define the base Camel component, the connector icon, global connector properties, and the connector actions.

10.3.1. Creating an Ignite Extension project for a custom connector

A Fuse Tooling Ignite extension project provides a starting point for a custom connector.

To create a Fuse Tooling Ignite extension project, follow these steps:

1. In Red Hat Developer Studio, select **New** → **Project** → **Red Hat Fuse** → **Fuse Ignite Extension Project**.
   The **New Fuse Ignite Extension Project** wizard opens.


New Fuse Ignite Extension Project

Choose a project name

Enter a name for your new project.

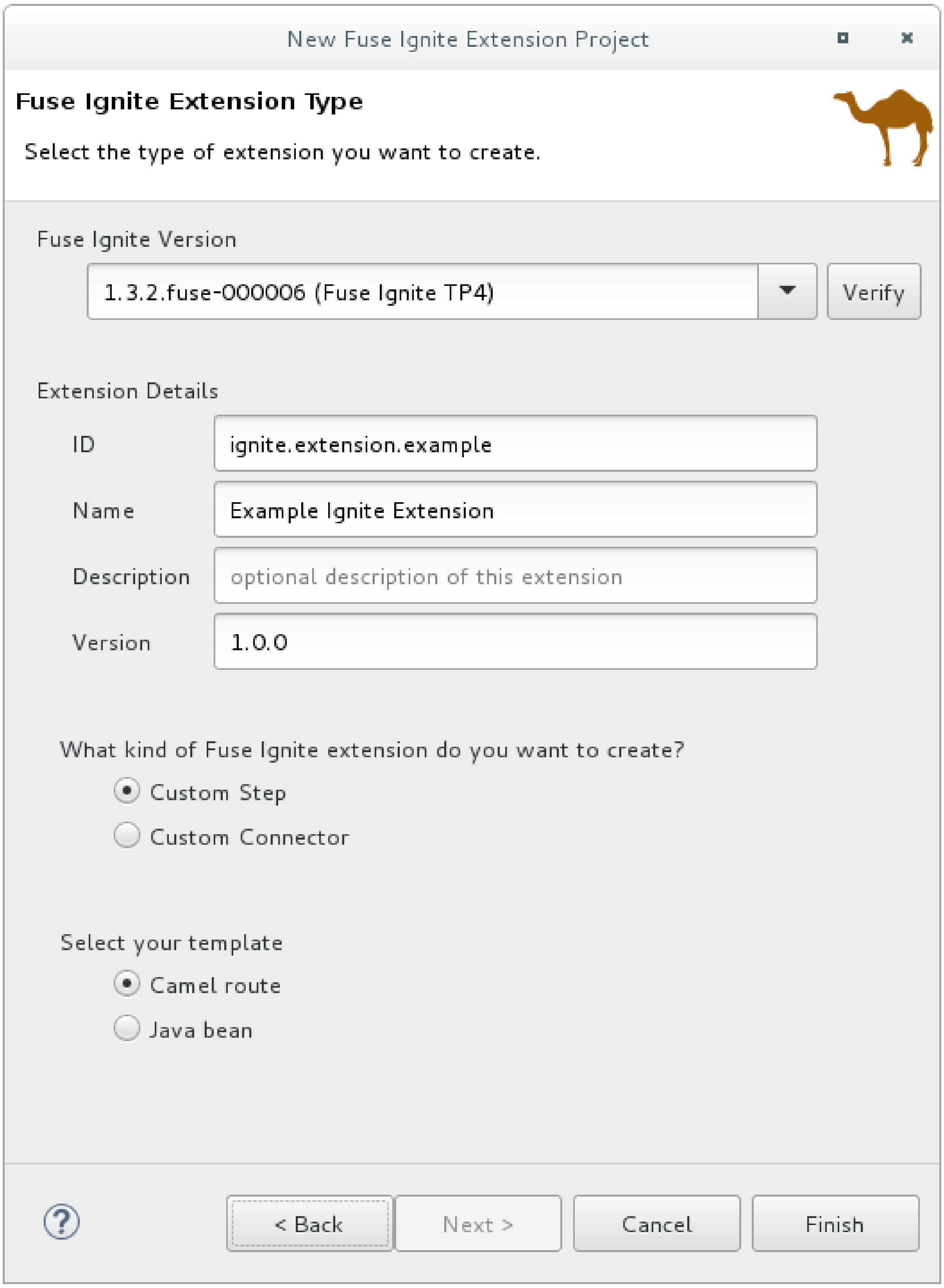
Project Name

Location

Path

[✓] Use default workspace location

2. Enter the name and location for the project, then click **Next**.


New Fuse Ignite Extension Project

Fuse Ignite Extension Type

Select the type of extension you want to create.

Fuse Ignite Version: 1.3.2.fuse-000006 (Fuse Ignite TP4)

Extension Details

ID	ignite.extension.example
Name	Example Ignite Extension
Description	optional description of this extension
Version	1.0.0

What kind of Fuse Ignite extension do you want to create?

[✓ Custom Step] Custom Connector

Select your template

[✓ Camel route] Java bean

3. Select the Fuse Ignite version.

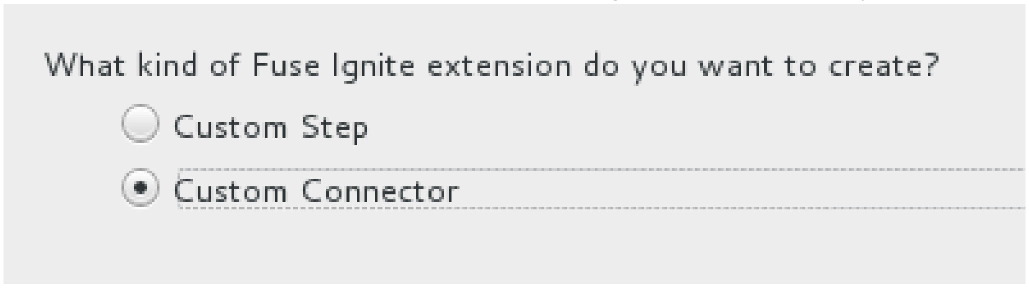
4. Specify the following extension details:

   - **ID** — A value that you define and that is unique in the Ignite environment. This value will be visible in Ignite when the Ignite user imports the extension **`.jar`** file.

   - **Name** — The name of the extension. This value will be visible in Ignite as the extension name. In Ignite, on the **Customizations** → **Extensions** tab, the user can see a list of the names and descriptions of extensions that have been uploaded to Ignite.

- **Description** — An optional description of the extension content.
- **Version** — The version of the extension. For example, if this is the initial version, you could type **`1.0`**. If you are updating a version, you could type **`1.1`** or **`2.0`**.

5. Select **Custom Connector** for the kind of Fuse Ignite extension that you want to create.



What kind of Fuse Ignite extension do you want to create?

Custom Step [✓ Custom Connector]

6. Click **Finish**.

The new project appears in the Red Hat Developer Studio **Project Explorer** view. It includes the following files for a custom connector extension:

- In the **`src/main/resources/META-INF/syndesis`** folder:
  - A descriptor file: **`syndesis-extension-definition.json`**
    This is the file that you edit to: * **Add top-level global properties, connector actions, and action properties.** * Change the Extension Id, Name, Version, or Description values.
  - A default icon image file: **`icon.png`**
    You can optionally replace this file with your own icon image (PNG or SVG) file.
- A Maven Project Object Model file: **`pom.xml`**
  This file contains information about the project and configuration details used by Maven to build the project, including default extension dependencies. You edit this file to add custom dependencies. The scope for any dependency that Red Hat ships is provided, for example:

```
<dependency>
    <groupId>io.syndesis.extension</groupId>
    <artifactId>extension-api</artifactId>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-core</artifactId>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter</artifactId>
    <scope>provided</scope>
</dependency>
```

10.3.2. Writing code for the custom connector

After you create the Ignite extension project, you write the code that defines the custom connector elements based on the description of the required functionality provided to you by the Ignite user. The Table 10.1, “Custom connector elements” table shows how the elements of the custom connector that you create in Fuse Tooling correspond to elements in Ignite.

Table 10.1. Custom connector elements

Fuse Tooling element	Ignite element	Description
Global (top-level) property	Connection configuration parameter	When an Ignite user creates a connection from this connector, the user specifies a value for this property as part of the configuration of the connection.
Action	Connection action	In Ignite, for a connection created from this connector, an Ignite user selects one of these actions.
Property defined in an action	An action configuration parameter	When an Ignite user configures the action that the connection performs, the Ignite user specifies a value for this property as part of the configuration of the action.

To write the code that implements a custom connector for Ignite:

1. Open the syndesis-extension-definition.json file in the Editor view and write the code that defines the global properties, the actions that the custom connector can perform, and each action’s properties.
   Each **global property** corresponds to a connection configuration parameter in Ignite. Each action property corresponds to an Ignite connection actionconfiguration parameter. In Ignite, when the user selects a custom connector, Ignite prompts for values for each connection configuration parameter. A custom connector can be for an application that uses the OAuth protocol. In this case, be sure to specify a global property for the OAuth client ID and another global property for the OAuth client secret. The Ignite user will need to specify values for these parameters for a connection created from this connector to work.

   Each **connector action** declares a base Camel component scheme.

   The example provided by the New Fuse Ignite Extension Project wizard uses the **`telegram`** Camel component scheme:

```
{
  "schemaVersion" : "v1",
  "name" : "Example Ignite Extension",
  "extensionId" : "ignite.extension.example",
  "version" : "1.0.0",
  "actions" : [ {
    "id" : "io.syndesis:telegram-chat-from-action",
    "name" : "Chat Messages",
    "description" : "Receive all messages sent to the chat bot",
    "descriptor" : {
      "componentScheme" : "telegram",
      "inputDataShape" : {
        "kind" : "none"
      },
      "outputDataShape" : {
        "kind" : "java",
        "type" :
"org.apache.camel.component.telegram.model.IncomingMessage"
      },
      "configuredProperties" : {
        "type" : "bots"
      }
    },
    "actionType" : "connector",
    "pattern" : "From"
  }, {
    "id" : "io.syndesis:telegram-chat-to-action",
    "name" : "Send a chat Messages",
    "description" : "Send a messages to the chat (through the
bot).",
    "descriptor" : {
      "componentScheme" : "telegram",
      "inputDataShape" : {
        "kind" : "java",
        "type" : "java.lang.String"
      },
      "outputDataShape" : {
        "kind" : "none"
      },
      "propertyDefinitionSteps" : [ {
        "description" : "Chat id",
        "name" : "chatId",
        "properties" : {
          "chatId" : {
            "kind" : "parameter",
            "displayName" : "Chat Id",
            "type" : "string",
            "javaType" : "String",
            "description" : "The telegram's Chat Id, if not set will
use CamelTelegramChatId from the incoming exchange."
          }
        }
      } ],
      "configuredProperties" : {
        "type" : "bots"
      }
    },
    "actionType" : "connector",
    "pattern" : "To"
  } ],
  "properties" : {
    "authorizationToken" : {
      "kind" : "property",
      "displayName" : "Authorization Token",
      "group" : "security",
      "label" : "security",
      "required" : true,
      "type" : "string",
      "javaType" : "java.lang.String",
      "secret" : true,
        "description" : "Telegram Bot Authorization Token"
      }
    }
  }
```

2. If the custom connector requires additional dependencies, add them to the project's **`pom.xml`** file. The default scope for dependencies is runtime. If you add a dependency that Red Hat ships, define its scope as provided, for example:

```
<dependencies>
    <dependency>
          <groupId>org.apache.camel</groupId>
          <artifactId>camel-telegram</artifactId>
          <scope>provided</scope>
    </dependency>
      </dependencies>
```

When you finish writing the code for the custom connector, build the **`.jar`** file as described in Section 10.5, "Building the Ignite extension JAR file".

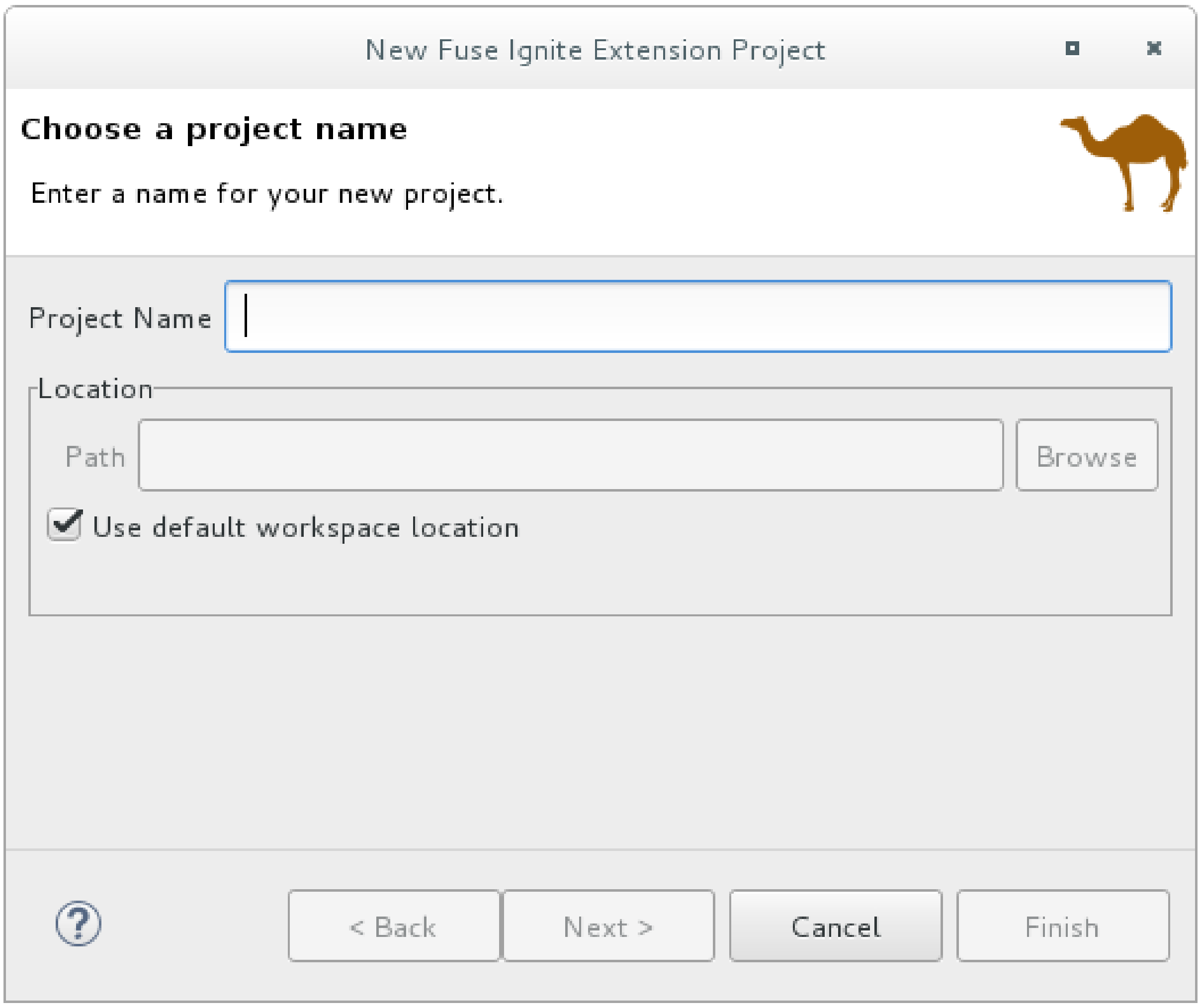
10.4. CREATING CUSTOM STEPS

After you create the Ignite extension project, you write the code that defines the custom steps based on the description of the required functionality provided to you by the Ignite user. Within a single extension, you can define more than one custom step and you can define each custom step with Camel routes or with Java beans.

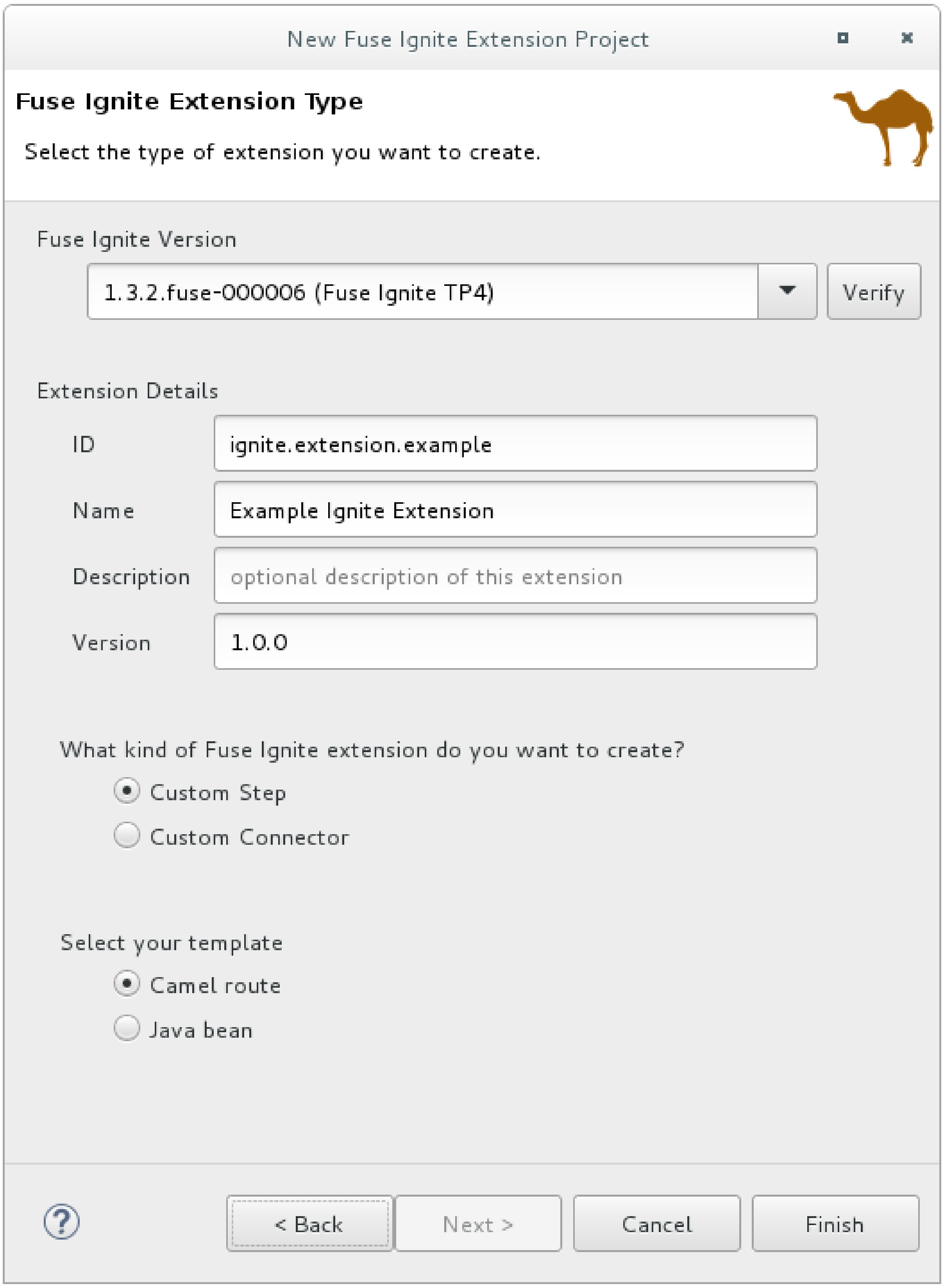
10.4.1. Creating an Ignite Extension project for custom steps

To create a Fuse Tooling Ignite extension project, follow these steps:

1. In Red Hat Developer Studio, select **New** → **Project** → **Red Hat Fuse** → **Fuse Ignite Extension Project**.
   The **New Fuse Ignite Extension Project** wizard opens.


New Fuse Ignite Extension Project
Choose a project name
Enter a name for your new project.
Project Name
Location
Path
Browse
Use default workspace location
< Back
Next >
Cancel
Finish

2. Enter the name and location for the project, then click **Next**.



Fuse Ignite Extension Type

Select the type of extension you want to create.

Fuse Ignite Version: 1.3.2.fuse-000006 (Fuse Ignite TP4)

Extension Details

ID	ignite.extension.example
Name	Example Ignite Extension
Description	optional description of this extension
Version	1.0.0

What kind of Fuse Ignite extension do you want to create?
[✓ Custom Step] Custom Connector

Select your template
[✓ Camel route] Java bean

3. Select the Fuse Ignite version.

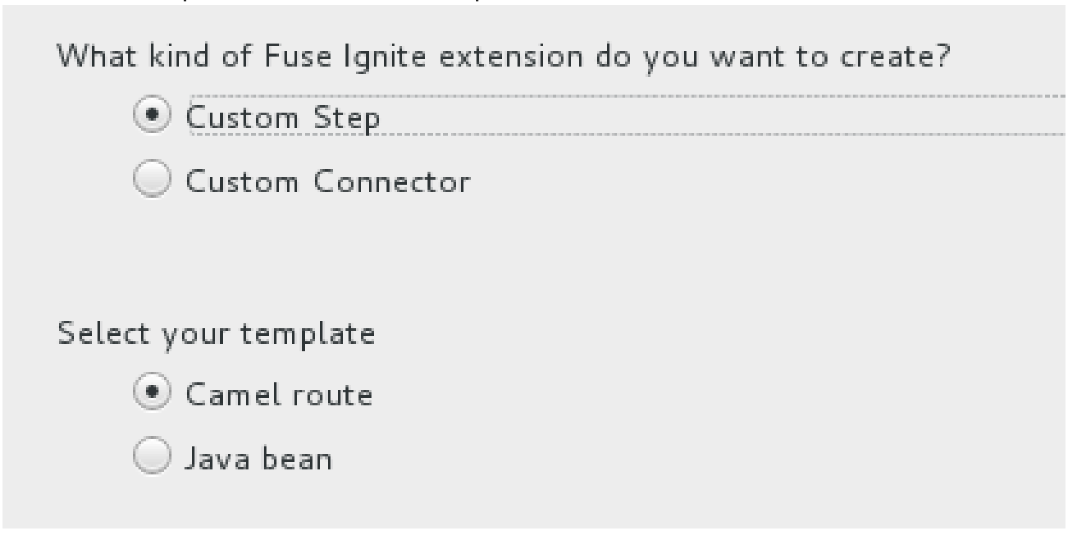
4. Specify the following extension details:

   - **ID** — A value that you define and that is unique in the Ignite environment. This value will be visible in Ignite when the Ignite user imports the extension **`.jar`** file.

   - **Name** — The name of the extension. This value will be visible in Ignite as the extension name. In Ignite, on the **Customizations** → **Extensions** tab, the user can see a list of the names and descriptions of extensions that have been uploaded to Ignite.

- **Description** — An optional description of the extension content.
- **Version** — The version of the extension. For example, if this is the initial version, you could type **`1.0`**. If you are updating a version, you could type **`1.1`** or **`2.0`**.

5. Select **Custom Step** for the kind of Fuse Ignite extension that you want to create.
6. Select the template for the custom step:



What kind of Fuse Ignite extension do you want to create?

(●) Custom Step

( ) Custom Connector

Select your template

(●) Camel route

( ) Java bean

- **Camel route** — Provides a sample Camel route.
- **Java bean** — Provides a sample Java bean.

NOTE

The template that you select provides a starting point for your project. If you want to create one or more custom steps based on Camel routes and one or more other custom steps based on Java beans within the same extension, start with one of the templates and then add the needed file and dependencies for the other type of custom step.

7. Click **Finish**.

The new project appears in the Red Hat Developer Studio **Project Explorer** view. It includes the following files depending on the template that you selected for the custom step:

- In the **`src/main/resources/META-INF/syndesis`** folder:
  - A descriptor file: **`syndesis-extension-definition.json`**
    This is the file that you edit to: * **Add one or more actions. An action in the `.json `file becomes a custom step in Ignite. In an action element, a property in the `.json `file becomes a step configuration parameter in Ignite.** * Change the Extension Id, Name, Version, or Description values.
  - For a Camel route template, a Camel context file: **`extensions/log-body-action.xml`**
    This file contains a sample route with a log component. You customize the Camel routes in this file.
  - For a Java bean template, a Java file: **`extensions/extension.java`**
    This file contains a sample POJO-based logging extension.

- A Maven Project Object Model file: **`pom.xml`**
  This file contains information about the project and configuration details used by Maven to build the project, including default extension dependencies. You edit this file to add custom dependencies. The scope for any dependency that Red Hat ships is provided, for example:

```
<dependency>
    <groupId>io.syndesis.extension</groupId>
    <artifactId>extension-api</artifactId>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-core</artifactId>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter</artifactId>
    <scope>provided</scope>
</dependency>
```

10.4.2. Writing code for the custom step

After you create the Ignite extension project, you write the code that defines the custom step(s)based on the description of the required functionality provided to you by the Ignite user.

Table 10.2, “Custom step elements” shows how the elements of the custom step that you create in Fuse Tooling correspond to elements in Ignite.

Table 10.2. Custom step elements

Fuse Tooling element	Ignite element	Description
Action	Custom Step	In Ignite, after the user imports the step extension, the custom step(s) appear(s) on the **Choose a step** page.
Property defined in an action	A custom step configuration parameter	In Ignite, when the user selects a custom step, Ignite prompts for values for configuration parameters.

To write the code that implements a custom step for Ignite:

1. For a Camel route-based step, in the **`extension.xml`** file, create routes that address the purpose of the extension. The entrypoint of each route must match the entrypoint that you define in the **`syndesis-extension-definition.json`** file, as described in Step 2.
   For a Java bean-based step, edit the **`java`** file.

2. In the **`syndesis-extension-definition.json`** file, write the code that defines the actions and their properties. You need a new action for each entrypoint.

Each action that you create corresponds to a custom step in Ignite. You can use different types of code for each action. That is, you can use a Camel route for one action and a Java bean for another action.

Each property corresponds to an Ignite step configuration parameter. In Ignite, when the user selects a custom step, Ignite prompts for values for configuration parameters. For example, a custom log step might have a level parameter that indicates how much information to send to the log.

Here is the template for the **`.json`** file that contains the extension metadata, including properties that will be filled in by the user in Ignite after uploading the extension and adding its custom step to an integration:

```
{
  "actions": [
    {
      "actionType": "extension",
      "id": "${actionId}",
      "name": "Action Name",
      "description": "Action Description",
      "tags": [
           "xml"
      ],
      "descriptor": {
        "kind": "ENDPOINT|BEAN|STEP",
        "entrypoint": "direct:${actionId}",
        "inputDataShape": {
          "kind": "any"
        },
        "outputDataShape": {
          "kind": "any"
        },
        "propertyDefinitionSteps": []
      }
    }
  ],
  "tags": [
    "feature",
    "experimental"
  ]
}
```

NOTE

The tags are ignored in this release. They are reserved for future use.

3. To edit the extension dependencies, open the `pom.xml `file in the editor. If you add a dependency, you must define its scope.

When you finish writing the code for the custom step(s), build the **`.jar`** file as described in Section 10.5, “Building the Ignite extension JAR file”.

10.5. BUILDING THE IGNITE EXTENSION JAR FILE

To build the `.jar` file for the extension:

1. In the **Project Explorer** view, right-click the project.

2. From the context menu, select **Run As → Maven clean verify**.

3. In the **Console** view, you can monitor the progress of the build.

4. When the build is complete, refresh the target folder in the **Project Explorer** view (select the project and then press **F5**).

5. In the **Project Explorer** view, open the target folder to see the generated `.jar` file:
   The name of the .jar file follows Maven defaults: **`${artifactId}-${version}.jar`**

   For example: **`custom:step-camel-1.0.0.jar`**

   This `.jar` file defines the extension, its required dependencies, and its metadata: Extension Id, Name, Version, Tags, and Description. For example:

```
{
    "schemaVersion" : "v1",
    "name" : "Example Ignite Extension",
    "description" : "Logs a message body with a prefix",
    "extensionId" : "ignite.extension.example",
    "version" : "1.0.0",
    "actions" : [ {
        "id" : "Log-body",
        "name" : "Log Body",
        "description" : "A simple xml Body Log with a prefix",
        "descriptor" : {
            "kind" : "ENDPOINT",
            "entrypoint" : "direct:log-xml",
            "resource" : "classpath:META-
INF/syndesis/extensions/log-body-action.xml",
            "inputDataShape" : {
                "kind" : "any"
            },
            "outputDataShape" : {
                "kind" : "any"
            },
            "propertyDefinitionSteps" : [ {
                "description" : "Define your Log message",
                "name" : "Log Body",
                "properties" : {
                    "prefix" : {
                        "componentProperty" : false,
                        "deprecated" : false,
                        "description" : "The Log body prefix
message",
                        "displayName" : "Log Prefix",
                        "javaType" : "String",
                        "kind" : "parameter",
                        "required" : false,
                        "secret" : false,
                        "type" : "string"
                    }
```

```
                  }
              } ]
          },
          "tags" : [ "xml" ],
          "actionType" : "step"
      } ],
      "dependencies" : [ {
          "type" : "MAVEN",
          "id" : "io.syndesis.extension:extension-api:jar:1.3.0.fuse-
000014"
      } ],
      "extensionType" : "Steps"
}
```

10.6. PROVIDING THE JAR FILE TO THE IGNITE USER

Provide the following to the Ignite user:

- The **`.jar`** file
- A document that describes the extension. For a step extension, include information about data shapes that each action in the step extension requires as input or provides as output (for data mapping).

In Ignite, the user uploads the **`.jar`** file as described in Integrating Applications with Ignite.

CHAPTER 11. CREATING A NEW CAMEL XML FILE

OVERVIEW

Apache Camel stores routes in an XML file that contains a camelContext element. When you create a new Fuse Integration project, the tooling provides an Apache Camel context (XML) file as part of the project by default.

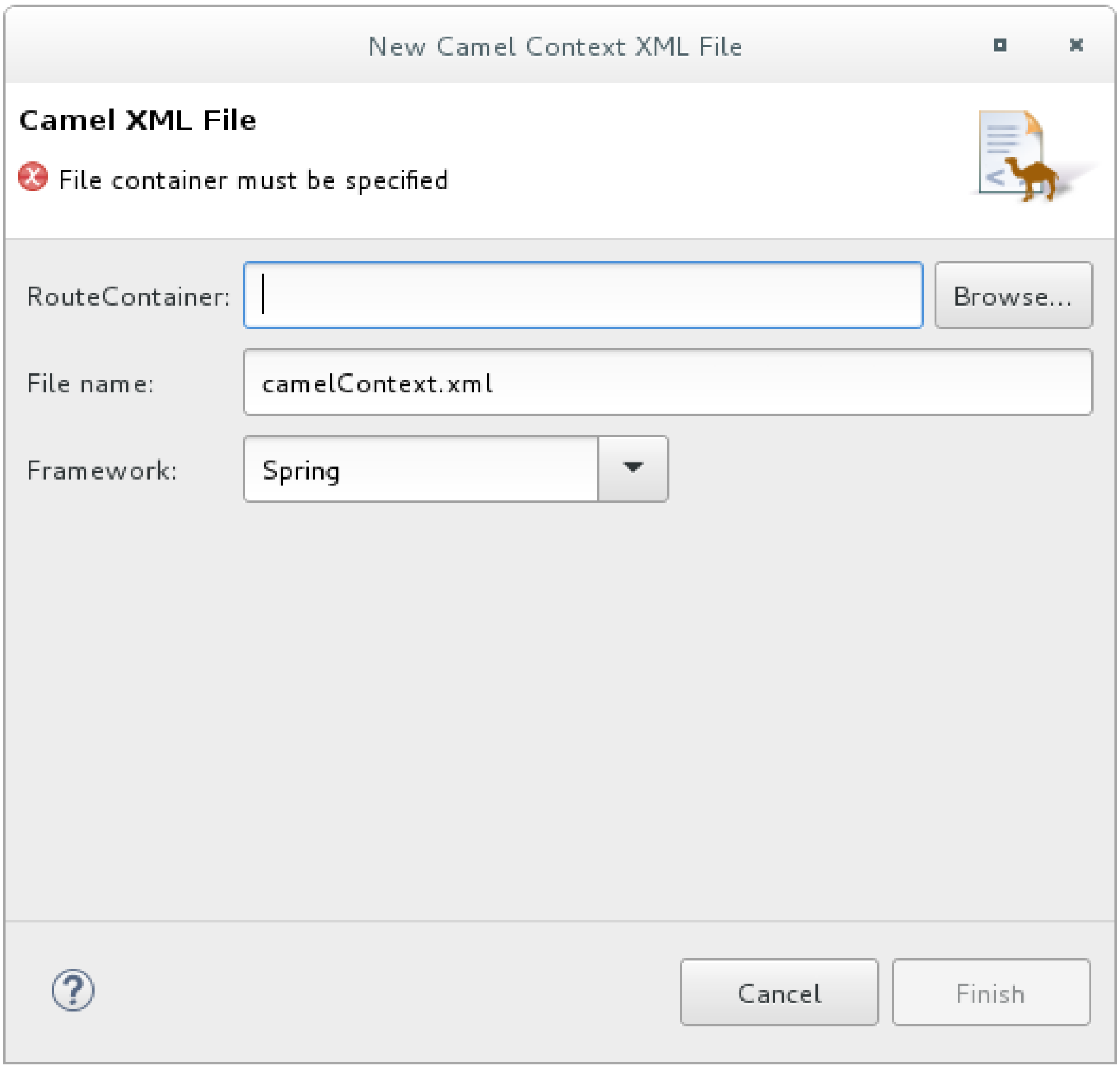
You can also add a new Camel XML file that includes all of the required namespaces preconfigured and a template camelContext element.

