Red Hat JBoss Fuse 6.2.1

Tooling User Guide

Developing and Debugging Applications

Last Updated: 2017-09-21

Red Hat JBoss Fuse 6.2.1 Tooling User Guide

Developing and Debugging Applications

JBoss A-MQ Docs Team
Content Services
fuse-docs-support@redhat.com

Legal Notice

Abstract

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

Table of Contents

ABOUT RED HAT JBOSS FUSE TOOLING

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

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

- Red Hat JBoss Fuse
- Red Hat JBoss A-MQ
- Apache Camel
- Apache CXF
- Apache Karaf

NOTE

You can also develop Fuse projects using Maven (for details, see Getting Started with Developing).

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

1. Create a Maven project for your application.

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

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

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

3. Edit the integration logic.

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

4. Debug your local Camel context.

   The tooling includes a full-featured Camel debugger for debugging locally running Camel contexts.

5. Test the application.

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

   - creating and using JUnit test cases on Apache Camel routes
   - using JMX to analyze running components

   - tracing messages through Apache Camel routes

6. Deploy the application.

   The tooling can deploy applications to a number of containers.

NOTE

See the Red Hat JBoss Fuse Tooling Tutorials for step-by-step instructions on:

- Creating a Route
- Running a Route
- Adding a Content-Based Router
- Adding Another Route to the CBR Routing Context
- Debugging a Routing Context
- Tracing a Message Through a Route
- Testing a Route with JUnit
- Publishing a Fuse Project to Red Hat JBoss Fuse

PART I. INTRODUCING THE FUSE TOOLING USER INTERFACE

Red Hat JBoss Fuse Tooling is an Eclipse-based IDE for creating and testing small- and large-scale integration applications. Its key features are a graphical routing editor, a full-featured Camel debugger for debugging locally running routing contexts, an integrated testing platform that uses JMX to communicate with the containers in a small-scale test environment, and tooling for using Fabric8 to test applications in large-scale, highly-distributed test environments.

CHAPTER 1. JBOSS PERSPECTIVE

OVERVIEW

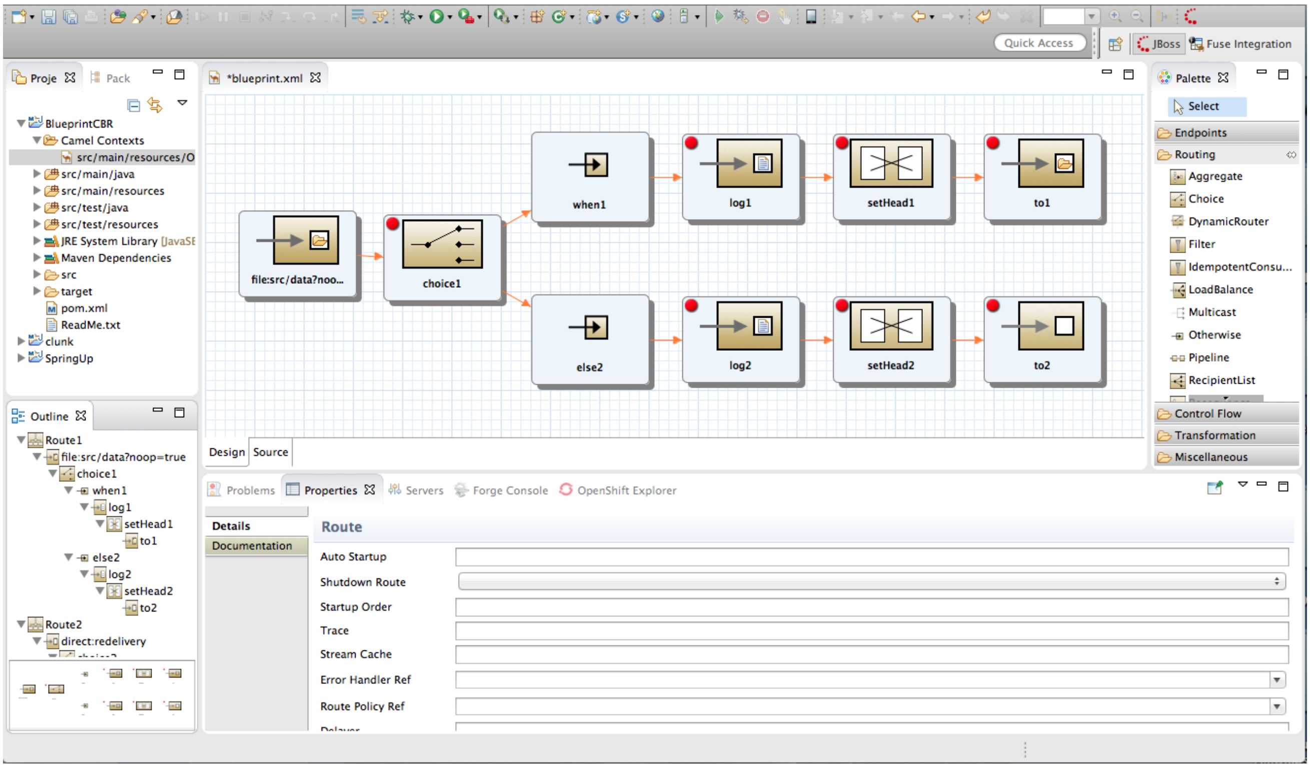
By default, Red Hat JBoss Developer Studio starts up in **JBoss** perspective. To access the design time tooling, you select **File** → **New** → **Fuse Project** on the menu bar.

NOTE

Though you can also do all of your development in **Fuse Integration** perspective, **JBoss** perspective provides more space for the route editor to expand to accommodate complex routes.

Creating a new Fuse project provides access to the route editor, the main design time tooling, shown in Figure 1.1.

Figure 1.1. route editor



It consists of four main areas:

- Canvas—the large grid area on which routes are constructed
- Palette—the pane to the right of the canvas from which Enterprise Integration Patterns (EIPs) are selected
- Project view—the pane on the left side of the canvas, which can display multiple views of the active project. The pane defaults to **Project Explorer**, which displays the project's folders and files in an hierarchical tree format.

NOTE

Red Hat recommends you to close the **`JMX Navigator`** view and then drag **`Outline`** view from the upper right side of the workspace to the lower left side, beneath **`Project Explorer`**, to provide optimal space for the route editor. **`Outline`** view displays, in an outline of EIP icons, the contents of the current **`<camelContext>`** element in the routing context file.

- Properties editor—the editor in which you configure the selected node's properties. It opens in a tab in the pane below the canvas.

CANVAS

The canvas is route editor's workbench and where you do most of your work. It displays a graphical representation of a route, which is made up of one or more connected EIP patterns (called nodes once they are placed on the canvas). Because the canvas displays only one route at a time, it delineates one route from another. You can add additional routes to your project, but each is displayed separately on its own slice of canvas.

Patterns are typically placed on the canvas by dragging them from the palette. Once on the canvas, a node's properties can be edited in the **`Properties`** editor. To do so, you need only select a node to open and populate the **`Properties`** editor with the properties that apply to that node. You can also edit a route's properties by clicking the canvas, which opens and populates the Properties editor with the properties that correspond to a single route in the routing context file.

NOTE

Clicking the **`Source`** tab at the lower, left of the canvas displays the XML code generated for the route, which you can then edit. The edits you make in **`Source`** view appear on the canvas when you switch back to **`Design`** View.

PALETTE

The Palette contains all of the patterns needed to construct a route. The Palette groups the patterns into categories according to function:

- **`Components`**—endpoint patterns that start or end a route
- **`Defined Endpoints`**—endpoint nodes that have been defined in the routing context
- **`Routing`**—patterns that direct the flow of messages based on specified criteria
- **`Control Flow`**—patterns that behave like control functions in a programming language. For example, some define loops, some handle error conditions, some handle transactions, and so on.
- **`Transformation`**—patterns that change the contents of a message as it passes through a route
- **`Miscellaneous`**—patterns that control the environment in which a route executes. For example, the **`Threads`** pattern specifies the number of threads available.

PROJECT EXPLORER

Project Explorer provides access to all of the components that make up a project. It provides the tools for managing, configuring, running, and debugging your project.

OUTLINE VIEW

Outline view provides a graphical outline of your route components. It enables you to switch easily between routes in a multiroute project. Clicking on a route in **Outline** view displays it on the route editor's canvas.

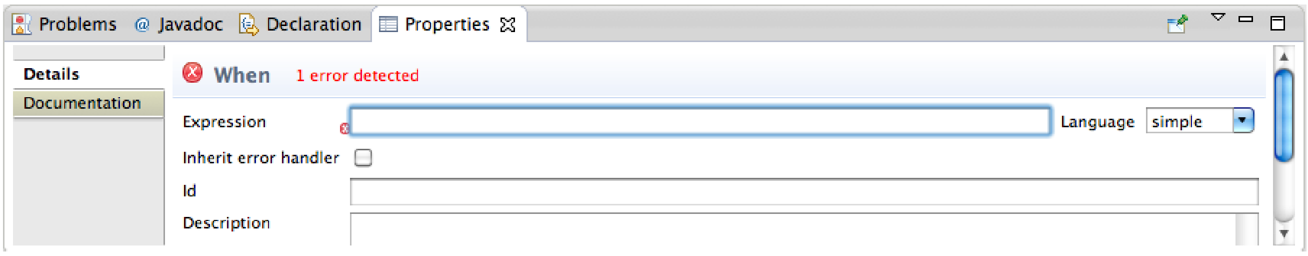
Outline view also provides a viewing portal when the route extends beyond the viewable area on the canvas. Sliding the view portal in **Outline** view over outlier nodes moves the canvas accordingly to bring those nodes into view.

PROPERTIES EDITOR

You use the **Properties** editor to configure the route and each node in it. Each type of node has a set of properties that can or must be set. The **Properties** editor opens populated with fields that apply to the selected node.

The **Properties** editor makes it clear which fields are required. ⊗ marks any required property that is not set, and the **Properties** editor displays an error message stating the number of required properties that still must be set, as shown in Figure 1.2.

Figure 1.2. Properties editor's error reporting



⊗ When 1 error detected

Expression ⊗

Language simple

Inherit error handler □

Id

Description

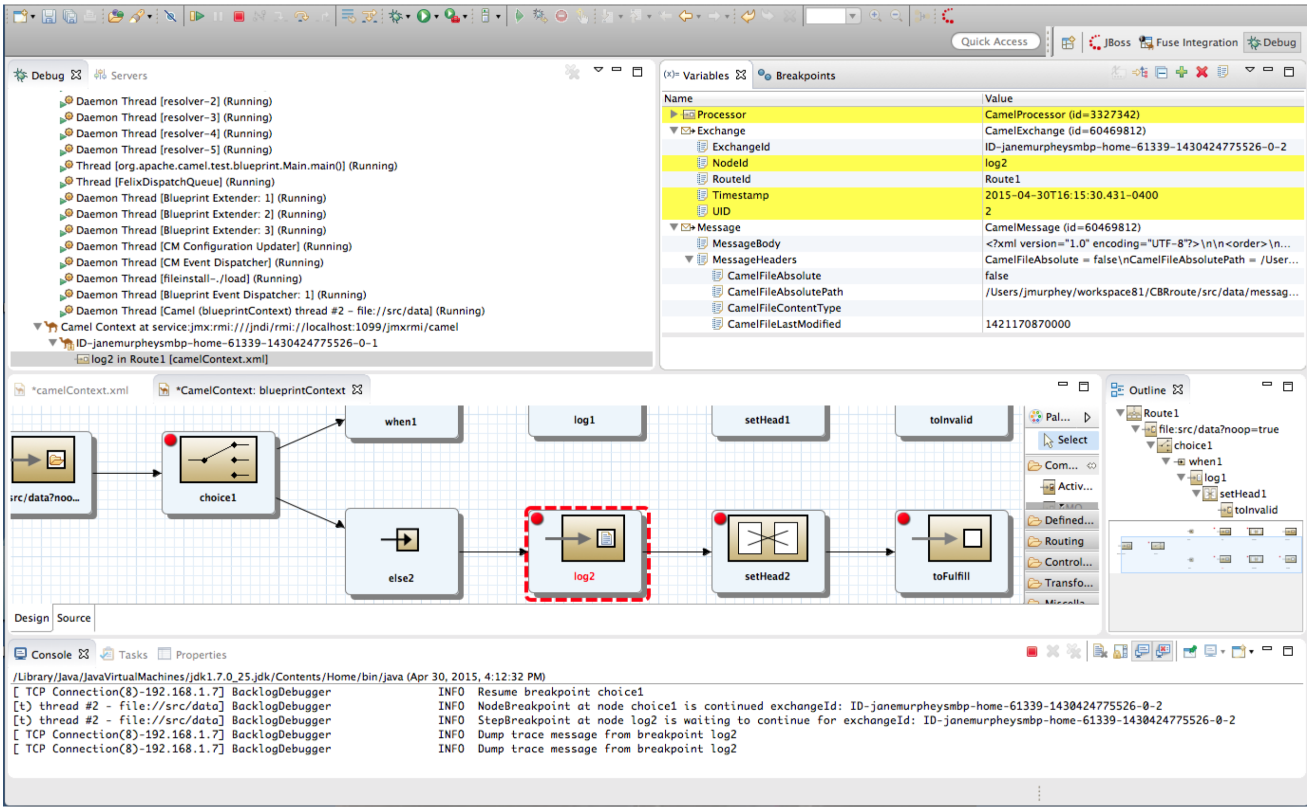
The **Properties** editor's **Documentation** tab displays information about how to configure the selected node.

CHAPTER 2. DEBUG PERSPECTIVE

OVERVIEW

The Debug perspective, shown in Figure 2.1, provides access to all of the Camel debugger's functions. It contains all of the views used to monitor and debug a running routing context.

Figure 2.1. Debug perspective



```
Daemon Thread [resolver-2] (Running)
Daemon Thread [resolver-3] (Running)
Daemon Thread [resolver-4] (Running)
Daemon Thread [resolver-5] (Running)
Thread [org.apache.camel.test.blueprint.Main.main()] (Running)
Thread [FelixDispatchQueue] (Running)
Daemon Thread [Blueprint Extender: 1] (Running)
Daemon Thread [Blueprint Extender: 2] (Running)
Daemon Thread [Blueprint Extender: 3] (Running)
Daemon Thread [CM Configuration Updater] (Running)
Daemon Thread [CM Event Dispatcher] (Running)
Daemon Thread [fileinstall-./load] (Running)
Daemon Thread [Blueprint Event Dispatcher: 1] (Running)
Daemon Thread [Camel (blueprintContext) thread #2 - file://src/data] (Running)
Camel Context at service:jmx:rmi:///jndi/rmi://localhost:1099/jmxrmi/camel
  ID-janemurpheysmbp-home-61339-1430424775526-0-1
    log2 in Route1 [camelContext.xml]
```

Name	Value
Processor	CamelProcessor (id=3327342)
Exchange	CamelExchange (id=60469812)
ExchangeId	ID-janemurpheysmbp-home-61339-1430424775526-0-2
NodeId	log2
RouteId	Route1
Timestamp	2015-04-30T16:15:30.431-0400
UID	2
Message	CamelMessage (id=60469812)
MessageBody	<?xml version="1.0" encoding="UTF-8"?>\n\n<order>\n...
MessageHeaders	CamelFileAbsolute = false\nCamelFileAbsolutePath = /User...
CamelFileAbsolute	false
CamelFileAbsolutePath	/Users/jmurphey/workspace81/CBRroute/src/data/messag...
CamelFileContentType	
CamelFileLastModified	1421170870000

```
Route1
  file:src/data?noop=true
    choice1
      when1
        log1
          setHead1
            toInvalid
```

```
/Library/Java/JavaVirtualMachines/jdk1.7.0_25.jdk/Contents/Home/bin/java (Apr 30, 2015, 4:12:32 PM)
[ TCP Connection(8)-192.168.1.7] BacklogDebugger            INFO  Resume breakpoint choice1
[t) thread #2 - file://src/data] BacklogDebugger            INFO  NodeBreakpoint at node choice1 is continued exchangeId: ID-janemurpheysmbp-home-61339-1430424775526-0-2
[t) thread #2 - file://src/data] BacklogDebugger            INFO  StepBreakpoint at node log2 is waiting to continue for exchangeId: ID-janemurpheysmbp-home-61339-1430424775526-0-2
[ TCP Connection(8)-192.168.1.7] BacklogDebugger            INFO  Dump trace message from breakpoint log2
[ TCP Connection(8)-192.168.1.7] BacklogDebugger            INFO  Dump trace message from breakpoint log2
```

- **Debug** view

  For the running routing context, **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 **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, in Figure 2.1, "Debug perspective", the breadcrumb ID for the first message flow is **ID-janemurpheysmbp-home-61339-11430424775526-0-1**, so the breadcrumbID for the second message flow would be **ID-janemurpheysmbp-home-61339-11430424775526-0-3**.

- **Variables** view

  For each node in the routing context that has a breakpoint set, **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:`** view

  Displays the running **`CamelContext:<`*`CamelId`*`>.xml`** 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 Properties view and then select the node on the canvas in **`Camel Context:`** view.

- **`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:`** view.