PROCEDURE

To add a new Apache Camel context file to your project:

1. Select **File** → **New** → **Camel XML File** from the main menu to open the **Camel XML File** wizard, as shown in Figure 11.1, “Camel XML File wizard”.

   Figure 11.1. Camel XML File wizard

   New Camel Context XML File

   Camel XML File

   File container must be specified

   RouteContainer:

   File name: camelContext.xml

   Framework: Spring

2. In **RouteContainer**, enter the location for the new file, or accept the default.

You can click Browse... to search for an appropriate location.

IMPORTANT

The Spring framework and the OSGi Blueprint framework require that all Apache Camel files be placed in specific locations under the project's **`META-INF`** or **`OSGI-INF`** folder:

- Spring - **`projectName/src/main/resources/META-INF/spring/`**
- OSGi Blueprint - **`projectName/src/main/resources/OSGI-INF/blueprint/`**

3. In **File Name**, enter a name for the new context file, or accept the default (**`camelContext.xml`**).
   The file's name cannot contain spaces or special characters, and it must be unique within the JVM.
4. In **Framework**, accept the default, or select which framework the routes will use:
   - **Spring** — [default] for routes that will be deployed in Spring containers, non-OSGi containers, or as standalone applications
   - **OSGi Blueprint** — for routes that will be deployed in OSGi containers
   - **Routes** — for routes that you can load and add into existing **`camelContext`**s
5. Click **Finish**.
   The new context file is added to the project and opened in the route editor.

CHAPTER 12. CHANGING THE CAMEL VERSION

As you work with a Fuse tooling project, you might want to change the Camel version that it uses. This can be helpful, for example, if you want to use a feature supported in a more recent Camel version or if you want to use a community version.

To change the Camel version that a project uses:

1. In **Project Explorer**, right-click the project for which you want to change the Camel version and select **Configure → Change Camel Version**.

2. In the **Change Camel Version** window, to the right of the **Camel Version** field, click the down caret to display available Camel versions.
   To use a community version of Apache Camel, you enter its version number, for example, **`2.19.2`**.

3. Select or enter the version you want and click **Finish**.

Fuse Tooling checks whether the version you selected is available and supported by Fuse Tooling. If it is then Fuse Tooling changes the Camel version and saves the project's updated **`pom.xml`** file. You receive an error message if the Camel version that you select is not available or not supported.

You can check the project's Camel version in its **`pom.xml`** file, in the **`<camel.version>`** element.

CHAPTER 13. IMPORTING AN EXISTING MAVEN PROJECT

OVERVIEW

You might want to import an existing project, for example, to use as a template or starting point for developing an application.

For example, the New Fuse Integration Project wizard points to the following Github repositories as sources for examples:

- https://github.com/apache/camel/tree/master/examples
- https://github.com/fabric8-quickstarts
- https://github.com/wildfly-extras/wildfly-camel-examples
- https://github.com/jboss-fuse/quickstarts

After you download an example project, you import it into Developer Studio.

PROCEDURE

To import an existing Maven project:

1. Select **File** → **Import** → **Maven** → **Existing Maven Projects**, and then click **Next**.
2. For the Root Directory, select the folder that contains the downloaded example projects.
3. In the list of Projects, check the projects that you want to import and then click **Finish**.

PART II. DEBUGGING ROUTING CONTEXTS

The Camel debugger includes many features for debugging locally and remotely running routing contexts:

- Setting conditional and unconditional breakpoints on nodes in the route editor
- Autolaunching the debugger and switching to the **Debug** perspective
- Interacting with the running routing context:
    - Switch between breakpoints to quickly compare variable values of message instances
    - Examine and change the value of variables of interest
    - Add variables of interest to the watch list to track them throughout the debug session
    - Disable and re-enable breakpoints on-the-fly
    - Track message flow graphically in the routing context runtime
    - Examine console logs to track Camel and debugger actions

NOTE

Before you can run the Camel debugger, you must set breakpoints on the nodes of interest displayed on the route editor’s canvas. Then you can run the Camel debugger on a project’s routing context `.xml` file to find the logic errors in it and fix them. Invoking the Camel debugger runs the routing context in debug mode and opens the Debug Perspective.

CHAPTER 14. SETTING BREAKPOINTS

OVERVIEW

To set breakpoints, your project's routing context `.xml` file must be open in the route editor's **Design** tab.

The Camel debugger supports two types of breakpoints:

- Unconditional breakpoints — triggered whenever one is encountered during a debugging session
- Conditional breakpoints — triggered only when the breakpoint's specified condition is met during a debugging session

NOTE

You cannot set breakpoints on consumer endpoints or on `when` or `otherwise` nodes.

SETTING UNCONDITIONAL BREAKPOINTS

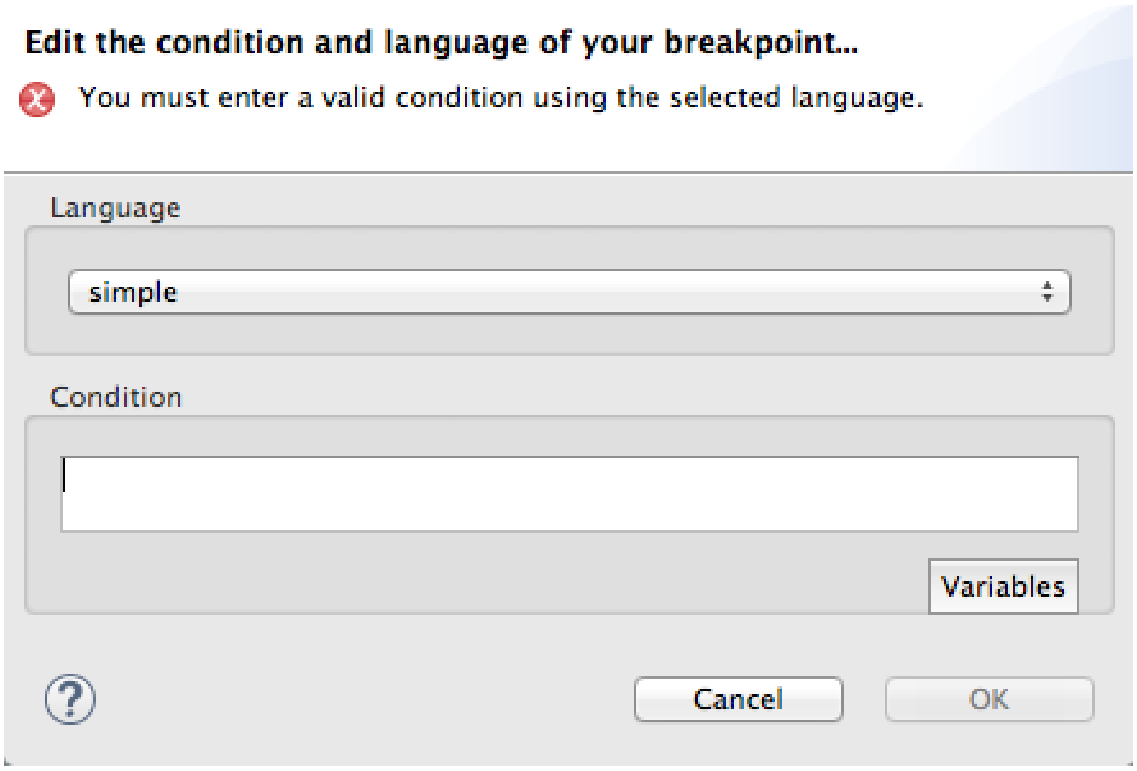
With your routing context displayed on the canvas in the **Design** tab:

1. Select a node whose state you want to examine during the debugging session.
2. Click its 🔴 icon to set an unconditional breakpoint.
3. Repeat these steps for each node on which you want to set an unconditional breakpoint.

SETTING CONDITIONAL BREAKPOINTS

With your routing context displayed on the canvas in the **Design** tab:

1. Select a node whose state you want to examine during the debugging session.
2. Click its 🟠 icon to set a conditional breakpoint and to open the **Edit the condition and language of your breakpoint…** dialog:


Edit the condition and language of your breakpoint...

You must enter a valid condition using the selected language.

Language: simple

Condition:

3. Click the **Language** drop-down menu and select the expression language to use to create the condition that will trigger the breakpoint.
   Fuse Tooling supports twenty-four expression languages. Some of these languages provide variables for creating conditional expressions, while others do not.

4. Click **Variables** to display a list of the selected language's supported variables.
   If a list appears, select in sequence one or more of the variables to create the condition for triggering the breakpoint. The variables you select appear in the **Condition** text box.

   If <nothing available> appears, enter the expression directly into the **Condition** text box.

5. Repeat steps [condBpFirst] through [condBpLast] for each node on which you want to set a conditional breakpoint.

DISABLING BREAKPOINTS

You can temporarily disable a breakpoint, leaving it in place, then enable it again later. The button skips over disabled breakpoints during debugging sessions.

To disable a breakpoint, select the node on the canvas and click its ● icon. The breakpoint turns gray, indicating that it has been disabled.

To enable a disabled breakpoint, select the node on the canvas and click its ● icon. Depending on whether the disabled breakpoint is conditional or unconditional, it turns yellow or red, respectively, to indicate that it has been re-enabled.

NOTE

You can also disable and re-enable breakpoints during debugging sessions. For details, see Chapter 19, *Disabling Breakpoints in a Running Context*.

DELETING BREAKPOINTS

You can delete individual breakpoints or all breakpoints.

- To delete individual breakpoints — in a route container, select the node whose breakpoint you want to delete, and click its icon.
- To delete all breakpoints in a particular route — right-click the target route's container, and select
  `Delete all breakpoints`.
- To delete all breakpoints in all routes — right-click the canvas, and select
  `Delete all breakpoints`.

RELATED TOPICS

- Chapter 15, *Running the Camel Debugger*

CHAPTER 15. RUNNING THE CAMEL DEBUGGER

You can run the Camel debugger on locally running routing contexts and on remotely running routing contexts. The same basic features and functionality are available in both debugging modes.

- Local debugging — Runs the debugger on the routing context running in the same JVM with Fuse Tooling. This mode is activated by selecting the project's routing context in the **Project Explorer** view and selecting **Debug As** → **Local Camel Context** from the context menu.
- Remote debugging — Runs the debugger on a routing context running in a separate JVM either on the local machine or on a remote machine. This mode requires a supported runtime server installed on the local machine or on an accessible remote machine. It is activated by creating and running a debug launch configuration that specifies the remote runtime's connection details.

NOTE

If your project contains Java code, you can use the standard Eclipse Java debugging tools to debug it. For local debugging, it is automatically available. For remote debugging, you need to use the **Remote Camel Context and Java** launch option (see Section 15.2, "Debugging a remotely running routing context").

NOTE

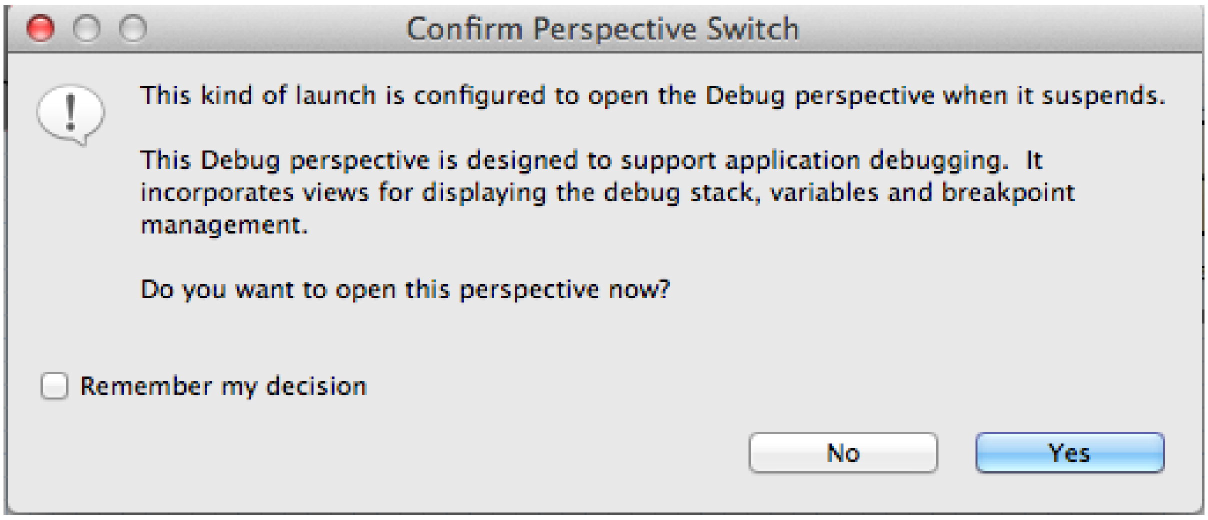
You must set breakpoints in your routing context file before you can start the Camel debugger.

15.1. DEBUGGING A LOCALLY RUNNING ROUTING CONTEXT

Procedure

1. In the **Project Explorer** view, select the routing context file you want to debug.
2. Right-click the selected file to open the context menu, and then select **Debug As** → **Local Camel Context**.
   Fuse Tooling builds the Camel route, starts Apache Camel, starts the routing context, enables JMX, starts the route(s) in the routing context, adds the breakpoints to the nodes, and enables the Camel debugger.

   The Camel debugger suspends execution of the routing context at the first breakpoint hit (received a message), and prompts you to indicate whether you want it to open the **Debug** perspective.



Confirm Perspective Switch

This kind of launch is configured to open the Debug perspective when it suspends.

This Debug perspective is designed to support application debugging. It incorporates views for displaying the debug stack, variables and breakpoint management.

Do you want to open this perspective now?

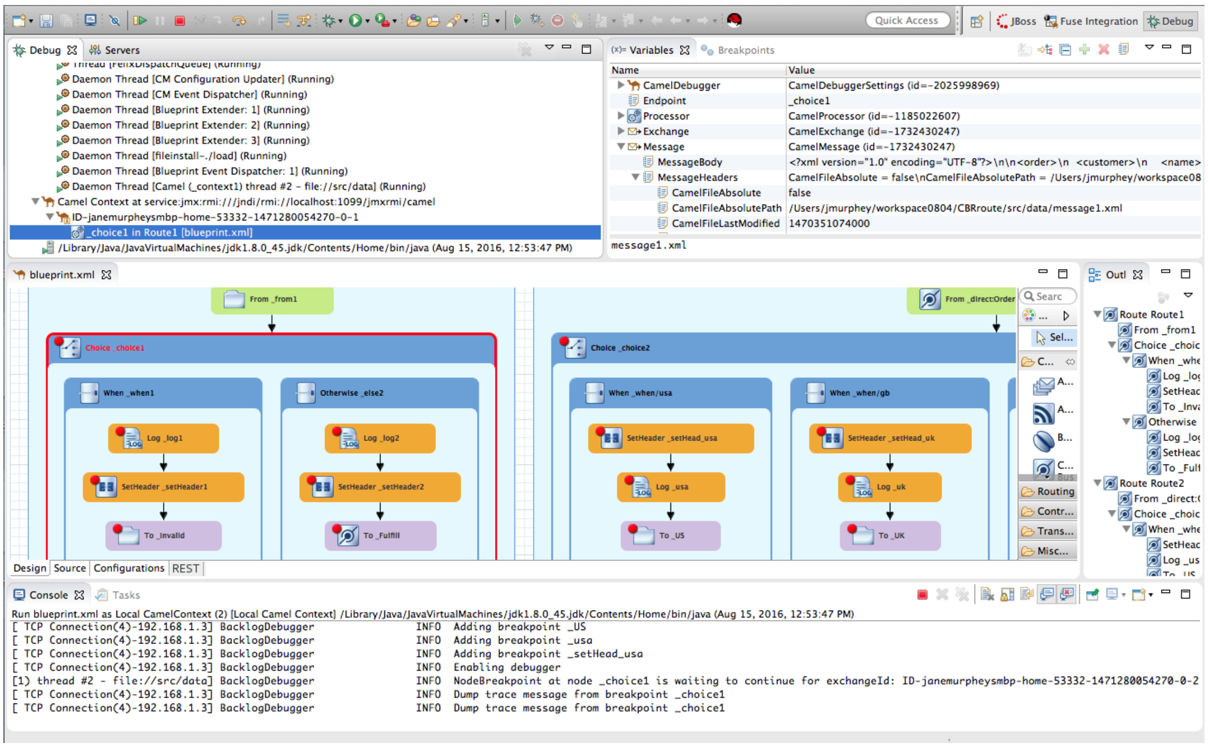
☐ Remember my decision

No Yes

3. Click **Yes** to open the **Debug** perspective.
   The **Debug** perspective opens with the routing context suspended at the first breakpoint encountered in the running routing context.

IMPORTANT

Breakpoints are held for a maximum of five minutes, after which debugging automatically resumes, moving on to the next breakpoint or to the end of the routing context.



```
Daemon Thread [CM Configuration Updater] (Running)
Daemon Thread [CM Event Dispatcher] (Running)
Daemon Thread [Blueprint Extender: 1] (Running)
Daemon Thread [Blueprint Extender: 2] (Running)
Daemon Thread [Blueprint Extender: 3] (Running)
Daemon Thread [fileinstall-./load] (Running)
Daemon Thread [Blueprint Event Dispatcher: 1] (Running)
Daemon Thread [Camel (_context1) thread #2 - file://src/data] (Running)
Camel Context at service:jmx:rmi:///jndi/rmi://localhost:1099/jmxrmi/camel
  ID-janemurpheysmbp-home-53332-1471280054270-0-1
    _choice1 in Route1 [blueprint.xml]
/Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/Contents/Home/bin/java (Aug 15, 2016, 12:53:47 PM)
```

Name	Value
CamelDebugger	CamelDebuggerSettings (id=-2025998969)
Endpoint	_choice1
Processor	CamelProcessor (id=-1185022607)
Exchange	CamelExchange (id=-1732430247)
Message	CamelMessage (id=-1732430247)
MessageBody	<?xml version="1.0" encoding="UTF-8"?>\n\n<order>\n <customer>\n <name>
MessageHeaders	CamelFileAbsolute = false\nCamelFileAbsolutePath = /Users/jmurphey/workspace08
CamelFileAbsolute	false
CamelFileAbsolutePath	/Users/jmurphey/workspace0804/CBRroute/src/data/message1.xml
CamelFileLastModified	1470351074000

message1.xml

```
Run blueprint.xml as Local CamelContext (2) [Local Camel Context] /Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/Contents/Home/bin/java (Aug 15, 2016, 12:53:47 PM)
[ TCP Connection(4)-192.168.1.3] BacklogDebugger                INFO  Adding breakpoint _US
[ TCP Connection(4)-192.168.1.3] BacklogDebugger                INFO  Adding breakpoint _usa
[ TCP Connection(4)-192.168.1.3] BacklogDebugger                INFO  Adding breakpoint _setHead_usa
[ TCP Connection(4)-192.168.1.3] BacklogDebugger                INFO  Enabling debugger
[1) thread #2 - file://src/data] BacklogDebugger                INFO  NodeBreakpoint at node _choice1 is waiting to continue for exchangeId: ID-janemurpheysmbp-home-53332-1471280054270-0-2
[ TCP Connection(4)-192.168.1.3] BacklogDebugger                INFO  Dump trace message from breakpoint _choice1
[ TCP Connection(4)-192.168.1.3] BacklogDebugger                INFO  Dump trace message from breakpoint _choice1
```

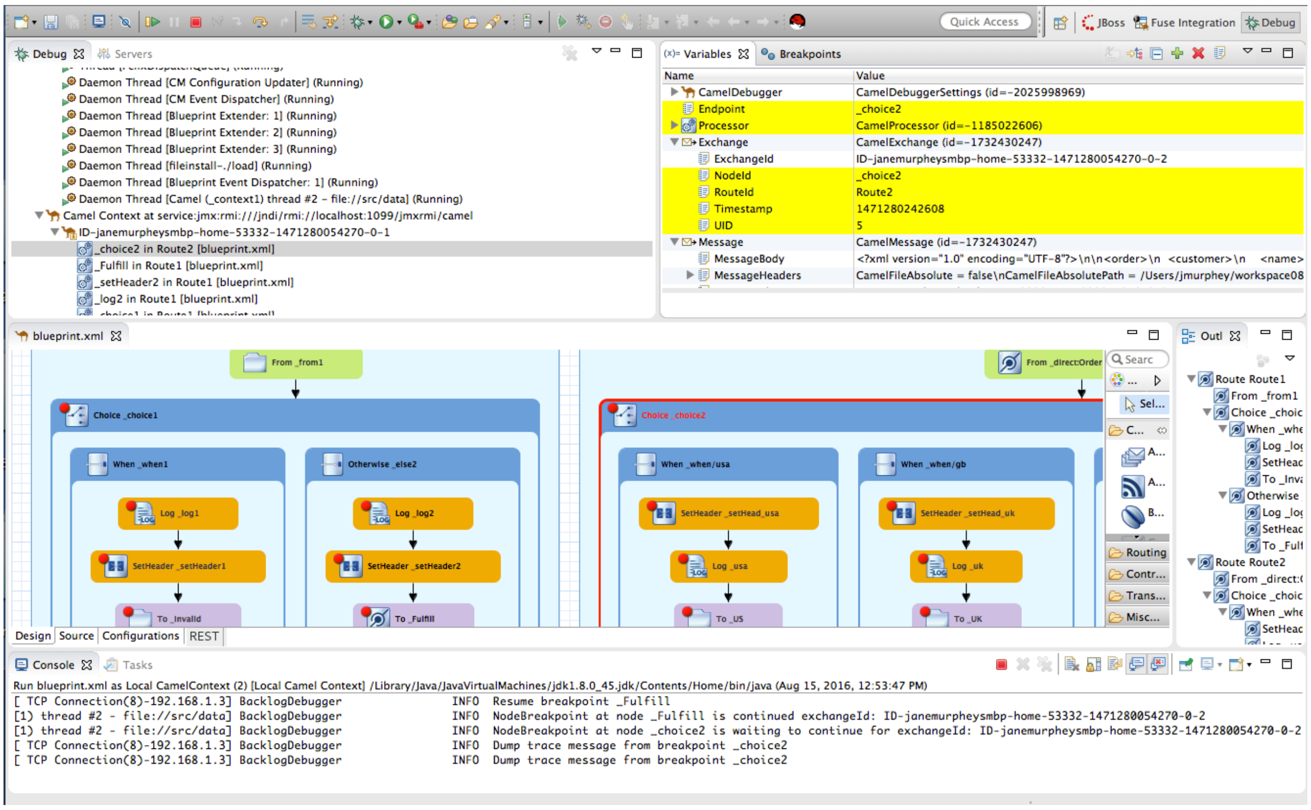
NOTE

To see the console output, open the **Console** view if it was not open when you switched perspectives.

> **NOTE**
>
> By default, the **Debug** perspective displays the **Outline** view, which provides the means to switch between separate routes in a running routing context. If your routing context contains a single route, closing the **Outline** view frees space to expand the other views, making it easier to access and examine debugger output.

Watching message exchanges progress through the routing context

Click (Step Over) to jump to the next node of execution in the routing context. Click (Resume) to continue execution at the next active breakpoint in the routing context.



```
Daemon Thread [CM Configuration Updater] (Running)
Daemon Thread [CM Event Dispatcher] (Running)
Daemon Thread [Blueprint Extender: 1] (Running)
Daemon Thread [Blueprint Extender: 2] (Running)
Daemon Thread [Blueprint Extender: 3] (Running)
Daemon Thread [fileinstall-./load] (Running)
Daemon Thread [Blueprint Event Dispatcher: 1] (Running)
Daemon Thread [Camel (_context1) thread #2 - file://src/data] (Running)
Camel Context at service:jmx:rmi:///jndi/rmi://localhost:1099/jmxrmi/camel
  ID-janemurpheysmbp-home-53332-1471280054270-0-1
    _choice2 in Route2 [blueprint.xml]
    _Fulfill in Route1 [blueprint.xml]
    _setHeader2 in Route1 [blueprint.xml]
    _log2 in Route1 [blueprint.xml]
```

Name	Value
CamelDebugger	CamelDebuggerSettings (id=-2025998969)
Endpoint	_choice2
Processor	CamelProcessor (id=-1185022606)
Exchange	CamelExchange (id=-1732430247)
ExchangeId	ID-janemurpheysmbp-home-53332-1471280054270-0-2
NodeId	_choice2
RouteId	Route2
Timestamp	1471280242608
UID	5
Message	CamelMessage (id=-1732430247)
MessageBody	<?xml version="1.0" encoding="UTF-8"?>\n\n<order>\n <customer>\n <name>
MessageHeaders	CamelFileAbsolute = false\nCamelFileAbsolutePath = /Users/jmurphey/workspace08

```
Run blueprint.xml as Local CamelContext (2) [Local Camel Context] /Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/Contents/Home/bin/java (Aug 15, 2016, 12:53:47 PM)
[ TCP Connection(8)-192.168.1.3] BacklogDebugger                INFO  Resume breakpoint _Fulfill
[1) thread #2 - file://src/data] BacklogDebugger                INFO  NodeBreakpoint at node _Fulfill is continued exchangeId: ID-janemurpheysmbp-home-53332-1471280054270-0-2
[1) thread #2 - file://src/data] BacklogDebugger                INFO  NodeBreakpoint at node _choice2 is waiting to continue for exchangeId: ID-janemurpheysmbp-home-53332-1471280054270-0-2
[ TCP Connection(8)-192.168.1.3] BacklogDebugger                INFO  Dump trace message from breakpoint _choice2
[ TCP Connection(8)-192.168.1.3] BacklogDebugger                INFO  Dump trace message from breakpoint _choice2
```

15.2. DEBUGGING A REMOTELY RUNNING ROUTING CONTEXT

Prerequisites

- Access to a Fuse runtime in one of the following ways:
  - Install a Fuse server on your local machine and edit its `Fuse_HOME/etc/users.properties` file to activate the `admin` user. For details, see Section 28.1, “Adding a Server”.
  - Be able to access a Fuse server that is installed on a remote machine. You must know its connection details, including the credentials of the `admin` user.
- Create a new Fuse Integration project, see the section called “Specifying the project name and workspace”. In the Fuse Integration project:
  - Select the Camel version that matches the version of the JBoss Fuse server runtime.

- Create an empty Blueprint DSL project or use one of the JBoss Fuse pre-defined templates with Blueprint DSL.

  NOTE

  Though not a requirement, it is a good idea to verify that you can run the project's routing context successfully as a **`Local Camel Context`**. For details, see Section 6.1, "Running routes as a local Camel context".

  NOTE

  The examples in the section called "Setting up and starting remote debugging" are based on the **Fuse** → **Content Based Router** pre-defined template and a Red Hat Fuse 6.3.0 runtime.

- In Red Hat Developer Studio:
  - Add the Fuse server to the **Servers** view. For details, see Section 28.1, "Adding a Server".
  - Start the Fuse server. For details, see Section 28.2, "Starting a Server".
  - Publish your project to the Fuse Runtime server. For details, see the section called "Publishing Fuse projects automatically when resources change".

    NOTE

    You cannot deploy a Fuse Integration project to a JBoss Fuse server running on a remote host from inside JBoss Developer Studio. Instead, you deploy the project's bundle directly on the remote host using one of two supported deployment methods (for details, see).

  - Verify that the project's bundle is deployed and active. For details, see the section called "Verifying the project was published to the server".
  - Stop the Fuse Runtime server. For details, see Section 28.5, "Stopping a Server".

Setting up and starting remote debugging

With the project deployed on Fuse and the server stopped, you need to set up and start remote debugging as follows:

- Start Fuse in debug mode outside of Red Hat Developer Studio.
- In Developer Studio, set breakpoints on nodes in the project's routing context, which is open in the route editor.
- Create a remote Camel context debug configuration and run it.
- Connect to Fuse runtime in the **JMX Navigator** view.
- Drop test messages in the **JMX Navigator** view on the input node of the Camel route running inside the Fuse runtime.
- Use any of the Camel debugger's tools for debugging routes.

Procedure

To set up and start remote debugging:

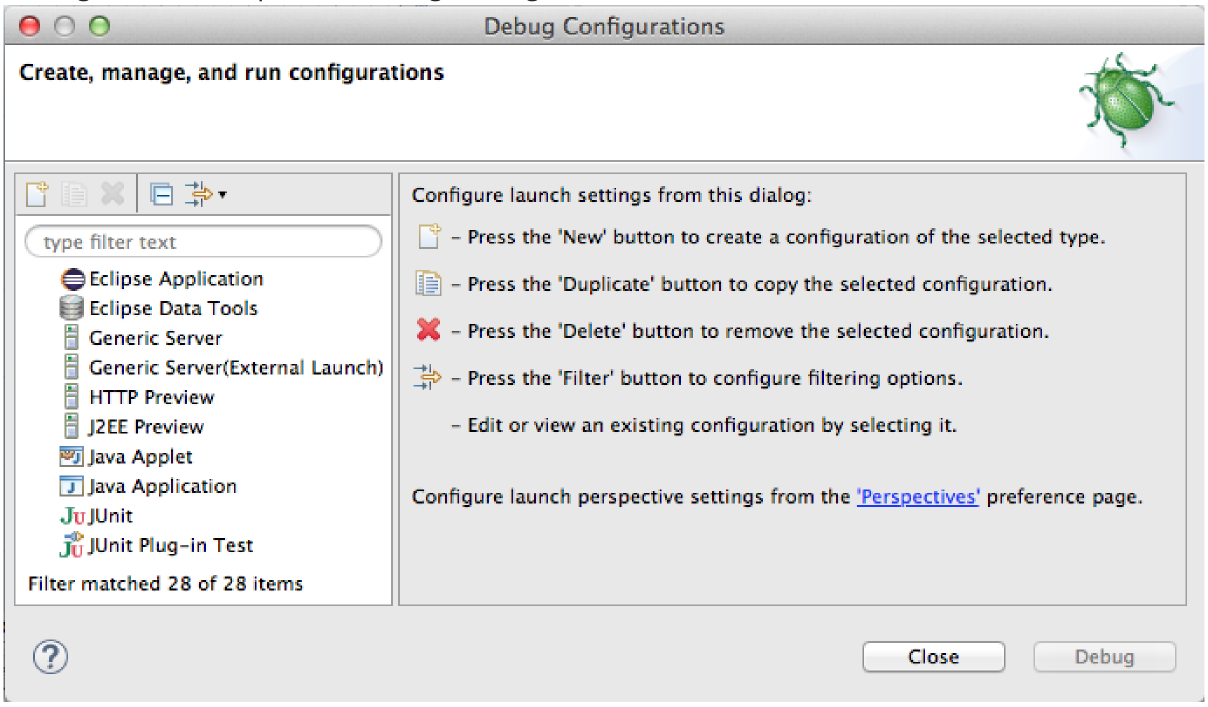
1. Open a terminal outside of Red Hat Developer Studio, and enter

```
$ [{prodname}_HOME]/bin/{prodname} debug
```

2. Wait for the Fuse splash screen to appear, and then return to the **Fuse Integration** perspective in JBoss Developer Studio.