RELATED TOPICS

Part III, “Debugging Routing Contexts”

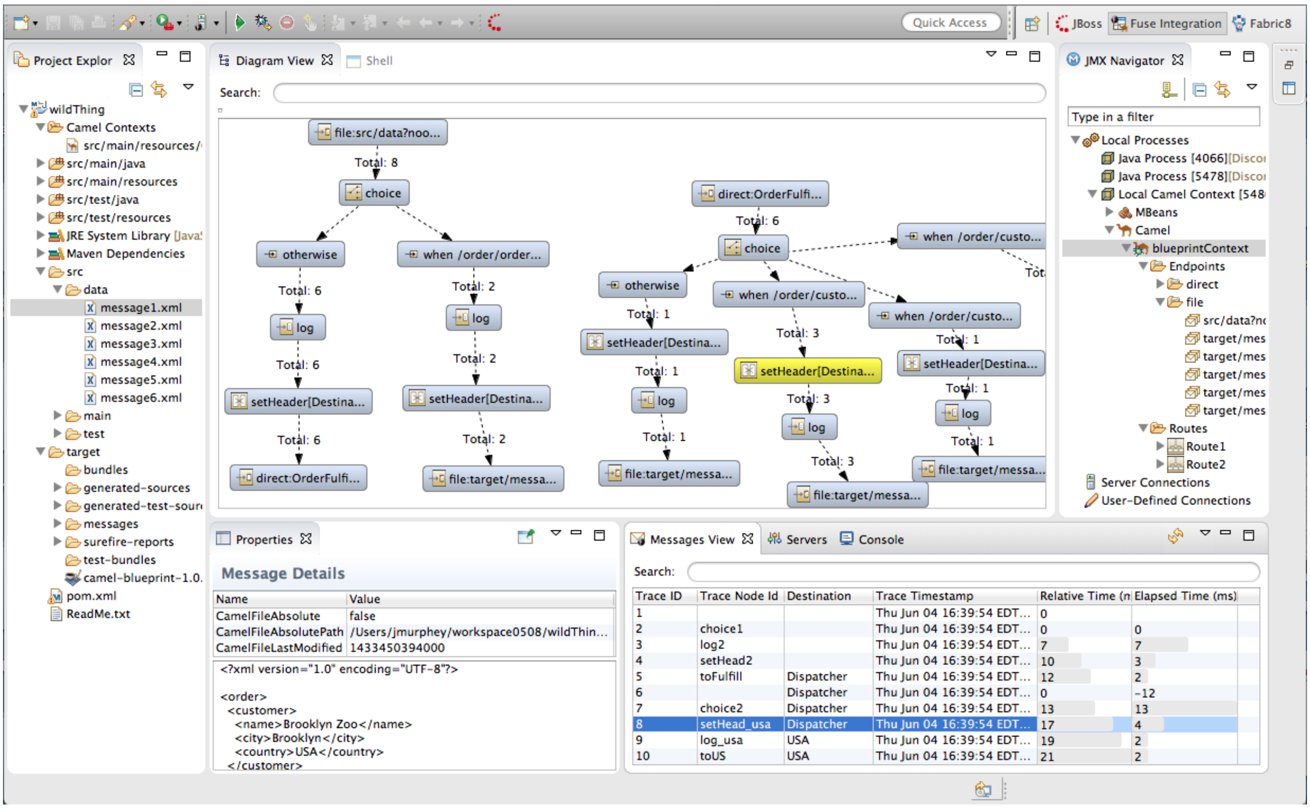
chapter "To Debug a Routing Context" in "Tooling Tutorials"

CHAPTER 3. FUSE INTEGRATION PERSPECTIVE

OVERVIEW

The **`Fuse Integration`** perspective, shown in Figure 3.1, provides access to all of the tooling for monitoring and testing your integration application.

Figure 3.1. Fuse Integration perspective



Message Details

Name	Value
CamelFileAbsolute	false
CamelFileAbsolutePath	/Users/jmurphey/workspace0508/wildThin...
CamelFileLastModified	1433450394000

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

<order>
  <customer>
    <name>Brooklyn Zoo</name>
    <city>Brooklyn</city>
    <country>USA</country>
  </customer>
```

Search:

Trace ID	Trace Node Id	Destination	Trace Timestamp	Relative Time (n	Elapsed Time (ms)
1			Thu Jun 04 16:39:54 EDT...	0	
2	choice1		Thu Jun 04 16:39:54 EDT...	0	0
3	log2		Thu Jun 04 16:39:54 EDT...	7	7
4	setHead2		Thu Jun 04 16:39:54 EDT...	10	3
5	toFulfill	Dispatcher	Thu Jun 04 16:39:54 EDT...	12	2
6		Dispatcher	Thu Jun 04 16:39:54 EDT...	0	-12
7	choice2	Dispatcher	Thu Jun 04 16:39:54 EDT...	13	13
8	setHead_usa	Dispatcher	Thu Jun 04 16:39:54 EDT...	17	4
9	log_usa	USA	Thu Jun 04 16:39:54 EDT...	19	2
10	toUS	USA	Thu Jun 04 16:39:54 EDT...	21	2

The Fuse Integration perspective consists of eight main areas:

- **`Project Explorer`**—displays all of the projects known to the tooling. You can view all of the artifacts that make up each project. **`Project Explorer`** also displays all of the routing context **`.xml`**. files for a project under its **`Camel Contexts`** node. This makes it easy for you to find and open the routing context file of any route included in a project.

- **`JMX Navigator`**—lists the JMX servers and the infrastructure they monitor. It allows you to browse JMX servers and the pocesses they are monitoring. It also identifies instances of Red Hat processes.

- **`Diagram View`**—displays a graphical tree representing the node selected in **`JMX Navigator`**. When you select a process, server, endpoint, or other node, **`Diagram`** view shows the selected node as the root and branches down to its children and grandchildren.

  When you select a broker, **`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, **`Diagram`** view displays all of the nodes in the route and shows the different paths that messages can take through it. It also displays timing metrics for each processing step in the route when route tracing is enabled.

- **Shell** view—Displays the command console of the connected container. You can control the container by entering commands in **Shell** view.

- **Messages View**—lists the messages that have passed through the selected JMS destination or through Apache Camel endpoints when route tracing is enabled.

- **Servers** view—displays a list of servers (Red Hat JBosss Fuse, Apache Karaf, or Apache ServiceMix) managed by the tooling. It displays their runtime status and provides controls for starting and stopping them and for debugging projects deployed on them.

- **Properties** view—displays the properties of the selected node.

- **Console** view— displays the console output for recently executed actions.

JMX NAVIGATOR

The **JMX Navigator** drives all monitoring and testing activities in **Fuse Integration** perspective. It determines which routes are displayed in **Diagram View**, the **Properties** viewer, and **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, **JMX Navigator** shows all of the Java processes running on your local machine. You can add JMX servers as needed to view infrastructure on other machines.

MESSAGES VIEW

The **Messages View** is used for browsing JMS destinations and for tracing messages that have passed through a route.

When a JMS destination is selected in **JMX Navigator**, the view lists all of the messages sitting in the destination.

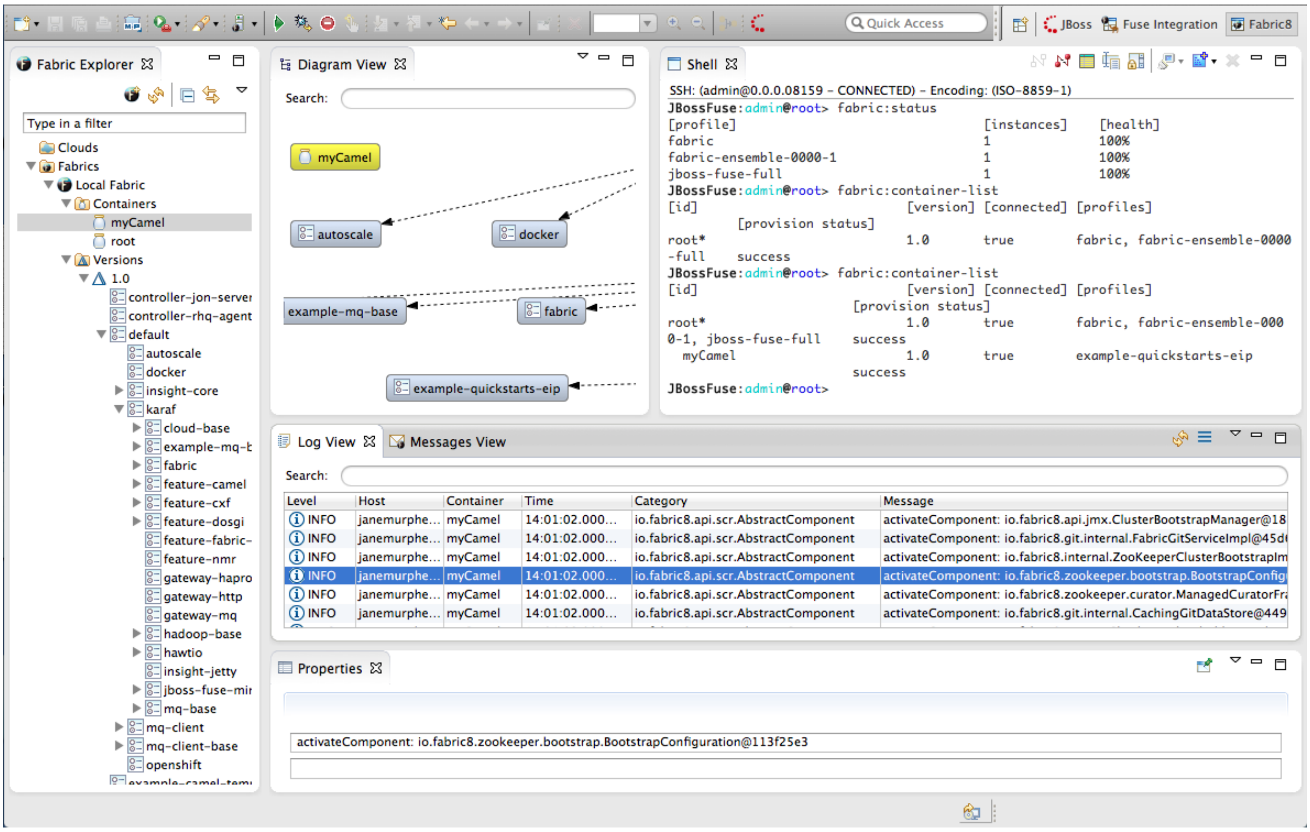
When route tracing is enabled, **Messages View** lists all of the messages that passed through the nodes in the route since tracing started. You can configure **Messages View** to display only the data in which you are interested and in your preferred sequence.

When a message trace in **Messages View** is selected, its details (message body and all message headers) are displayed in the **Properties** viewer. In **Diagram View**, the step in the route associated with the selected message trace is also highlighted.

CHAPTER 4. FABRIC8 PERSPECTIVE

The Fabric8 perspective, shown in Figure 4.1, is where you access the fabric development and debugging tools.

Figure 4.1. Fabric8 perspective