3. In the tooling's route editor, with the project's routing context open in the **Design** tab, set breakpoints on the nodes of interest. For details, see Chapter 14, *Setting Breakpoints*.

4. In the **Project Explorer** view, right-click the project's root and select **Debug As** → **Debug Configurations** to open the **Debug Configurations** wizard:



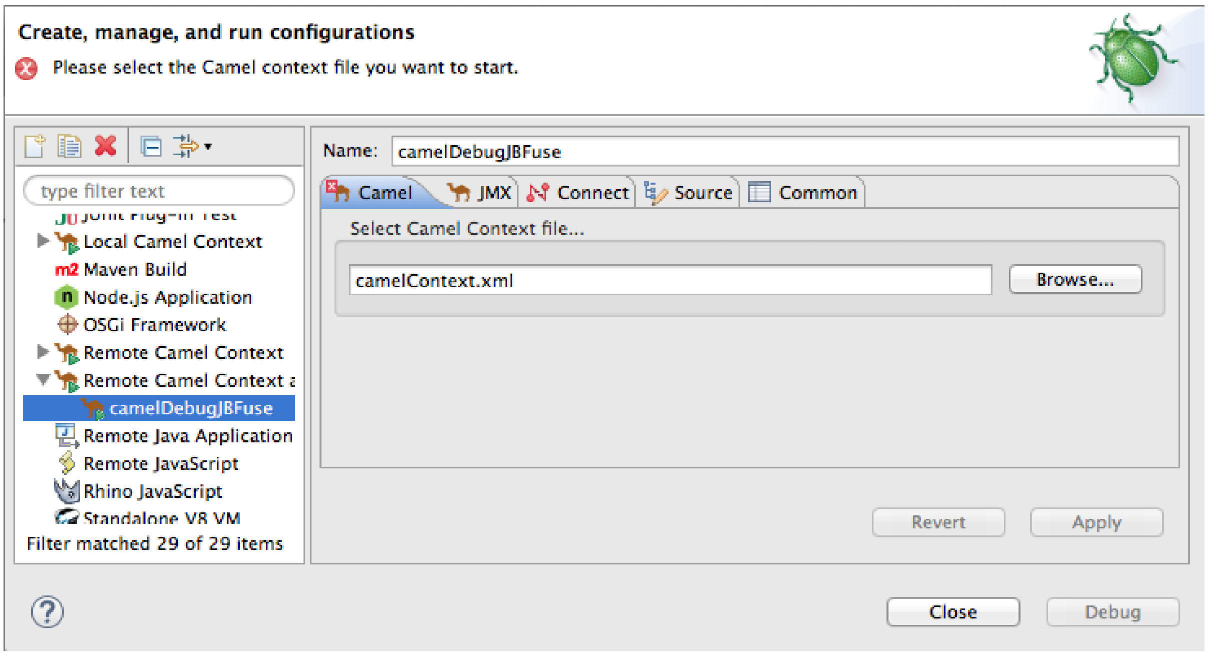
Create, manage, and run configurations

Eclipse Application
Eclipse Data Tools
Generic Server
Generic Server(External Launch)
HTTP Preview
J2EE Preview
Java Applet
Java Application
JUnit
JUnit Plug-in Test
Filter matched 28 of 28 items

Configure launch settings from this dialog:
- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.

Configure launch perspective settings from the 'Perspectives' preference page.

5. In the configuration type pane, select either **Remote Camel Context** or **Remote Camel Context and Java** , and then click :


Create, manage, and run configurations

Please select the Camel context file you want to start.

Name: camelDebugJBFuse

Select Camel Context file...

camelContext.xml

For both the **Remote Camel Context** and **Remote Camel Context and Java** options, you need to specify configuration details on the **Camel** and **JMX** tabs.

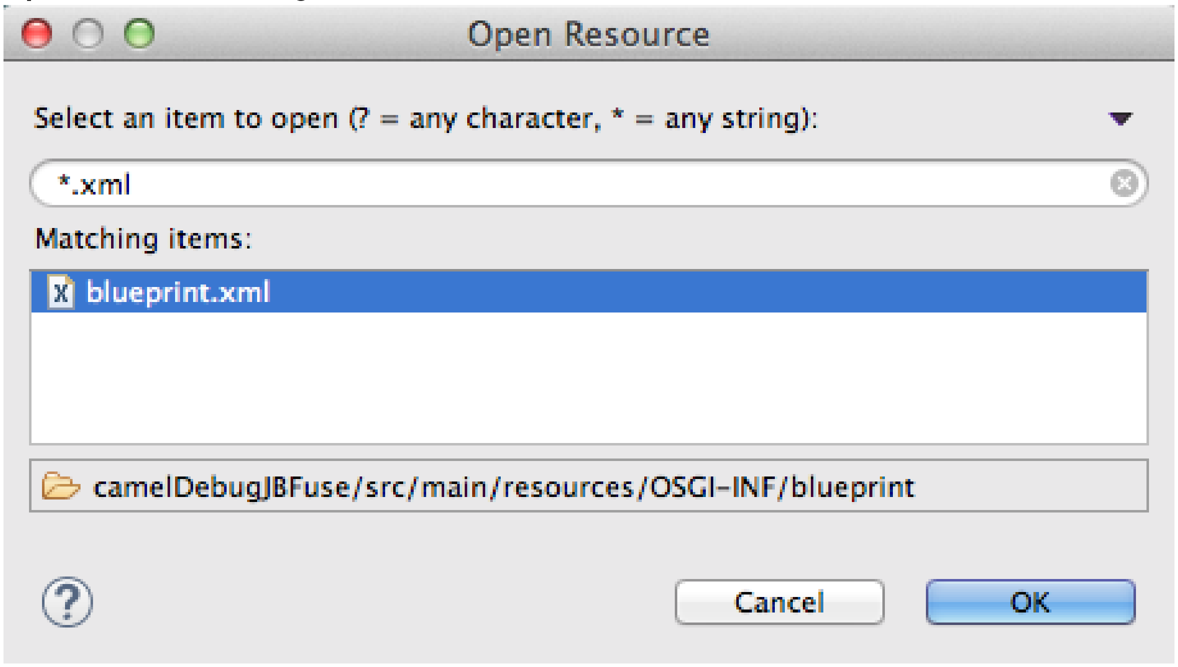
For the **Remote Camel Context and Java** option only, you also need to specify configuration details on the **Connect** tab.

NOTE

Unless your project contains Java code that you want to debug using the standard Eclipse Java debugging tools, select the **Remote Camel Context** option.

6. In the **Name** field, enter a name for the new launch configuration.

7. On the **Camel** tab, click the **Browse** button to locate the project's routing context `.xml` file in the **Open Resources** dialog:


Open Resource

Select an item to open (? = any character, * = any string):

*.xml

Matching items:

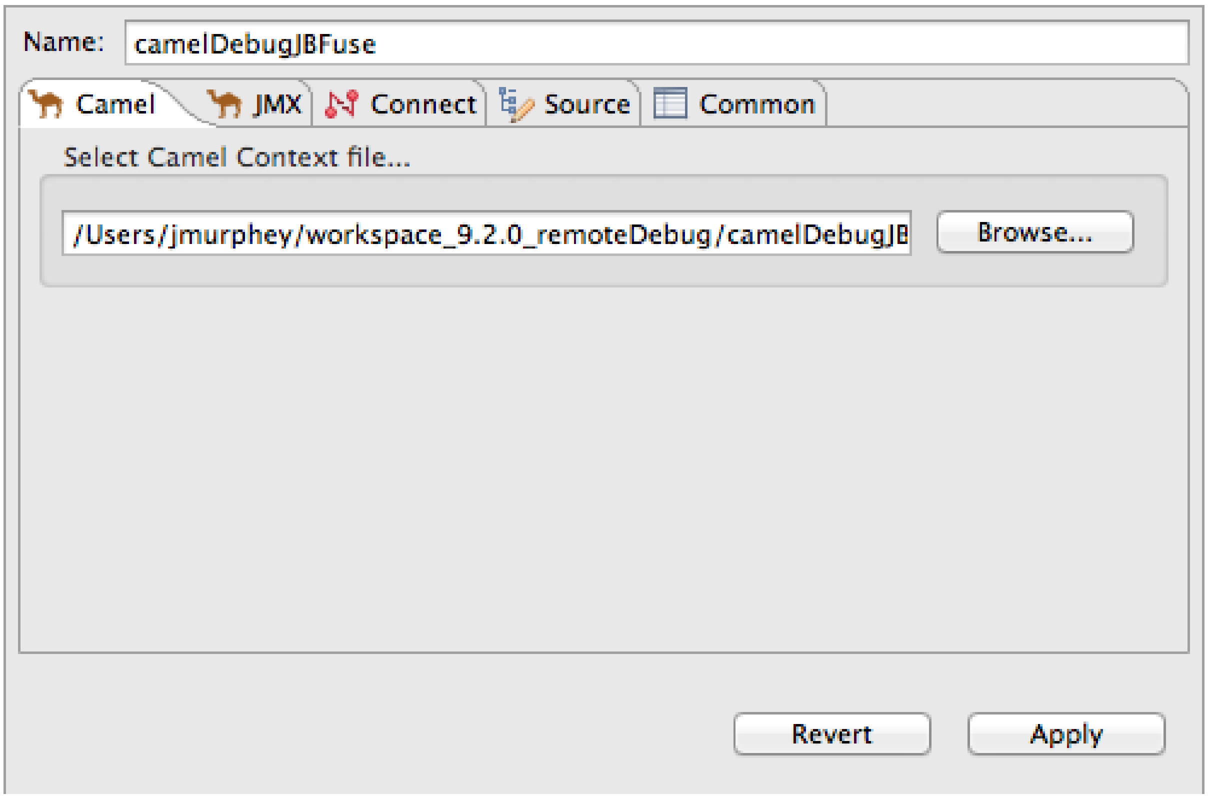
blueprint.xml

camelDebugJBFuse/src/main/resources/OSGI-INF/blueprint

NOTE

When you select a file in the **Matching items** pane, the tooling displays the file's location, relative to the project root, at the bottom of the pane.

8. In the **Matching items** pane, select your project's routing context file from the list, and then click **OK**.
   The tooling inserts the file's path into the **Select Camel Context** file field:

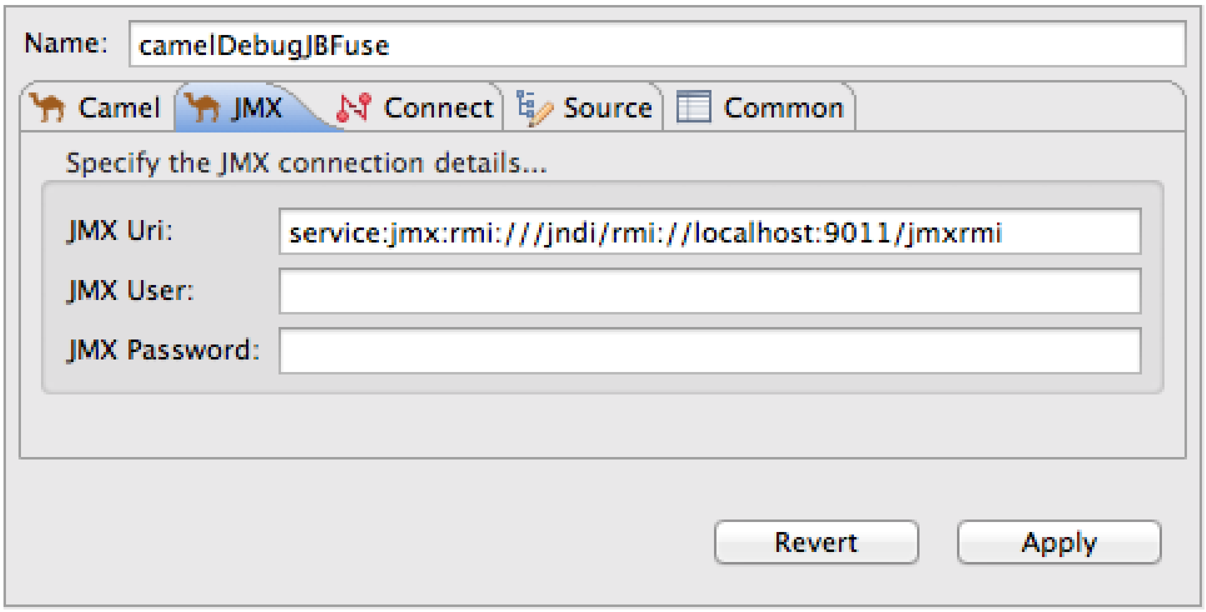


Name: camelDebugJBFuse

Select Camel Context file...

/Users/jmurphey/workspace_9.2.0_remoteDebug/camelDebugJB

9. Click the **JMX** tab:



Name: camelDebugJBFuse

Specify the JMX connection details...

JMX Uri: service:jmx:rmi:///jndi/rmi://localhost:9011/jmxrmi

JMX User:

JMX Password:

Edit the JMX connection details as follows:

- **JMX Uri** — change `:9011/jmxrmi` to `:1099/karaf-root`

If the Fuse server is running on a remote host, replace **`localhost`** with the DNS name or IP address of the remote host.

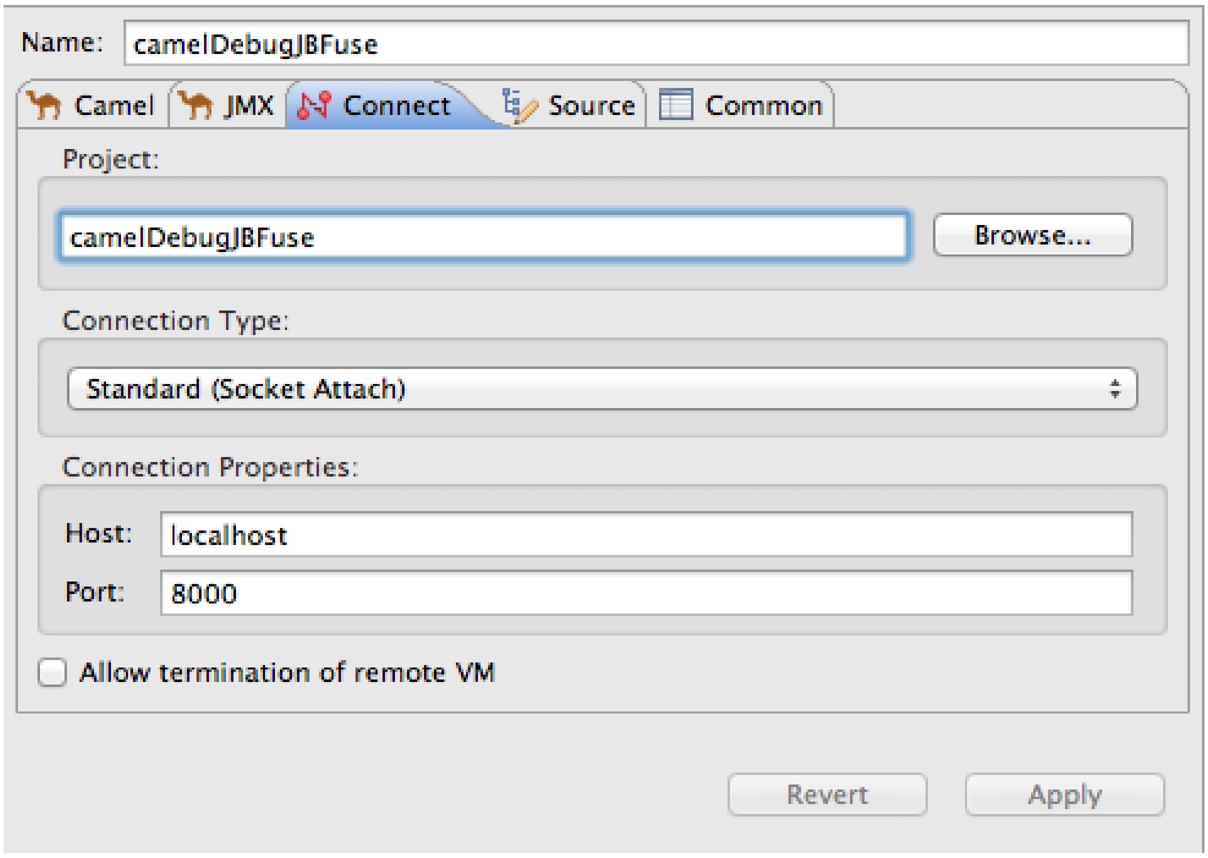
- **JMX User** — enter **`admin`**.
- **JMX Password** — enter **`admin`**.

IMPORTANT

The values shown for **JMX User** and **JMX Password** are the Fuse **admin** user defaults, stored in the **`Fuse_HOME/etc/users.properties`** file. If your setup is different, enter the values specific to it.

If you are creating a **Remote Camel Context** debug launch configuration, you are done. Skip to [debugCfgGo].

10. Click the **Connect** tab:



Name: camelDebugJBFuse

Camel | JMX | Connect | Source | Common

Project: camelDebugJBFuse [Browse...]

Connection Type: Standard (Socket Attach)

Connection Properties:

Host: localhost

Port: 8000

☐ Allow termination of remote VM

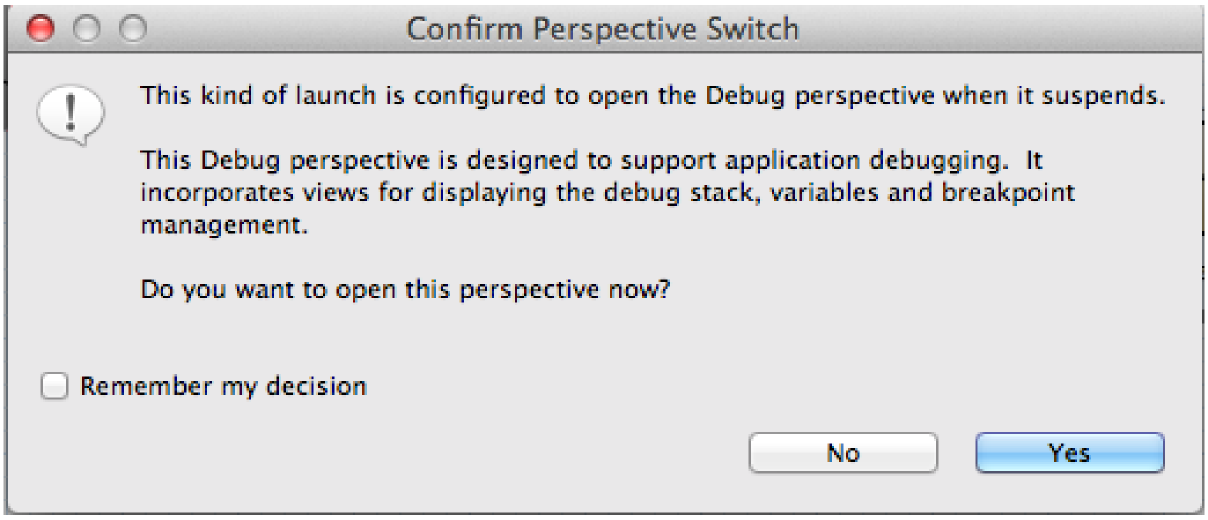
[Revert] [Apply]

Change the **Port** value from **`8000`** to **`5005`**. Leave each of the other properties as is.

11. Click **Apply** and then click **Debug**.

12. In the **JMX Navigator** view, double-click **Fuse [xxx] [Disconnected]** to connect to it, and then expand its tree.

13. In the **Project Explorer** view, drag a test message from **`src/test/resources/data`** and drop it on the **`cbr-example-context/Endpoints/file/work/cbr/input`** folder in the **JMX Navigator** view.
When the message hits the first breakpoint that is set in the routing context, the tooling asks you to switch to the **Debug** perspective:


Confirm Perspective Switch

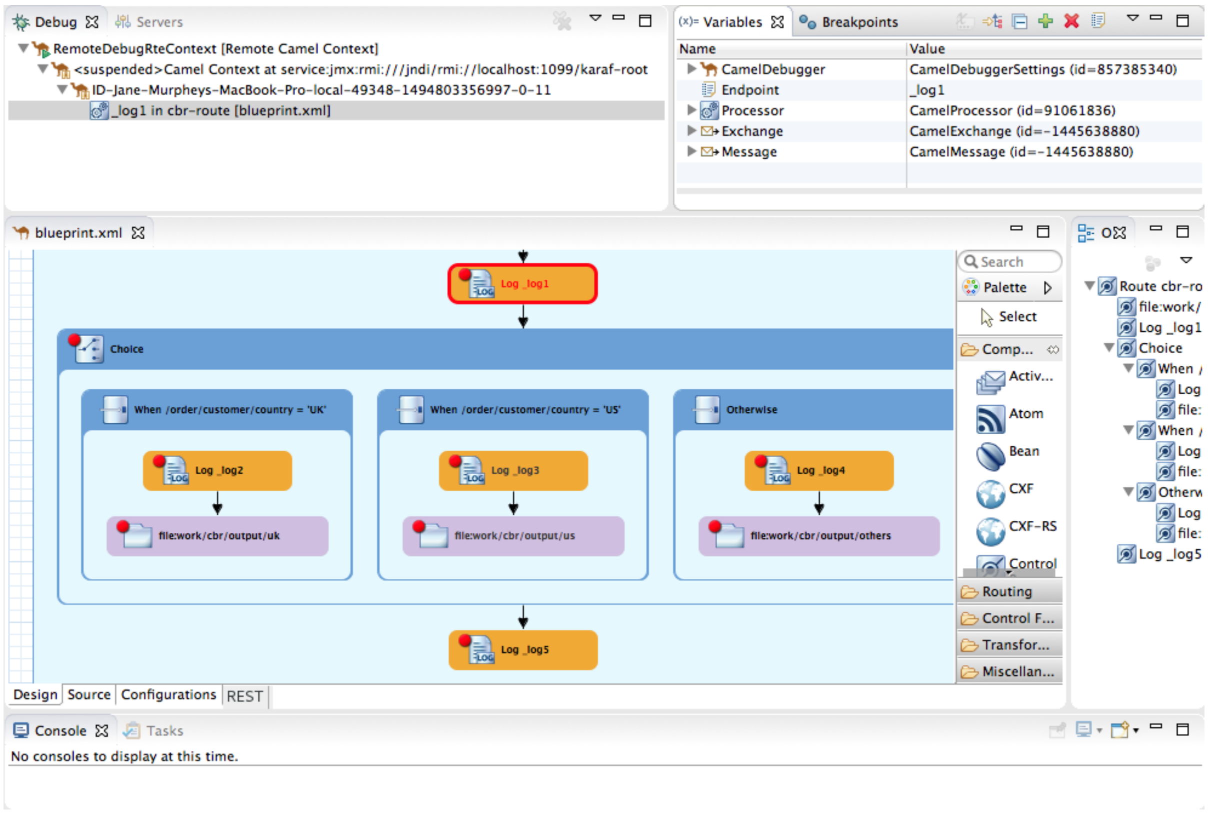
This kind of launch is configured to open the Debug perspective when it suspends.

This Debug perspective is designed to support application debugging. It incorporates views for displaying the debug stack, variables and breakpoint management.

Do you want to open this perspective now?

Remember my decision

14. Click **Yes**.



```
RemoteDebugRteContext [Remote Camel Context]
  <suspended>Camel Context at service:jmx:rmi:///jndi/rmi://localhost:1099/karaf-root
    ID-Jane-Murpheys-MacBook-Pro-local-49348-1494803356997-0-11
      _log1 in cbr-route [blueprint.xml]
```

Name	Value
CamelDebugger	CamelDebuggerSettings (id=857385340)
Endpoint	_log1
Processor	CamelProcessor (id=91061836)
Exchange	CamelExchange (id=-1445638880)
Message	CamelMessage (id=-1445638880)

No consoles to display at this time.

At this point, you can use any of the Camel debugger’s tools to debug your routing context.

NOTE

In remote debugging sessions, the **Console** view does not display log output.

NOTE

When one message reaches the end of the routing context, the debugger is suspended. To continue debugging, switch back to the **Fuse Integration** perspective and drop another message on the input node in the **JMX Navigator** view. Each time you do so, the tooling asks you to confirm the switch to the **Debug** perspective.

RELATED TOPICS

- Chapter 16, *Stopping the Camel Debugger*

CHAPTER 16. STOPPING THE CAMEL DEBUGGER

OVERVIEW

The way to stop the Camel debugger depends on the mode in which it is running:

- Local debugging (Section 15.1, “Debugging a locally running routing context”)

  To stop the Camel debugger, click 🟥 on the menu bar once if the debugging session has ended. Otherwise, click 🟥 twice: once to terminate the currently running node thread and once to terminate the `Camel Context` thread (both displayed in the **Debug** view).

  NOTE

  Terminating the Camel debugger also terminates the console but does not clear its output. To clear the output, click (Clear Console) on the **Console** view’s menu bar.

- Remote debugging (Section 15.2, “Debugging a remotely running routing context”)
  To stop the Camel debugger, select the `[Remote Camel Context]` thread or the `[Remote Camel Context and Java]` thread in the **Debug** view, and then click 🟥 on the menu bar.

  NOTE

  Terminating the Camel debugger does not close the connection to the remote runtime server in the **Servers** view nor in the **JMX Navigator** view.

CLOSING THE CAMEL DEBUGGER

After you have finished debugging your project, you may want to close the **Debug** perspective to make more space for your workbench.

To do so, right-click Debug on the right side of the Developer Studio toolbar, and then select **Close**.

RELATED TOPICS

- Chapter 15, *Running the Camel Debugger*

CHAPTER 17. CHANGING VARIABLE VALUES

OVERVIEW

When the Camel debugger hits a breakpoint, the **Variables** view displays the values of all variables available at that point in the routing context. Some variables are editable, allowing you to change their value. This enables you to see how the application handles changes in program state.

NOTE

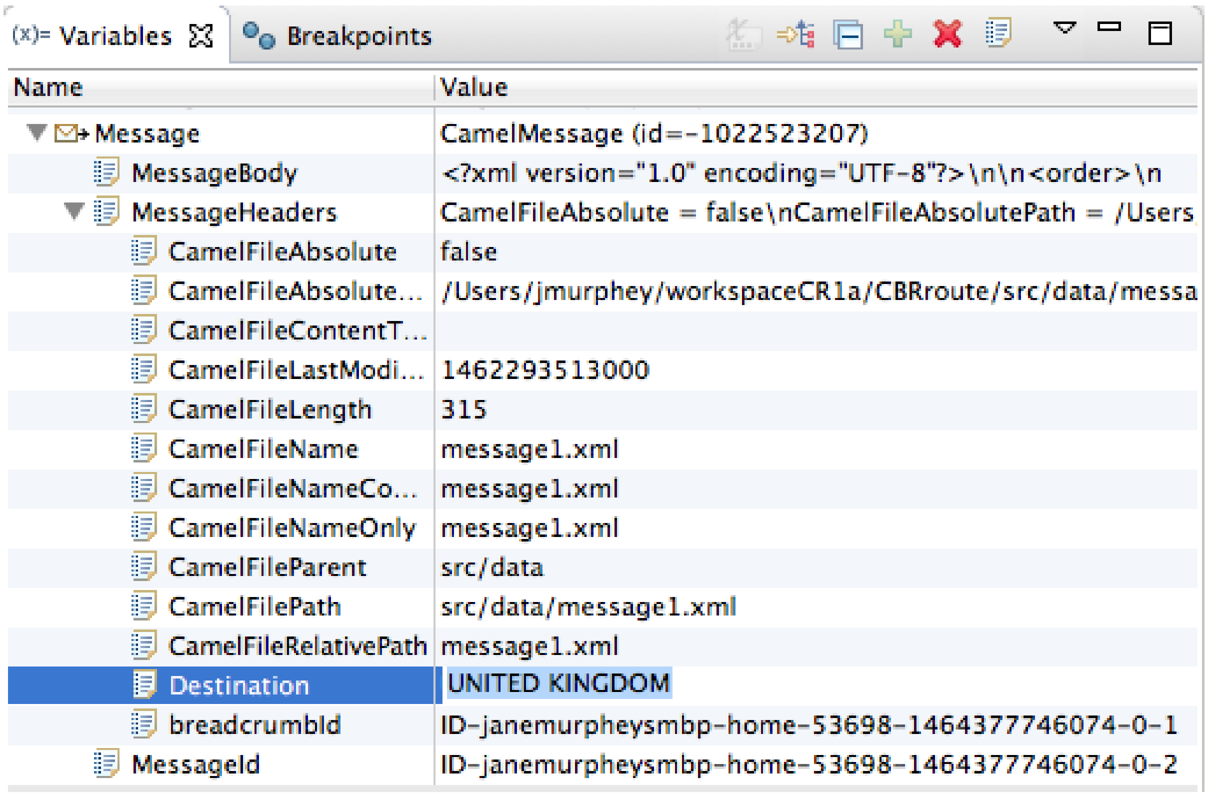
Not all variables are editable. The context menu of editable variables displays the **Change Value…** option.

PROCEDURE

To change the value of a variable:

1. If necessary, start the debugger. See Chapter 15, *Running the Camel Debugger*.

2. In the **Variables** view, select a variable whose value you want to change, and then click its **Value** field.



Name	Value
Message	CamelMessage (id=-1022523207)
MessageBody	<?xml version="1.0" encoding="UTF-8"?>\n\n<order>\n
MessageHeaders	CamelFileAbsolute = false\nCamelFileAbsolutePath = /Users
CamelFileAbsolute	false
CamelFileAbsolute…	/Users/jmurphey/workspaceCR1a/CBRroute/src/data/messa
CamelFileContentT…	
CamelFileLastModi…	1462293513000
CamelFileLength	315
CamelFileName	message1.xml
CamelFileNameCo…	message1.xml
CamelFileNameOnly	message1.xml
CamelFileParent	src/data
CamelFilePath	src/data/message1.xml
CamelFileRelativePath	message1.xml
Destination	UNITED KINGDOM
breadcrumbId	ID-janemurpheysmbp-home-53698-1464377746074-0-1
MessageId	ID-janemurpheysmbp-home-53698-1464377746074-0-2

The variable's **value** field turns a lighter shade of blue, indicating that it is in edit mode.

NOTE

Alternatively, you can right-click the variable to open its context menu, and select **Change Value…** to edit its value.

3. Enter the new value and then click **Enter**.
The **Console** view displays an **`INFO`** level log entry noting the change in the variable's value (for

example, **`Breakpoint at node to1 is updating message header on exchangeId: ID-dhcp-97-16-bos-redhat-com-52574-1417298894070-0-2 with header: Destination and value: UNITED KINGDOM`**).

4. Continue stepping through the breakpoints and check whether the message is processed as expected. At each step, check the **Debug** view, the **Variables** view, and the **Console** output.

RELATED TOPICS

- Chapter 19, *Disabling Breakpoints in a Running Context*
- Chapter 18, *Adding Variables to the Watch List*

CHAPTER 18. ADDING VARIABLES TO THE WATCH LIST

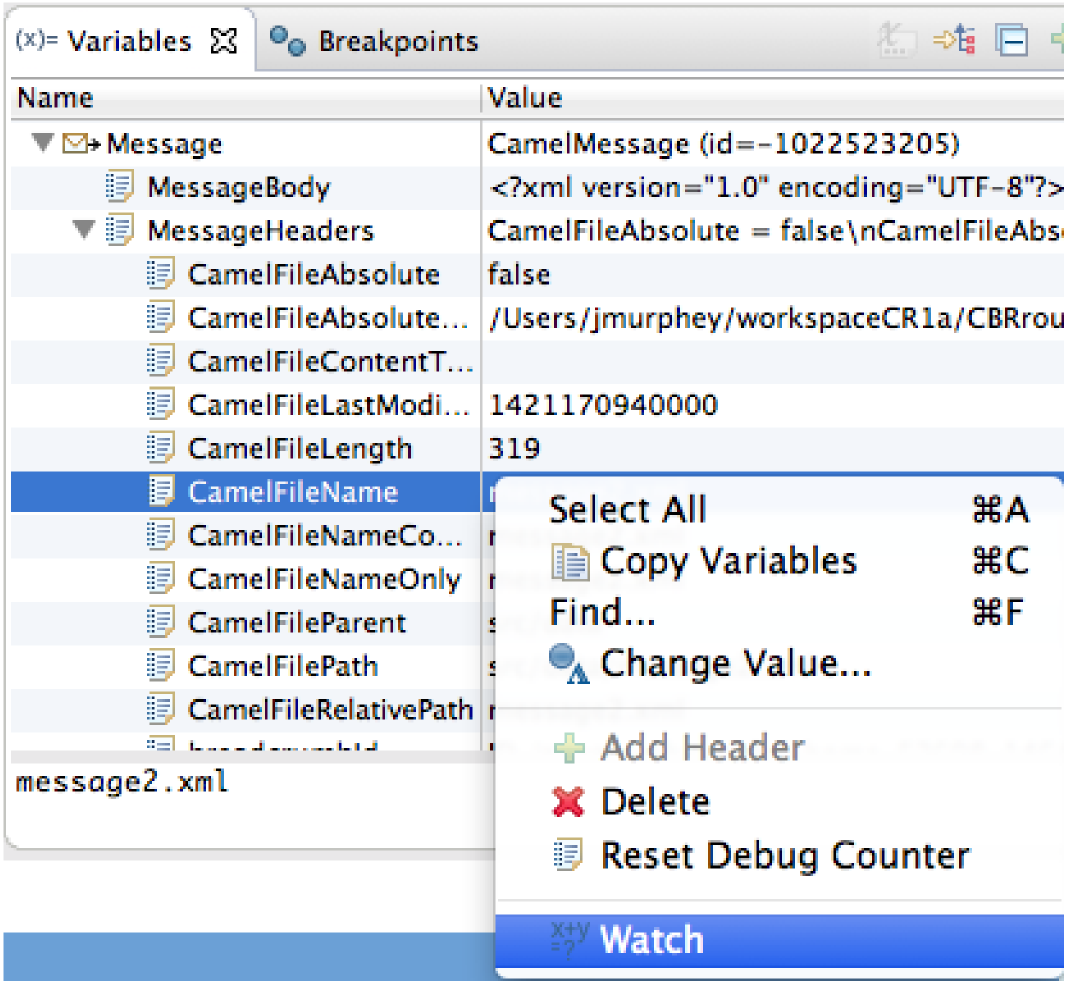
OVERVIEW

By adding variables to the watch list, you can focus on particular variables to see whether their values change as expected as they flow through the routing context.

PROCEDURE

To add a variable to the watch list:

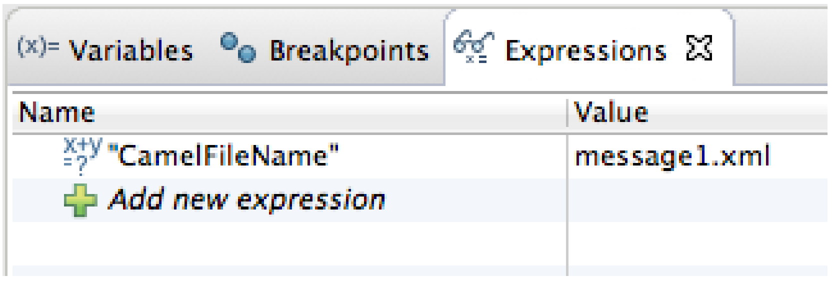
1. If necessary, start the debugger. See Chapter 15, *Running the Camel Debugger*.
2. In the **Variables** view, right-click a variable you want to track to open the context menu.



Name	Value
Message	CamelMessage (id=-1022523205)
MessageBody	<?xml version="1.0" encoding="UTF-8"?>
MessageHeaders	CamelFileAbsolute = false\nCamelFileAbs
CamelFileAbsolute	false
CamelFileAbsolute...	/Users/jmurphey/workspaceCR1a/CBRrou
CamelFileContentT...	
CamelFileLastModi...	1421170940000
CamelFileLength	319
CamelFileName	r
CamelFileNameCo...	r
CamelFileNameOnly	r
CamelFileParent	s
CamelFilePath	s
CamelFileRelativePath	r

```
message2.xml
```

3. Select **Watch**.
   A new view, **Expressions**, opens next to the **Breakpoints** view. The **Expressions** view displays the name of the variable being watched and its current value, for example:



Name	Value
"CamelFileName"	message1.xml
Add new expression	

4. Repeat [watch1] and [watch2] to add additional variables to the watch list.

NOTE

The variables you add remain in the watch list until you remove them. To stop watching a variable, right-click it in the list to open the context menu, and then click **Remove**.

5. With the **Expressions** view open, step through the routing context to track how the value of each variable in the watch list changes as it reaches each step in the route.

RELATED TOPICS

- Chapter 17, *Changing Variable Values*

CHAPTER 19. DISABLING BREAKPOINTS IN A RUNNING CONTEXT

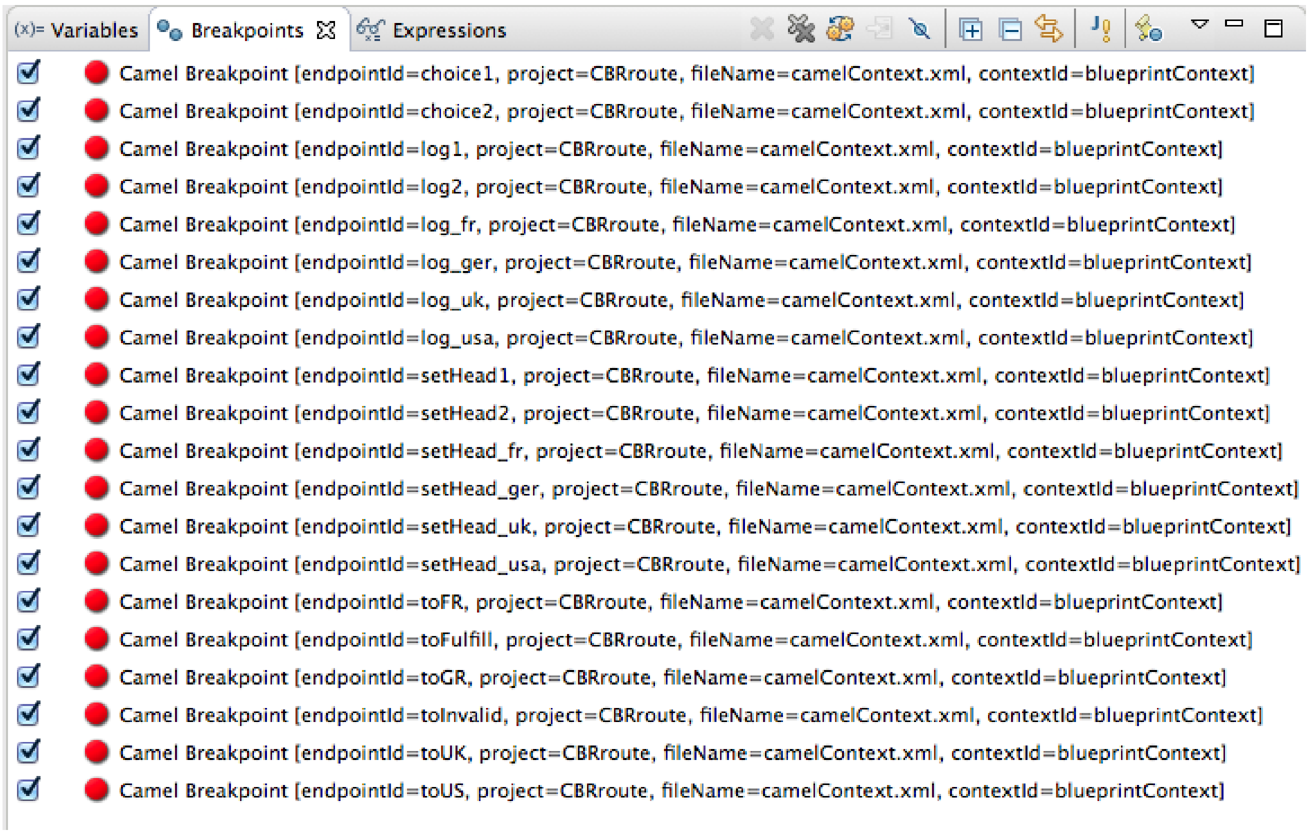
OVERVIEW

You can disable and re-enable breakpoints in a running routing context in the **Breakpoints** view.

When a breakpoint is disabled, the button causes the debugger to skip over it during the debugging session.

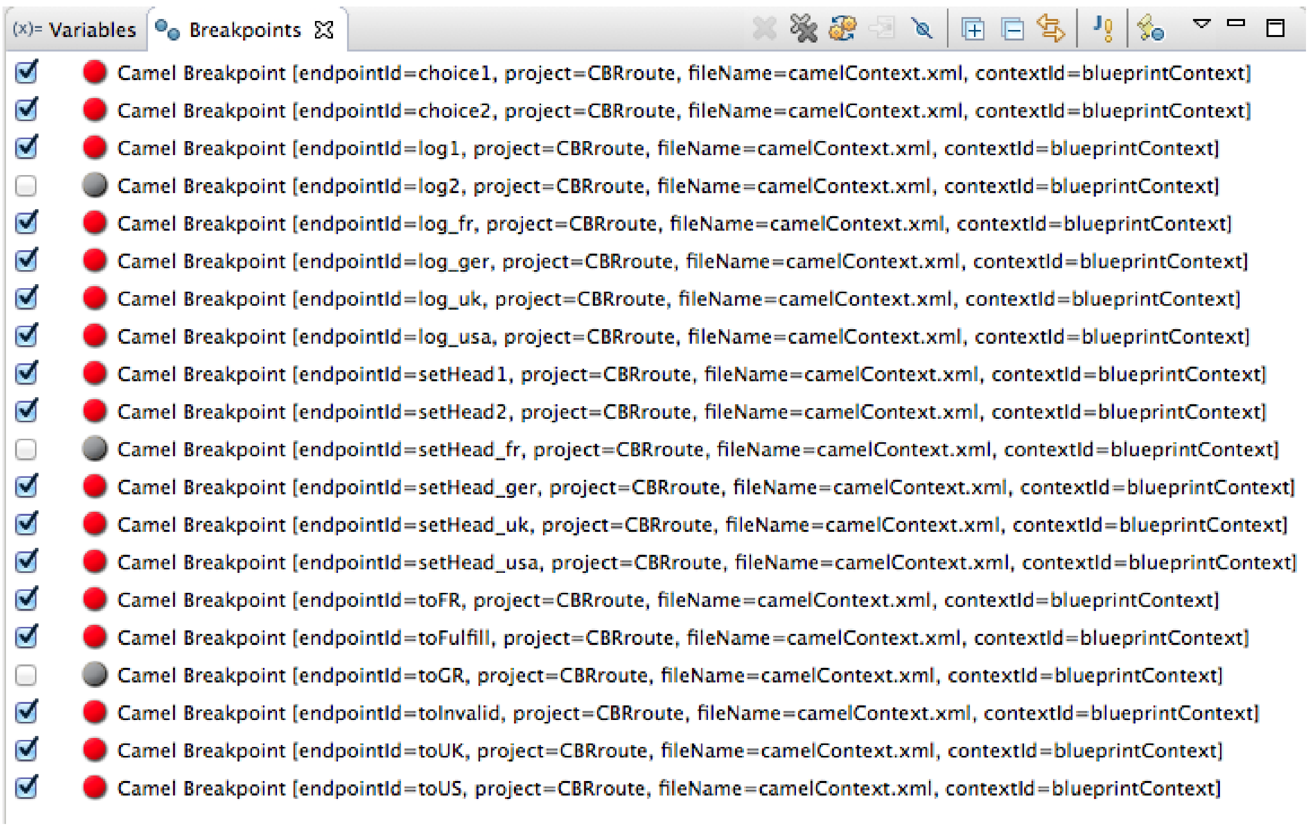
DISABLING AND ENABLING BREAKPOINTS IN BREAKPOINTS VIEW

The **Breakpoints** view opens with all set breakpoints enabled.



```
✓ ● Camel Breakpoint [endpointId=choice1, project=CBRroute, fileName=camelContext.xml, contextId=blueprintContext]
✓ ● Camel Breakpoint [endpointId=choice2, project=CBRroute, fileName=camelContext.xml, contextId=blueprintContext]
✓ ● Camel Breakpoint [endpointId=log1, project=CBRroute, fileName=camelContext.xml, contextId=blueprintContext]
✓ ● Camel Breakpoint [endpointId=log2, project=CBRroute, fileName=camelContext.xml, contextId=blueprintContext]
✓ ● Camel Breakpoint [endpointId=log_fr, project=CBRroute, fileName=camelContext.xml, contextId=blueprintContext]
✓ ● Camel Breakpoint [endpointId=log_ger, project=CBRroute, fileName=camelContext.xml, contextId=blueprintContext]
✓ ● Camel Breakpoint [endpointId=log_uk, project=CBRroute, fileName=camelContext.xml, contextId=blueprintContext]
✓ ● Camel Breakpoint [endpointId=log_usa, project=CBRroute, fileName=camelContext.xml, contextId=blueprintContext]
✓ ● Camel Breakpoint [endpointId=setHead1, project=CBRroute, fileName=camelContext.xml, contextId=blueprintContext]
✓ ● Camel Breakpoint [endpointId=setHead2, project=CBRroute, fileName=camelContext.xml, contextId=blueprintContext]
✓ ● Camel Breakpoint [endpointId=setHead_fr, project=CBRroute, fileName=camelContext.xml, contextId=blueprintContext]
✓ ● Camel Breakpoint [endpointId=setHead_ger, project=CBRroute, fileName=camelContext.xml, contextId=blueprintContext]
✓ ● Camel Breakpoint [endpointId=setHead_uk, project=CBRroute, fileName=camelContext.xml, contextId=blueprintContext]
✓ ● Camel Breakpoint [endpointId=setHead_usa, project=CBRroute, fileName=camelContext.xml, contextId=blueprintContext]
✓ ● Camel Breakpoint [endpointId=toFR, project=CBRroute, fileName=camelContext.xml, contextId=blueprintContext]
✓ ● Camel Breakpoint [endpointId=toFulfill, project=CBRroute, fileName=camelContext.xml, contextId=blueprintContext]
✓ ● Camel Breakpoint [endpointId=toGR, project=CBRroute, fileName=camelContext.xml, contextId=blueprintContext]
✓ ● Camel Breakpoint [endpointId=toInvalid, project=CBRroute, fileName=camelContext.xml, contextId=blueprintContext]
✓ ● Camel Breakpoint [endpointId=toUK, project=CBRroute, fileName=camelContext.xml, contextId=blueprintContext]
✓ ● Camel Breakpoint [endpointId=toUS, project=CBRroute, fileName=camelContext.xml, contextId=blueprintContext]
```

To disable a breakpoint, clear its check box.



☑ ● Camel Breakpoint [endpointId=choice1, project=CBRoute, fileName=camelContext.xml, contextId=blueprintContext]
☑ ● Camel Breakpoint [endpointId=choice2, project=CBRoute, fileName=camelContext.xml, contextId=blueprintContext]
☑ ● Camel Breakpoint [endpointId=log1, project=CBRoute, fileName=camelContext.xml, contextId=blueprintContext]
☐ ● Camel Breakpoint [endpointId=log2, project=CBRoute, fileName=camelContext.xml, contextId=blueprintContext]
☑ ● Camel Breakpoint [endpointId=log_fr, project=CBRoute, fileName=camelContext.xml, contextId=blueprintContext]
☑ ● Camel Breakpoint [endpointId=log_ger, project=CBRoute, fileName=camelContext.xml, contextId=blueprintContext]
☑ ● Camel Breakpoint [endpointId=log_uk, project=CBRoute, fileName=camelContext.xml, contextId=blueprintContext]
☑ ● Camel Breakpoint [endpointId=log_usa, project=CBRoute, fileName=camelContext.xml, contextId=blueprintContext]
☑ ● Camel Breakpoint [endpointId=setHead1, project=CBRoute, fileName=camelContext.xml, contextId=blueprintContext]
☑ ● Camel Breakpoint [endpointId=setHead2, project=CBRoute, fileName=camelContext.xml, contextId=blueprintContext]
☐ ● Camel Breakpoint [endpointId=setHead_fr, project=CBRoute, fileName=camelContext.xml, contextId=blueprintContext]
☑ ● Camel Breakpoint [endpointId=setHead_ger, project=CBRoute, fileName=camelContext.xml, contextId=blueprintContext]
☑ ● Camel Breakpoint [endpointId=setHead_uk, project=CBRoute, fileName=camelContext.xml, contextId=blueprintContext]
☑ ● Camel Breakpoint [endpointId=setHead_usa, project=CBRoute, fileName=camelContext.xml, contextId=blueprintContext]
☑ ● Camel Breakpoint [endpointId=toFR, project=CBRoute, fileName=camelContext.xml, contextId=blueprintContext]
☑ ● Camel Breakpoint [endpointId=toFulfill, project=CBRoute, fileName=camelContext.xml, contextId=blueprintContext]
☐ ● Camel Breakpoint [endpointId=toGR, project=CBRoute, fileName=camelContext.xml, contextId=blueprintContext]
☑ ● Camel Breakpoint [endpointId=toInvalid, project=CBRoute, fileName=camelContext.xml, contextId=blueprintContext]
☑ ● Camel Breakpoint [endpointId=toUK, project=CBRoute, fileName=camelContext.xml, contextId=blueprintContext]
☑ ● Camel Breakpoint [endpointId=toUS, project=CBRoute, fileName=camelContext.xml, contextId=blueprintContext]

For each breakpoint you disable, the **Console** view displays an **`INFO`** level log entry noting that it has been disabled (for example, **`Removing breakpoint log2`**). Likewise, for each breakpoint you re-enable, the **Console** view displays an **`INFO`** level log entry noting that it has been enabled (for example, **`Adding breakpoint log2`**).

NOTE

To re-enable a disabled breakpoint, click its check box. The **Console** view displays an **`INFO`** level log entry noting that the breakpoint has been added to the selected node.

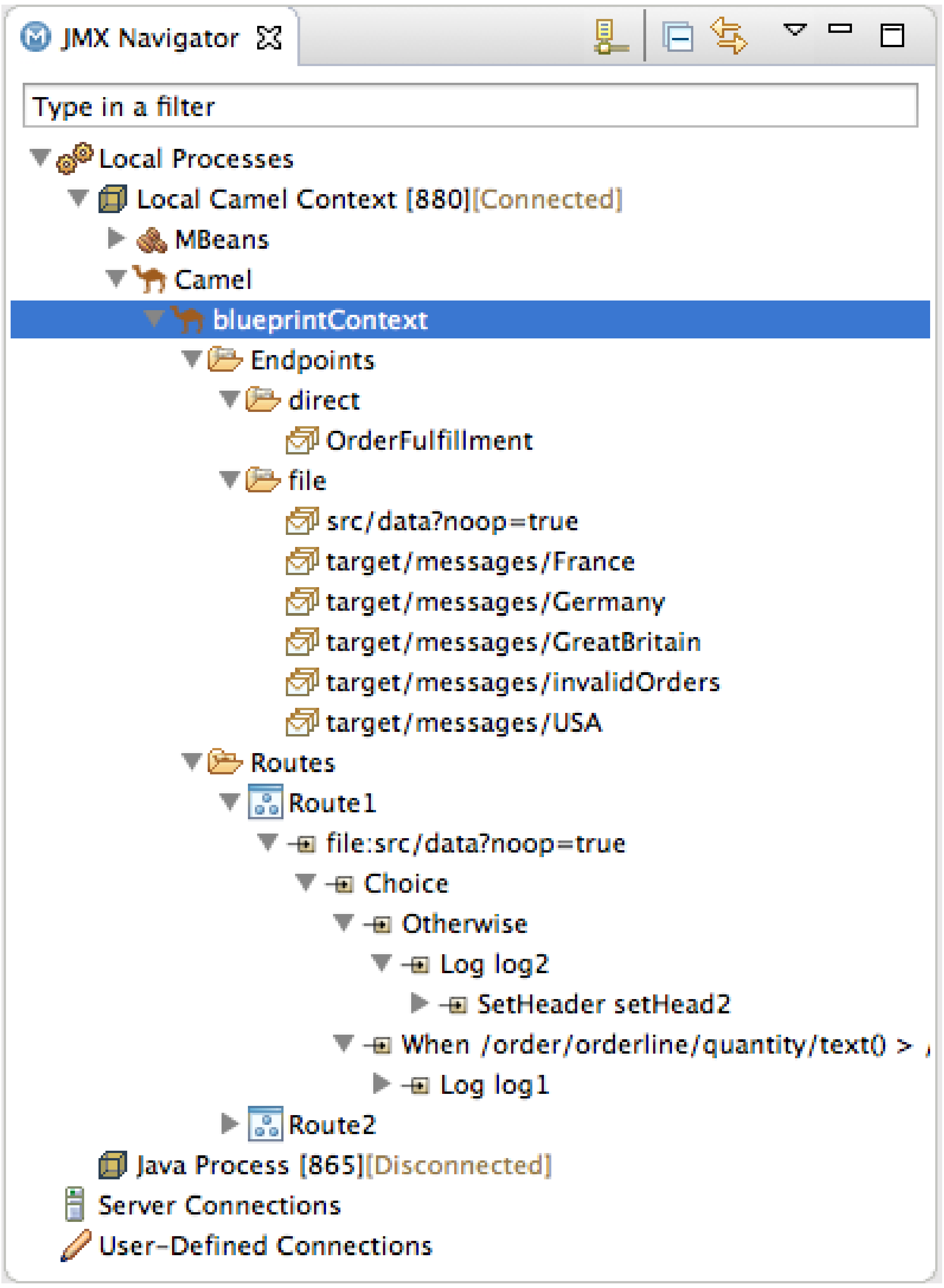
PART III. MONITORING AND TESTING APPLICATIONS

The *JMX Navigator^ view provides numerous ways to monitor and test your Fuse applications.

CHAPTER 20. JMX NAVIGATOR

The **JMX Navigator** view, shown in Figure 20.1, “JMX Navigator view”, displays all processes that are running in your application and it drives all interactions with the monitoring and testing features. Other areas of the **Fuse Integration** perspective adapt to display information related to the node selected in the **JMX Navigator** view. In the **JMX Navigator** view, its context menu provides the commands needed to activate route tracing and to add JMS destinations.

Figure 20.1. JMX Navigator view



```
JMX Navigator
Type in a filter
Local Processes
    Local Camel Context [880][Connected]
        MBeans
        Camel
            blueprintContext
                Endpoints
                    direct
                        OrderFulfillment
                    file
                        src/data?noop=true
                        target/messages/France
                        target/messages/Germany
                        target/messages/GreatBritain
                        target/messages/invalidOrders
                        target/messages/USA
                Routes
                    Route1
                        file:src/data?noop=true
                            Choice
                                Otherwise
                                    Log log2
                                        SetHeader setHead2
                                When /order/orderline/quantity/text() > ,
                                    Log log1
                    Route2
    Java Process [865][Disconnected]
Server Connections
User-Defined Connections
```

By default, the **JMX Navigator** view discovers all JMX servers running on the local machine and lists them under the following categories:

- Local Processes
- Server Connections

- User-Defined Connections

> **NOTE**
>
> You can add other JMX servers by using a server's JMX URL. For details, see Section 20.2, "Adding a JMX server".

20.1. VIEWING PROCESSES IN JMX

Overview

The **JMX Navigator** view lists all known processes in a series of trees. The root for each tree is a JMX server.

The first tree in the list is a special **Local Processes** tree that contains all JMX servers that are running on the local machine. You must connect to one of the JMX servers to see the processes it contains.

Viewing processes in a local JMX server

To view information about processes in a local JMX server:

1. In the **JMX Navigator** view, expand **Local Processes**.
2. Under **Local Processes**, double-click one of the top-level entries to connect to it.
3. Click the ▶ icon that appears next to the entry to display a list of its components that are running in the JVM.

Viewing processes in alternate JMX servers

To view information about processes in an alternate JMX server:

1. Section 20.2, "Adding a JMX server" the JMX server to the **JMX Navigator** view.
2. In the **JMX Navigator** view, expand the server's entry by using the ▶ icon that appears next to the entry. This displays a list of that JMX server's components that are running in the JVM.

20.2. ADDING A JMX SERVER

Overview

In the **JMX Navigator** view, under the **Local Processes** branch of the tree, you can see a list of all local JMX servers. You may need to connect to specific JMX servers to see components deployed on other machines.

To add a JMX server, you must know the JMX URL of the server you want to add.

Procedure

To add a JMX server to the **JMX Navigator** view:

1. In the **JMX Navigator** view, click **New Connection** .

2. In the **Create a new JMX connection** wizard, select **Default JMX Connection**.
3. Click **Next**.
4. Select the **Advanced** tab.
5. In the **Name** field, enter a name for the JMX server.
   The name can be any string. It is used to label the entry in the **JMX Navigator** tree.
6. In the **JMX URL** field, enter the JMX URL of the server.
7. If the JMX server requires authentication, enter your user name and password in the **Username** and **Password** fields.
8. Click **Finish**.
   The new JMX server appears as a branch in the **User-Defined Connections** tree.

CHAPTER 21. VIEWING A COMPONENT'S JMX STATISTICS

OVERVIEW

The tooling collects all JMX statistics reported by Fuse components and displays them in the **Properties** view. This statistical information can provide significant insight into what is happening in your integration application.

JMX statistics are grouped into three categories: **Properties**, **Processor**, and **Profile**.

PROCEDURE

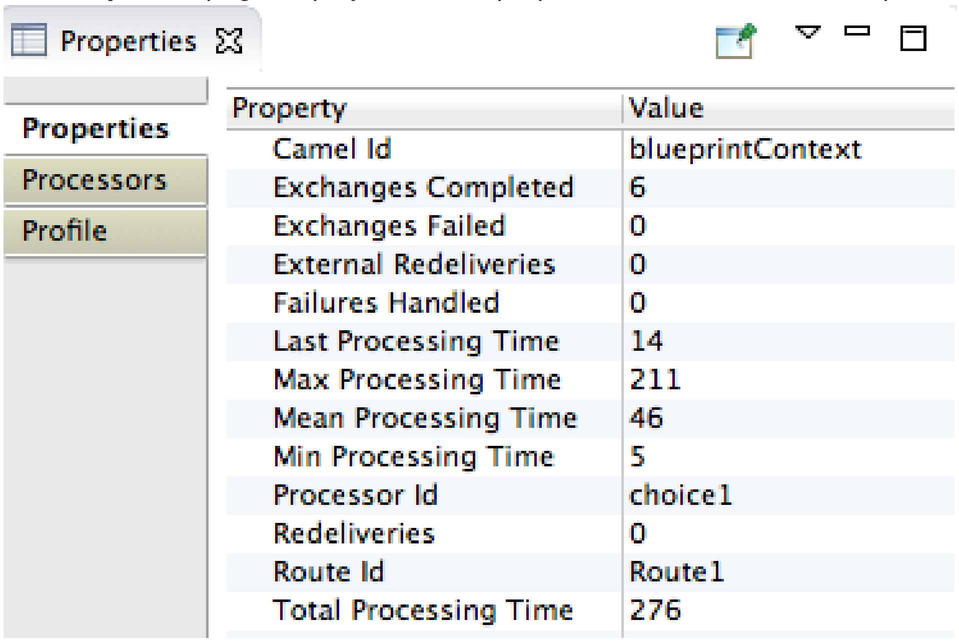
To see a Fuse component's statistics:

1. In the **JMX Navigator** view, locate the node for the component.
   You may have to expand nodes on the tree to locate low-level components.

2. Select the node of the Fuse component whose statistics you want to review.

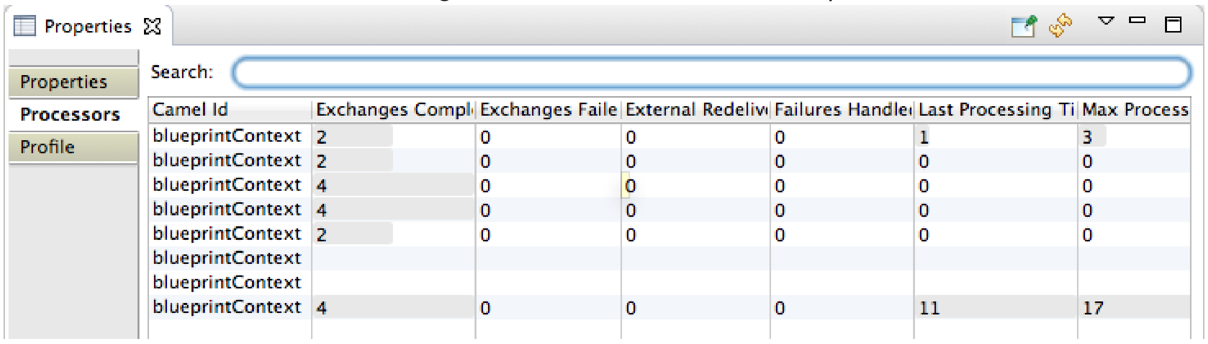
3. Open the **Properties** view.