```
SSH: (admin@0.0.0.08159 - CONNECTED) - Encoding: (ISO-8859-1)
JBossFuse:admin@root> fabric:status
[profile]                                [instances]    [health]
fabric                                   1              100%
fabric-ensemble-0000-1                   1              100%
jboss-fuse-full                          1              100%
JBossFuse:admin@root> fabric:container-list
[id]                          [version] [connected] [profiles]
        [provision status]
root*                         1.0       true        fabric, fabric-ensemble-0000
-full    success
JBossFuse:admin@root> fabric:container-list
[id]                          [version] [connected] [profiles]
                         [provision status]
root*                         1.0       true        fabric, fabric-ensemble-000
0-1, jboss-fuse-full     success
  myCamel                     1.0       true        example-quickstarts-eip
                         success
JBossFuse:admin@root>
```

Search:

Level	Host	Container	Time	Category	Message
INFO	janemurphe...	myCamel	14:01:02.000...	io.fabric8.api.scr.AbstractComponent	activateComponent: io.fabric8.api.jmx.ClusterBootstrapManager@18
INFO	janemurphe...	myCamel	14:01:02.000...	io.fabric8.api.scr.AbstractComponent	activateComponent: io.fabric8.git.internal.FabricGitServiceImpl@45d
INFO	janemurphe...	myCamel	14:01:02.000...	io.fabric8.api.scr.AbstractComponent	activateComponent: io.fabric8.internal.ZooKeeperClusterBootstrapIm
INFO	janemurphe...	myCamel	14:01:02.000...	io.fabric8.api.scr.AbstractComponent	activateComponent: io.fabric8.zookeeper.bootstrap.BootstrapConfig
INFO	janemurphe...	myCamel	14:01:02.000...	io.fabric8.api.scr.AbstractComponent	activateComponent: io.fabric8.zookeeper.curator.ManagedCuratorFr
INFO	janemurphe...	myCamel	14:01:02.000...	io.fabric8.api.scr.AbstractComponent	activateComponent: io.fabric8.git.internal.CachingGitDataStore@449

activateComponent: io.fabric8.zookeeper.bootstrap.BootstrapConfiguration@113f25e3

The Fabric8 perspective consists of six main areas:

- **`Fabric Explorer`**—lists the fabrics and the containers, profiles, and versions of which they consist.
- **`Diagram View`**—provides a graphical representation of a node selected in **`Fabric Explorer`**.
- **`Shell`**— provides console access to any container running on the fabric.
- **`Log`** view—lists the log entries of the selected container or the selected JMX process.
- **`Messages View`**—lists the message instances that passed through the nodes in a selected route, after tracing was enabled on the route.
- **`Properties`** view—displays the properties of an object selected in **`Fabric Explorer`**.

FABRIC EXPLORER

`Fabric Explorer` provides access to your fabrics. Once the tooling is connected to a fabric, **`Fabric Explorer`** provides access to all of the fabric's components (containers, profiles, and profile versions) enabling you to configure, run, and debug your project in the fabric environment.

DIAGRAM VIEW

Use **Diagram View** in conjunction with **Messages View** to track a message through the nodes of a route for which tracing has been activated. In **Diagram View**, the node which corresponds to the message instance selected in **Messages View** is highlighted. **Messages View** displays metrics for the selected message instance, including the time at which it exited the highlighted node.

LOG VIEW

Log view displays the log entries of the container selected in **Fabric Explorer**. This feature enables you to see the log messages of any container running in a fabric.

Double-clicking a process in a container's JMX tree opens the process in a jconsole-like viewer, where you can browse its attributes, operations, notifications, and other information, all of which are presented in tabular format.

MESSAGES VIEW

The **Messages View** is used for tracing messages through a route in a fabric environment.

When an Apache Camel route for which route tracing has been activated is selected in **Fabric Explorer**, **Messages View** lists all messages that pass through the route from the time tracing was activated. **Messages View** also provides metrics for each message instance, making it easier to debug routes.

Use the view's **Refresh** button periodically to update its display of message traces.

PROPERTIES VIEW

For container nodes, the **Properties** view also provides the **Profiles** tab, for viewing the profiles assigned to the container, and the **Profile Details** tab, for viewing the details of each assigned profile.

The **Profiles** editor enables you to change the profiles associated with the selected container, and the **Profile Details** editor enables you to modify the configuration of the selected profile.

JMX VIEWER

Double-clicking a process in a container's JMX tree opens the process in a jconsole-like viewer, where you can browse its attributes, operations, notifications, and other information, all of which are presented in tabular format.

PART II. DEVELOPING APPLICATIONS

We reccomend that you develop your applications in `Chapter 1, JBoss Perspective` because it provides more room to expand the route editor's canvas when you are building complex routes. If it is not already open, click JBoss on the right side of the JBoss Developer Studio tool bar.

CHAPTER 5. CREATING A NEW FUSE PROJECT

OVERVIEW

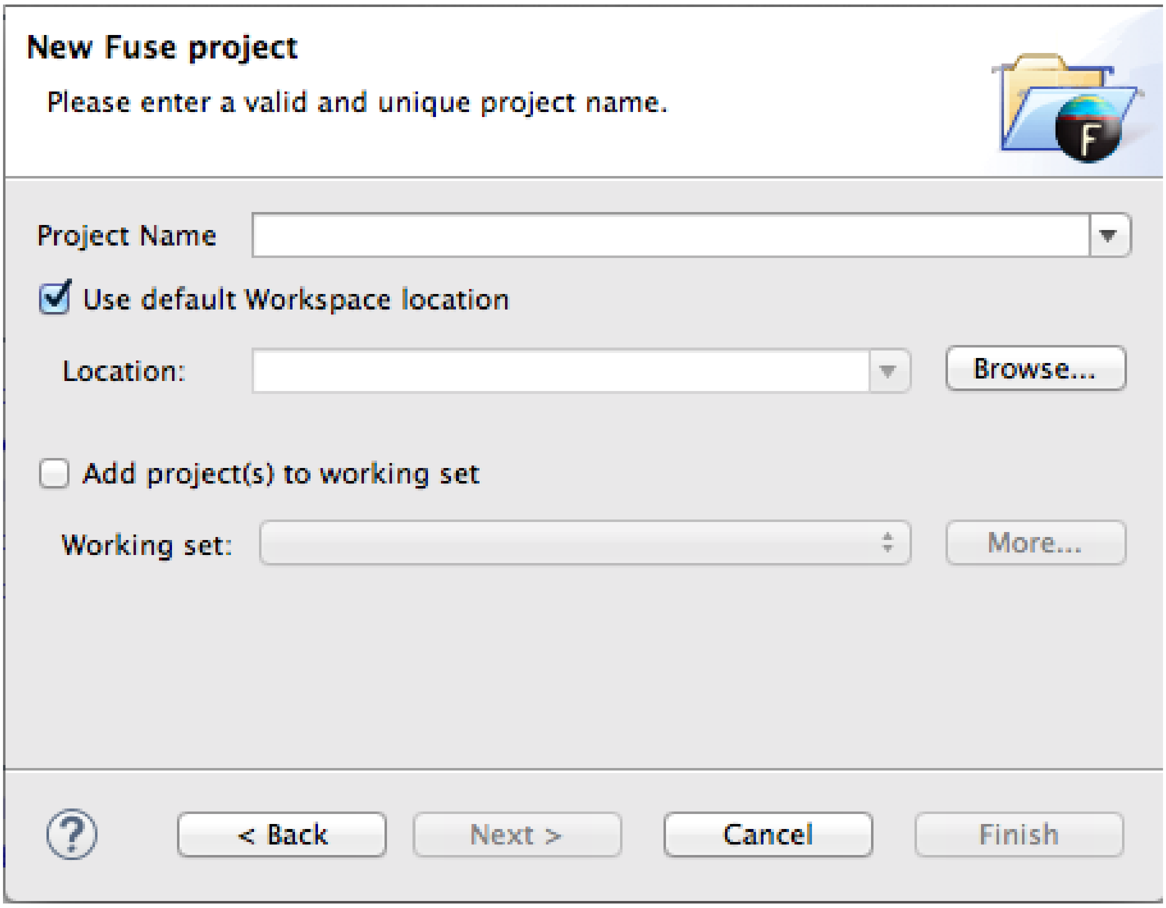
The tooling uses Maven archetypes to generate a project with all of the dependencies preconfigured. The archetypes also create the POM file needed to run and deploy your application, as well as sample code and data to get you started.

PROCEDURE

To create a Fuse project:

1. Select **File** → **New** → **Fuse Project** to open the **`New Fuse Project`** wizard shown in Figure 5.1, “New Fuse Project Wizard's Project Location Page”.

   Figure 5.1. New Fuse Project Wizard's Project Location Page

   New Fuse project

   Please enter a valid and unique project name.

   ☑ Use default Workspace location

   □ Add project(s) to working set

   The wizard opens with the **`Use default workspace location`** option selected on the location page.

2. Enter the name of the Fuse project.

3. Specify the location where the data for the project will be stored.

   - To use the default workspace select **`Use default workspace location`**.

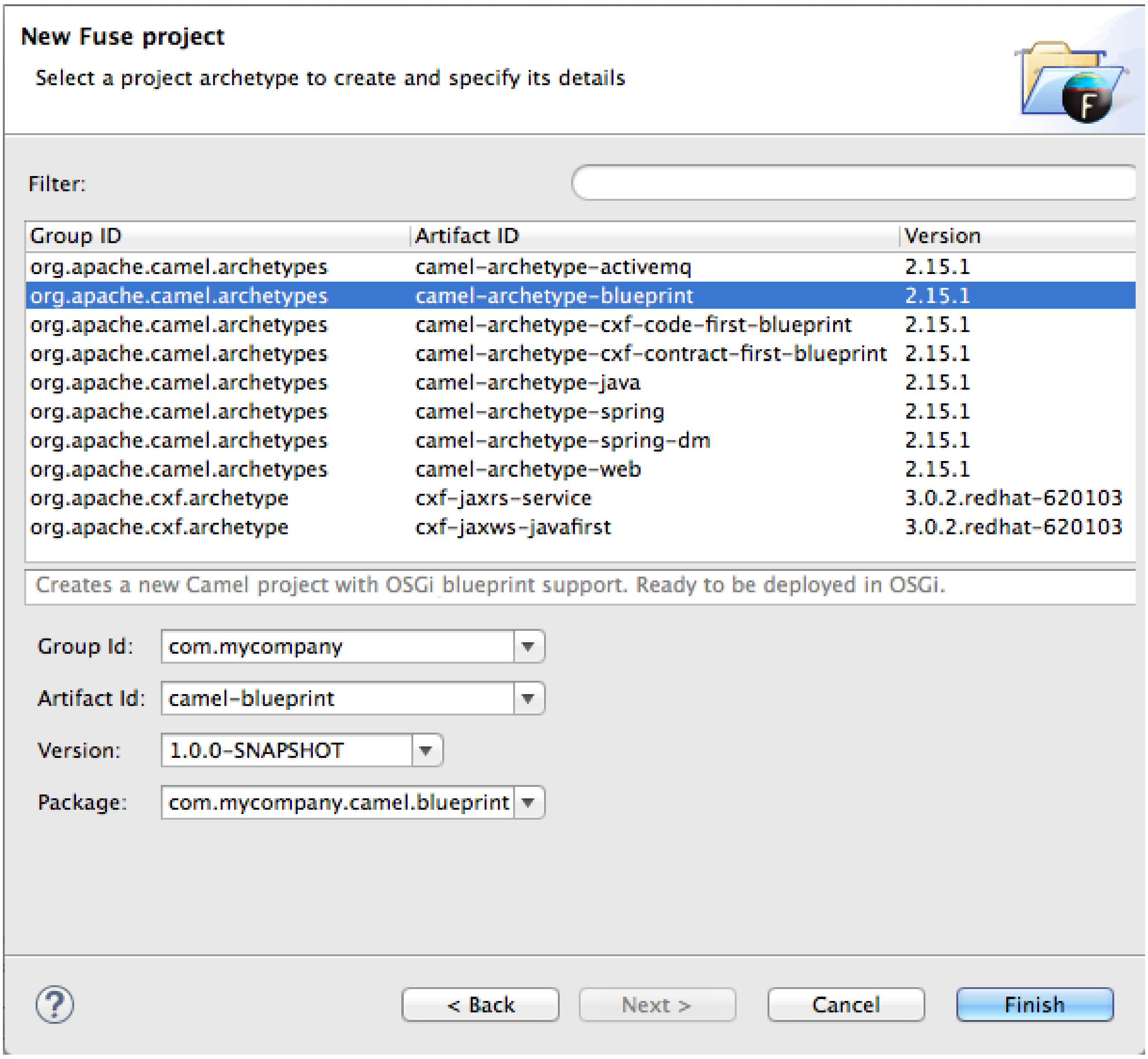
   - To use an alternative location clear **`Use default workspace location`** and specify a new location in the provided field.

Clicking Browse... opens a file browser.

4. If you want to add the new project to an Eclipse working set, select **`Add project(s) to working set`** and enter the name of the working set.

5. Click Next > to open the **New Fuse Project** details page shown in Figure 5.2.

Figure 5.2. New Fuse Project wizard's details page



New Fuse project

Select a project archetype to create and specify its details

Filter:

Group ID	Artifact ID	Version
org.apache.camel.archetypes	camel-archetype-activemq	2.15.1
org.apache.camel.archetypes	camel-archetype-blueprint	2.15.1
org.apache.camel.archetypes	camel-archetype-cxf-code-first-blueprint	2.15.1
org.apache.camel.archetypes	camel-archetype-cxf-contract-first-blueprint	2.15.1
org.apache.camel.archetypes	camel-archetype-java	2.15.1
org.apache.camel.archetypes	camel-archetype-spring	2.15.1
org.apache.camel.archetypes	camel-archetype-spring-dm	2.15.1
org.apache.camel.archetypes	camel-archetype-web	2.15.1
org.apache.cxf.archetype	cxf-jaxrs-service	3.0.2.redhat-620103
org.apache.cxf.archetype	cxf-jaxws-javafirst	3.0.2.redhat-620103

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

Group Id: com.mycompany

Artifact Id: camel-blueprint

Version: 1.0.0-SNAPSHOT

Package: com.mycompany.camel.blueprint

6. Select a project type from the list.

NOTE

The route editor works with these project types:

- **`camel-archetype-activemq`**

  Creates a new Apache Camel project that configures and interacts with Apache ActiveMQ.

- **`camel-archetype-blueprint`**

  Creates a new Apache Camel project with support for OSGi blueprint that is deployment-ready for OSGi.

- **`camel-archetype-cxf-code-first-blueprint`**

  Creates a new Apache Camel project with Apache CXF code-first example using OSGi Blueprint.

- **`camel-archetype-cxf-contract-first-blueprint`**

  Creates a new Apache Camel project with Apache CXF contract-first example using OSGi Blueprint.

- **`camel-archetype-java`**

  Creates a new Apache Camel project using Java DSL.

  You cannot edit Java DSL in the route editor.

- **`camel-archetype-spring`**

  Creates a new Apache Camel project with added support for Spring DSL.

- **`camel-archetype-spring-dm`**

  Deprecated Creates a new Apache Camel project with added support for Spring DSL that is deployment-ready for OSGi.

- **`camel-archetype-web`**

  Creates a new Apache Camel web project that deploys the Camel routes as a WAR.

- **`cxf-jaxrs-service`**

  Creates a simple CXF JAX-RS web application service using Spring configuration.

- **`cxf-jaxws-javafirst`**

  Creates a project for developing a Web service starting from Java code.

7. Enter a group ID for the project in the **`Group Id`** field, or accept the default.

   The tooling uses the group ID to form the first part of the dot-separated package name. For example, if you enter **`Demo`** for the group ID, it appears in the **`Package`** field as **`Demo.`**.

8. Enter an artifact ID for the project in the **`Artifact Id`** field, or accept the default.

   The tooling uses the artifact ID as the name of the project and to form the second part of the dot-separated package name. For example, if you enter **`BigRoute`** for the artifact ID, it appears in the **`Package`** field as **`Demo.BigRoute`**.

9. Enter a version for the project in the **`Version`** field, or accept the default.

10. If you want to change the package name generated for the artifacts, enter the new package name in the **`Package`** field.

11. Click **`Finish`** to create the Maven project, which contains starting point artifacts.

RESOLVING MAVEN DEPENDENCY ERRORS

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

Though it can happen at other times, it more typically occurs when you create a project based on any of the supplied archetypes for the first time, but then cancel the project before the process finishes. 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 easily resolve these dependency errors by updating Maven dependencies this way:

1. In **`Project Explorer`**, select the root project just created.

2. Right-click it to open the context menu.

3. Select **Maven → Update Project...**

4. In the **`Update Maven Project`** wizard, check **`Force Update of Snapshots/Releases`**.

5. Click **`OK`**.

   In the bottom, right corner of the workbench, you may see the progress status bar churning as missing dependencies are downloaded from the Maven repositories.

RELATED TOPICS

New Fuse Project
Red Hat JBoss Fuse Tooling Tutorials, To Create a New Route

CHAPTER 6. CREATING A NEW CAMEL XML FILE

OVERVIEW

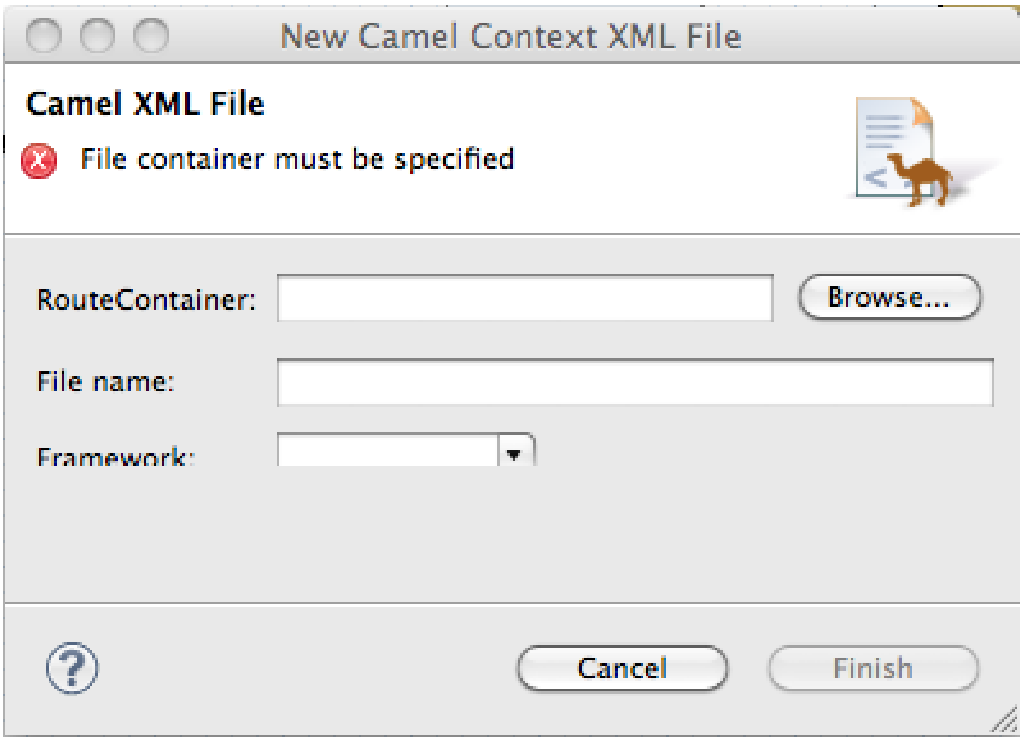
Apache Camel stores routes in an XML file that contains a **`camelContext`** element. The tooling includes a wizard that simplifies adding an Apache Camel context file to your project. It creates a new XML file that has all of the required namespaces preconfigured and a template **`camelContext`** element.

PROCEDURE

To add an Apache Camel context file to your project:

1. Select **File** → **New** → **Other...** → **Fuse Tooling** → **Camel XML File** from the main menu to open the **`Camel XML File`** wizard, as shown in Figure 6.1, “Camel XML File wizard”.

   Figure 6.1. Camel XML File wizard

   New Camel Context XML File

   Camel XML File

   File container must be specified

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.

   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`**—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.

RELATED TOPICS

New Camel XML File

CHAPTER 7. EDITING A ROUTING CONTEXT IN THE ROUTE EDITOR

Developing an Apache Camel application typically consists of the following tasks:

1. Adding one or more routes to the routing context.
2. Adding a starting point pattern to a route.
3. Adding one or more endpoint patterns to a route.
4. Adding one or more processor patterns that represent how messages will be transformed and routed between the starting point and endpoint(s).
5. Connecting the patterns (referred to as nodes once they are placed on the canvas).
6. Configuring the details for each of the endpoints and processors that make up the route.
7. Adding any configuration beans to the context.

7.1. 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, but the route editor's canvas can display only one of the routes at a time. Therefore the canvas is each route's delineator, and each route displayed on the canvas maps to a **`route`** element in the generated **`camelContext`** element.

When you create a new Fuse Project, the tooling creates an example **`camelContext.xml`** file that is defined either as **`camel-context.xml`** (Spring) or **`blueprint.xml`** (Blueprint). You can view and edit the contents of the **`camelContext.xml`** file in the route editor's **`Source`** view. In **`Design`** view, the route editor presents an empty canvas, which represents the empty **`route`** element. You can drag patterns from the **`Palette`** and drop them onto the canvas to create a route. The design time tooling updates the empty **`route`** element with XML code generated from the patterns you drop onto the canvas.

The tooling provides three methods for adding a new route:

- From the menu bar, by opening the **Routes** menu, and then selecting **Add Route**
- In **`Design`** view, by right-clicking the canvas or a node to access the context menu, and then selecting **Add** → **Route**
- In **`Source`** view, by adding a **`<route/>`** element to the existing list within the **`camelContext`** element

Procedure

To add another route to the camelContext:

1. Select one of the methods for adding a route.