4. The **Properties** page displays the JMX properties for the selected component:



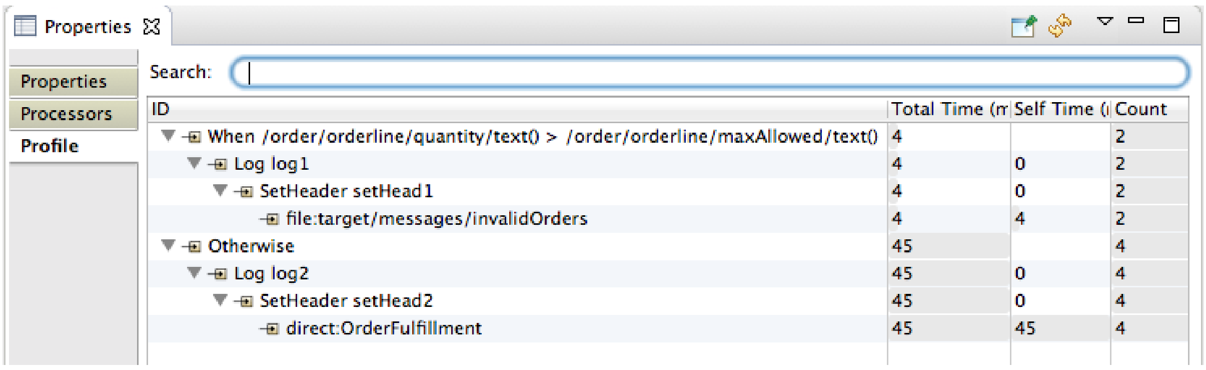
Property	Value
Camel Id	blueprintContext
Exchanges Completed	6
Exchanges Failed	0
External Redeliveries	0
Failures Handled	0
Last Processing Time	14
Max Processing Time	211
Mean Processing Time	46
Min Processing Time	5
Processor Id	choice1
Redeliveries	0
Route Id	Route1
Total Processing Time	276

5. Click **Processors** to check exchange metrics for the selected component:



Camel Id	Exchanges Compl	Exchanges Faile	External Redeliv	Failures Handle	Last Processing Ti	Max Process
blueprintContext	2	0	0	0	1	3
blueprintContext	2	0	0	0	0	0
blueprintContext	4	0	0	0	0	0
blueprintContext	4	0	0	0	0	0
blueprintContext	2	0	0	0	0	0
blueprintContext						
blueprintContext						
blueprintContext	4	0	0	0	11	17

6. Click **Profile** to check message metrics for the selected node and its subnodes:



Properties

Properties

Processors

Profile

Search:

ID	Total Time (m	Self Time (i	Count
When /order/orderline/quantity/text() > /order/orderline/maxAllowed/text()	4		2
Log log1	4	0	2
SetHeader setHead1	4	0	2
file:target/messages/invalidOrders	4	4	2
Otherwise	45		4
Log log2	45	0	4
SetHeader setHead2	45	0	4
direct:OrderFulfillment	45	45	4

CHAPTER 22. BROWSING MESSAGES

OVERVIEW

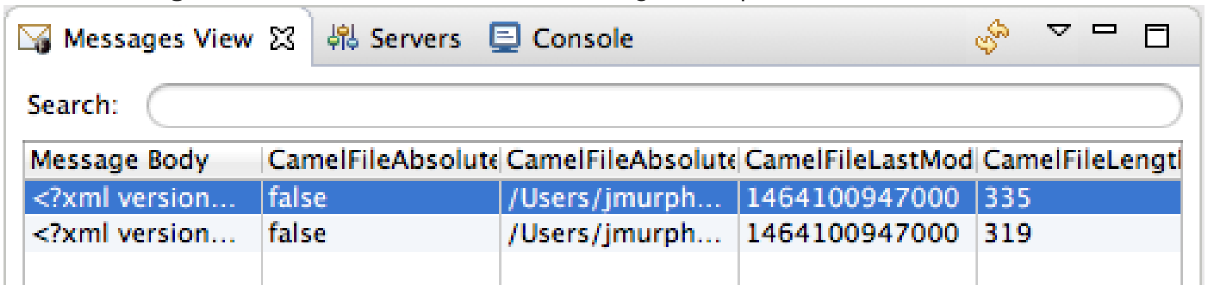
A key tool in debugging applications in a distributed environment is seeing all of the messages stored in the JMS destinations and route endpoints in the application. The tooling can browse the following:

- JMS destinations
- JMS routing endpoints
- Apache Camel routing endpoints
- SEDA routing endpoints
- Browse routing endpoints
- Mock routing endpoints
- VM routing endpoints
- DataSet routing endpoints

PROCEDURE

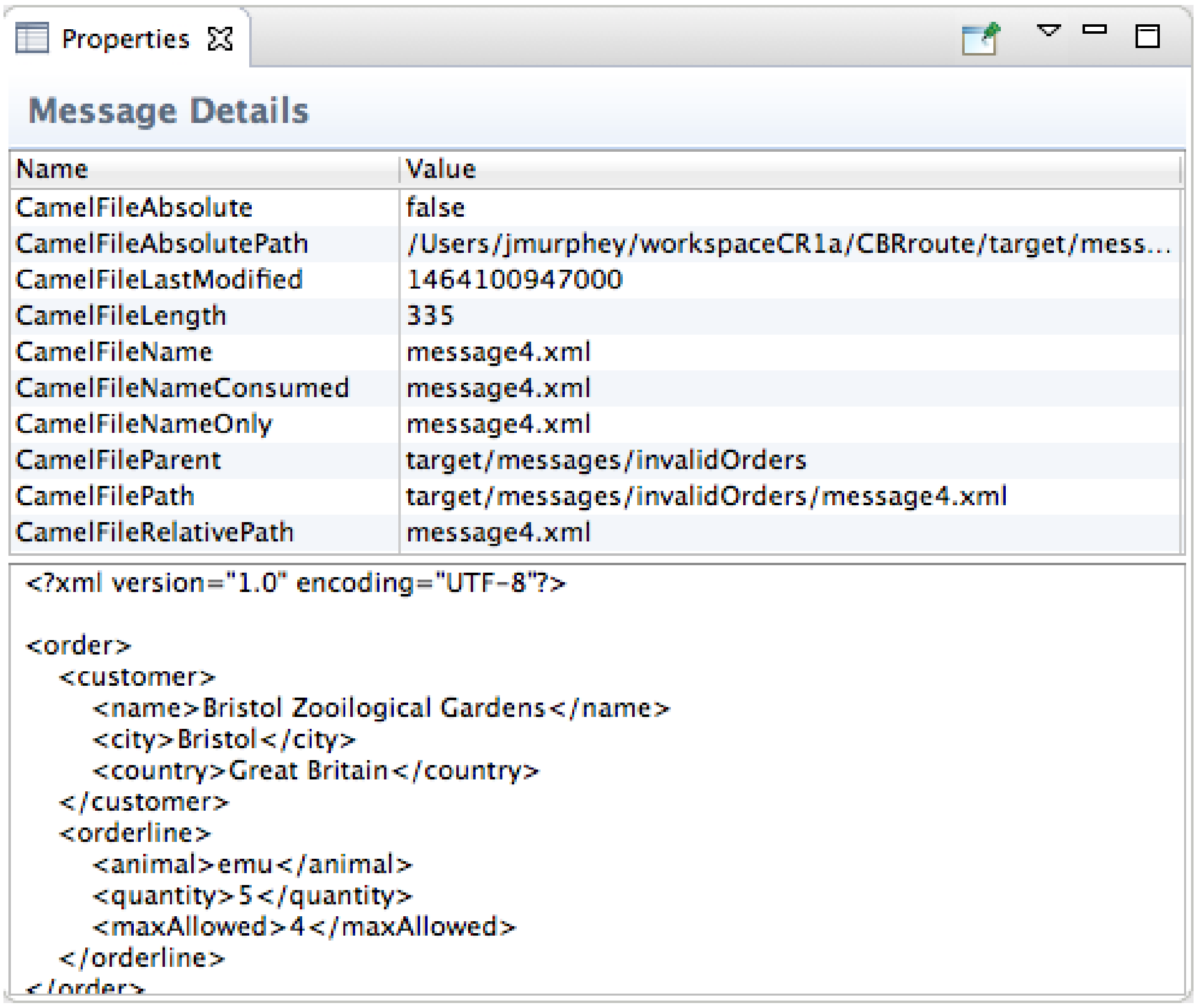
To browse messages:

1. In the **JMX Navigator** view, select the JMS destination or endpoint you want to browse.
   The list of messages appears in the **Messages View**.
2. In the **Messages View**, select an individual message to inspect.



Message Body	CamelFileAbsolute	CamelFileAbsolute	CamelFileLastMod	CamelFileLengt
<?xml version...	false	/Users/jmurph...	1464100947000	335
<?xml version...	false	/Users/jmurph...	1464100947000	319

Message details and content are displayed in the **Properties** view:


Properties

Message Details

Name	Value
CamelFileAbsolute	false
CamelFileAbsolutePath	/Users/jmurphey/workspaceCR1a/CBRroute/target/mess...
CamelFileLastModified	1464100947000
CamelFileLength	335
CamelFileName	message4.xml
CamelFileNameConsumed	message4.xml
CamelFileNameOnly	message4.xml
CamelFileParent	target/messages/invalidOrders
CamelFilePath	target/messages/invalidOrders/message4.xml
CamelFileRelativePath	message4.xml

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

<order>
   <customer>
      <name>Bristol Zooilogical Gardens</name>
      <city>Bristol</city>
      <country>Great Britain</country>
   </customer>
   <orderline>
      <animal>emu</animal>
      <quantity>5</quantity>
      <maxAllowed>4</maxAllowed>
   </orderline>
</order>
```

RELATED TOPICS

- Section 23.3, “Tracing messages through a routing context”

CHAPTER 23. TRACING ROUTES

Debugging a route often involves solving one of two problems:

- A message was improperly transformed.
- A message failed to reach its destination endpoint.

Tracing one or more test messages through the route is the easiest way to discover the source of such problems.

The tooling's route tracing feature enables you to monitor the path a message takes through a route and see how the message is transformed as it passes from processor to processor.

The **Diagram View** displays a graphical representation of the route, which enables you to see the path a message takes through it. For each processor in a route, it also displays the average processing time, in milliseconds, for all messages processed since route start-up and the number of messages processed since route start-up.

The **Messages View** displays the messages processed by a JMS destination or route endpoint selected in the **JMX Navigator** tree. Selecting an individual message trace in the **Messages View** displays the full details and content of the message in the **Properties** view and highlights the correspoding node in the **Diagram View**.

Tracing messages through a route involves the following steps:

1. Section 23.1, "Creating test messages for route tracing"
2. Section 23.2, "Activating route tracing"
3. Section 23.3, "Tracing messages through a routing context"
4. Section 23.4, "Deactivating route tracing"

23.1. CREATING TEST MESSAGES FOR ROUTE TRACING

Overview

Route tracing works with any kind of message structure. The **Fuse Message** wizard creates an empty `.xml` message, leaving the structuring of the message entirely up to you.

NOTE

If the folder where you want to store the test messages does not exist, you need to create it before you create the messages.

Creating a new folder to store test messages

To create a new folder:

1. In the **Project Explorer** view, right-click the project root to open the context menu.
2. Select **New** → **Folder** to open the **New Folder** wizard.
   The project root appears in the **Enter or select the parent folder** field.

3. Expand the nodes in the graphical representation of the project’s hierarchy, and select the node you want to be the parent folder.

4. In the **Folder name** field, enter a name for the new folder.

5. Click **Finish**.
   The new folder appears in the **Project Explorer** view, under the selected parent folder.

   NOTE

   If the new folder does not appear, right-click the parent foler and select **Refresh**.

Creating a test message

To create a test message:

1. In the **Project Explorer** view, right-click the project to open the context menu.

2. Select **New** → **Fuse Message** to open the **New File** wizard.

3. Expand the nodes in the graphical representation of the project’s hierarchy, and select the folder in which you want to store the new test message.

4. In the **File name** field, enter a name for the message, or accept the default (`message.xml`).

5. Click **Finish**.
   The new message opens in the XML editor.

6. Enter the message contents, both body and header text.

   NOTE

   You may see the warning, **`No grammar constraints (DTD or XML Schema) referenced in the document`**, depending on the header text you entered. You can safely ignore this warning.

Related topics

- Section 23.3, “Tracing messages through a routing context”

23.2. ACTIVATING ROUTE TRACING

Overview

You must activate route tracing for the routing context before you can trace messages through that routing context.

Procedure

To activate tracing on a routing context:

1. In the **JMX Navigator** view, select the running routing context on which you want to start tracing.

NOTE

You can select any route in the context to start tracing on the entire context.

2. Right-click the selected routing context to open the context menu, and then select **Start Tracing** to start the trace.
   If **Stop Tracing Context** is enabled on the context menu, then tracing is already active.

Related topics

- Section 23.3, “Tracing messages through a routing context”
- Section 23.4, “Deactivating route tracing”

23.3. TRACING MESSAGES THROUGH A ROUTING CONTEXT

Overview

The best way to see what is happening in a routing context is to watch what happens to a message at each stop along the way. The tooling provides a mechanism for dropping messages into a running routing context and tracing the path the messages take through it.

Procedure

To trace messages through a routing context:

1. Create one or more test messages as described in Section 23.1, “Creating test messages for route tracing”.
2. In the **Project Explorer** view, right-click the project’s Camel context file to open the context menu, and select **Run As → Local Camel Context (without Tests)**.

   NOTE

   Do not run it as **Local Camel Context** unless you have created a comprehensive JUnit test for the project.

3. Activate tracing for the running routing context as described in Section 23.2, “Activating route tracing”.
4. Drag one of the test messages from the **Project Explorer** view onto the routing context’s starting point in the **JMX Navigator** view.
5. In the **JMX Navigator** view, select the routing context being traced.
   The tooling populates the **Messages View** with message instances that represent the message at each stage in the traced context.

   The **Diagram View** displays a graphical representation of the selected routing context.
6. In the **Messages View**, select one of the message instances.
   The **Properties** view displays the details and content of the message instance.

In the **Diagram View**, the route step corresponding to the selected message instance is highlighted. If the route step is a processing step, the tooling tags the exiting path with timing and processing metrics.

7. Repeat this prodedure as needed.

Related topics

- Section 23.1, “Creating test messages for route tracing”
- Section 23.2, “Activating route tracing”
- Section 23.4, “Deactivating route tracing”

23.4. DEACTIVATING ROUTE TRACING

Overview

When you are finished debugging the routes in a routing context, you should deactivate tracing.

IMPORTANT

Deactivating tracing stops tracing and flushes the trace data for all of the routes in the routing context. This means that you cannot review any past tracing sessions.

Procedure

To stop tracing for a routing context:

1. In the **JMX Navigator** view, select the running routing context for which you want to deactivate tracing.

   NOTE

   You can select any route in the context to stop tracing for the context.

2. Right-click the selected routing context to open the context menu, and then select **Stop Tracing Context**.
   If **Start Tracing** appears on the context menu, tracing is not activated for the routing context.

Related topics

- Section 23.2, “Activating route tracing”
- Section 23.3, “Tracing messages through a routing context”

CHAPTER 24. MANAGING JMS DESTINATIONS

The **JMX Navigator** view lets you add or delete JMS destinations in a running instance of Red Hat Fuse.

IMPORTANT

These changes are not persistent across broker restarts.

24.1. ADDING A JMS DESTINATION

Overview

When testing a new scenario, it is convenient to add a new JMS destination to one of your brokers.

Procedure

To add a JMS destination to a broker:

1. In the **JMX Navigator** view, under the broker node for which you want to add a destination, select either the **Queues** child or the **Topics** child.
2. Right-click the selected node to open the context menu, and then select either **Create Queue** or **Create Topic**.
3. In either the **Create Queue** or **Create Topic** dialog, enter a name for the new destination.
4. Click **OK**.
5. Right-click either the **Queues** or the **Topics** child, and then select **Refresh**.
   The new destination appears in the **JMX Navigator** view under the **Queues** child or the **Topics** child.

Related topics

- Section 24.2, “Deleting a JMS destination”

24.2. DELETING A JMS DESTINATION

Overview

When testing failover scenarios or other scenarios that involve handling failures, it is helpful to be able to easily remove a JMS destination.

Procedure

To delete a JMS destination:

1. In the **JMX Navigator** view, under the **Queues** child or the **Topics** child, select the JMS destination you want to delete.
2. Right-click the selected destination to open the context menu, and then select **Delete Queue/Topic**.

Related topics

- Section 24.1, “Adding a JMS destination”

CHAPTER 25. MANAGING ROUTING ENDPOINTS

The **JMX Navigator** view lets you add or delete routing endpoints.

IMPORTANT

These changes are not persistent across routing context restarts.

25.1. ADDING A ROUTING ENDPOINT

Overview

When testing a new scenario, you might want to add a new endpoint to a routing context.

Procedure

To add an endpoint to a routing context:

1. In the **JMX Navigator** view, under the routing context node, select the **Endpoints** child to which you want to add an endpoint.
2. Right-click the selected node to open the context menu, and then select **Create Endpoint**.
3. In the **Create Endpoint** dialog, enter a URL that defines the new endpoint, for example, `file://target/messages/validOrders`.
4. Click **OK**.
5. Right-click the routing context node, and select **Refresh**.
   The new destination appears in the **JMX Navigator** view under the **Endpoints** node, in a folder that corresponds to the type of endpoint it is, for example, **file**.

Related topics

- Section 25.2, “Deleting a routing endpoint”

25.2. DELETING A ROUTING ENDPOINT

Overview

When testing failover scenarios or other scenarios that involve handling failures, it is helpful to be able to remove an endpoint from a routing context.

Procedure

To delete a routing endpoint:

1. In the **JMX Navigator** view, select the endpoint you want delete.
2. Right-click the selected endpoint to open the context menu, and then select **Delete Endpoint**.
   The tooling deletes the endpoint.

3. To remove the deleted endpoint from the view, right-click the **Endpoints** node, and select **Refresh**.
   The endpoint disappears from the **JMX Navigator** view.

NOTE

To remove the endpoint's node from the **Project Explorer** view without rerunning the project, you need to explicitly delete it by right-clicking the node and selecting **Delete**. To remove it from view, refresh the project display.

Related topics

- Section 25.1, "Adding a routing endpoint"

CHAPTER 26. EDITING RUNNING ROUTES

OVERVIEW

You can experiment with changes to a running route without changing your project's routing context.

To do so:

- In the **JMX Navigator** view, enable the **Edit Routes** option on the running routing context.
  This opens an in-memory model of it — **Remote CamelContext:<camelContextId>** — in the route editor.

- In the route editor, make your changes to the in-memory model of the routing context. At the same time, you can set breakpoints on relevant nodes to use the Camel debugger and all of its features.
  You can edit the in-memory model to add, remove, or rearrange nodes; to add or remove properties of existing nodes; and to modify property values set on existing nodes. You must save changes made to the in-memory model to update the running context and to see results in the **Debug** perspective if you set breakpoints.

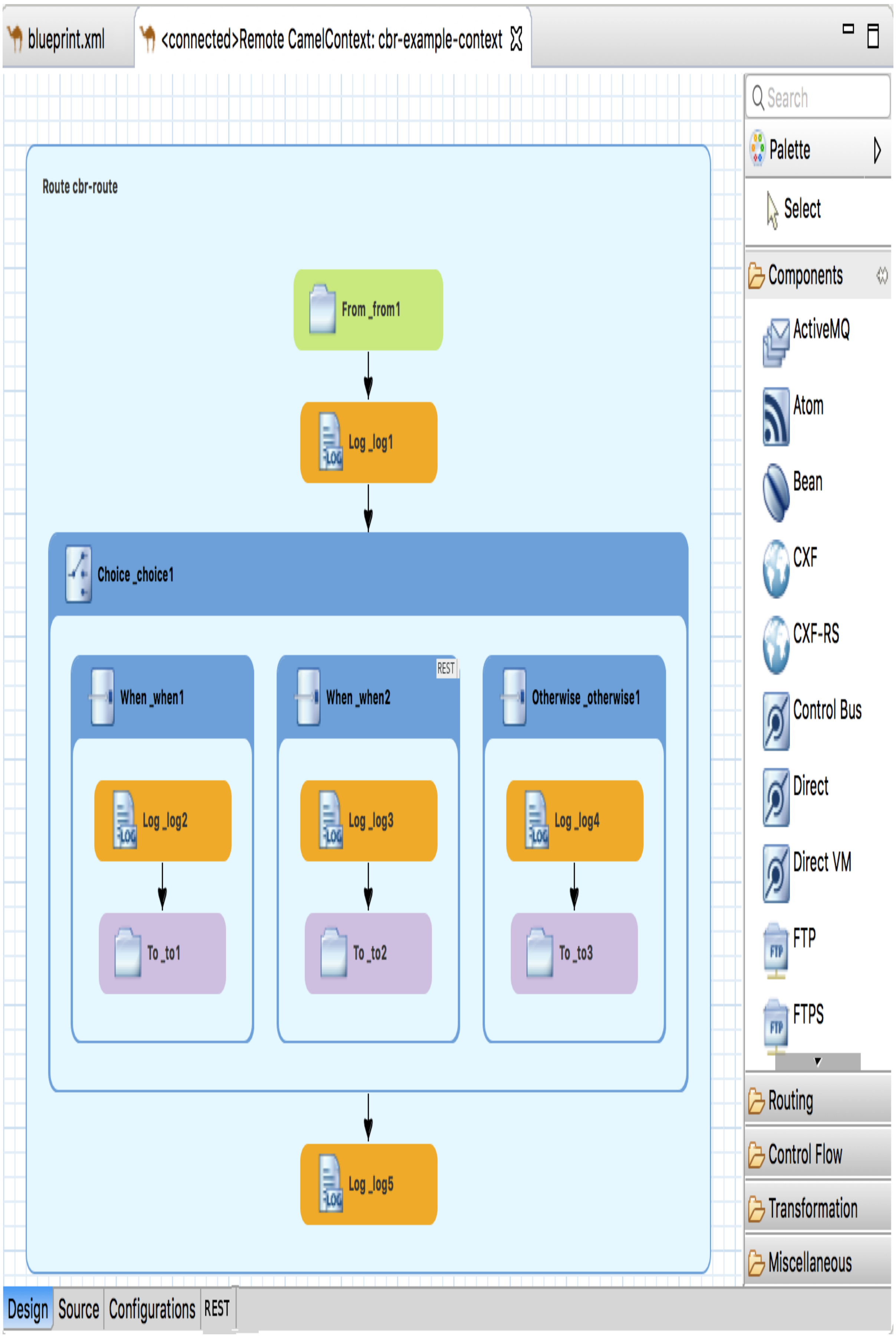
- In the **JMX Navigator** view, drop messages on the running routing context, or wait for messages to arrive from a timer, ActiveMQ, file, or other continuous input node.

- In the **Debug** perspective, evaluate results and use the Camel debugger to gain deeper insight into your routing context.

MODIFYING A RUNNING ROUTE AND EVALUATING RESULTS

1. In the **JMX Navigator** view, select the routing context that contains the routes you want to edit.

2. Right-click the selected routing context to open the context menu, and select **Edit Routes**.
   The route editor opens an in-memory model of the routing context, **Remote CamelContext: <contextId>**, and displays all routes in the context, for example:


blueprint.xml
<connected>Remote CamelContext: cbr-example-context
Route cbr-route
From _from1
Log _log1
Choice _choice1
When _when1
When _when2
REST
Otherwise _otherwise1
Log _log2
Log _log3
Log _log4
To _to1
To _to2
To _to3
Log _log5
Search
Palette
Select
Components
ActiveMQ
Atom
Bean
CXF
CXF-RS
Control Bus
Direct
Direct VM
FTP
FTPS
Routing
Control Flow
Transformation
Miscellaneous
Design
Source
Configurations
REST

> **NOTE**
>
> `<contextId>` is the ID of the `camelContext` element in the project's routing context `.xml` file. In this example, which is based on the **Fuse → Content Based Router** built-in template, the ID is `cbr-example-context`.

3. Edit the route as described in Chapter 2, *Editing a routing context in the route editor*, then select **File → Save** to save the changes you made to the in-memory model and to update the running routing context.

4. Set breakpoints on the relevant nodes as described in Chapter 14, *Setting Breakpoints*.

5. In the **JMX Navigator** view, drop a message on the running routing context's input node.
   If your project does not include test messages, you can create them as described in Section 23.1, "Creating test messages for route tracing".

6. Click **Yes** to confirm the switch to the **Debug** perspective.

7. In the Camel debugger, step the message through the breakpoints as you normally would (see Chapter 15, *Running the Camel Debugger*) to see the results your changes generated.
   The Camel debugger behaves the same in **Edit Routes** mode as in normal debug mode, so you can use any of the Camel debugger's features while a message is transiting the routing context.

> **NOTE**
>
> When a message reaches the end of the routing context, the debugger is suspended. To continue debugging, switch back to the **Fuse Integration** perspective and drop another message on the input node in the **JMX Navigator** view. Each time you do so, the tooling asks you to confirm the switch to the **Debug** perspective.

> **NOTE**
>
> During a route editing session, it is possible to lose the connection to the running routing context. If this happens, then in the **JMX Navigator** view, you would see something like this: **Local Processes → maven[xxxx][Disconnected]**. To continue the session, you must reconnect to the running routing context, select it in the **JMX Navigator** view, and then re-select **Edit Routes**.

TERMINATING THE ROUTE EDITING SESSION

1. In the **Debug** perspective's **Debug** view, select the **Remote Camel Debug - camelContext--<contextId>--xxxxxxxxxxxxxxxxxx.xml [Remote Camel Context]** thread, and then click 🟥 on the menu bar to terminate the debugging session.

2. On **Console** view's menu bar, click 🟥 to terminate the routing context.

3. If you want to clear console output, click on the **Console** view's menu bar.

4. Switch to the **Fuse Integration** perspective, and in the route editor, click on the **Remote CamelContext:<contextId>** tab to close the in-memory model of the routing context file.

RELATED TOPICS

- Chapter 2, *Editing a routing context in the route editor*
- Part II, “Debugging Routing Contexts”

CHAPTER 27. MANAGING ROUTING CONTEXTS

The **JMX Navigator** view lets you suspend and resume running routing contexts.

27.1. SUSPENDING OPERATION OF A ROUTING CONTEXT

Overview

The tooling enables you to suspend the operation of a routing context in the **JMX Navigator** view. Suspending context operation gracefully shuts down all routes in the context, but keeps them loaded in memory, so that they can resume operation.

Procedure

To suspend operation of a routing context:

1. In the **JMX Navigator** view, expand the project's **Camel Contexts** node, and select the routing context whose operation you want to suspend.
2. Right-click the selected routing context to open the context menu, and then select **Suspend Camel Context**.

NOTE

If **Resume Camel Context** appears on the context menu, operation of the context is already suspended.

Related topics

- Section 27.2, "Resuming operation of a routing context"

27.2. RESUMING OPERATION OF A ROUTING CONTEXT

Overview

The tooling lets you resume operation of a suspended routing context in the **JMX Navigator** view. Resuming operation of a context restarts all of the routes in it so that they can process messages.

Procedure

To resume operation of a routing context:

1. In the **JMX Navigator** view, expand the project's **Camel Contexts** node, and select the routing context whose operation you want to resume.
2. Right-click the selected context to open the context menu, and then select **Resume Camel Context**.

NOTE

If **Suspend Camel Context** appears in the context menu, the context and its routes are running.

Related topics

- Section 27.1, “Suspending operation of a routing context”

PART IV. PUBLISHING APPLICATIONS TO A CONTAINER

To publish Fuse Integration projects to a server container, you must first add the server and its runime definition to the tooling’s **Servers** list. Then you can assign projects to the server runtime and set the publishing options for it.

CHAPTER 28. MANAGING SERVERS

The **Servers** view lets you run and manage servers in your Red Hat Developer Studio environment. The supported servers are:

- Red Hat Fuse Server — version 7.0
- Red Hat Enterprise Application Platform — version 7.1

NOTE

For step by step instructions on how to publish a Camel project to Red Hat Fuse, see Chapter 29, *Publishing Fuse Integration Projects to a Server*.

28.1. ADDING A SERVER

Overview

For the tooling to manage a server, you need to add the server to the **Servers** list. Once added, the server appears in the **Servers** view, where you can connect to it and publish your Fuse Integration projects.

NOTE

If adding a Red Hat Fuse, it is recommended that you edit its *installDir*`/etc/users.properties` file and add user information, in the form of `user=password,role`, to enable the tooling to establish an SSH connection to the server.

Procedure

There are three ways to add a new server to the **Servers** view:

- In the **Servers** view, click `No servers are available. Click this link to create a new server…`.

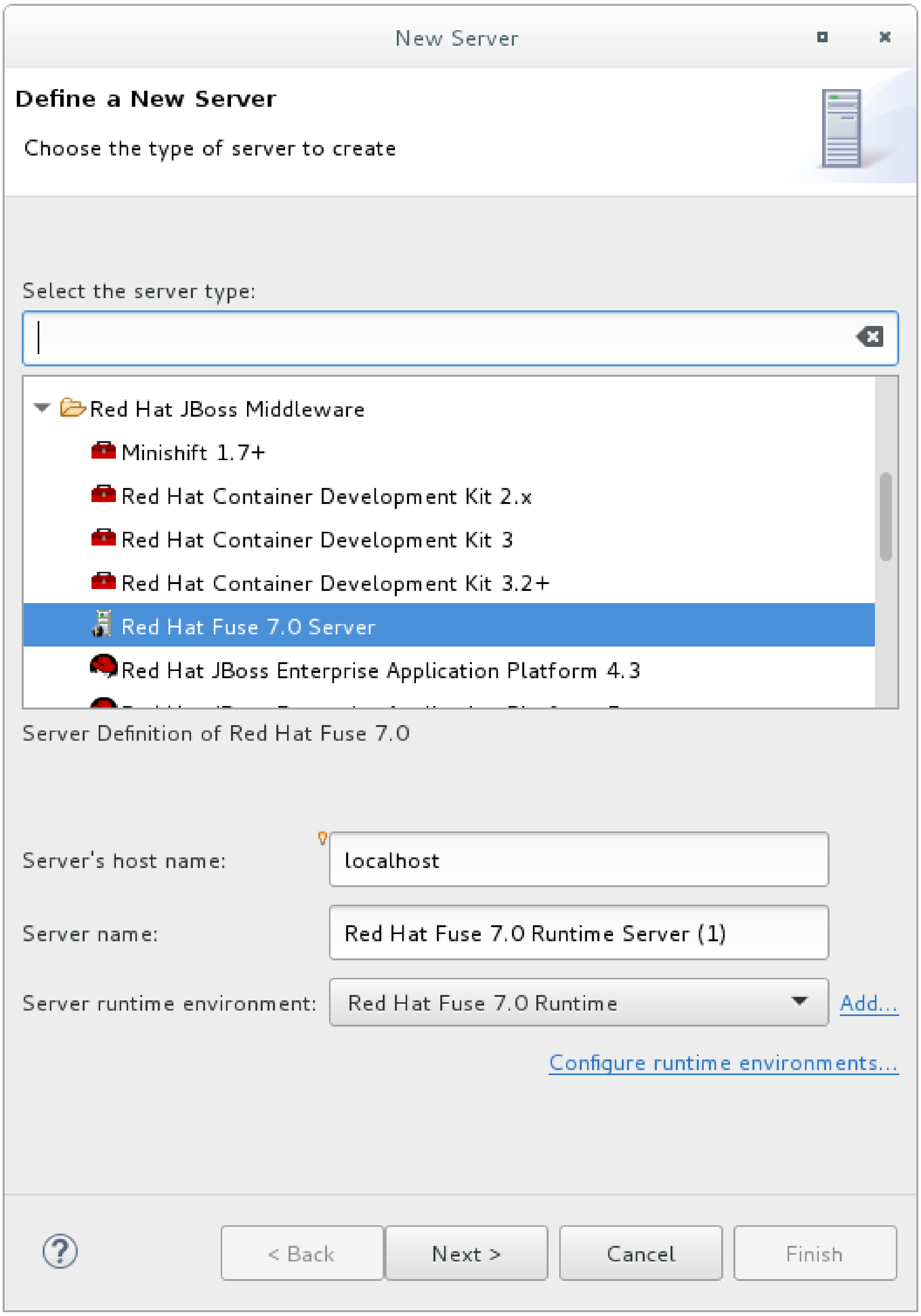
  NOTE

  This link appears in the **Servers** view only when no server has been defined. If you defined and added a server when you first created your project, the **Servers** view displays that server.

- In the **Servers** view, right-click to open the context menu and select**New** → **Server**.
- On the menu bar, select **File** → **New** → **Other** → **Server** → **Server**.

In the **Define a New Server** dialog, to add a new server:

1. Expand the **Red Hat JBoss Middleware** node to expose the list of available server options:



Define a New Server

Choose the type of server to create

Select the server type:

Red Hat JBoss Middleware
- Minishift 1.7+
- Red Hat Container Development Kit 2.x
- Red Hat Container Development Kit 3
- Red Hat Container Development Kit 3.2+
- Red Hat Fuse 7.0 Server
- Red Hat JBoss Enterprise Application Platform 4.3

Server Definition of Red Hat Fuse 7.0

Server's host name: localhost

Server name: Red Hat Fuse 7.0 Runtime Server (1)

Server runtime environment: Red Hat Fuse 7.0 Runtime

2. Click the server that you want to add.

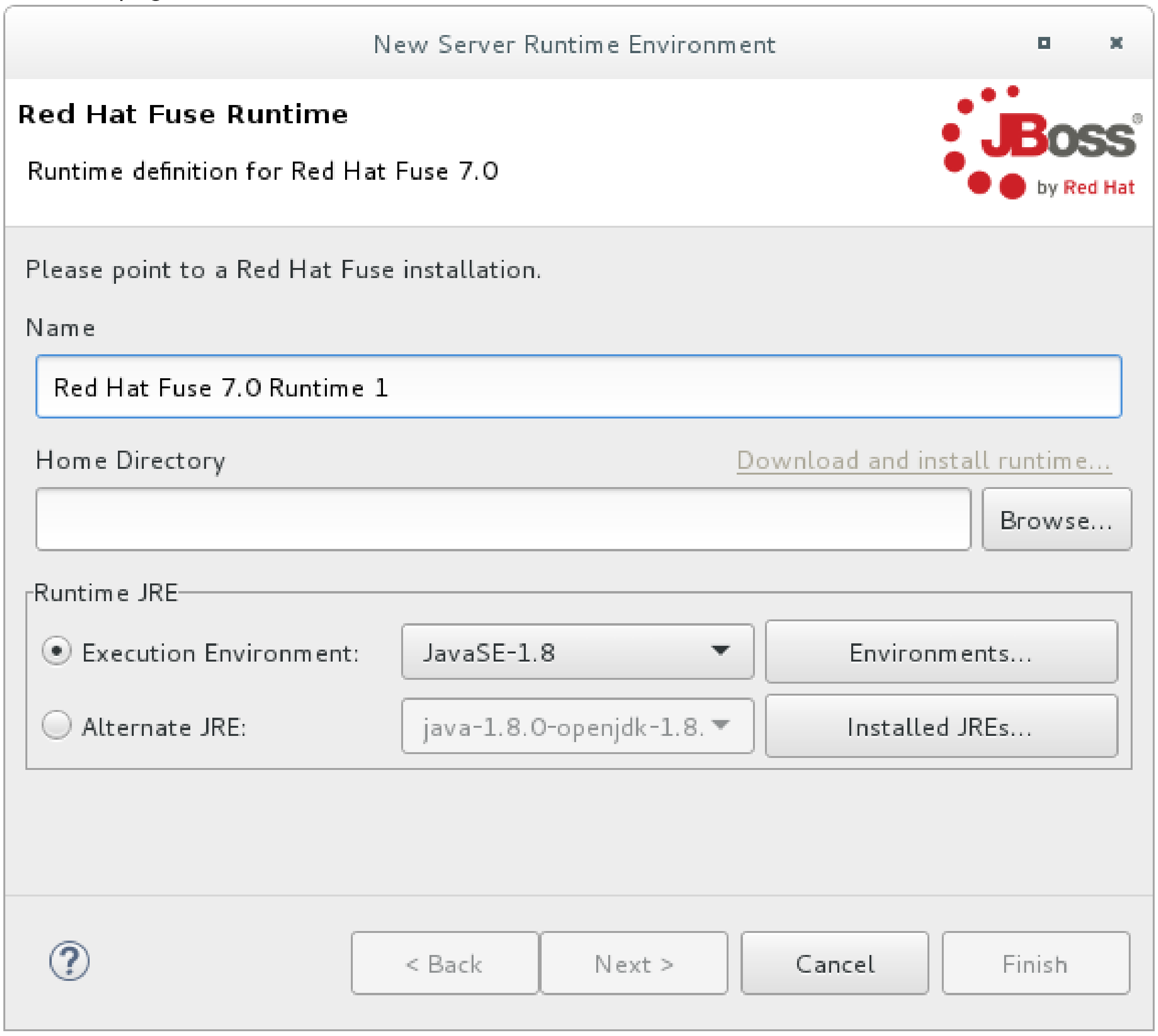
3. In the **Server's host name** field, accept the default (`localhost`).

NOTE

The address of `localhost` is `0.0.0.0`.

4. In the **Server name** field, accept the default, or enter a different name for the runtime server.

5. For **Server runtime environment**, accept the default or click **Add** to open the server's runtime definition page:


New Server Runtime Environment

Red Hat Fuse Runtime

Runtime definition for Red Hat Fuse 7.0

Please point to a Red Hat Fuse installation.

Name: Red Hat Fuse 7.0 Runtime 1

Home Directory

Runtime JRE

Execution Environment: JavaSE-1.8

Alternate JRE: java-1.8.0-openjdk-1.8.

NOTE

If the server is not already installed on your machine, you can install it now by clicking **Download and install runtime…** and following the site's download instructions. Depending on the site, you might be required to provide valid credentials before you can continue the download process.

6. Accept the default for the installation **Name**.

7. In the **Home Directory** field, enter the path where the server runtime is installed, or click **Browse** to find and select it.

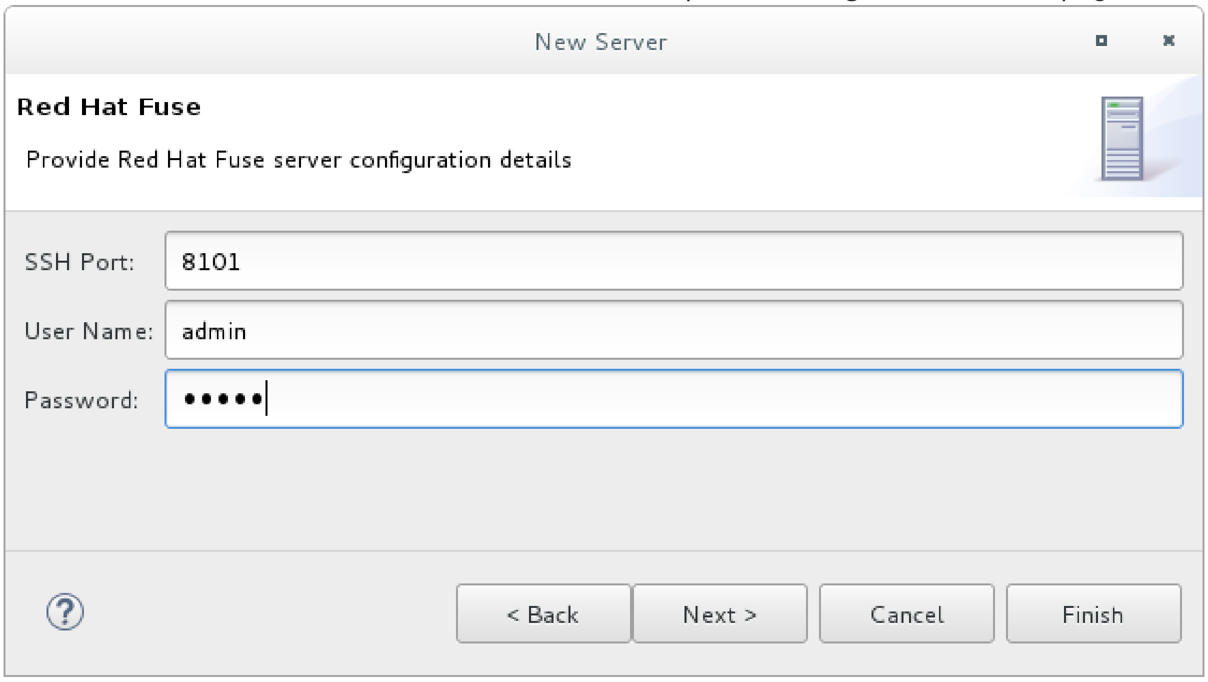
8. Next to **Execution Environment**, select the runtime JRE from the drop-down menu. If the version you want does not appear in the list, click **Environments** and select the version from the list that appears. The JRE version you select must be installed on your machine.

NOTE

Red Hat Fuse 7.0 requires JRE version 1.8.

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

10. Click **Next** to save the server's runtime definition and open its **Configuration details** page:



SSH Port: 8101
User Name: admin
Password: •••••

11. 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 for your setup, you can discover the correct port number by looking in the server's *installDir*/**`etc/org.apache.karaf.shell.cfg`** file.

12. In the **User Name** field, enter the name used to log into the server.
    For Red Hat Fuse, 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 the **User Name** and **Password** fields with the default user's name and password, as shown in [servCnfigDetails].

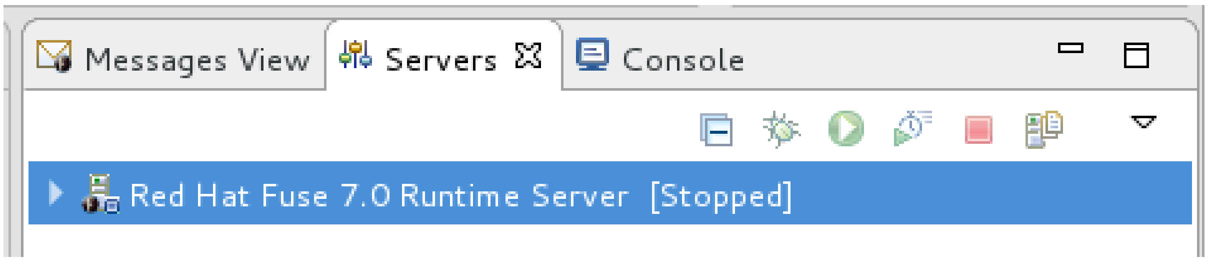
    If a user has not been set up, you can either add one to that file by using the format **`user=password,role`** (for example, **`joe=secret,Administrator`**), or you can set one using the karaf **`jaas`** command set:

    - **`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`**.

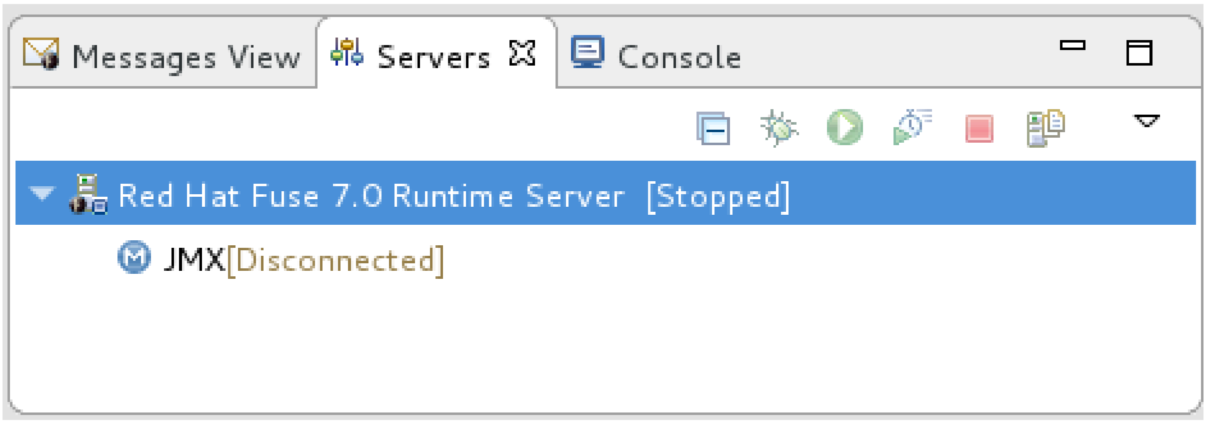
13. In the **Password** field, enter the password required for **User Name** to log into the server.

14. Click **Finish** to save the server's configuration details.
The server runtime appears in the **Servers** view. For example:



Red Hat Fuse 7.0 Runtime Server [Stopped]

Expanding the server node exposes the server's JMX node:



Red Hat Fuse 7.0 Runtime Server [Stopped]
JMX[Disconnected]

28.2. STARTING A SERVER

Overview

When you start a configured server, the tooling opens the server's remote management console in the **Terminal** view. This allows you to easily manage the container while testing your application.

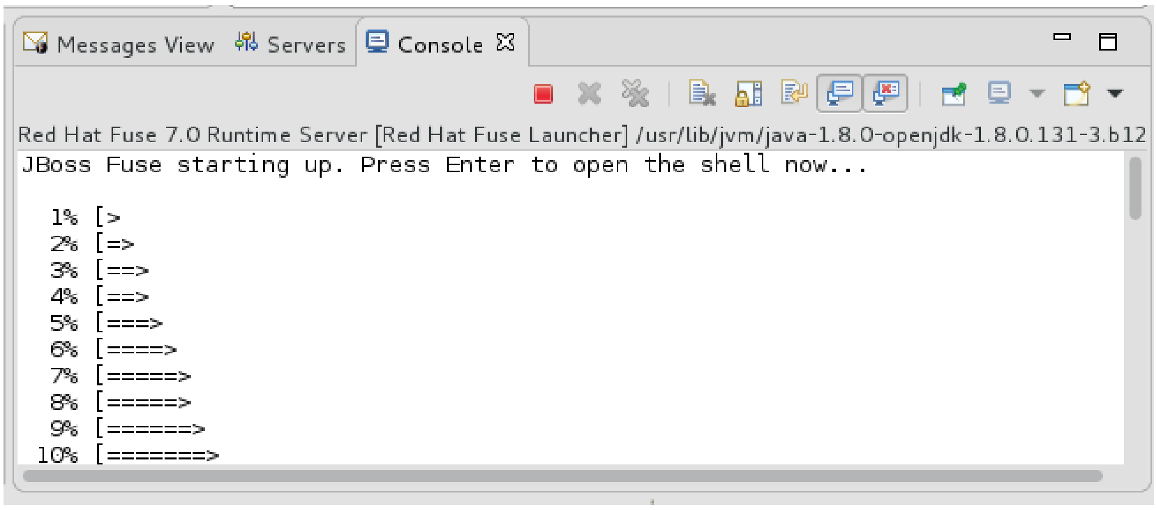
Procedure

To start a server:

1. In the **Servers** view, select the server you want to start.

2. Click ▶ .

   - The **Console** view opens and displays a message asking you to wait while the container is starting, for example:

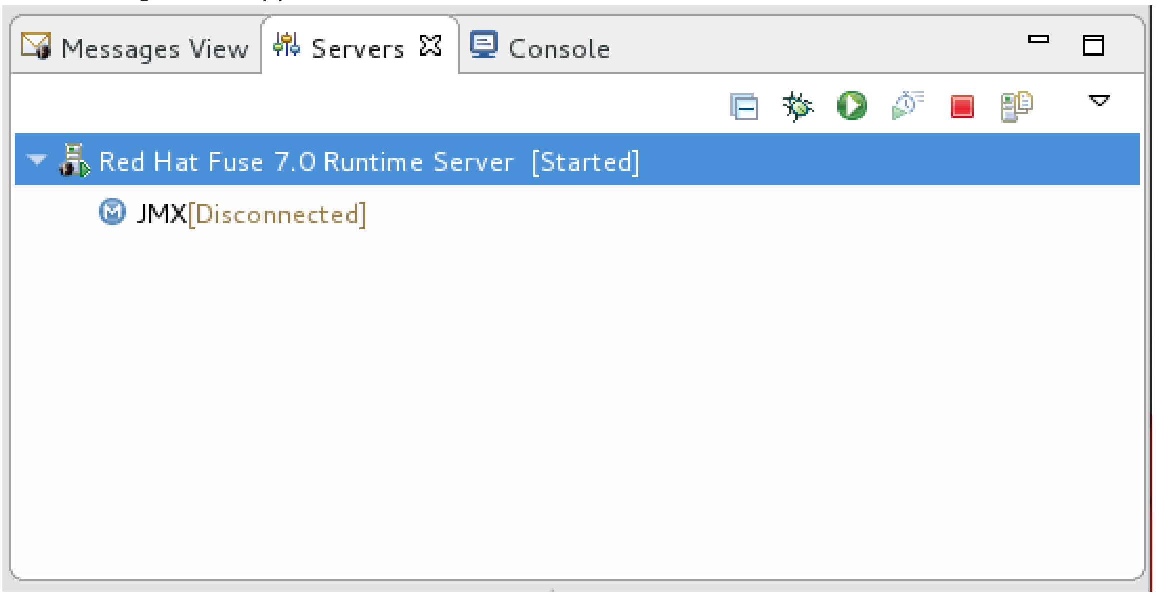


```
Red Hat Fuse 7.0 Runtime Server [Red Hat Fuse Launcher] /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.131-3.b12
JBoss Fuse starting up. Press Enter to open the shell now...

  1% [>
  2% [=>
  3% [==>
  4% [==>
  5% [===>
  6% [====>
  7% [=====>
  8% [=====>
  9% [======>
 10% [=======>
```

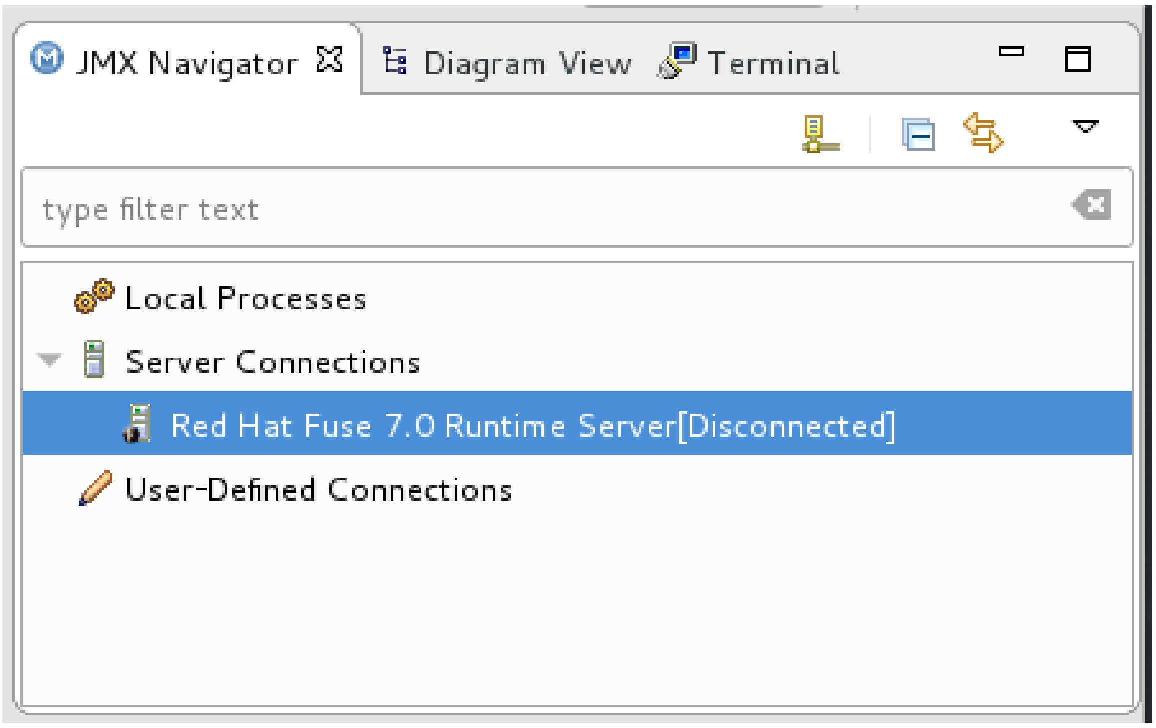
NOTE

If you did not properly configure the user name and password for opening the remote console, a dialog opens asking you to enter the proper credentials. See Section 28.1, “Adding a Server”.

- After the container has started up, the **Terminal** view opens to display the container’s management console.
- The running server appears in the **Servers** view:



- The running server also appears in the **JMX Navigator** view under **Server Connections**:



NOTE

If the server is running on the same machine as the tooling, the server also has an entry under **Local Processes**.

28.3. CONNECTING TO A RUNNING SERVER

Overview

After you start a configured server, it appears in the **Servers** view and in the **JMX Navigator** view under the **Server Connections** node. You may need to expand the **Server Connections** node to see the server.

To publish and test your Fuse project application on the running server, you must first connect to it. You can connect to a running server either in the **Servers** view or in the **JMX Navigator** view.

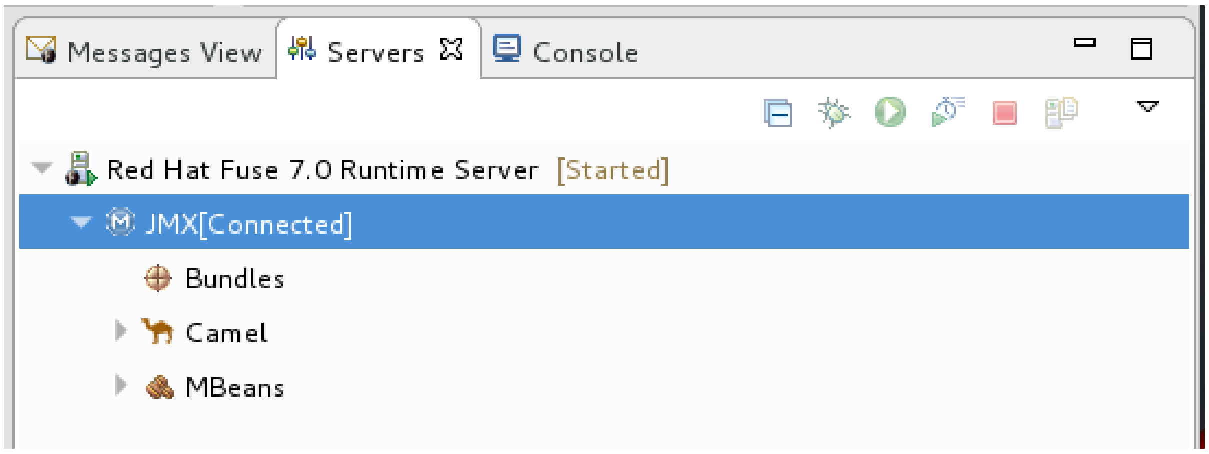
NOTE

The **Servers** view and the **JMX Navigator** view are synchronized with regards to server connections. That is, connecting to a server in the **Servers** view also connects it in the **JMX Navigator** view, and vice versa.

Connecting to a running server in the Servers view

1. In the **Servers** view, expand the server runtime to expose its **JMX[Disconnected]** node.

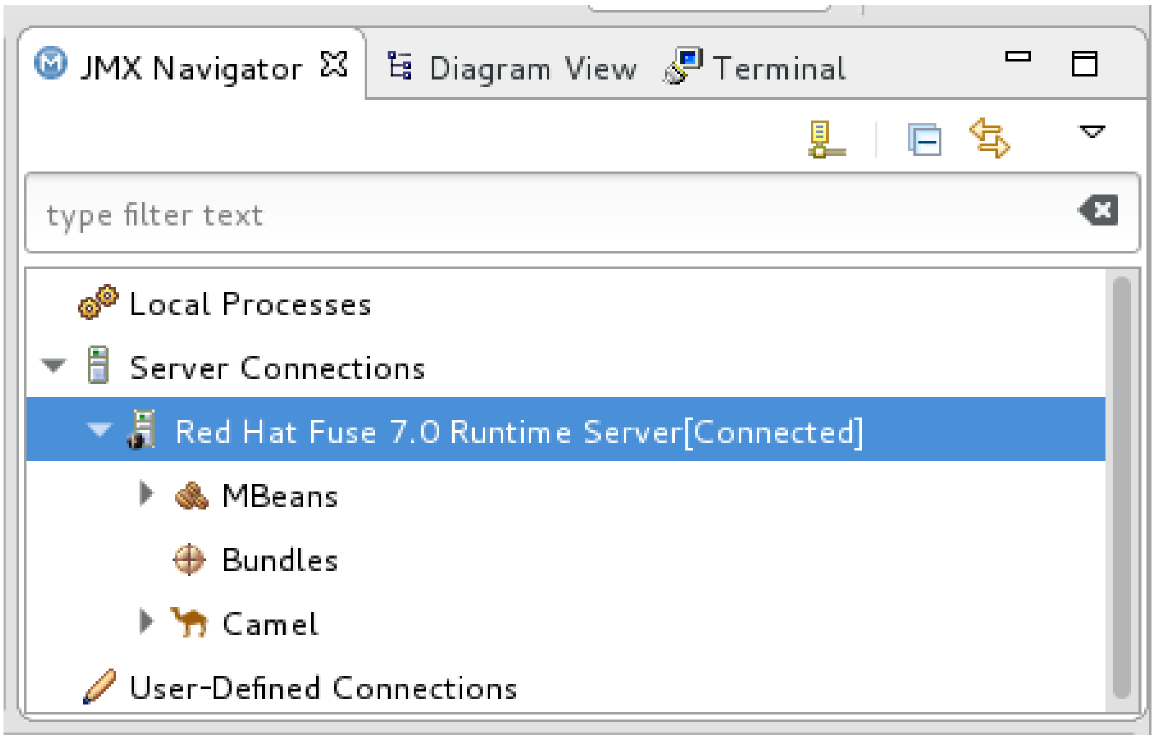
2. Double-click the **JMX[Disconnected]** node:



Connecting to a running server in the JMX Navigator view

1. In the **JMX Navigator** view, under the **Server Connections** node, select the server to which you want to connect.

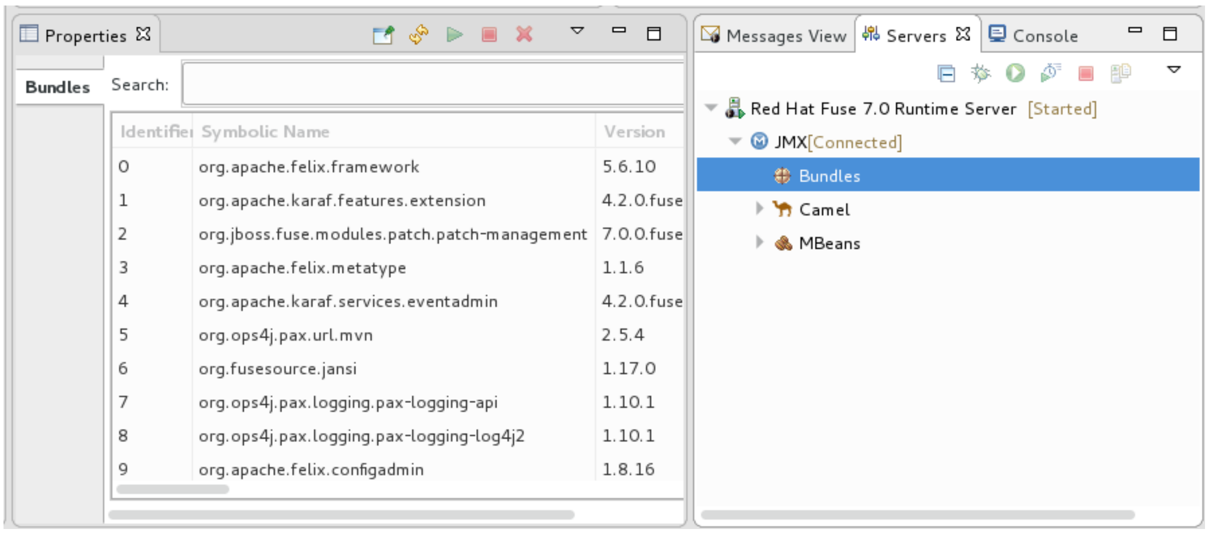
2. Double-click the selected server:



Viewing bundles installed on the connected server

1. In either the **Servers** view or the **JMX Navigator** view, expand the server runtime tree to expose the **Bundles** node, and select it.

2. The tooling populates the **Properties** view with a list of bundles that are installed on the server:



Identifier	Symbolic Name	Version
0	org.apache.felix.framework	5.6.10
1	org.apache.karaf.features.extension	4.2.0.fuse
2	org.jboss.fuse.modules.patch.patch-management	7.0.0.fuse
3	org.apache.felix.metatype	1.1.6
4	org.apache.karaf.services.eventadmin	4.2.0.fuse
5	org.ops4j.pax.url.mvn	2.5.4
6	org.fusesource.jansi	1.17.0
7	org.ops4j.pax.logging.pax-logging-api	1.10.1
8	org.ops4j.pax.logging.pax-logging-log4j2	1.10.1
9	org.apache.felix.configadmin	1.8.16