In **`Design`** view, a route icon appears in **`Outline`** view, and the **`Properties`** editor displays the list of the new route's properties for you to edit.

2. In the **`Properties`** editor, enter an ID (for example, Route2) for the new route in the route's **`ID`** field.

   In **`Outline`** view, the new ID displays next to the new route icon.

3. In the **`Properties`** editor, enter a description of the route in the **`Description`** field.

4. On the menu bar, select **File** → **Save** to save the changes you made to the routing context file.

NOTE

To switch between multiple routes, select the route you want to display on the canvas by clicking it in **`Outline`** view or by selecting it from the **Routes** menu on the menu bar.

Related topics

Red Hat JBoss Fuse Tooling Tutorials, To Add a Content-Based Router
Red Hat JBoss Fuse Tooling Tutorials, To Add Another Route to the CBR Routing Context

7.2. ADDING PATTERNS TO A ROUTE

Routes consist of a sequence of connected patterns, referred to as nodes once they are placed on the canvas. To add a pattern to a route you can either:

- Drag the pattern from the palette
- Use the context menu

Both methods provide access to all of the available patterns. Dragging patterns from the palette and dropping them onto the canvas is an easy and convenient way to add multiple patterns to the canvas quickly. Using a node's context menu, you can add and automatically connect a pattern to the node.

7.2.1. Drag and drop a pattern

Overview

The **`Palette`** lists all of the available patterns, grouped by function. For example, the Loop pattern is part of the **`Control Flow`** group.

You can select any pattern from the **`Palette`** and drag it to the canvas, which represents a single route.

Procedure

To drag a pattern onto a route:

1. In the **`Palette`**, locate the desired pattern.

2. Select the pattern, drag it to the canvas, and then release the mouse button.

NOTE

You need not place the pattern in its intended location. You can easily move any node on the canvas by dragging it to a new location.

The new pattern appears on the canvas and becomes the selected node. The `Properties` editor displays a list of the node's properties for you to edit and configure.

Related topics

Section 7.5, “Rearranging patterns on the canvas”
Section 7.4, “Configuring a pattern”
Section 7.6, “Removing patterns from a route”

7.2.2. Using the context menu

Overview

Nodes on the canvas have a context menu that includes an **Add** option for adding new patterns to a route.

Procedure

To add and connect a pattern to a node in a route:

1. In the canvas, select the node to which you want to connect the new pattern.
2. Right-click it to open the context menu.
3. Select **Add** to display the list of patterns that can be connected to the selected node. The patterns are grouped according to function.
4. Select the pattern to add to the route.

The new pattern is added to the route and automatically linked to the selected node. The new pattern becomes the currently selected node, and the `Properties` editor displays a list of the node's properties for you to edit and configure.

Related topics

Section 7.5, “Rearranging patterns on the canvas”
Section 7.4, “Configuring a pattern”
Section 7.6, “Removing patterns from a route”

7.3. CONNECTING PATTERNS TO MAKE A ROUTE

Overview

Until the patterns on the canvas (referred to as nodes) are connected, they do not constitute a route. The editor does not save a `route` element in the context file until all nodes are linked together to form a complete route. A complete route typically consists of a starting endpoint, a string of processing nodes, and one or more destination endpoints.

Connecting two nodes on the canvas is as simple as dragging a line from one to the other. Each node has a connector arrow. Selecting a node and dragging its connector arrow to a target node establishes a connection between the two nodes.

Procedure

To connect two nodes:

1. On the canvas, select the source node to display its connector arrow.

2. Click-drag the source node's connector arrow (↗) to the target node.

   The direction of the connection represents the direction messages flow between the nodes in the route.

3. While hovering over the target node, release the mouse button to drop the connector on it.

   The route editor updates the `<route>` element with the xml generated from the connection. You can view the xml in `Source` view.

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

7.4. CONFIGURING A PATTERN

Overview

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

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

- validating that all required properties have values
- 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 `Details` tab in the `Properties` editor lists all of the selected node's properties for you to edit. The `Documentation` tab describes the pattern and each of its properties.

2. Edit the fields in the `Properties` editor to configure the node.

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

Related topics

7.5. REARRANGING PATTERNS ON THE CANVAS

As your route grows and changes, you'll likely need to change how it is laid out on the canvas. Changing the layout does not change the underlying DSL, but it does make it easier to visualize the route.

The tooling provides two ways to rearrange the patterns on the canvas:

- dragging and dropping
- the context menu's

7.5.1. Rearranging patterns by dragging them

Overview

You can select one or more nodes on the canvas and drag them to a new location. Each node's connections are retained and move along with it. However, the layout of the route will continue to change as it grows in scope and complexity.

Procedure

To drag a node to a new location on the canvas:

1. Determine which node or nodes you want to move.

2. Select one node, or select a group of nodes by dragging a box around them.

3. Drag the node or nodes to their new location, then release the mouse button.

NOTE

You can also rearrange the connectors between nodes to accommodate increasing numbers of patterns on the canvas. Clicking a connector joining two nodes on the canvas reveals a line segmented by a bendpoint. You can drag the bendpoint to another location on the canvas while leaving the two nodes in place. Dragging a bendpoint creates two new bendpoints on either side of it, further segmenting the connector. This feature makes it easy to organize the nodes in a complex route for visual clarity.

7.5.2. Automatically aligning patterns

Overview

The **Layout Diagram** command on the canvas' context menu automatically aligns all nodes on the canvas relative to the route's start point. By default, the route editor arranges the nodes in a route from left to right across the canvas, according to the order in which they are connected to each other. It moves the start point to the left edge of the canvas, then aligns the processing nodes and endpoints accordingly.

NOTE

You can change the default layout direction (**Right**). To do so, on the menu bar select **Window** → **Preferences** → **Fuse Tooling** → **Editor**, and then select **Down** from the **`Choose the layout direction for the diagram editor`**'s drop-down list.

You can continue adding more patterns to your route after you execute the **Layout Diagram** command.

Procedure

To direct the tooling to automatically align all nodes on the canvas:

1. Right-click the canvas to open the context menu.

2. Select **Layout Diagram**.

7.6. 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 **Remove** option on the nodes' context menu makes this easy to do. When you delete a node from the canvas, all of its connections with all other nodes on the canvas are also deleted.

NOTE

You can also remove a node by right-clicking it in **`Outline`** view and selecting **Remove**, or by selecting **Edit** → **Delete** from the menu bar.

Procedure

To remove a node from a route:

1. Select the node you want to delete.

2. Right-click it to open the context menu.

3. Select **Remove**.

The node and all of its connections are deleted from the canvas.

Related topics

Section 7.2, “Adding patterns to a route”

7.7. DISCONNECTING TWO PATTERNS

Overview

As you develop and update a route, you may need to disconnect some nodes. The **Remove** option on the connector context menu makes this easy to do.

> **NOTE**
>
> You can also delete a connector by selecting **Edit** → **Delete** from the menu bar.

Procedure

To disconnect two nodes:

1. Select the connector you want to delete.

2. Right-click it to open the context menu.

3. Select **Remove**.

The connector between the nodes is removed.

Related topics

Section 7.3, “Connecting patterns to make a route”

7.8. DELETING A ROUTE

Overview

In some cases you made need to delete an entire route from your routing context. The **Remove Route** option on the canvas' context menu makes this easy to do. When you delete a route, all of the nodes the route contains are also deleted.

> **NOTE**
>
> You can also remove a route from the **Routes** menu on the menu bar.

Procedure

To delete a route:

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

2. Right-click on the canvas to open the context menu.

3. Select **Remove Route**.

The route is removed from the canvas and from the camelContext.

7.9. ADDING BEANS AND CONFIGURATION

Overview

Many routing patterns rely on references to Java objects (beans) for configuration or for implementation details. You add the beans into the routing context file using the route editor's **`Source`** tab.

Procedure

To add beans to your routing context file:

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

2. Click the **`Source`** tab at the bottom of the route editor's canvas to switch to Source view, so you can edit the XML that defines the route.

3. Enter the **`bean`** elements needed by your route before the **`camelContext`** element.

   NOTE

   Use the **`id`** attribute to identify the bean, not the **`name`** attribute.

   IMPORTANT

   Do not edit the contents of the **`camelContext`** element. Red Hat JBoss Fuse Tooling overwrites the **`camelContext`** element when changes are made to the route diagram in **`Design`** view.

4. Save your changes by selecting **File** → **Save** on the menu bar.

5. Click the **`Design`** tab at the bottom of the route editor's canvas to return to Design view and the route diagram.

Related topics

Chapter 9, *The Source View*
Apache Camel Development Guide , Part IV Programming EIP Components

7.10. CONFIGURING THE ROUTE EDITOR

Overview

Using JBoss Developer Studio's preference panel, you can specify many aspects of the route editor's behavior:

- The default language to use for expressions in EIPs

- The method for labeling nodes on the canvas

- The direction in which patterns flow on the canvas when creating routes

- Whether the canvas displays a grid overlay

- Colors used for various diagrammatic components

Procedure

To configure the route editor:

1. In JBoss Developer Studio, click **Window** → **Preferences** to open the **`Preferences`** dialog.
2. Click **Fuse Tooling** → **Editor** to show the **`Editor`** preferences.
3. Select, from the drop-down list, the default language you want to use for expressions in EIP nodes.
4. Click the checkbox next to **`If enabled the ID values will be use for labels if existing`** to enable or disable using node IDs as labels.
5. Select, from the drop-down list, the direction in which you want the route editor to align the patterns in a route.
6. Click the checkbox next to **`Show diagram grid in Routes Editor`** to enable or disable displaying the grid overlay on the canvas.
7. Click **`Apply`** to apply the changes to the **`Editor`** preferences.
8. Click **Fuse Tooling** → **Colors** to show the **`Colors`** preferences.
9. For each component whose color you want to change, click its color icon to open the color palette, and then select a color from the palette.
10. When done, click **`Apply`** and then **`OK`** to close the **`Preferences`** dialog.

    You can restore the route editor's original color scheme at any time by returning to **Fuse Tooling** → **Colors** and clicking **`Restore Defaults`**.

Related topics

Editor
Colors

CHAPTER 8. 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.

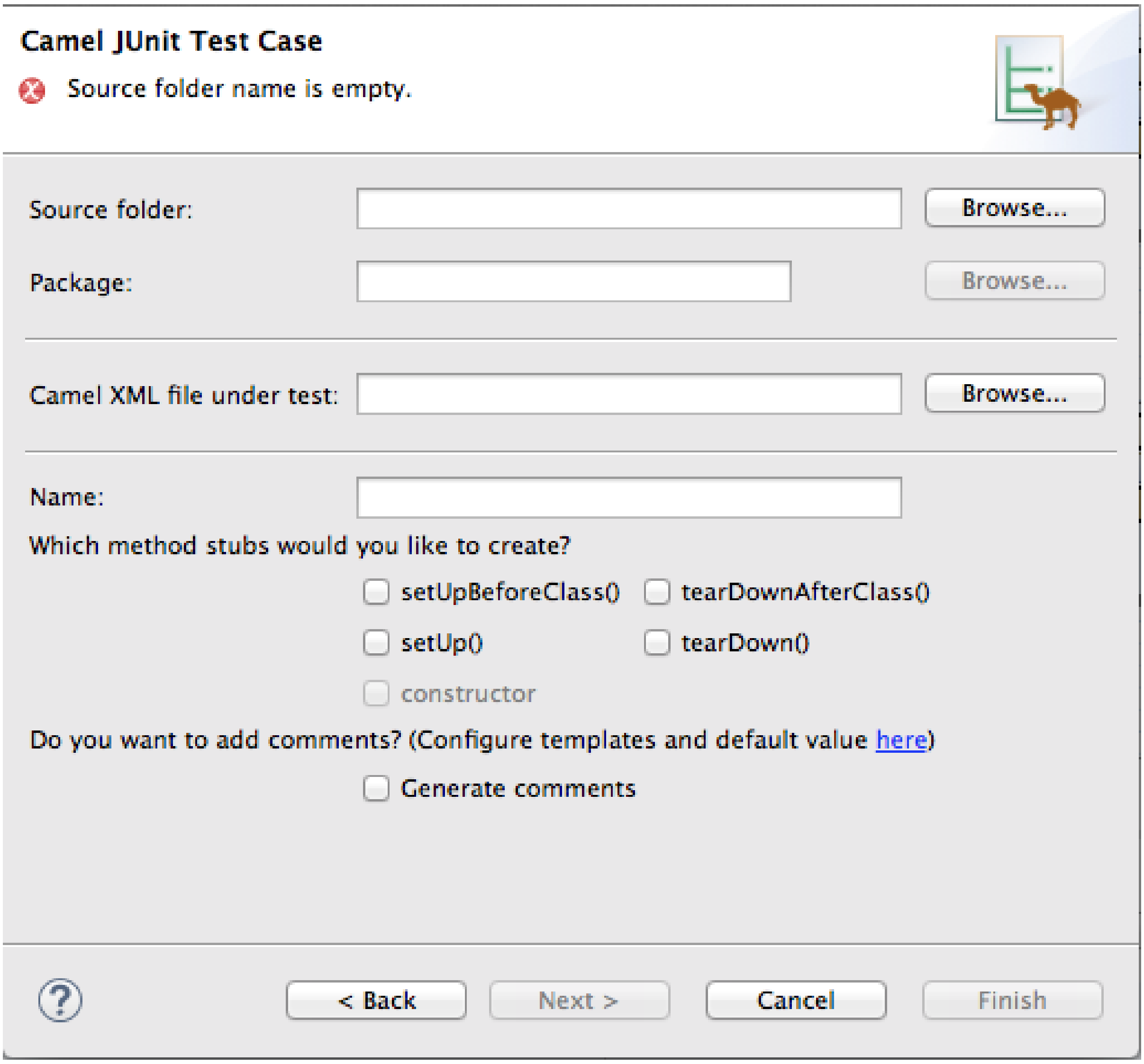
PROCEDURE

To create a new JUnit test case for your route:

1. In `Project Explorer`, select the `camel-context.xml` file in your routing 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 8.1, “New Camel JUnit Test Case wizard”.

Figure 8.1. New Camel JUnit Test Case wizard



Camel JUnit Test Case

Source folder name is empty.

Source folder:

Package:

Camel XML file under test:

Name:

Which method stubs would you like to create?

□ setUpBeforeClass() □ tearDownAfterClass()

□ setUp() □ tearDown()

□ constructor

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

□ Generate comments

Alternatively, you can open the wizard by selecting **File** → **New** → **Other...** → **Fuse Tooling** → **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.

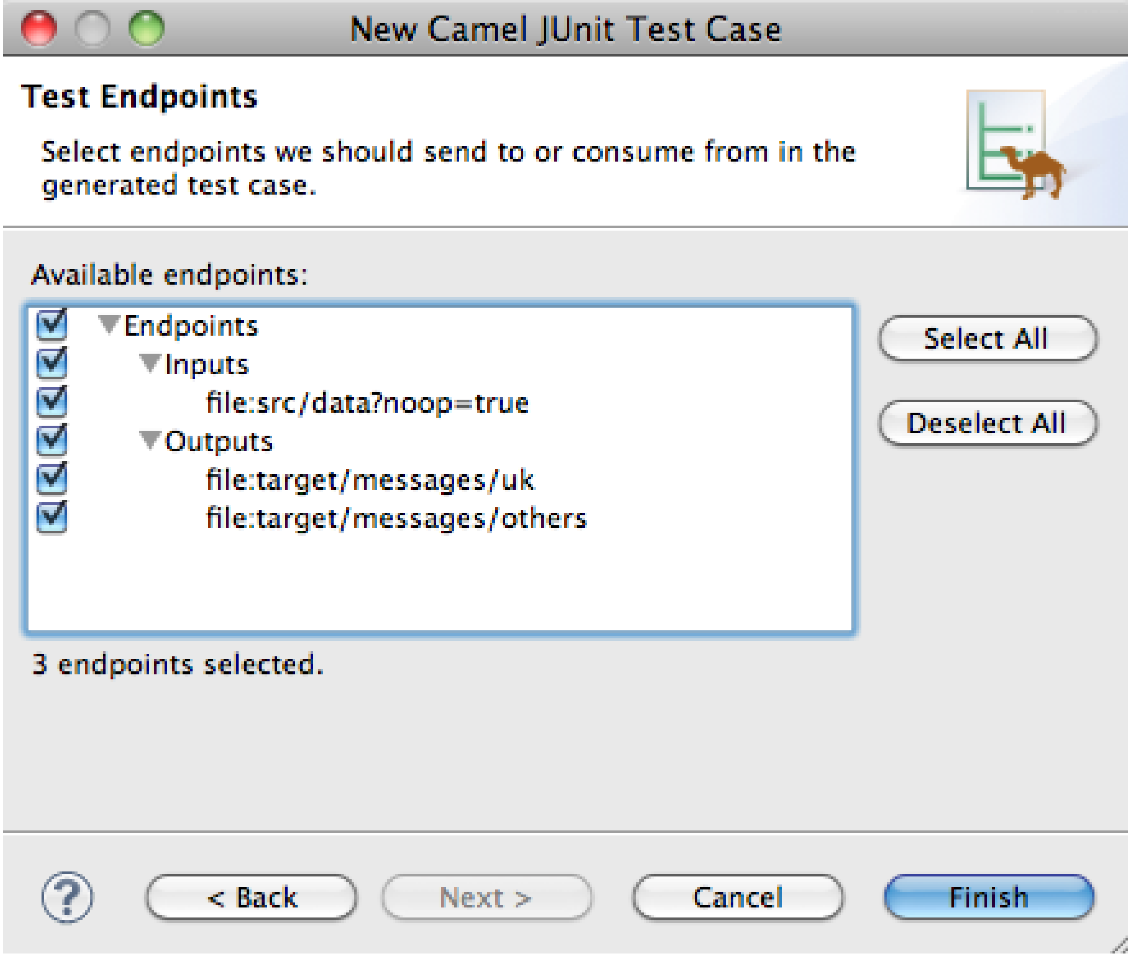
6. In **`Name`**, accept the default name for the generated test class, or enter another name.

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 8.2, “New Camel JUnit Test Case page” shows a route's input and output file endpoints selected.

Figure 8.2. New Camel JUnit Test Case page



New Camel JUnit Test Case

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 **Project Explorer** under **src/test/java**. The class implementing the test case opens in the Java editor.

RELATED TOPICS

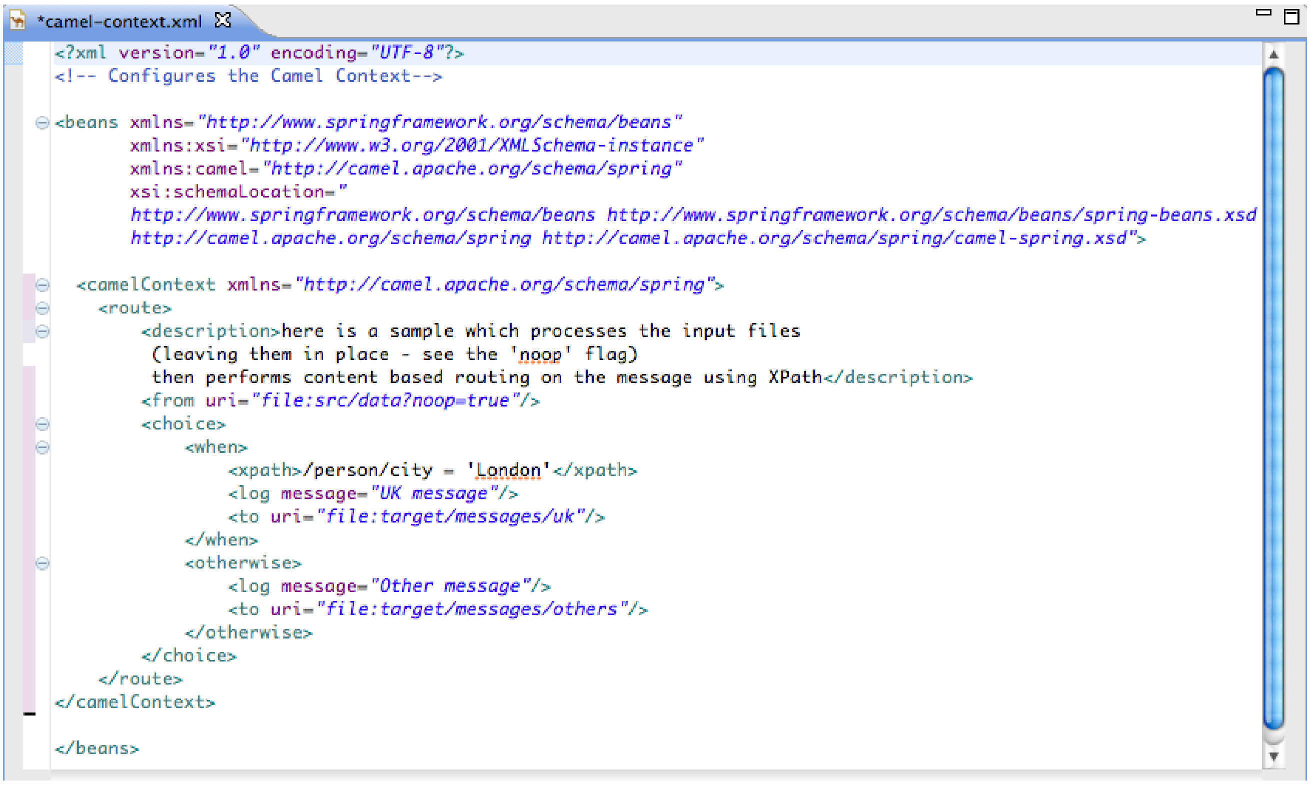
New Camel JUnit Test Case
Test Endpoints

Red Hat JBoss Fuse Tooling Tutorials, To Test a Route with JUnit

CHAPTER 9. THE SOURCE VIEW

The tooling's **Source** tab opens the route editor in **Source** view (Figure 9.1) to display the contents of an Apache Camel route file as XML.

Figure 9.1. Source view



```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Configures the Camel Context-->

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:camel="http://camel.apache.org/schema/spring"
       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 xmlns="http://camel.apache.org/schema/spring">
    <route>
        <description>here is a sample which processes the input files
         (leaving them in place - see the 'noop' flag)
         then performs content based routing on the message using XPath</description>
        <from uri="file:src/data?noop=true"/>
        <choice>
            <when>
                <xpath>/person/city = 'London'</xpath>
                <log message="UK message"/>
                <to uri="file:target/messages/uk"/>
            </when>
            <otherwise>
                <log message="Other message"/>
                <to uri="file:target/messages/others"/>
            </otherwise>
        </choice>
    </route>
</camelContext>

</beans>
```

The **Source** view is useful for editing and adding any configuration, comments, or beans to the routing context file. The Content Assist functionality helps you when working with configuration files. In the **Source** view, press **Ctrl+Space** to see a list of possible values that can be inserted into your project.

While the route editor allows you to change the contents of the **camelContext** element in the routing context file, not all changes are preserved. Comments inserted into the **camelContext** element are lost upon switching to **Design** view.

CHAPTER 10. RUNNING ROUTES INSIDE RED HAT JBOSS FUSE TOOLING

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

- As a local camel context (with or without tests)
- Using Maven

10.1. 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 **`Project Explorer`**'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 **`Project Explorer`**, 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 10.3.1, “Editing a Local Camel Context runtime profile”
Red Hat JBoss Fuse Tooling Tutorials, To Run a Route

10.2. 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 **`Project Explorer`**, select the root of the project .
2. Right-click it to open the context menu, and then select **Run As** → **Maven build**.
   - 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:

     1. 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 will be similar to **`~/workspace/simple-router`**.

     2. In the **`Goals:`** field, enter **`camel:run`**.
     3. Click **`Apply`** and then **`Run`**.
   - 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 10.3.2, “Editing a Maven runtime profile”

10.3. WORKING WITH RUNTIME PROFILES

Red Hat JBoss 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.

10.3.1. 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:

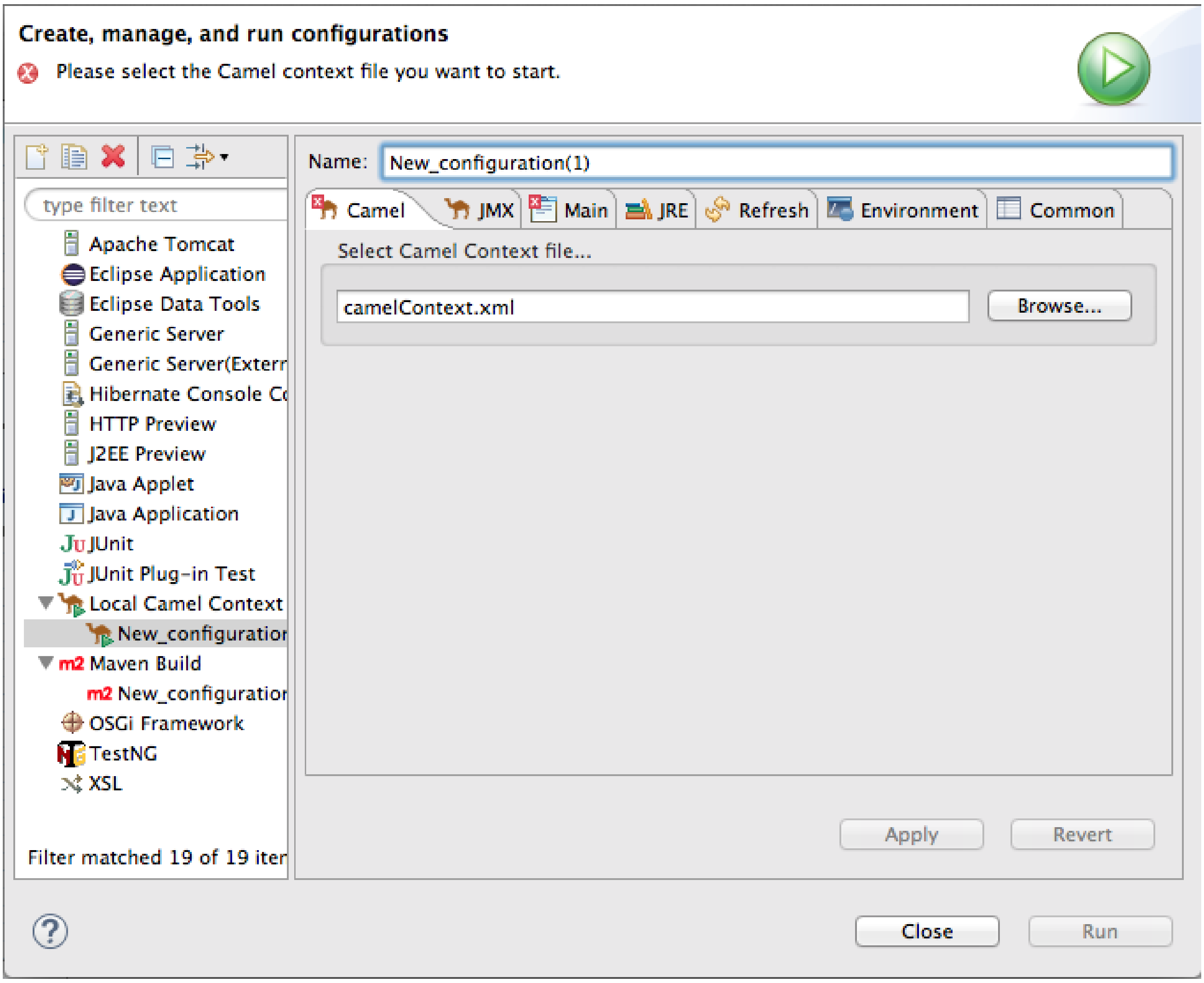
- **Camel Context File**—specifies the name of the new configuration and the full path of the routing context file that contains your routes.
- **Main**—specifies the fully qualified name of the project's base directory, a few options for locating the base directory, any goals required to execute before 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 JBoss 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 **Project Explorer**, select the camelContext 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 10.1. Runtime configuration editor for Local Camel Context



Create, manage, and run configurations

Please select the Camel context file you want to start.

Name: New_configuration(1)

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**`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 10.1, “Running routes as a local Camel context”

10.3.2. 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.

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.

- **`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 using Maven, you must create a default runtime profile for it.

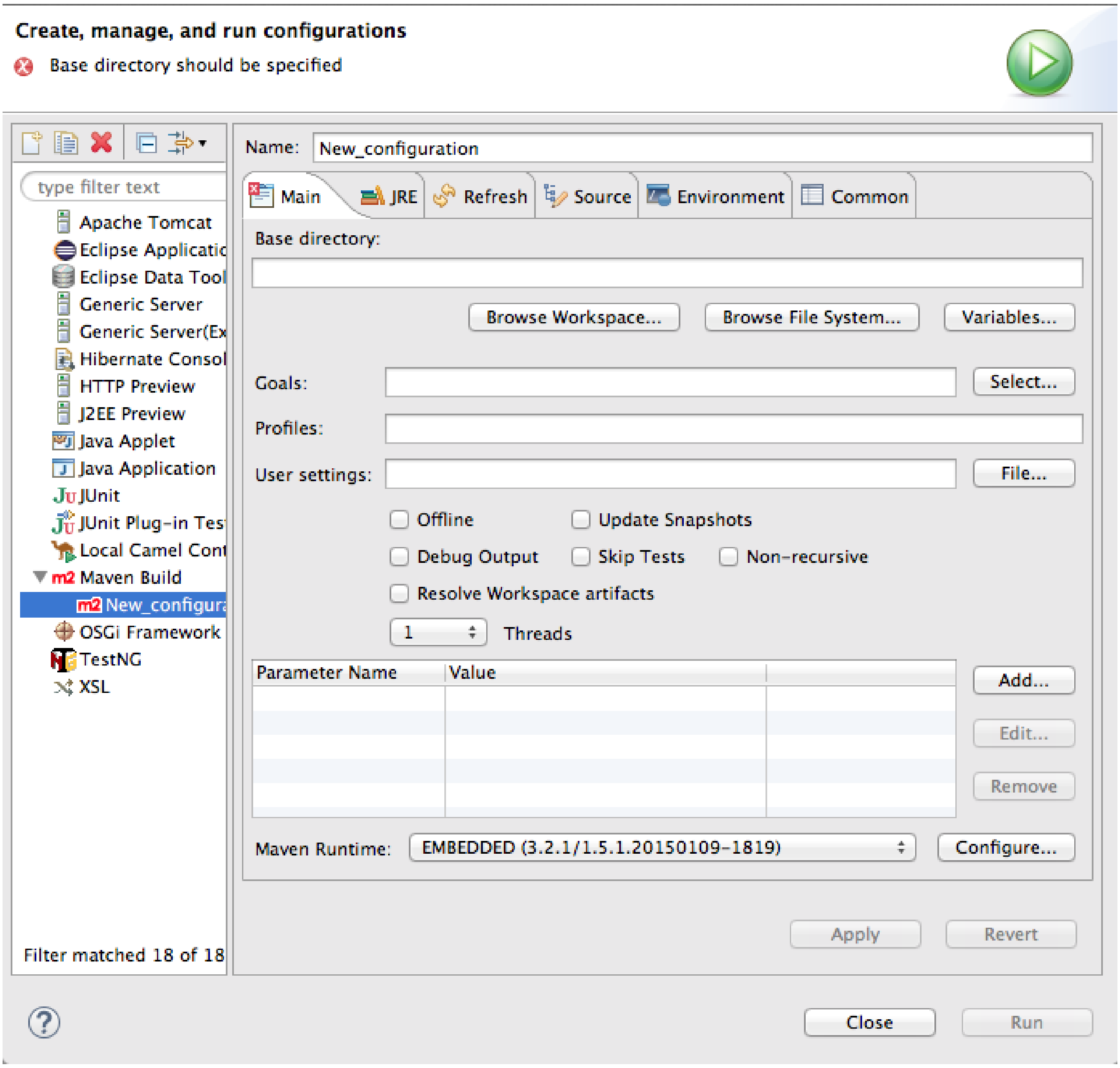
Accessing the Maven runtime configuration editor

1. In **`Project Explorer`**, 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 **m2 Maven Build**, and then click 🗋 at the top, left of the context selection pane.

Figure 10.2. Runtime configuration editor for Maven



Create, manage, and run configurations

☒ Base directory should be specified

Name: New_configuration

Base directory:

Goals:

Profiles:

User settings:

□ Offline □ Update Snapshots

□ Debug Output □ Skip Tests □ Non-recursive

□ Resolve Workspace artifacts

1 Threads

Parameter Name	Value	

Maven Runtime: EMBEDDED (3.2.1/1.5.1.20150109-1819)

Filter matched 18 of 18

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.

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 JBoss 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 **`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 10.2, “Running routes using Maven”

CHAPTER 11. USING THE JBOSS FUSE SAP TOOL SUITE

Abstract

The JBoss 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). Note that the SAP Tool Suite depends on the JCo and IDoc client libraries from SAP. *You must have an SAP Service Marketplace Account*, in order to install and use these libraries.

11.1. INSTALLING THE JBOSS FUSE SAP TOOL SUITE

Overview

The JBoss Fuse SAP Tool Suite provides a SAP Connections view for configuring SAP connections and a collection of SAP components for integrating your Camel routes with an SAP server. 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 JBoss Fuse Supported Configurations.

Prerequisites

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

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

Please note the following points:

- You will require an SAP Service Marketplace Account to download the JCo and IDoc client archives.
- Choose the appropriate JCo and IDoc libraries for your operating system.
- 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 JBoss Fuse SAP Tool Suite into JBoss Developer Studio, perform the following steps:

1. In JBoss Developer Studio, select **File** → **Import** to open the `Import` wizard.

2. In the **Select** screen of the **Import** wizard, select **JBoss Fuse → Install JBoss Fuse SAP Tool Suite**, and click **Next**.

3. The **Install the JBoss 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:

   - JBoss 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.

11.2. CREATE AND TEST AN SAP DESTINATION CONNECTION

Overview

The JBoss Fuse SAP Tools Suite provides an SAP Connections view, which enables you to create and manage SAP destination connections and SAP server connections. This section describes how to create and test an SAP destination connection in the SAP Connections view.

Procedure

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