```
Red Hat Fuse 7.0 Runtime Server  [Started]
  JMX[Connected]
    Bundles
    Camel
    MBeans
```

Using the **Properties** view's **Search** tool, you can search for bundles by their **Symbolic Name** or by their **Identifier**, if you know it. As you type the symbolic name or the identifier, the list updates, showing only the bundles that match the current search string.

NOTE

Alternatively, you can issue the **`osgi:list`** command in the **Terminal** view to see a generated list of bundles installed on the Red Hat Fuse server runtime. The tooling uses a different naming scheme for OSGi bundles displayed by the **`osgi:list`** command.

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. For more details, see the section called "Verifying the project was published to the server".

28.4. DISCONNECTING FROM A SERVER

Overview

When you are done testing your application, you can disconnect from the server without stopping it.

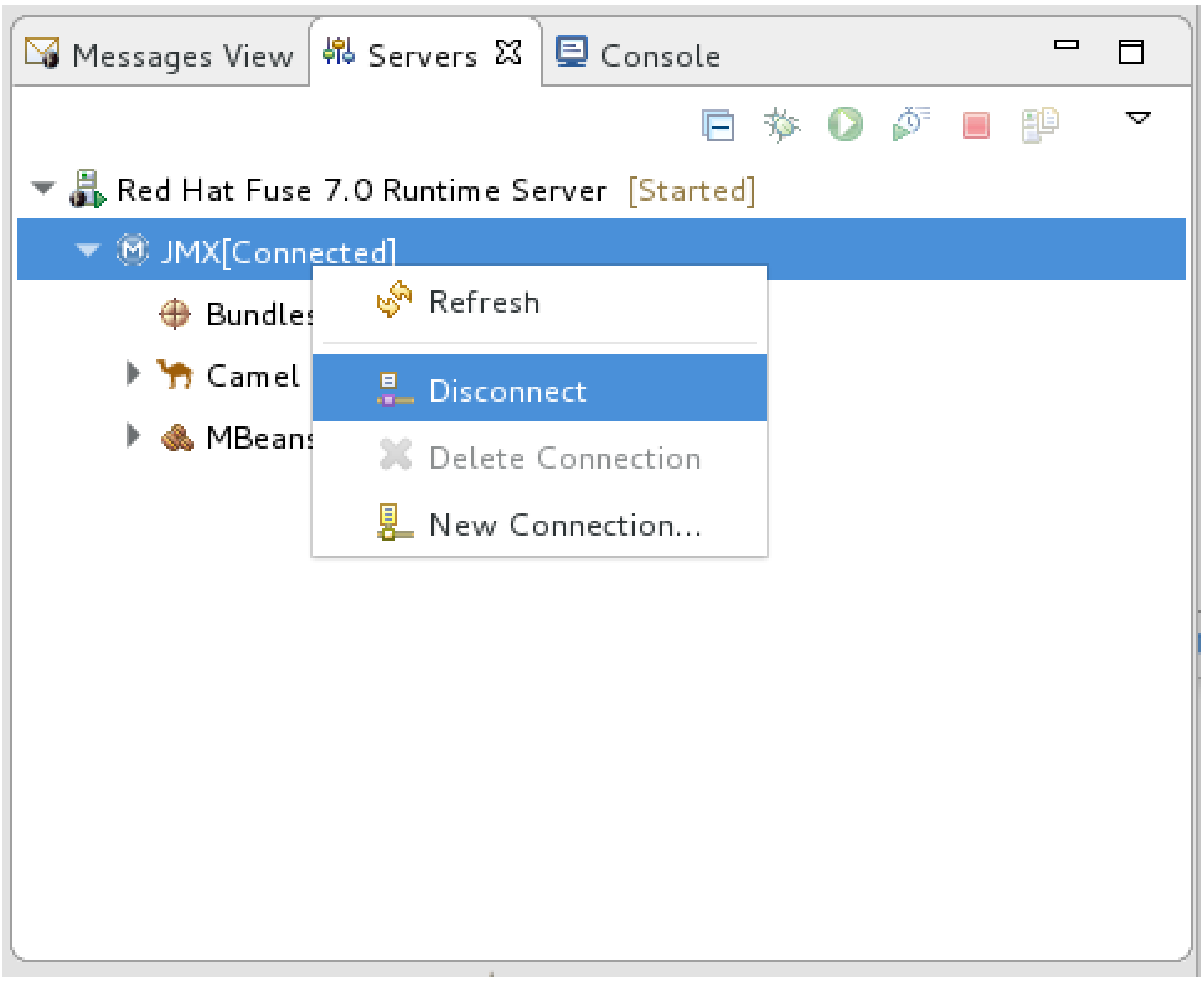
NOTE

The **Servers** view and the **JMX Navigator** view are synchronized with regards to server connections. That is, disconnecting from a server in the **Servers** view also disconnects it in the **JMX Navigator** view, and vice versa.

Disconnecting from a server in the Servers view

1. In the **Servers** view, expand the server runtime to expose its **JMX[Connected]** node.

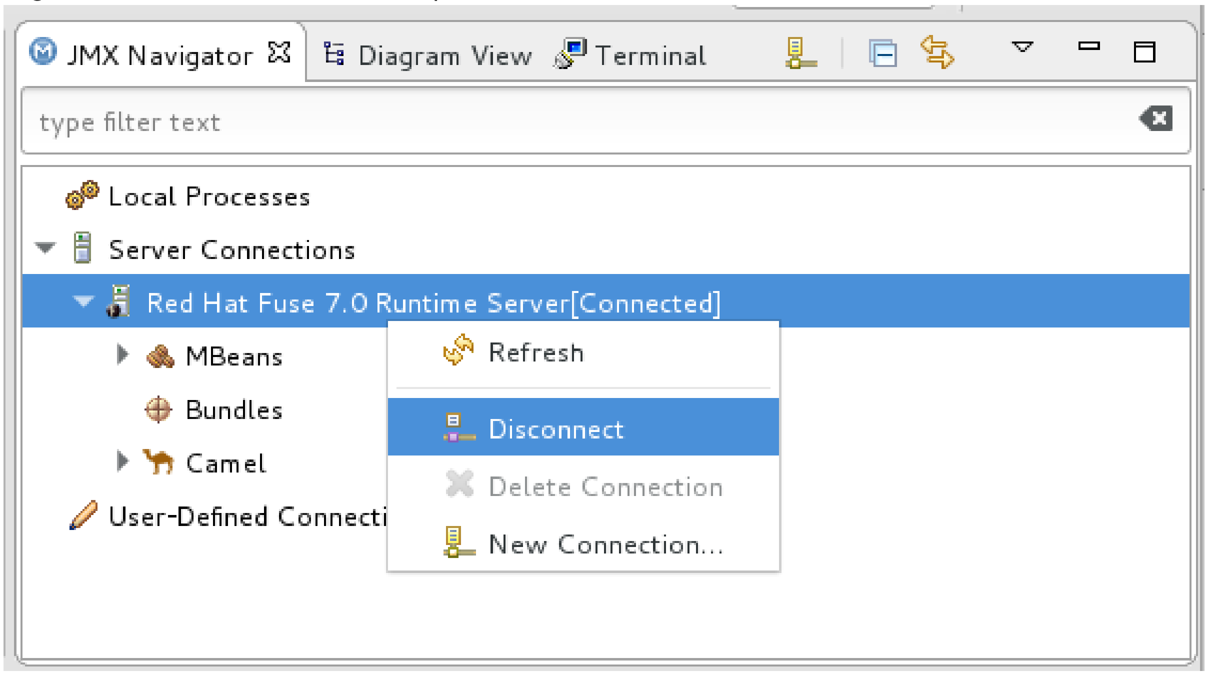
2. Right-click the **JMX[Connected]** node to open the context menu, and then select **Disconnect**.



Disconnecting from a server in the JMX Navigator view

1. In the **JMX Navigator** view, under **Server Connections**, select the server from which you want to disconnect.

2. Right-click the selected server to open the context menu, and then select **Disconnect**.



28.5. STOPPING A SERVER

Overview

You can shut down a server in the **Servers** view or in the server’s remote console in the **Terminal** view.

Using the Servers view

To stop a server:

1. In the **Servers** view, select the server you want to stop.
2. Click 🟥 .

Using the remote console

To stop a server:

1. Open the **Terminal** view that is hosting the server’s remote console.
2. Press: **CTRL+D**

28.6. DELETING A SERVER

Overview

When you are finished with a configured server, or if you misconfigure a server, you can delete it and its configuration.

First, delete the server from the **Servers** view or from the **JMX Navigator** view. Next, delete the server’s configuration.

Deleting a server

1. In the **Servers** view, right-click the server you want to delete to open the context menu.
2. Select **Delete**.
3. Click **OK**.

Deleting the server’s configuration

1. On the menu bar, select **Developer Studio** → **Preferences** → **Server**.

   NOTE

   On Linux and Windows machines, select **Window** → **Preferences**.

2. Expand the **Server** folder, and then select **Runtime Environments** to open the **Server Runtime Environments** page.
3. From the list, select the runtime environment of the server that you previously deleted from the **Servers** view, and then click **Remove**.
4. Click **OK**.

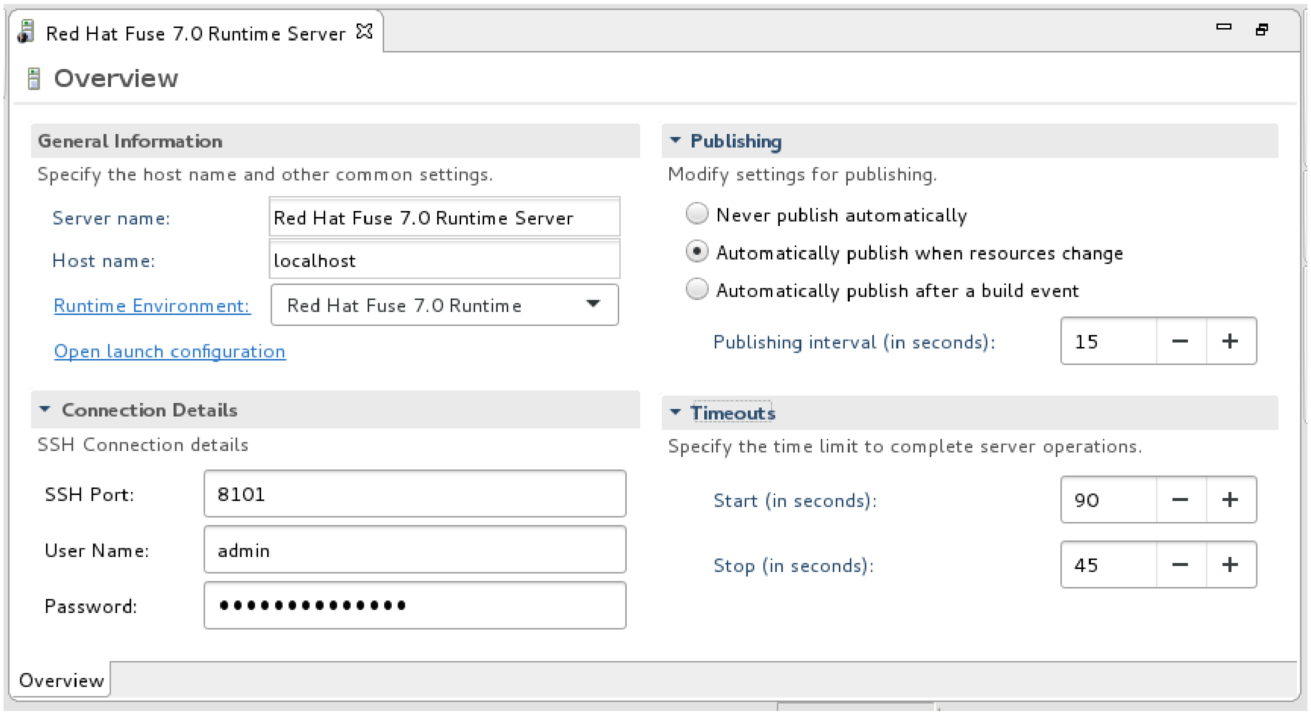
CHAPTER 29. PUBLISHING FUSE INTEGRATION PROJECTS TO A SERVER

You deploy Fuse Integration projects into a server runtime using the Eclipse publishing mechanism. To do so, you must have defined and added the server to the **Servers** view in the **Fuse Integration** perspective. For a step-by-step demonstration, see .

OVERVIEW

You can set up supported servers to publish assigned Fuse projects automatically or to publish them only when you manually invoke the publish command.

Each server runtime added to the **Servers** view has its own **Overview** page that contains its configuration, connection, and publishing details:


Red Hat Fuse 7.0 Runtime Server

Overview

General Information

Specify the host name and other common settings.

Server name: Red Hat Fuse 7.0 Runtime Server

Host name: localhost

Runtime Environment: Red Hat Fuse 7.0 Runtime

Open launch configuration

Connection Details

SSH Connection details

SSH Port: 8101

User Name: admin

Password: ••••••••••••••

Publishing

Modify settings for publishing.

Never publish automatically [✓ Automatically publish when resources change] Automatically publish after a build event

Publishing interval (in seconds): 15

Timeouts

Specify the time limit to complete server operations.

Start (in seconds): 90

Stop (in seconds): 45

You might need to expand **Publishing** to expose the server runtime publishing options and default settings:

- **Never publish automatically** — You must select this option to manually publish projects.

  IMPORTANT

  You must also disable the **If server started, publish changes immediately** option on the server's **Add and Remove** page (for details see, the section called "Publishing Fuse projects manually".

- **Automatically publish when resources change** — [default] Enable this option to automatically publish or republish a Fuse project when you save changes made to it. How quickly projects are published depends on the **Publishing interval** (default is 15 seconds).

- **Automatically publish after a build event** — For Fuse projects, works the same as **Automatically publish when resources change**.

PUBLISHING FUSE PROJECTS AUTOMATICALLY WHEN RESOURCES CHANGE

The default publishing option for server runtimes is **Automatically publish when resources change**.

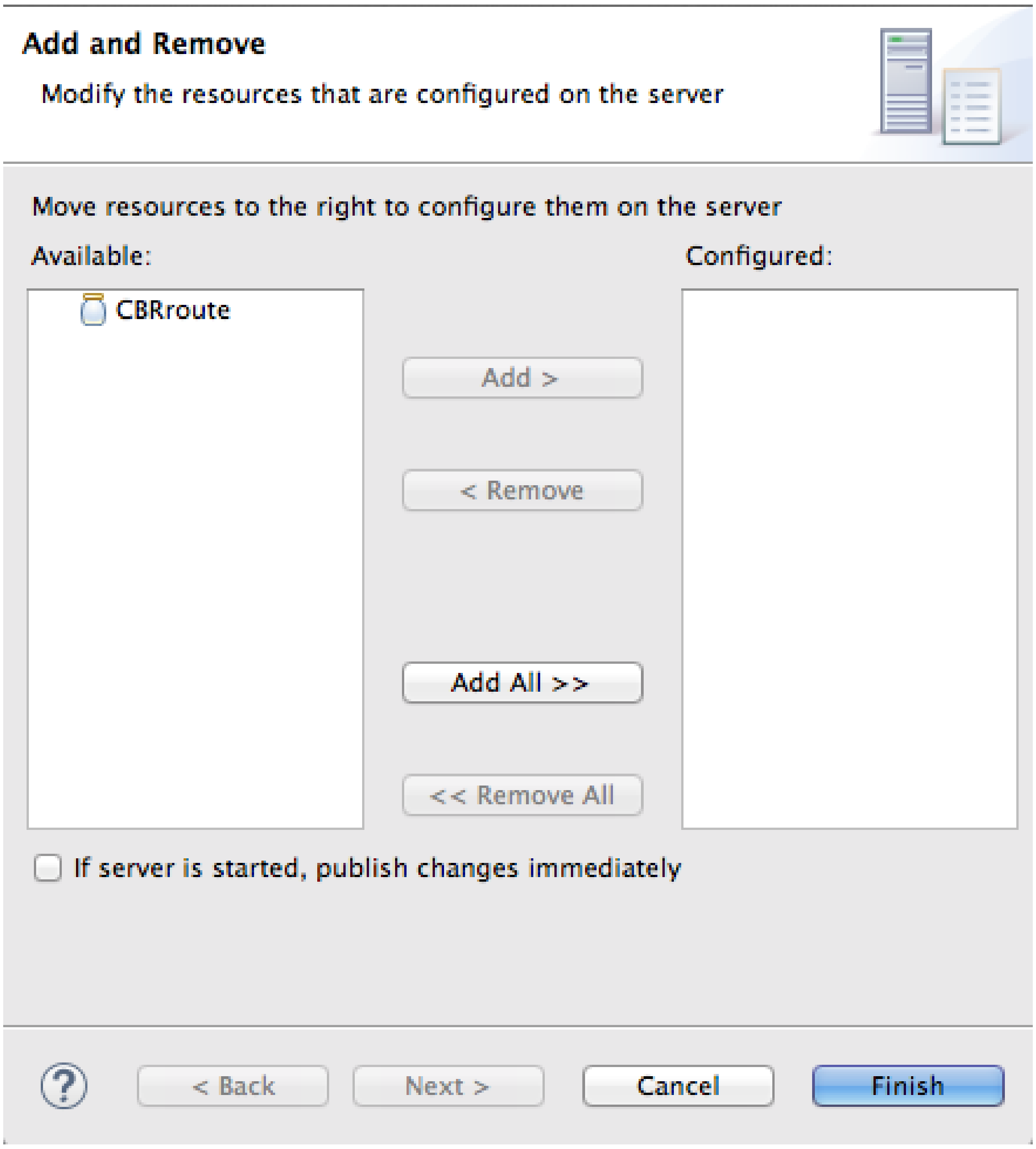
1. If necessary, start the server runtime to which you want to publish a Fuse project. For details, see Section 28.2, “Starting a Server”.

2. In the **Servers** view, double-click the server runtime to open its **Overview** page.

3. Expand **Publishing**, and then select **Automatically publish when resources change**.

4. To increase or decrease the interval between publishing cycles, click the radio button next to **Publishing interval (in seconds)** up or down, as appropriate.

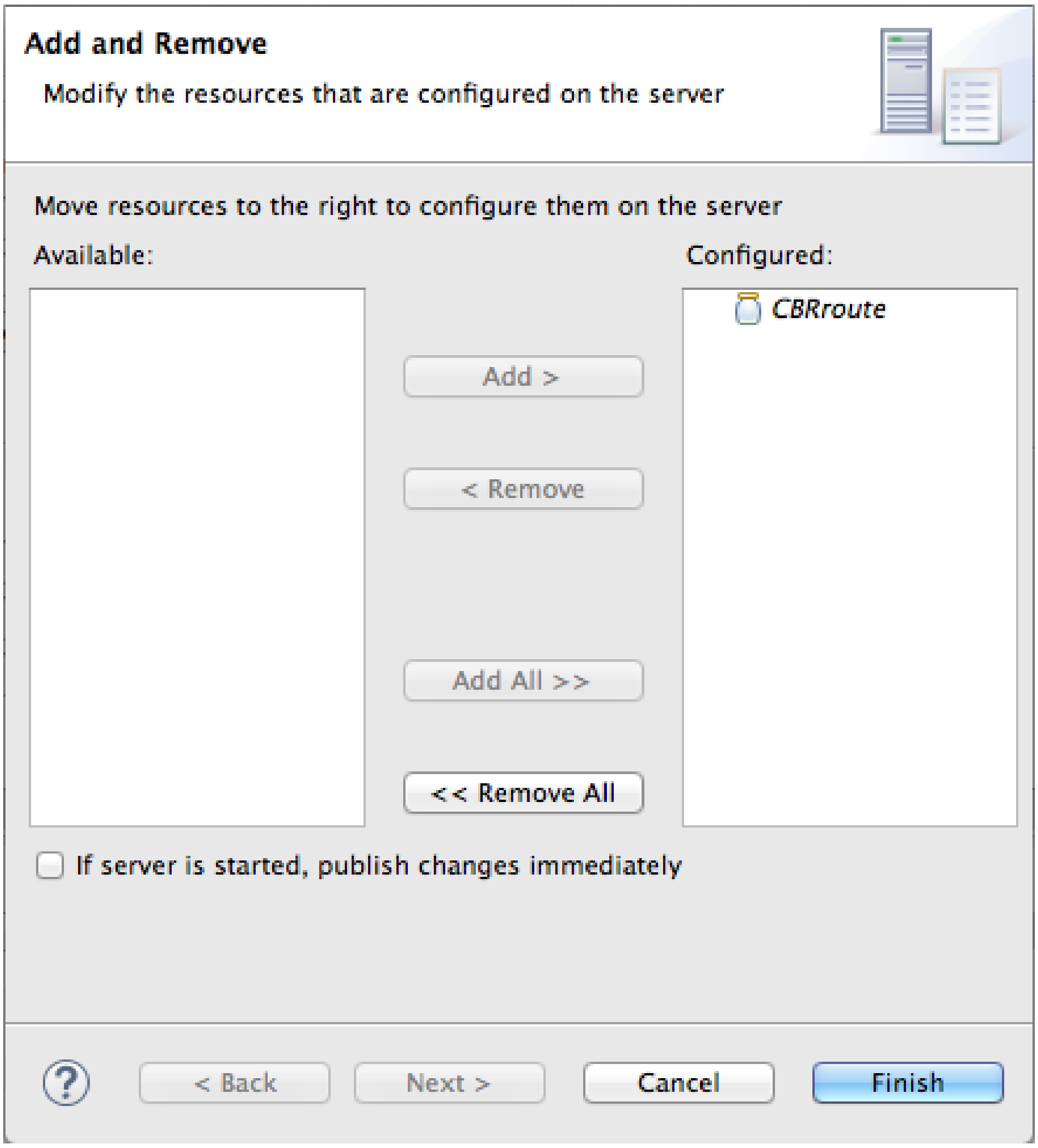
5. In the **Servers** view, right-click the server runtime to open the context menu, and then select **Add and Remove**.


Add and Remove
Modify the resources that are configured on the server
Move resources to the right to configure them on the server
Available:
Configured:
CBRroute
Add >
< Remove
Add All >>
<< Remove All
If server is started, publish changes immediately
< Back
Next >
Cancel
Finish

All resources available for publishing appear in the **Available** column.

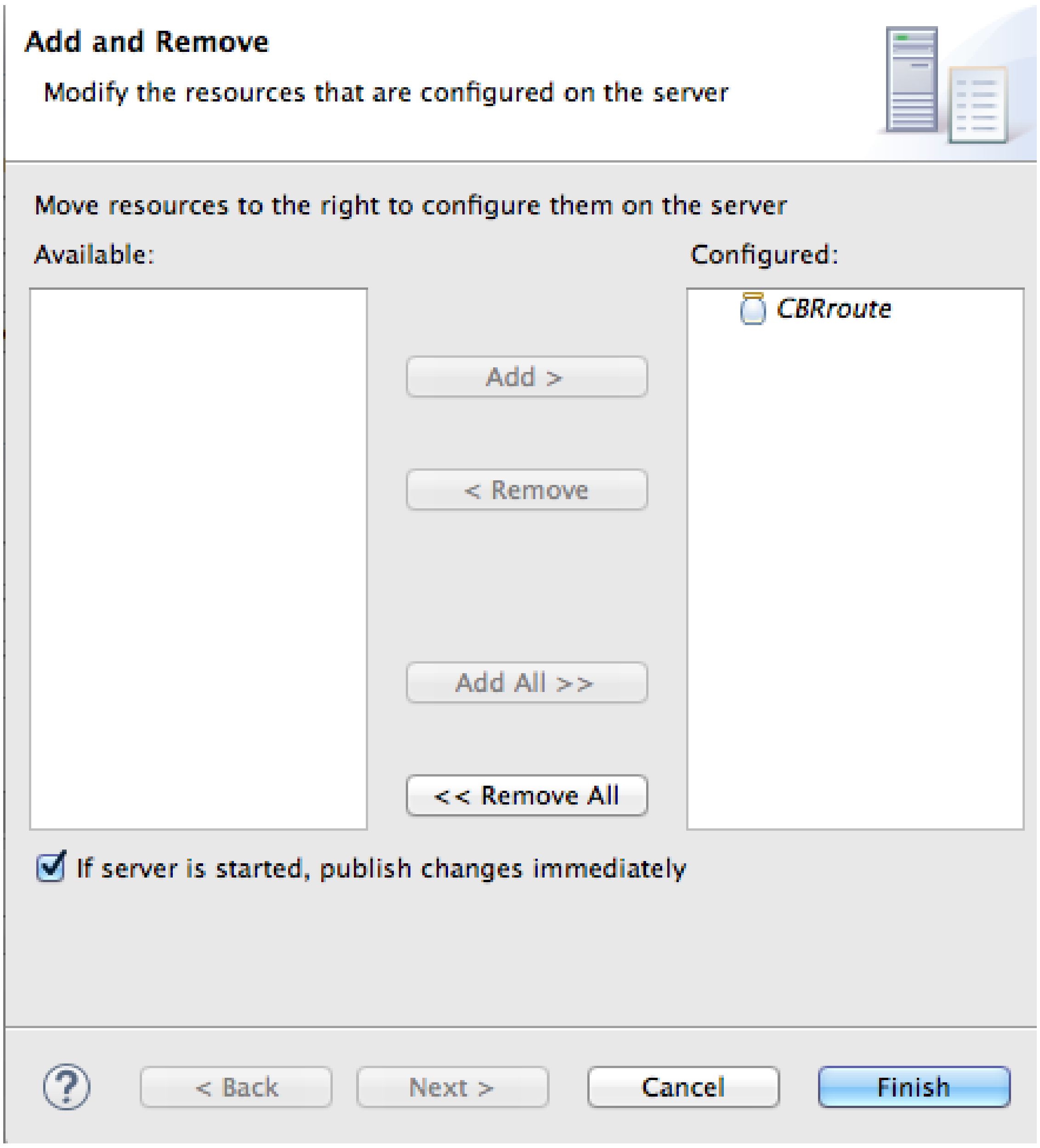
6. To assign a resource (in this case, the **CBRroute** Fuse project) to the server runtime:

- Double-click it, or
- Select it, and click **Add**.
  The selected resource moves to the **Configured** column:


Add and Remove
Modify the resources that are configured on the server
Move resources to the right to configure them on the server
Available:
Configured:
CBRroute
Add >
< Remove
Add All >>
<< Remove All
If server is started, publish changes immediately
< Back
Next >
Cancel
Finish

At this stage, the time at which the assigned resource would actually be published depends on whether the server runtime was running and on the **Publishing interval** setting. However, if the server was stopped, you would have to manually publish the project after you started the server (for details, see the section called “Publishing Fuse projects manually”).

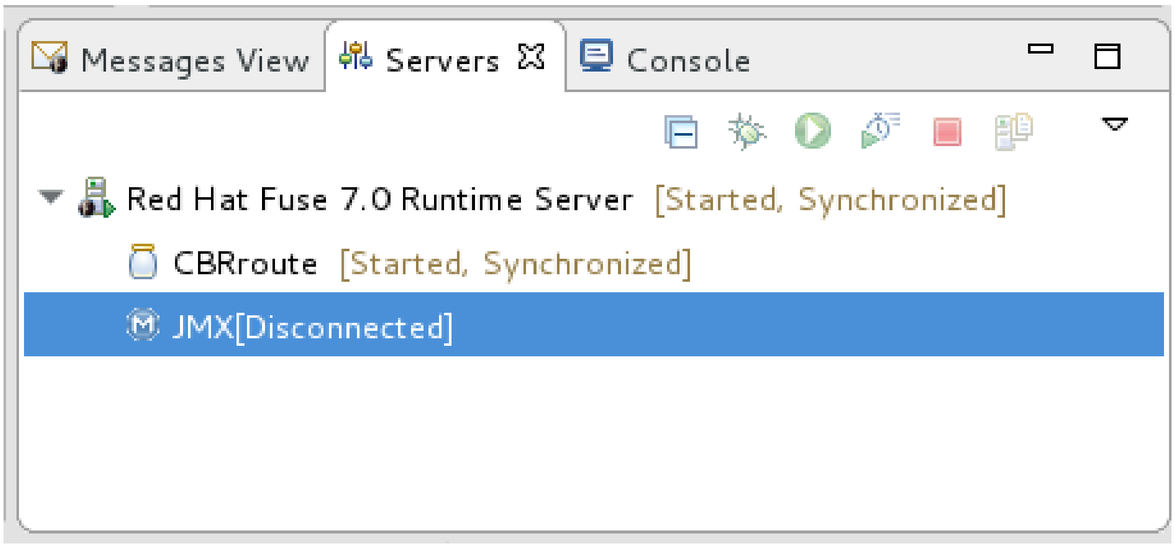
7. Click the **If server started, publish changes immediately** option to enable it:


Add and Remove
Modify the resources that are configured on the server
Move resources to the right to configure them on the server
Available:
Configured:
CBRroute
Add >
< Remove
Add All >>
<< Remove All
If server is started, publish changes immediately
< Back
Next >
Cancel
Finish

This option ensures that the configured project is published immediately once you click **Finish**. The **Automatically publish when resources change** option on the server runtime **Overview** page ensures that the configured project is republished whenever changes made to the local project are saved.

8. Click **Finish**.
   The project appears in the **Servers** view under the server runtime node, and the server runtime status reports **[Started,Publishing...]**.

   When publishing is done, the status of both the server runtime and the project report is **[Started,Synchronized]**:



Red Hat Fuse 7.0 Runtime Server [Started, Synchronized]
CBRroute [Started, Synchronized]
JMX[Disconnected]

NOTE

For a server runtime, **Synchronized** means that all published resources on the server are identical to their local counterparts. For a published resource, **Synchronized** means that it is identical to its local counterpart.

PUBLISHING FUSE PROJECTS MANUALLY

1. If necessary, start the server runtime to which you want to publish a Fuse project. For details, see Section 28.2, “Starting a Server”.

2. In the **Servers** view, double-click the server runtime to open its **Overview** page.

3. Expand **Publishing**, and then select **Never publish automatically**.

4. Click **File** → **Save** to save the publishing option changes.

5. If the Fuse project has already been assigned to the server runtime, make sure this option is disabled: **If server started, publish changes immediately**:

   a. In the **Servers** view, right-click the server runtime to open the context menu.

   b. Click **Add and Remove…** to open the server’s **Add and Remove** page.

   c. If the following option is enabled, disable it: **If server started, publish changes immediately**.

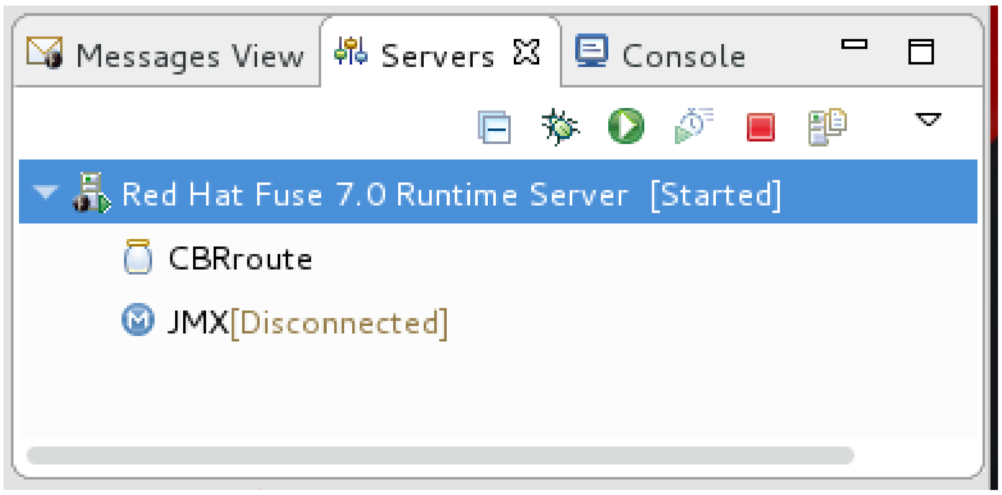
   d. Skip to [finish].

6. If the Fuse project has not been assigned to the server runtime, assign it now:

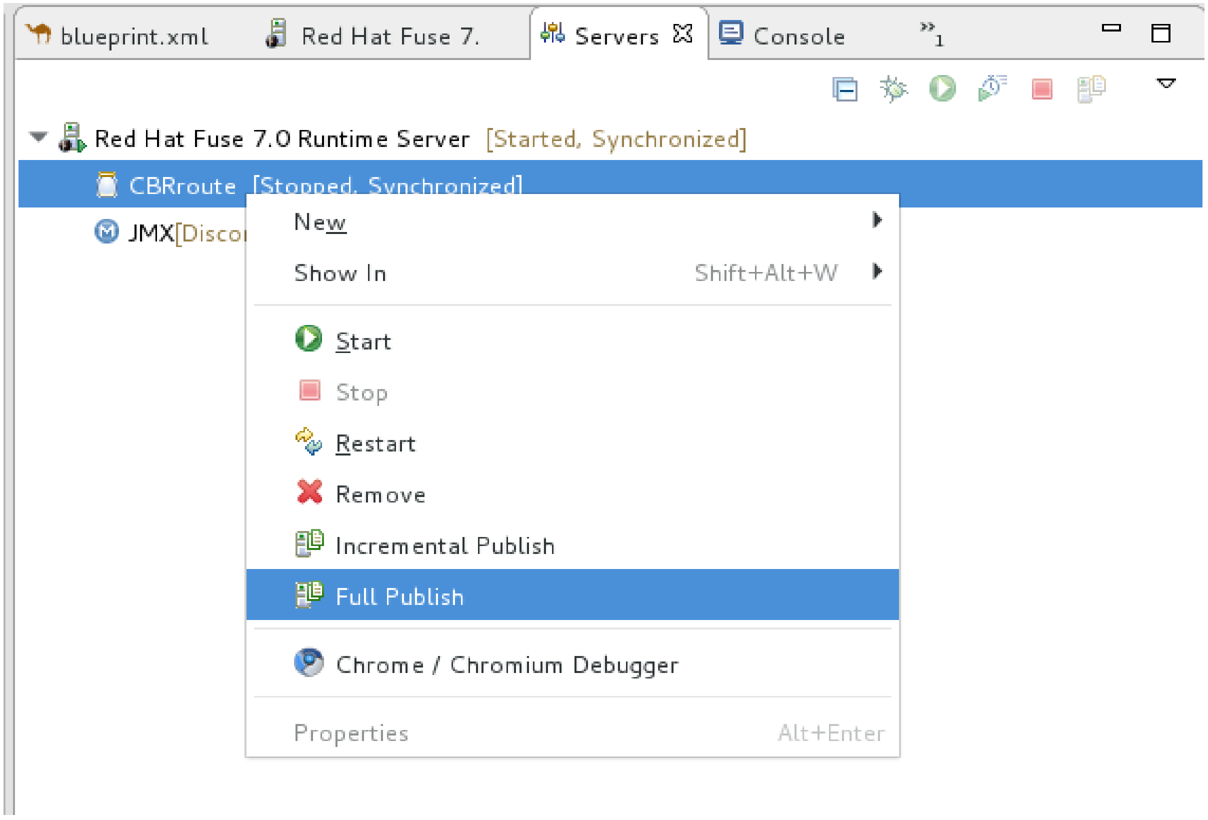
   a. Follow [startAssignResource] through [stopAssignResource] in the section called “Publishing Fuse projects automatically when resources change”.

   b. Do not enable the **If server started, publish changes immediately** option.

7. Click **Finish**.
   The project appears in the **Servers** view under the server runtime node, and the server runtime status reports **[Started]**:



8. In the **Servers** view, right-click the project’s node. In this example, select the **CBRroute** Fuse project to open the context menu:



9. Select **Full Publish**.
   During the publishing operation, the status of both the server runtime and the project report **[Started,Republish]**.

   When publishing is done, the status of both the server runtime and the project report **[Started,Synchronized]**:

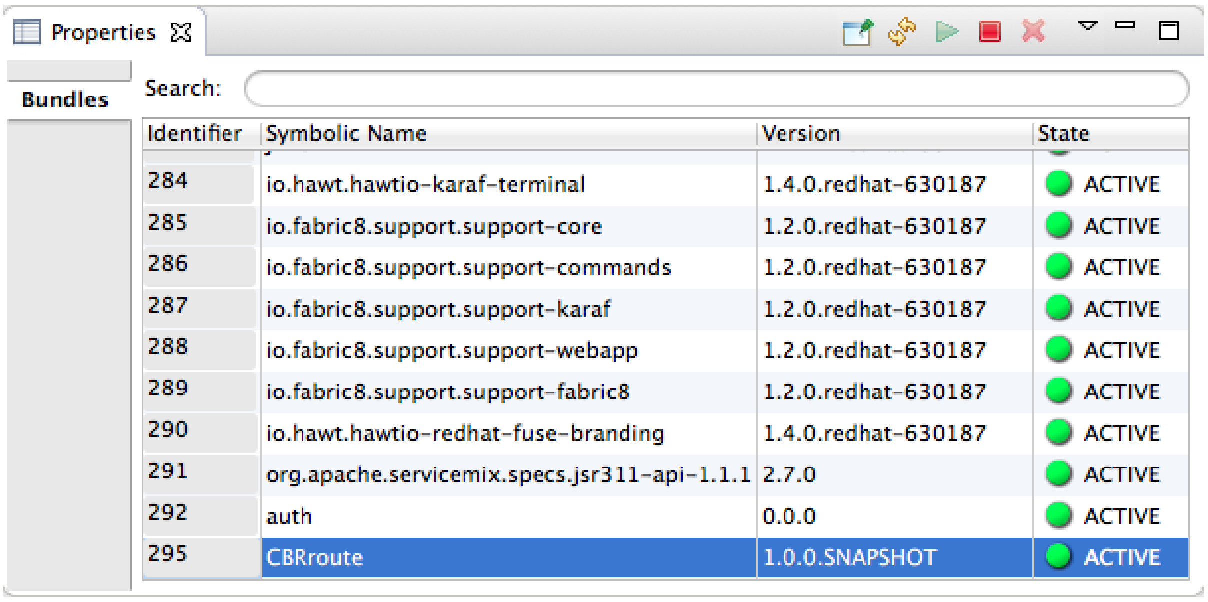
> **NOTE**
>
> The tooling does not support the **Incremental Publish** option. Clicking **Incremental Publish** results in a full publish.

VERIFYING THE PROJECT WAS PUBLISHED TO THE SERVER

After you have published a Fuse project to a server runtime, you can connect to the server and check that the project’s bundle was installed on it.

1. Connect to the server runtime. For details see the section called “Connecting to a running server in the Servers view”.

2. In the **Servers** view, expand the server runtime tree to expose the **Bundles** node and select it. The tooling populates the **Properties** view with a list of bundles that are installed on the server:



Properties

Bundles

Search:

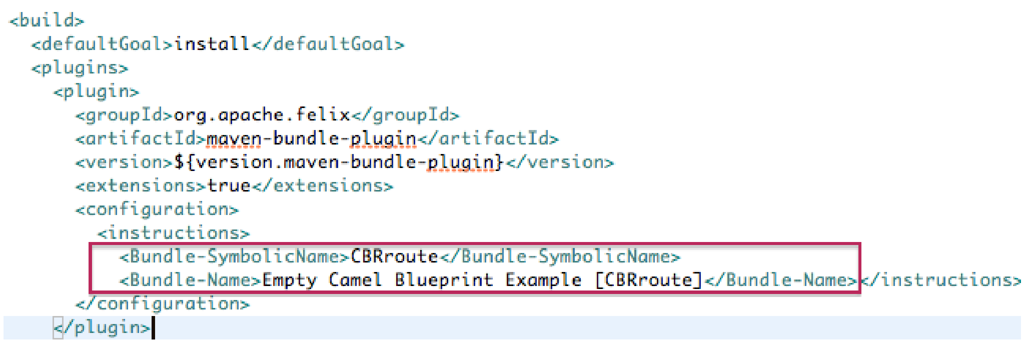
Identifier	Symbolic Name	Version	State
284	io.hawt.hawtio-karaf-terminal	1.4.0.redhat-630187	● ACTIVE
285	io.fabric8.support.support-core	1.2.0.redhat-630187	● ACTIVE
286	io.fabric8.support.support-commands	1.2.0.redhat-630187	● ACTIVE
287	io.fabric8.support.support-karaf	1.2.0.redhat-630187	● ACTIVE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

3. To find your project’s bundle, either scroll down to the bottom of the list, or start typing the bundle’s **Symbolic Name** in the **Properties** view’s **Search** box. The bundle’s **Symbolic Name** is the name you gave your project when you created it.

NOTE

Alternatively, you can issue the **`osgi:list`** command in the **Terminal** view to see a generated list of bundles installed on the Fuse server runtime. The tooling uses a different naming scheme for OSGi bundles displayed by the **`osgi:list`** command.

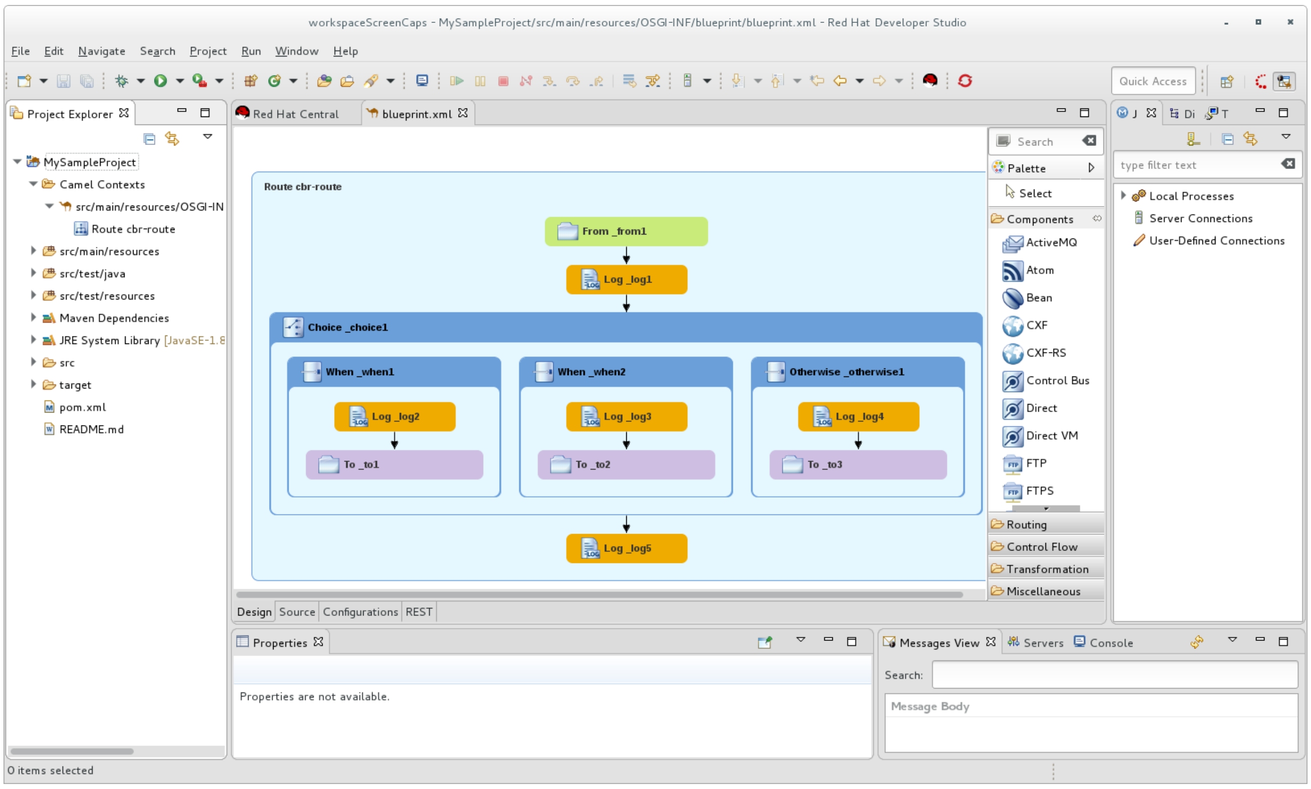
In the **`<build>`** section of project’s **`pom.xml`** file, you can find the bundle’s symbolic name and its bundle name (OSGi) listed in the **`maven-bundle-plugin`** entry; for example:



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

APPENDIX A. FUSE INTEGRATION PERSPECTIVE

Use the **Fuse Integration** perspective to design, monitor, test, and publish your integration application.

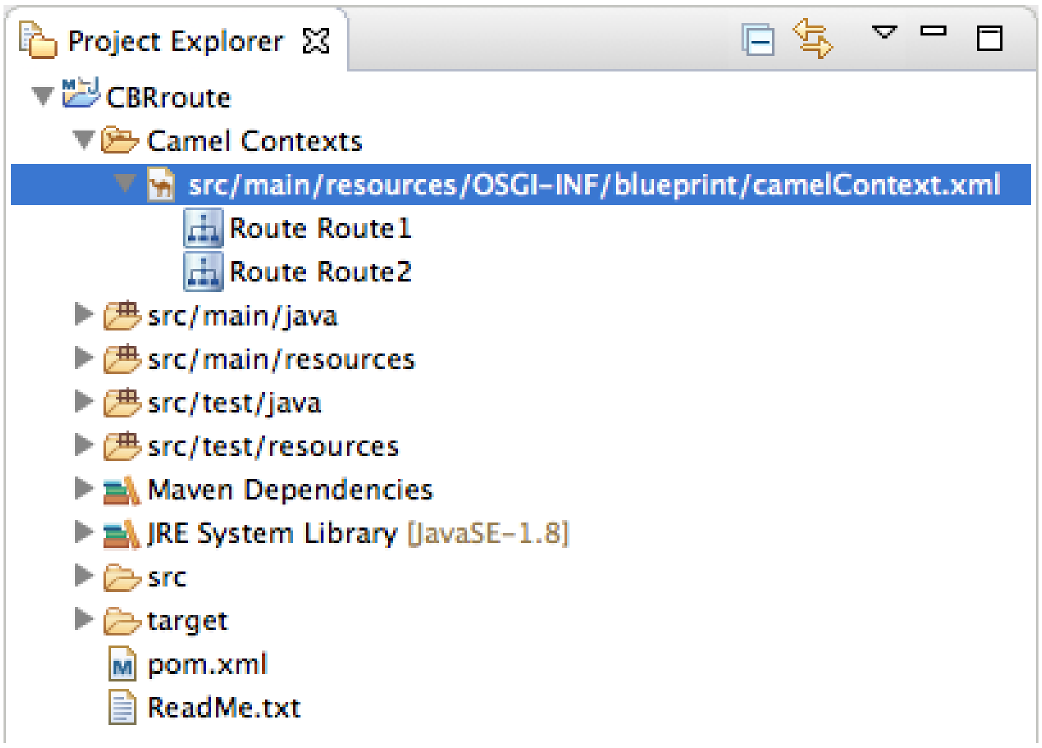


You can open the **Fuse Integration** perspective in the following ways:

- When you create a new Fuse Integration project (see Chapter 1, *Creating a New Fuse Integration Project*), the tooling switches to the **Fuse Integration** perspective.
- Click on the right side of the Developer Studio tool bar. If the icon is not available on the tool bar, click and then select **Fuse Integration** from the list of available perspectives.
- Select **Window** → **Perspective** → **Open Perspective** → **Fuse Integration**.

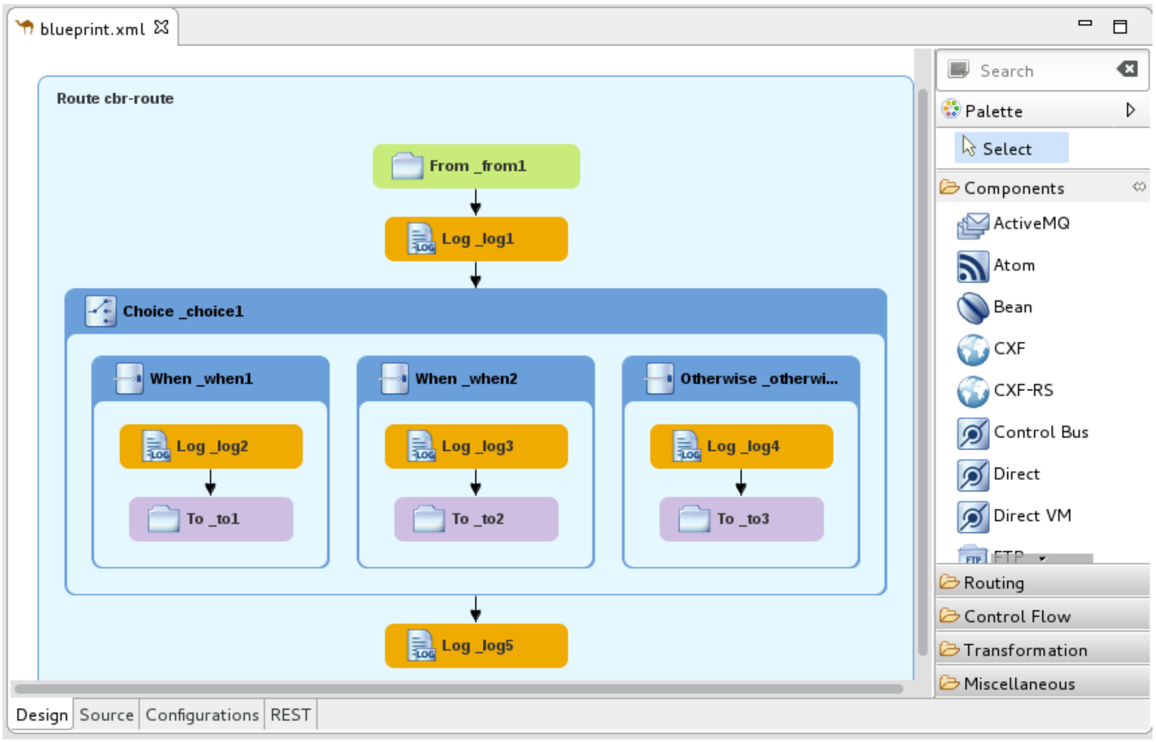
The **Fuse Integration** perspective consists of nine main areas:

- **Project Explorer** view
  Displays all projects known to the tooling. You can view all artifacts that make up each project. The **Project Explorer** view also displays all routing context `.xml`. files for a project under its `Camel Contexts` node. This enables you to find and open a routing context file included in a project. Under each routing context `.xml` file, the **Project Explorer** view displays all routes defined within the context. For multiroute contexts, this lets you focus on a specific route on the canvas.



- The route editor
  Provides the main design-time tooling and consists of three tabs:

  - **Design** — Displays a large grid area on which routes are constructed and a palette from which Enterprise Integration Patterns (EIPs) and Camel components are selected and then connected on the canvas to form routes.



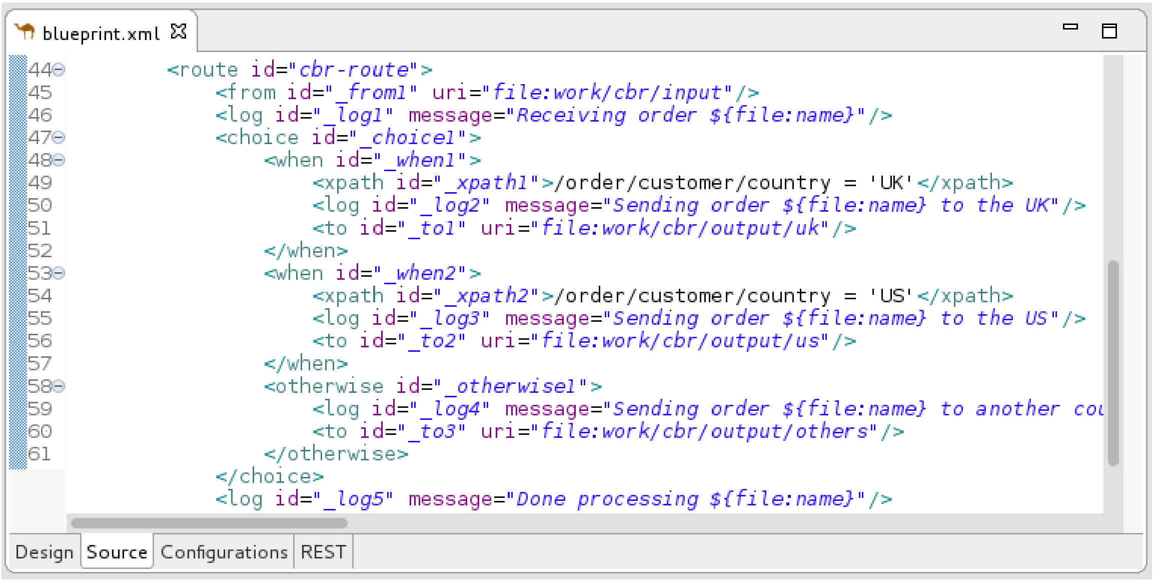
The canvas is the route editor's workbench and where you do most of your work. It displays a graphical representation of one or more routes, which are made up of connected EIPs and Camel components (called nodes once they are placed on the canvas).

Selecting a node on the canvas populates the **Properties** view with the properties that apply to the selected node, so you can edit them.

The **Palette** contains all of the patterns and Camel components needed to construct a route and groups them according to function — **Components**, **Routing**, **Control Flow**, **Transformation**, and **Miscellaneous**.

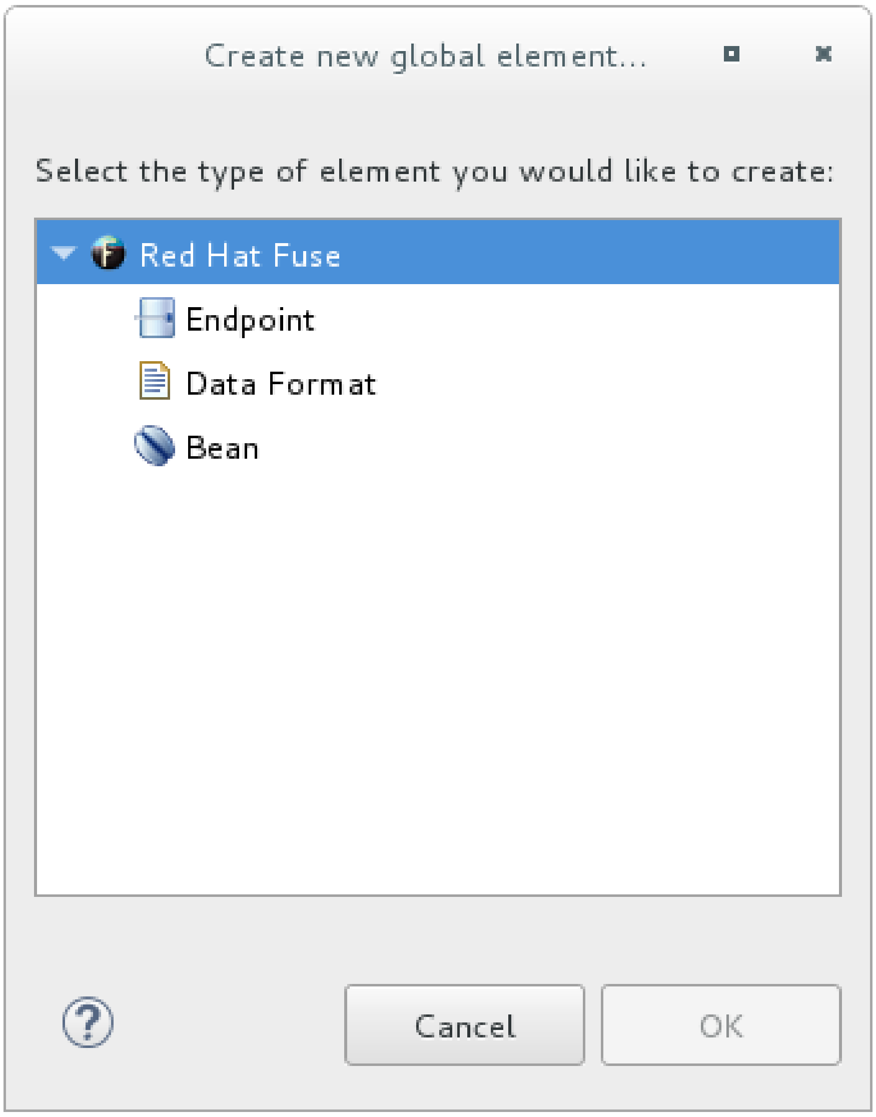
- **Source**
  Displays the contents of the .xml file for the routes constructed on the route editor’s canvas.

  You can edit the routing context in the **Source** tab as well as in the **Design** tab. The **Source** tab is useful for editing and adding any configuration, comments, or beans to the routing context file. The content assist feature helps you when working with configuration files. In the **Source** tab, press **`Ctrl`+Space** to see a list of possible values that can be inserted into your project.



```
        <route id="cbr-route">
            <from id="_from1" uri="file:work/cbr/input"/>
            <log id="_log1" message="Receiving order ${file:name}"/>
            <choice id="_choice1">
                <when id="_when1">
                    <xpath id="_xpath1">/order/customer/country = 'UK'</xpath>
                    <log id="_log2" message="Sending order ${file:name} to the UK"/>
                    <to id="_to1" uri="file:work/cbr/output/uk"/>
                </when>
                <when id="_when2">
                    <xpath id="_xpath2">/order/customer/country = 'US'</xpath>
                    <log id="_log3" message="Sending order ${file:name} to the US"/>
                    <to id="_to2" uri="file:work/cbr/output/us"/>
                </when>
                <otherwise id="_otherwise1">
                    <log id="_log4" message="Sending order ${file:name} to another cou
                    <to id="_to3" uri="file:work/cbr/output/others"/>
                </otherwise>
            </choice>
            <log id="_log5" message="Done processing ${file:name}"/>
```

- **Configurations** — Provides an easy way to add shared configuration (global endpoints, data formats, beans) to a multi-route, routing context. For details see Section 2.6, “Adding global endpoints, data formats, or beans”.


Create new global element...
Select the type of element you would like to create:
Red Hat Fuse
Endpoint
Data Format
Bean
Cancel
OK

- **Properties** view
  Displays the properties of the node selected on the canvas.

- **JMX Navigator** view
  Lists the JMX servers and the infrastructure they monitor. It enables you to browse JMX servers and the pocesses they are monitoring. It also identifies instances of Red Hat processes.

  The **JMX Navigator** view drives all monitoring and testing activities in the **Fuse Integration** perspective. It determines which routes are displayed in the **Diagram View**, the **Properties** view, and the **Messages View**. It is also provides menu commands for activating route tracing, adding and deleting JMS destinations, and starting and suspending routes. It is also the target for dragging and dropping messages onto a route.

  By default, the **JMX Navigator** view shows all Java processes that are running on your local machine. You can add JMX servers as needed to view infrastructure on other machines.

- **Diagram View**
  Displays a graphical tree representing the node selected in the **JMX Navigator** view. When you select a process, server, endpoint, or other node, the **Diagram View** shows the selected node as the root with branches down to its children and grandchildren.

  When you select a broker, the **Diagram View** displays up to three children: connections, topics, and queues. It also shows configured connections and destinations as grandchildren.

When you select a route, the **Diagram View** displays all nodes in the route and shows the different paths that messages can take through the route. It also displays timing metrics for each processing step in the route when route tracing is enabled.

- **Messages View**
  Lists the messages that have passed through the selected JMS destination or through Apache Camel endpoints when route tracing is enabled.

  When a JMS destination is selected in the **JMX Navigator** view, the view lists all messages that are at the destination.

  When route tracing is enabled, the **Messages View** lists all messages that passed through the nodes in the route since tracing started. You can configure the **Messages View** to display only the data in which you are interested and in your preferred sequence.

  When a message trace in the **Messages View** is selected, its details (message body and all message headers) appear in the **Properties** view. In the **Diagram View**, the step in the route associated with the selected message trace is highlighted.

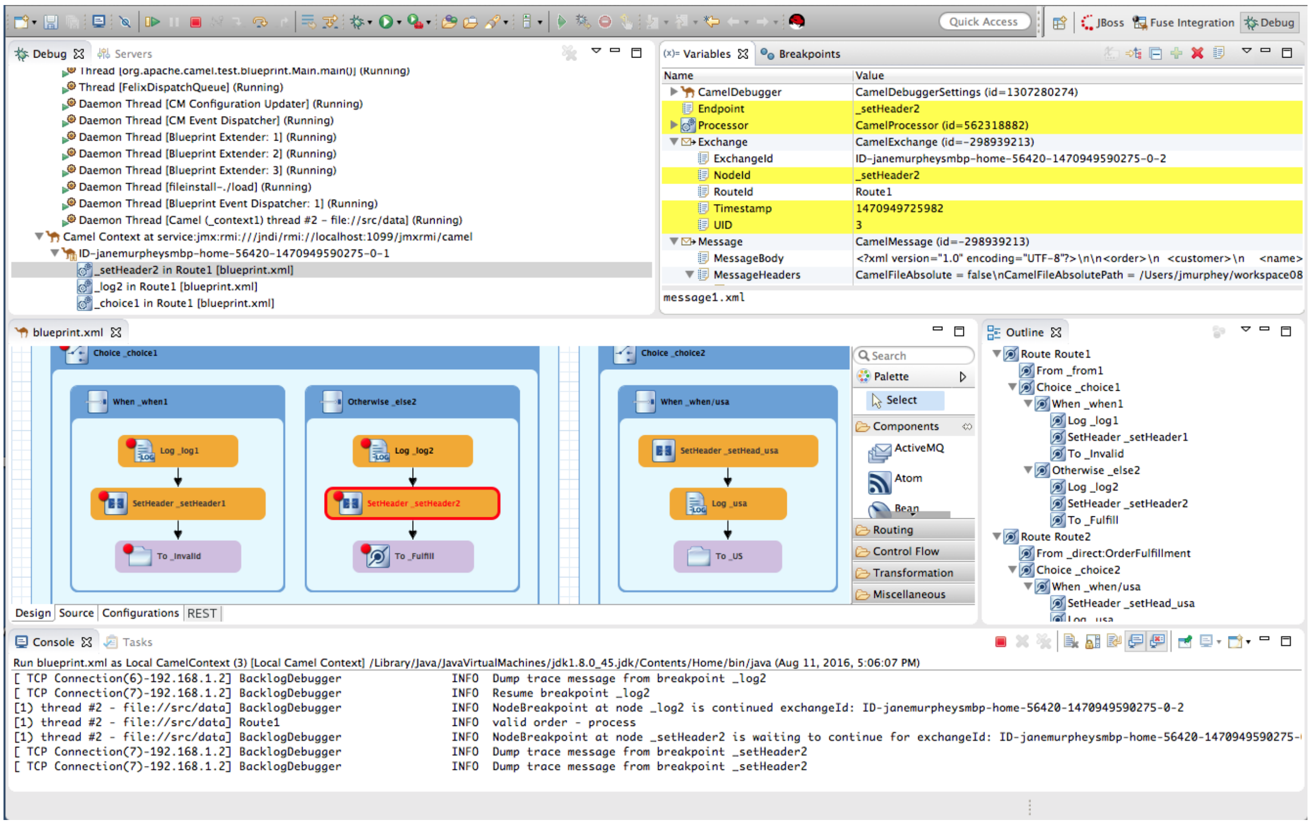
- **Servers** view
  Displays a list of servers managed by the tooling. It displays their runtime status and provides controls for adding, starting and stopping them and for publishing projects to them.

- **Terminal** view
  Displays the command console of the connected container. You can control the container by entering commands in the **Terminal** view.

- **Console** view
  Displays the console output for recently executed actions.

APPENDIX B. DEBUG PERSPECTIVE

Use the **Debug** perspective to monitor and debug a running Camel context.



```
Thread [org.apache.camel.test.blueprint.Main.main()] (Running)
Thread [FelixDispatchQueue] (Running)
Daemon Thread [CM Configuration Updater] (Running)
Daemon Thread [CM Event Dispatcher] (Running)
Daemon Thread [Blueprint Extender: 1] (Running)
Daemon Thread [Blueprint Extender: 2] (Running)
Daemon Thread [Blueprint Extender: 3] (Running)
Daemon Thread [fileinstall-./load] (Running)
Daemon Thread [Blueprint Event Dispatcher: 1] (Running)
Daemon Thread [Camel (_context1) thread #2 - file://src/data] (Running)
Camel Context at service:jmx:rmi:///jndi/rmi://localhost:1099/jmxrmi/camel
  ID-janemurpheysmbp-home-56420-1470949590275-0-1
    _setHeader2 in Route1 [blueprint.xml]
    _log2 in Route1 [blueprint.xml]
    _choice1 in Route1 [blueprint.xml]
```

Name	Value
CamelDebugger	CamelDebuggerSettings (id=1307280274)
Endpoint	_setHeader2
Processor	CamelProcessor (id=562318882)
Exchange	CamelExchange (id=-298939213)
ExchangeId	ID-janemurpheysmbp-home-56420-1470949590275-0-2
NodeId	_setHeader2
RouteId	Route1
Timestamp	1470949725982
UID	3
Message	CamelMessage (id=-298939213)
MessageBody	<?xml version="1.0" encoding="UTF-8"?>\n\n<order>\n <customer>\n <name>
MessageHeaders	CamelFileAbsolute = false\nCamelFileAbsolutePath = /Users/jmurphey/workspace08

```
message1.xml
```

```
Run blueprint.xml as Local CamelContext (3) [Local Camel Context] /Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/Contents/Home/bin/java (Aug 11, 2016, 5:06:07 PM)
[ TCP Connection(6)-192.168.1.2] BacklogDebugger              INFO  Dump trace message from breakpoint _log2
[ TCP Connection(7)-192.168.1.2] BacklogDebugger              INFO  Resume breakpoint _log2
[1) thread #2 - file://src/data] BacklogDebugger              INFO  NodeBreakpoint at node _log2 is continued exchangeId: ID-janemurpheysmbp-home-56420-1470949590275-0-2
[1) thread #2 - file://src/data] Route1                       INFO  valid order - process
[1) thread #2 - file://src/data] BacklogDebugger              INFO  NodeBreakpoint at node _setHeader2 is waiting to continue for exchangeId: ID-janemurpheysmbp-home-56420-1470949590275-
[ TCP Connection(7)-192.168.1.2] BacklogDebugger              INFO  Dump trace message from breakpoint _setHeader2
[ TCP Connection(7)-192.168.1.2] BacklogDebugger              INFO  Dump trace message from breakpoint _setHeader2
```

- **Debug** view
  For the running Camel context, the **Debug** view displays the debug stack.

  You can switch between breakpoints within the same message flow, listed under the **`Camel Context at service:jmx:rmi://jndi/rmi://localhost:1099/jmxrmi/camel`** entry, to review and compare variable values in the **Variables** view.

  Messages flows are identified by their unique breadcrumb ID, and the breadcrumb ID of each subsequent message flow is incremented by 2. For example, if the breadcrumb ID for the first message flow is **`ID-janemurpheysmbp-home-54620-1470949590275-0-1`**, the breadcrumbID for the second message flow would be **`ID-janemurpheysmbp-home-54620-1470949590275-0-3`**.

- **Variables** view
  For each node in the routing context that has a breakpoint set, the **Variables** view displays the value of the available variables when the breakpoint is hit. Each variable who's value changed since the preceding breakpoint is highlighted in yellow.

  You can change the value of editable variables to check whether such changes produce the expected results and to test the robustness of your routing context.

  You can also add variables to the watch list, so you can quickly and easily see whether their values change as expected at the expected point in the message flow.

- **Breakpoints** view
  Displays a list of the breakpoints set in the routing context, and shows whether they are enabled or disabled. You can enable and disable individual breakpoints by checking (enabling) or unchecking (disabling) them. This enables you to temporarily focus on nodes in your routing

context that are behaving problematically.

The button skips over disabled breakpoints to jump to the next active breakpoint in the routing context. In contrast, the button jumps to the next node of execution in the routing context, regardless of breakpoints.

- **camel Context.xml** view
  Displays the running routing context file in graphical mode. For nodes set with breakpoints, it shows the type of breakpoint set and whether the breakpoint is enabled or disabled. When a breakpoint is hit, its corresponding node on the canvas is outlined in red.

  To check a node’s configuration, open the **Properties** view and then select the node on the canvas in **`camel Context.xml`**.

- **Console** view
  Displays the log output generated by the Camel debugger as it executes the routing context.

- **Properties** view
  Displays the properties set for the node selected on the canvas in **`CamelContext.xml`**.