1. If you have not already added the **`SAP Connections`** view to your perspective, select **Window → Show View → Other** to open the **`Show View`** dialog. Select **`SAP Connections`** and click **`OK`**. The **`SAP Connections`** view opens on the left of your perspective (under **`Project Explorer`**).

2. In the **`SAP Connections`** view, right-click on **`Destination Data Store`** and select **`New Destination`**, to open the **`Create Destination`** dialog.

3. Enter a name for the destination in the **`New Destination Name`** field and click **`OK`**.

4. Select the newly created destination in the **`Destination Data Store`** tree, so that its properties show up in the **`Properties`** view.

5. In the **`Properties`** view, click on 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`**

   These are standard SAP client connection properties. If you need more information about these settings, please consult the SAP documentation.

6. You are now ready to test the destination connection. In the **`SAP Connection`** view, right-click on the destination name and select **`Test`**.

7. The **`Test Destination Connection`** dialog opens. Click **`Test`**.

8. The dialog uses the current destination configuration settings to connect to the SAP server. 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.

9. Click **`Close`**, to close the **`Test Destination Connection`** dialog.

11.3. CREATE AND TEST AN SAP SERVER CONNECTION

Overview

The JBoss Fuse SAP Tools Suite provides an SAP Connections view, which enables you to create and manage SAP destination connections and SAP server connections. This section describes how to create and test an SAP server connection in the SAP Connections view.

Procedure

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

1. If you have not already added the **SAP Connections** view to your perspective, select **Window → Show View → Other** to open the **Show View** dialog. Select **SAP Connections** and click **OK**. The **SAP Connections** view opens on the left of your perspective (under **Project Explorer**).

2. In the **SAP Connections** view, right-click on **Server Data Store** and select **New Server**, to open the **Create Server** dialog.

3. Enter a name for the server connection in the **New Server Name** field and click **OK**.

4. Select the newly created server connection in the **Server Data Store** tree, so that its properties show up in the **Properties** view.

5. In the **Properties** view, click on the **Mandatory** tab to configure the basic properties required for an SAP server connection. In this tab, fill in the following property fields to configure the connection:

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

   In the **Repository Destination** field, you can fill in one of the destination names from the **Destination Data Store** tree (see Section 11.2, “Create and Test an SAP Destination Connection”). An SAP server endpoint uses this destination to retrieve meta-data from a remotely hosted meta-data repository.

   The other properties are standard SAP server connection properties. Please consult the SAP documentation for more information.

6. You are now ready to test the server connection. In the **SAP Connection** view, right-click on the server connection name and select **Test**.

7. The **Test Server Connection** dialog opens. Click **Server**.

8. The dialog uses the current server configuration settings to start up a server endpoint and connect to the gateway host. If the test is successful, you will see the following messages in the status area:

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

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

9. Click **Stop**, to shut down the test server.

10. Click **`Close`**, to close the **`Test Server Connection`** dialog.

11.4. EXPORT SAP CONNECTION CONFIGURATION TO A FILE

Overview

After creating some SAP destination connections (see Section 11.2, “Create and Test an SAP Destination Connection”) and SAP server connections (see Section 11.3, “Create and Test an SAP Server Connection”), you can export the SAP connection configuration to a file (either in Blueprint XML or Spring XML format). You will need this exported configuration data when you come to creating an SAP endpoint (the connection configuration is required in order to configure an SAP endpoint).

Procedure

To export SAP connection configurations to a file, perform the following steps:

1. In the **`SAP Connections`** view, right-click on **`Sap Connection Configuration`** and select **`Export`** to open the **`Export SAP Connection Configuration`** dialog.

2. Enter the location of the export file in the **`Export Location`** field, either directly or using the **`Browse`** button.

3. Select the **`Export File Type`** using the drop-down menu. The supported file types are: **`Blueprint XML`** or **`Spring XML`**. Select the file type that matches the file type you will use to create your Camel routes.

4. Click **`Finish`**.

11.5. CREATE A NEW SAP ENDPOINT

Overview

You can use the Components palette in the Camel editor to add SAP components to a route. But you must also 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, and have exported this configuration to a file of the appropriate type (Blueprint XML or Spring XML). See Section 11.4, “Export SAP Connection Configuration to a File”.

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 Camel editor. If you have already installed the JBoss Fuse SAP Tool Suite, you should be able to see the SAP components under the **`Components`** palette in the Camel 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`** view of the Camel 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 JBoss Fuse SAP Tool Suite. It is hosted in the Apache Camel project.

3. Switch to the **`Source`** view of the Camel editor, by clicking on the **`Source`** tab at the bottom of the canvas. In this view, you can see the XML source of the routes.

4. Open an exported SAP connection configuration file that you exported earlier (see Section 11.4, “Export SAP Connection Configuration to a File”), making sure that the type of the exported SAP connection configuration file matches the type of your Camel XML file (Blueprint XML or Spring XML).

5. The SAP connection configuration data is encoded as a **`bean`** element, with the **`id`** value, **`sap-configuration`**. Copy this **`bean`** element.

6. Return to the **`Source`** view of the Camel editor. Paste the **`bean`** element as a child of the root **`beans`** element in the Camel XML file.

7. You might not need all of the entries in the SAP connection configuration. Delete any unneeded destination connection and server connection entries (**`entry`** elements).

NOTE

Remember that server connections usually also depend on a destination connection (for retrieving meta-data from a remote repository). So, even if you only need a server connection, do not delete all of the destination connection entries.

8. 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 12. GETTING STARTED WITH DATA TRANSFORMATION

12.1. 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.

The Fuse Transformation tooling provided with JBoss Fuse is a GUI to assist developers in implementing data translations as part of Camel routes.

12.2. DATA TRANSFORMATION TUTORIAL

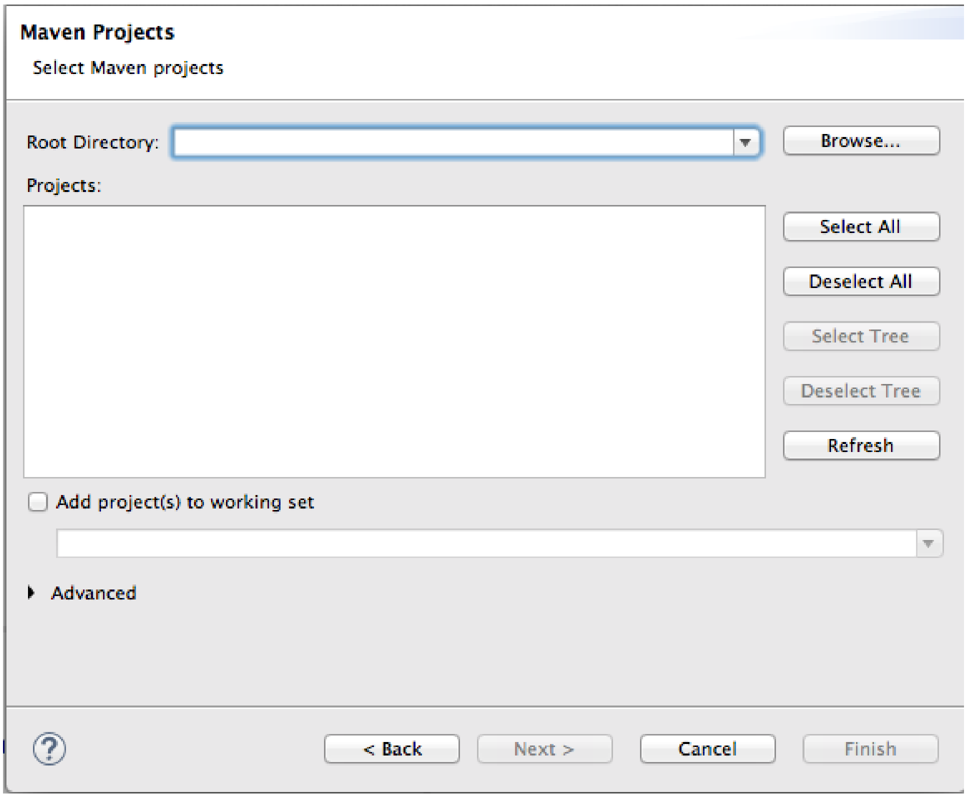
In this tutorial you will learn how to use the data transformation tooling to include data transformation in a predefined Camel route. The Camel route directs messages from a source endpoint that produces XML data to a target endpoint that consumes JSON data. You will add and define a data transformation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

Prerequisites

- JBoss Fuse tooling installed on JBoss Developer Studio. See Install Red Hat JBoss Developer Studio Integration Stack
- Maven installed and configured correctly. See Red Hat JBoss Fuse Maven Repositories
- the transformation quickstart applications downloaded and installed

Importing the `starter` quickstart application

1. Right-click in **`Project Explorer`** 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.

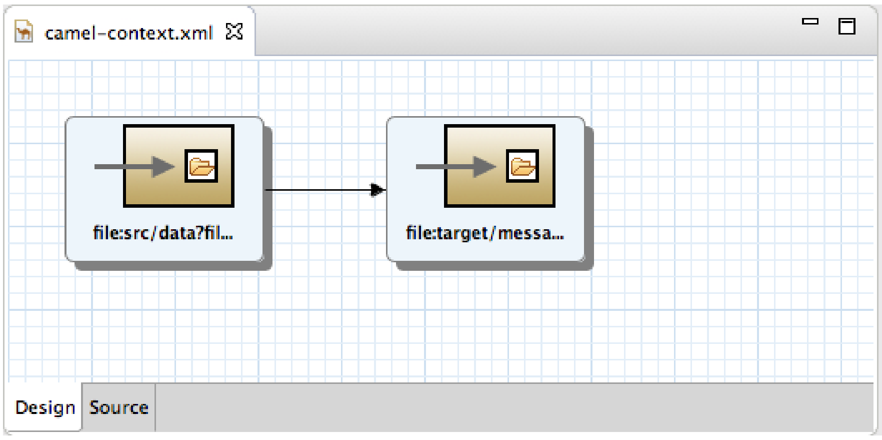
   If the browse operation finds multiple projects, make sure you select the **starter** quickstart application. The full path to the **starter** quickstart application appears in the **Projects** pane.

6. Click **Finish**.

   After the import operation finishes, the **starter** project appears in **Project Explorer**.

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

8. Double-click **starter/src/main/resources/META-INF/spring/camel-context.xml** to open it in the route editor.



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.

10. Click the **Design** tab to return to **Design** view.

11. Remove the arrow connecting the source and target endpoints.

    NOTE

    Do not save your project now. If you do, the endpoints will disappear because they are not connected, which makes the route invalid.

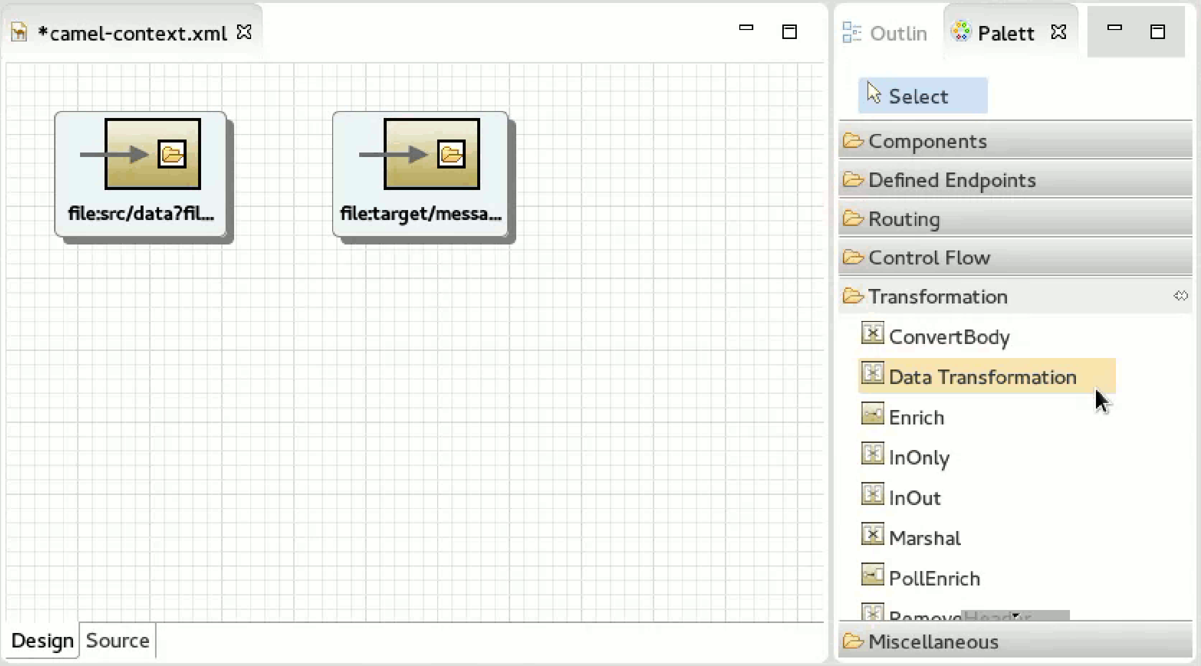
12. If **Console** view is not already open, open it now by clicking **Window → Show View → Console**.

NOTE

Although the following mini tutorials are written to be run in consecutive order, you can run through them in any order, but the Console output per tutorial will differ from that shown.

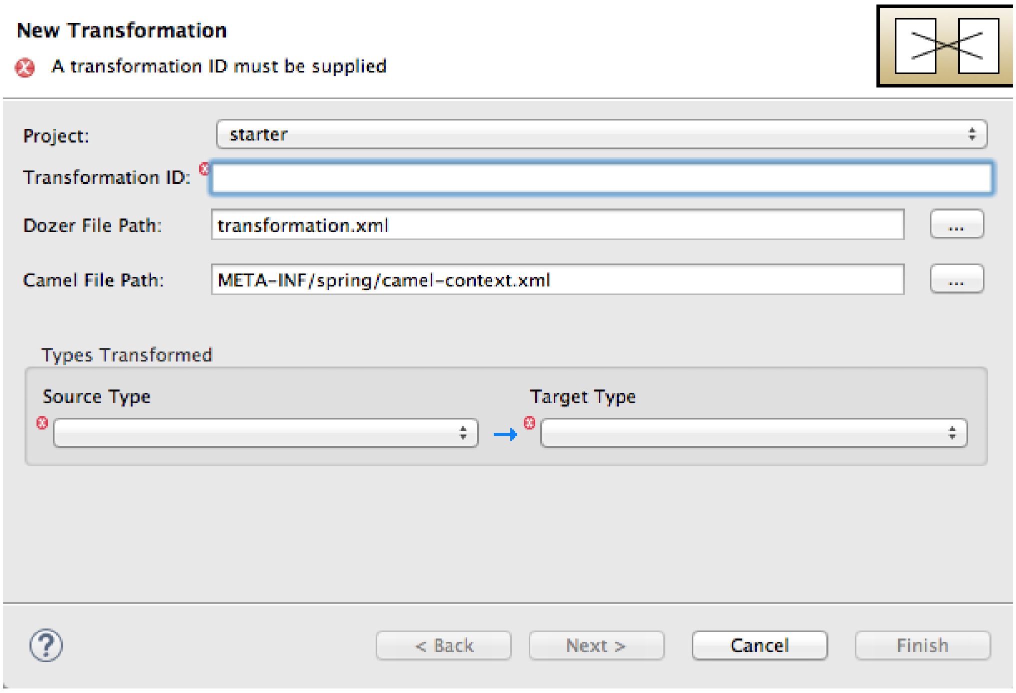
Adding a data transformation node to the Camel route

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



2. Drag a **Data Transformation** component onto the canvas.

   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

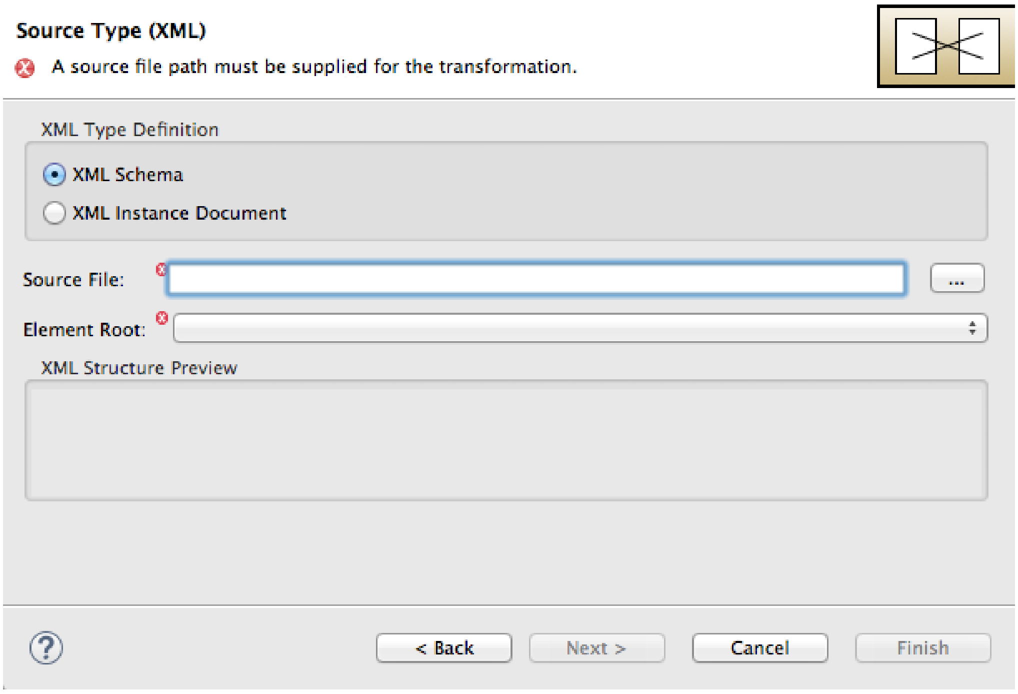
Types Transformed

Source Type

Target Type

3. Fill in the remaining fields:
    - In **`Transformation ID`**, 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.

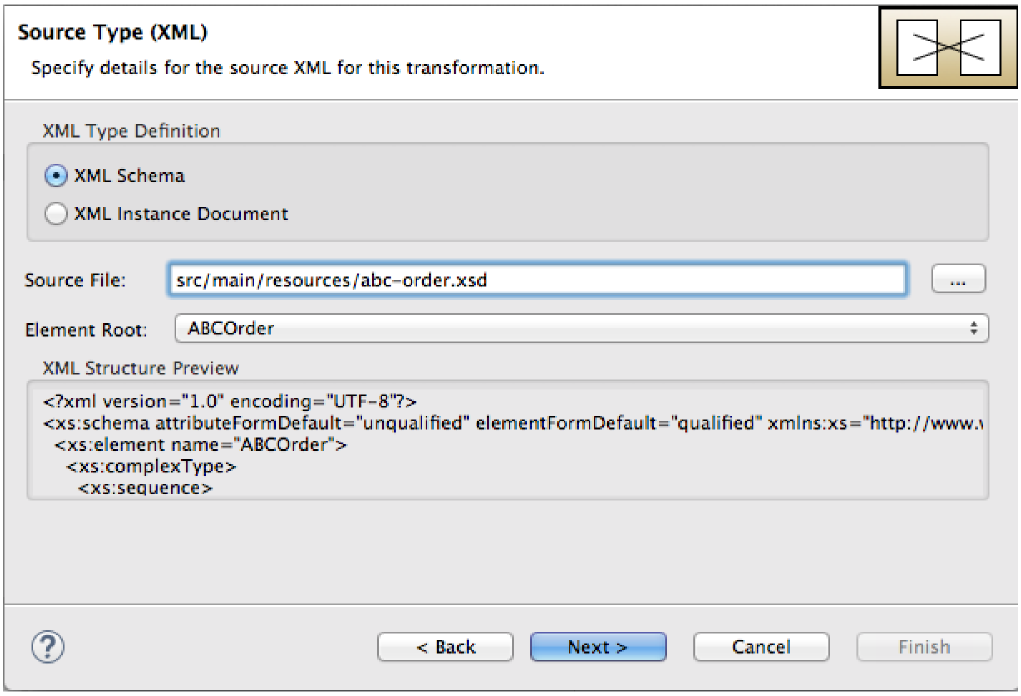
6. For **`Source file`**, browse to the location of the XML schema file or the XML instance file to use for the type definition of the source data, and select it (in this case, **`abc-order.xsd`**).

   The XML Structure Preview pane displays a preview of the XML structure.

7. In **`Element root`**, 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:



Source Type (XML)

Specify details for the source XML for this transformation.

XML Type Definition

● XML Schema

○ XML Instance Document

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

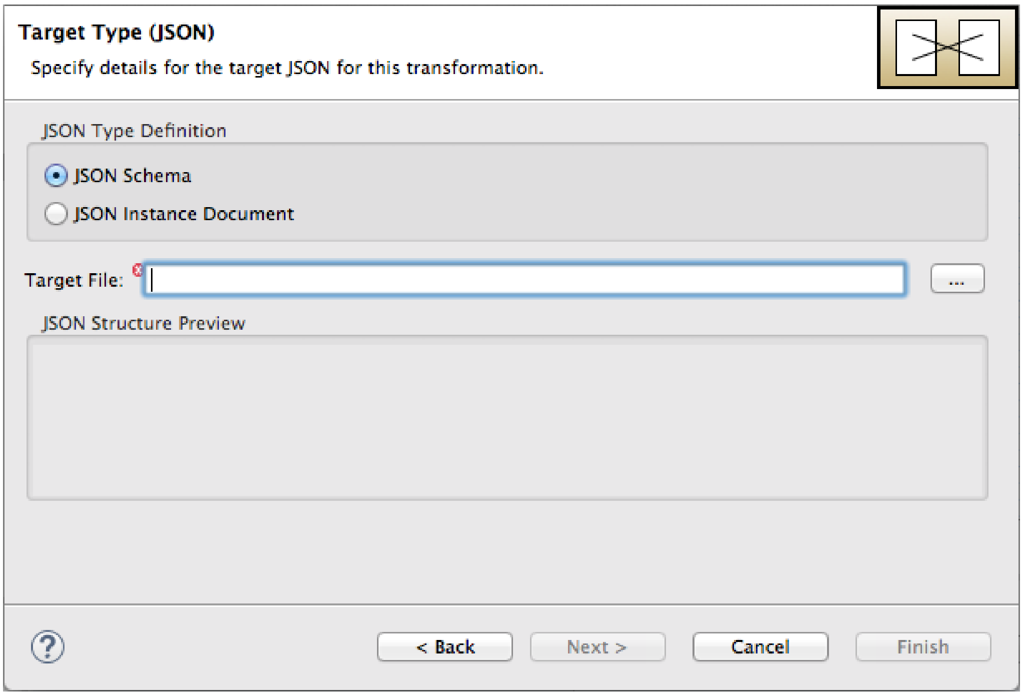
Element Root: ABCOrder

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`**.

   The **`Target Type (JSON)`** definition page opens, 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

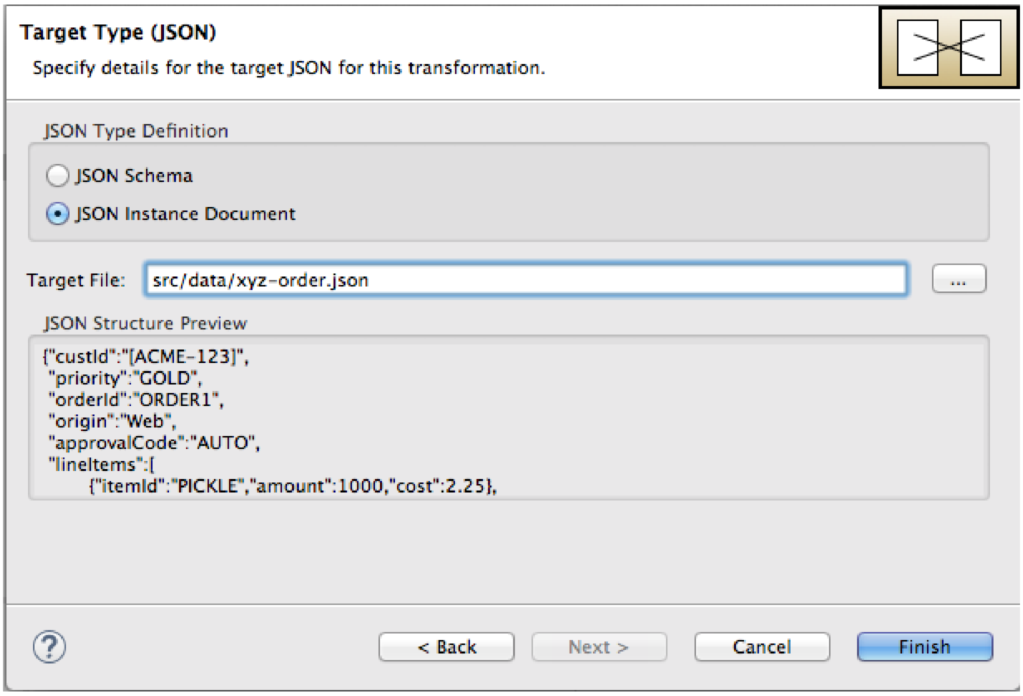
○ JSON Instance Document

Target File:

JSON Structure Preview

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

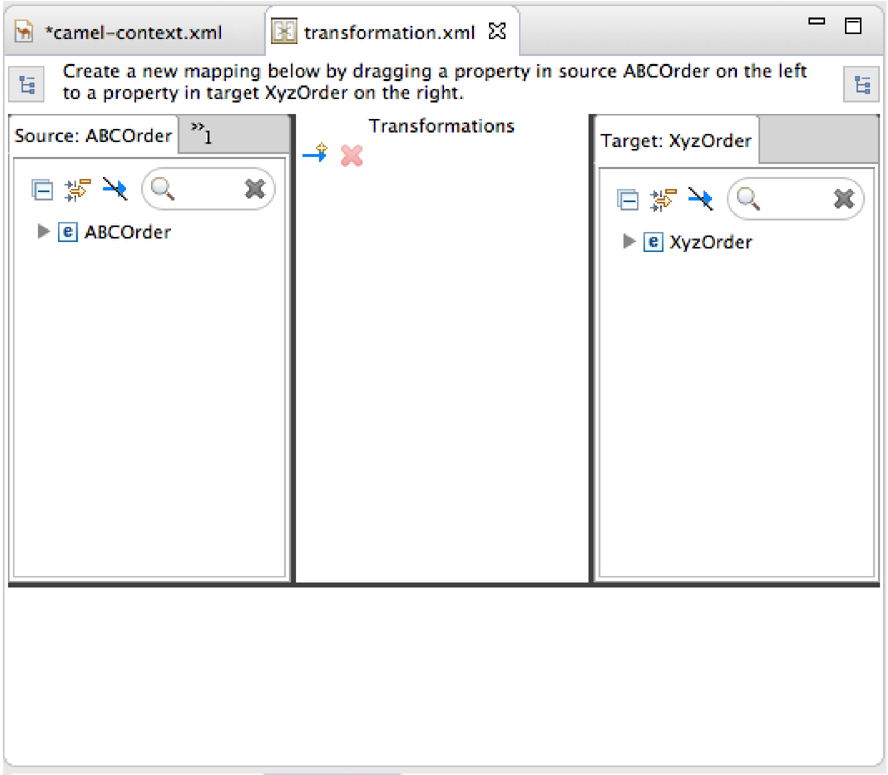
Target File: src/data/xyz-order.json

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, so you can map data items in your XML source to data items in your JSON target.

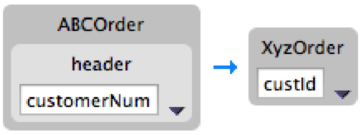


Create a new mapping below by dragging a property in source ABCOrder on the left to a property in target XyzOrder on the right.

The Transformation editor is composed of three panels:

- **`Source`**—lists the available data items of the source
- **`Transformations`**—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 view, 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:

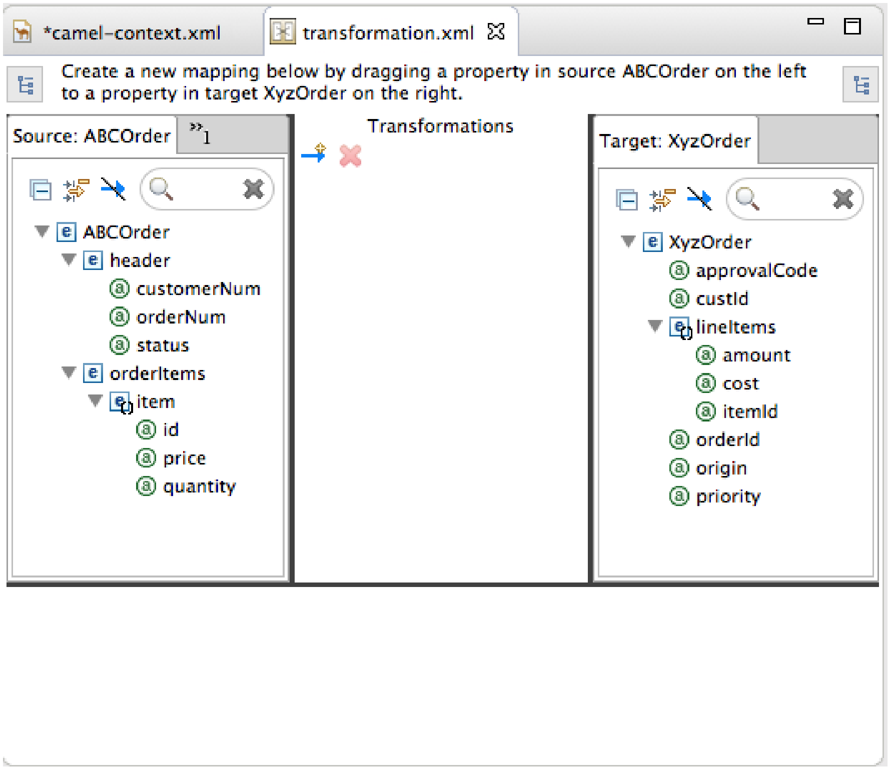


Using Details view, you can customize the mapping for the selected source and target data items:

- **`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 (see the section called “Mapping an expression to a data item”)
- **`Add function`**—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 function`**—Modify the value of a mapped data item using Java method you create or one you previously created (see the section called “Adding a custom function to a mapped data item”)

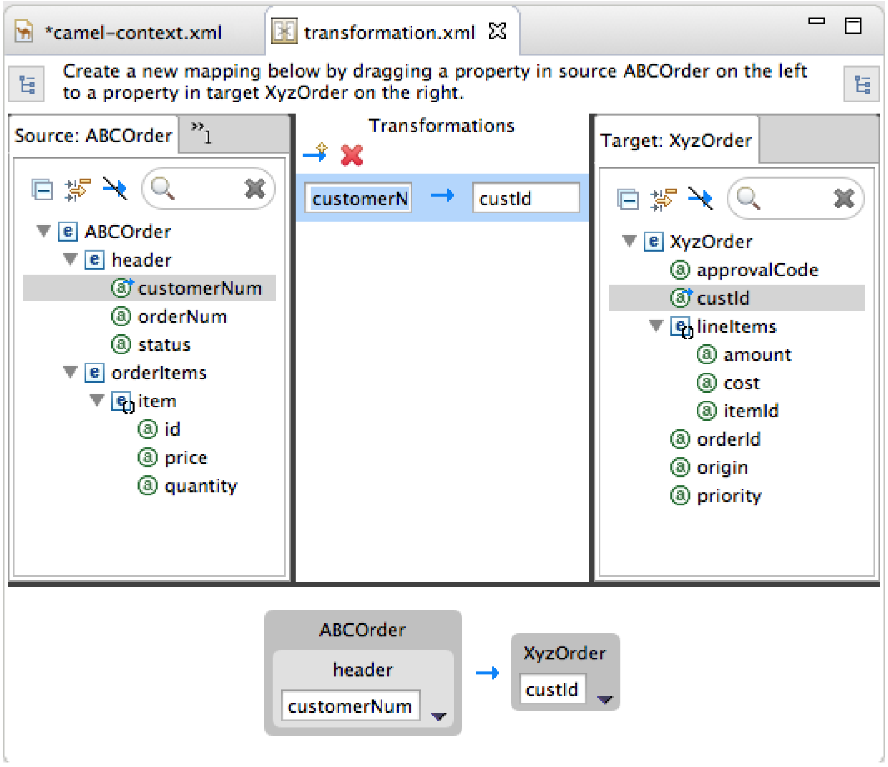
Mapping source data items to target data items

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


Create a new mapping below by dragging a property in 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.


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

   The mapping appears in the **Transformations** panel, and the details of both the **Source** and **Target** data items appear below in Details view.

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`**

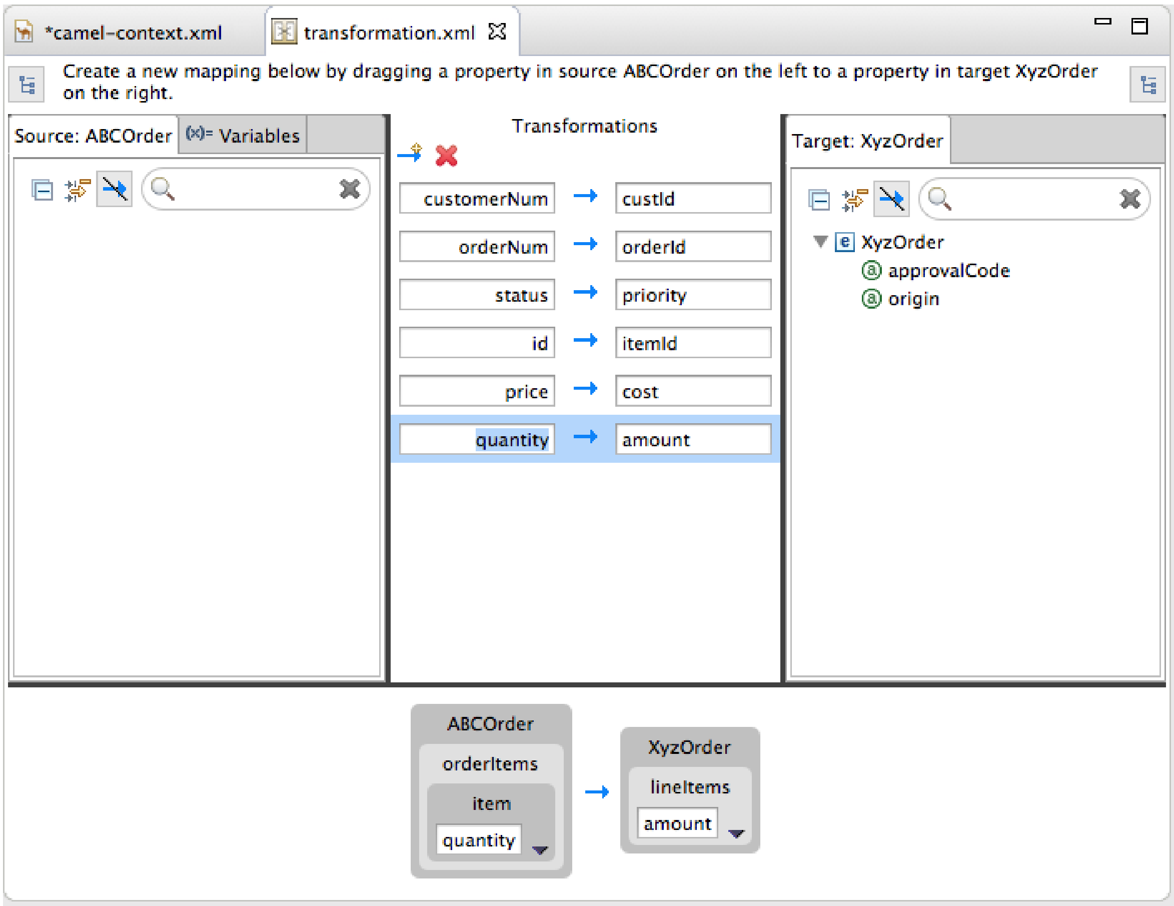
NOTE

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.

NOTE

To easily discover which data items are collections, click in both the **`Source`** and **`Target`** panels. (Brackets appear before the name of any data item that is part of a collection.) For an example, see Step 1 in the section called “Adding a custom function to a mapped data item”.

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



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

Transformations

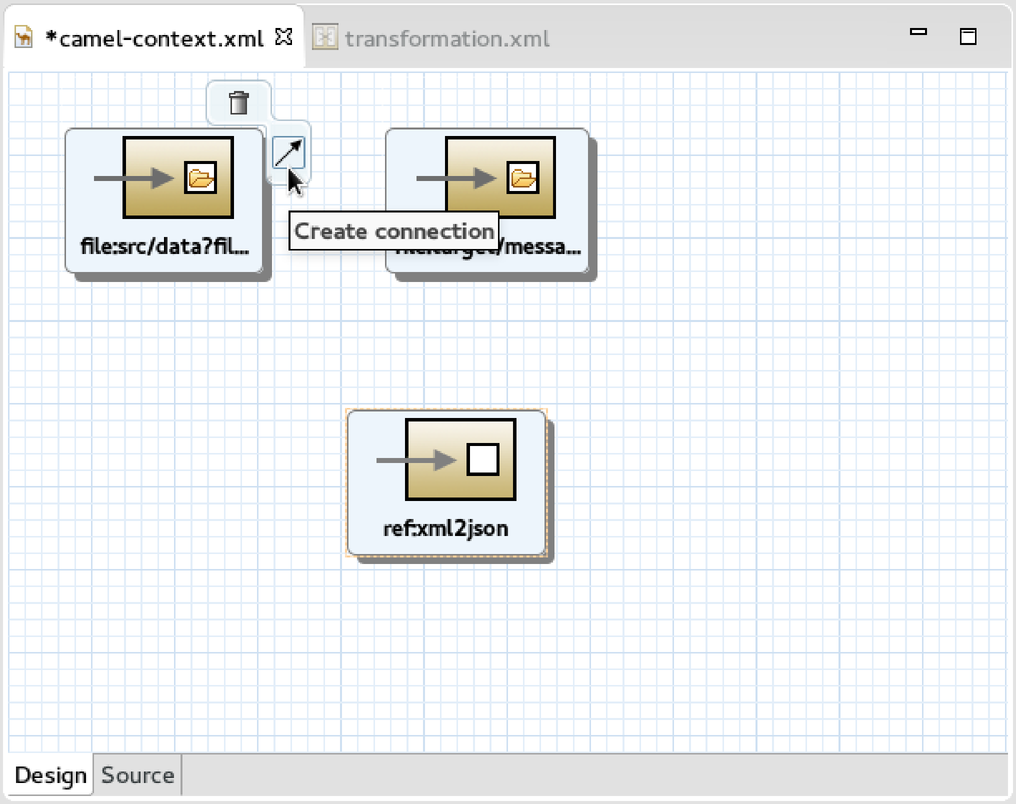
customerNum → custId
orderNum → orderId
status → priority
id → itemId
price → cost
quantity → amount

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

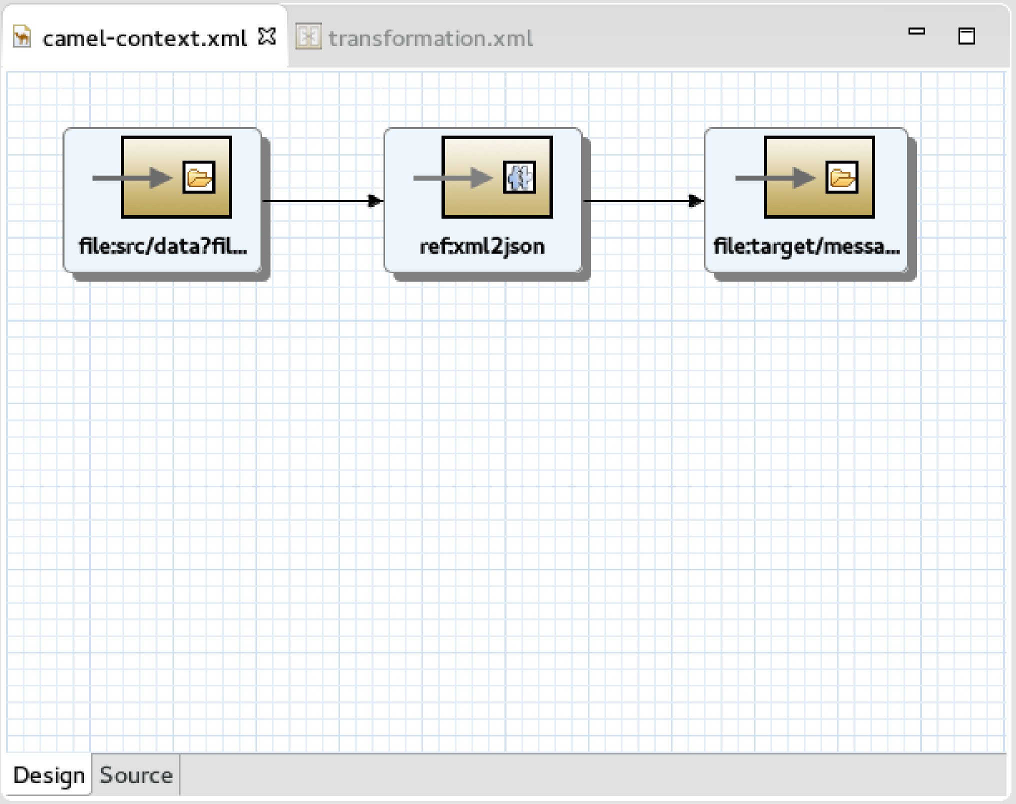
In the **starter** example, the remaining unmapped **Target** attributes are ***approvalCode*** and ***origin***.

5. Click the **camel-context.xml** tab to return to the route editor in **Design** view.

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 **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 **Project Explorer**, 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 **Project Explorer**, 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");
   ```

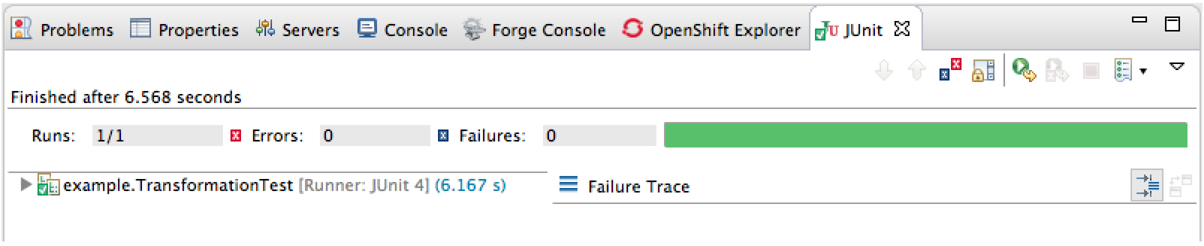
7. Click **File** → **Save**.

   You can now run a JUnit test on your transformation file at any point in these tutorials.

8. In **`Project Explorer`**, 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 **`Console`** view.

   Finished after 6.568 seconds

   Runs: 1/1 Errors: 0 Failures: 0

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

   Failure Trace

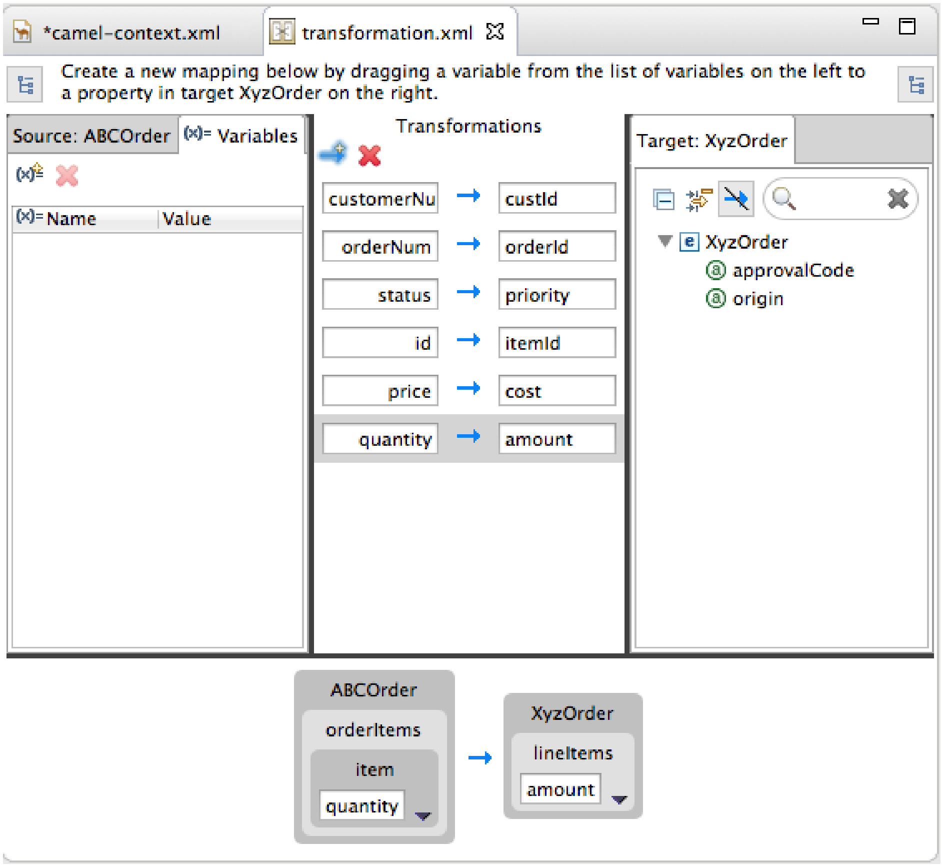
10. Open **`Console`** view to see the log output.

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.

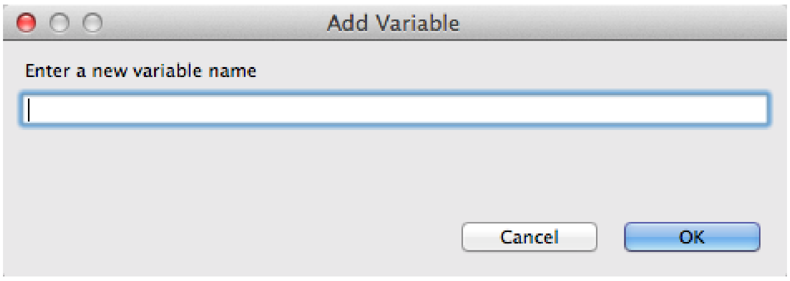
In the **`starter`** example, the target data item ***`origin`*** does not have a corresponding source data item. To map the ***`origin`*** attribute to a constant variable:

1. In the **`Source`** panel, click the **`Variables`** tab to open **`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.

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



Enter a new variable name

3. Enter a name for the variable you want to create.

   For the **starter** example, enter **ORIGIN**.

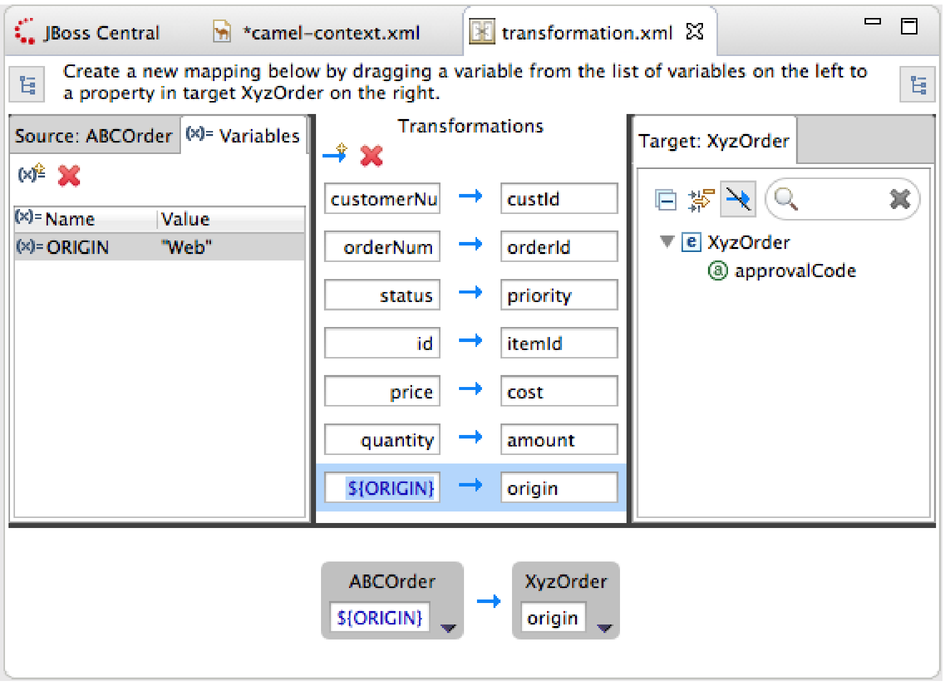
4. Click **OK**.

   The newly created variable ***ORIGIN*** appears in **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.



The new mapping of the variable **`$(ORIGIN)`** appears in the **`Transformations`** panel and in Details view.

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”.

   Console view displays the json-formatted output data:

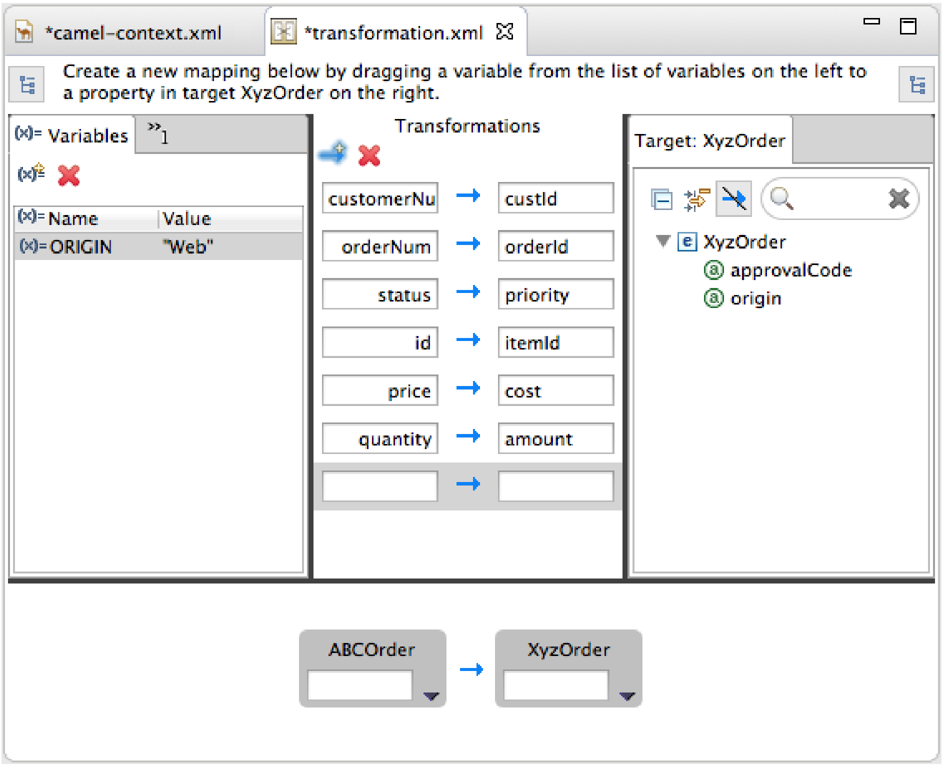
```
{"custId":"ACME-
123","priority":"GOLD","orderId":"ORDER1","origin":"Web",
"approvalCode":"AUTO_OK","lineItems":
[{"itemId":"PICKLE","amount":1000,"cost":2.25},
{"itemId":"BANANA","amount":400,"cost":1.25}]}
```

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 **`Transformations`** panel.

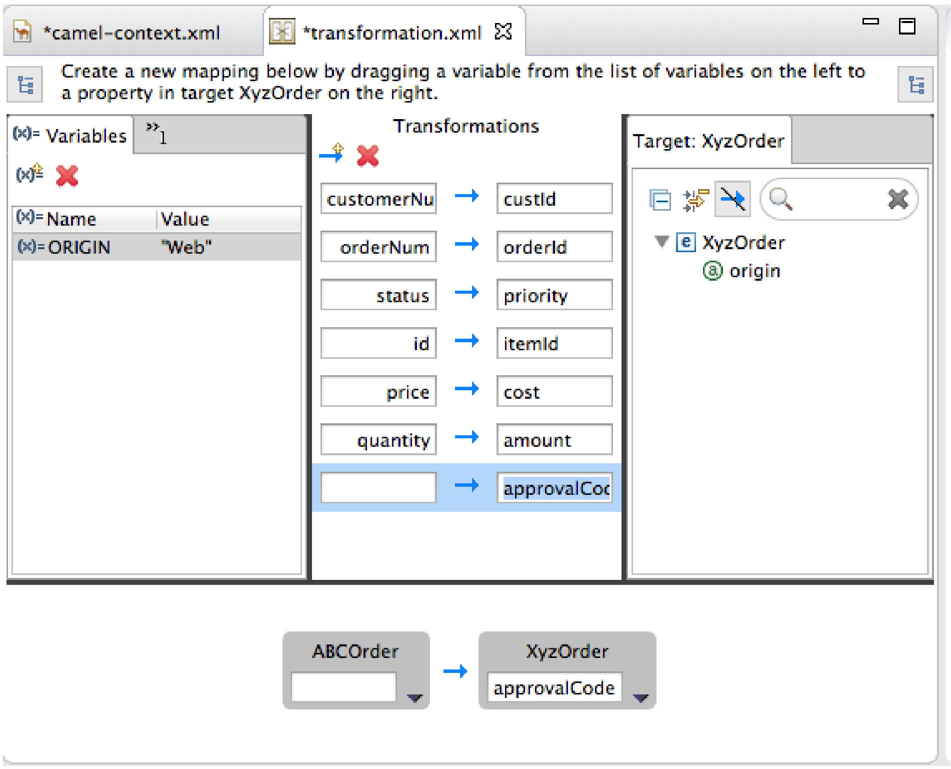


Create a new mapping below by dragging a variable from the list of variables on the left to a property in target XyzOrder on the right.

Name	Value
ORIGIN	"Web"

Transformations	
customerNu	custId
orderNum	orderId
status	priority
id	itemId
price	cost
quantity	amount

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



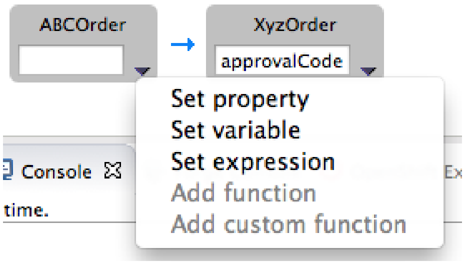
Create a new mapping below by dragging a variable from the list of variables on the left to a property in target XyzOrder on the right.

Name	Value
ORIGIN	"Web"

Transformations	
customerNu	custId
orderNum	orderId
status	priority
id	itemId
price	cost
quantity	amount
	approvalCoc

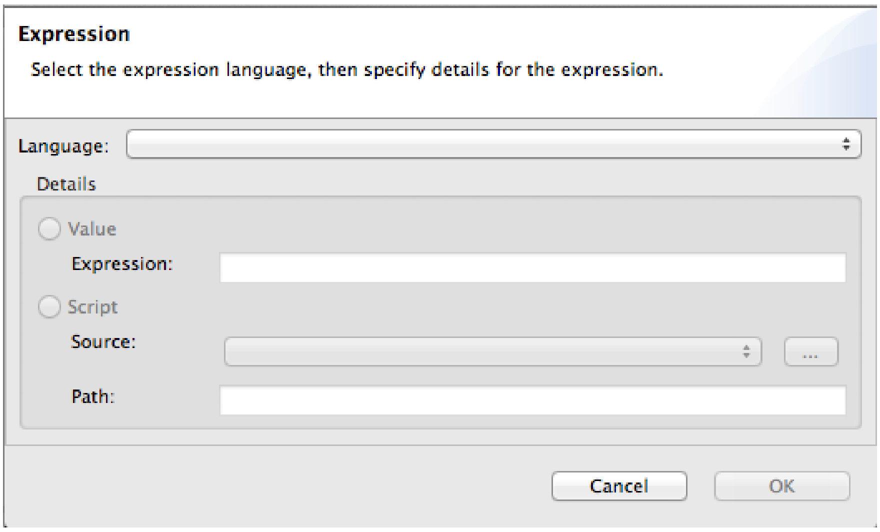
The approvalCode data item also appears in Details view's target box.

3. In Details view, 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.



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.

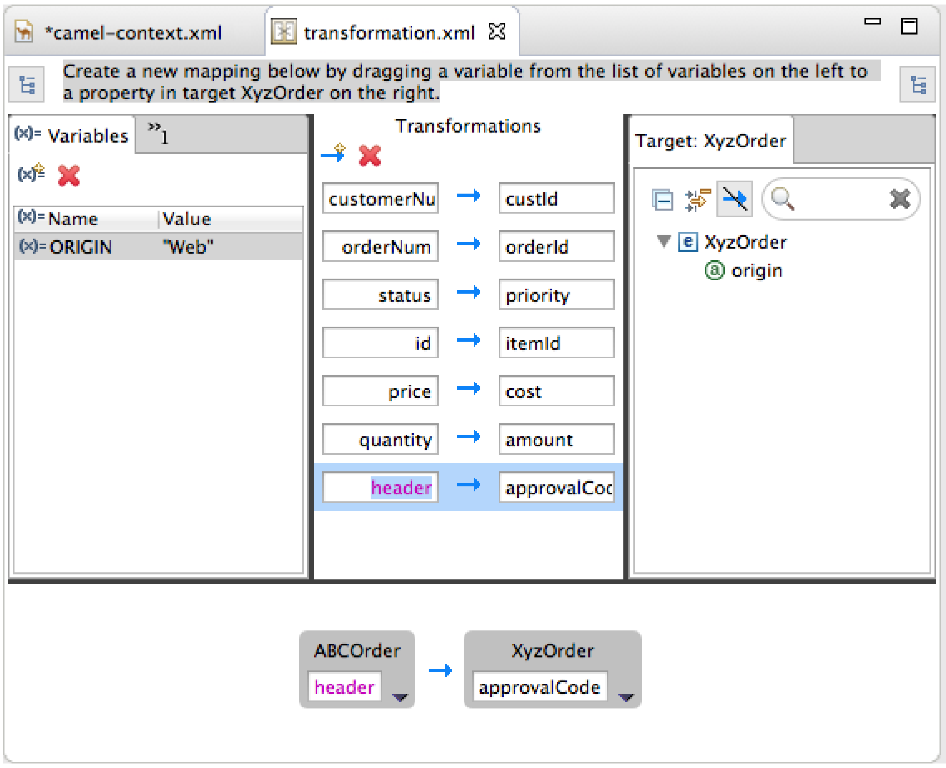
   For the **`starter`** example, select **Header**.

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`**.



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"

customerNu → custId
orderNum → orderId
status → priority
id → itemId
price → cost
quantity → amount
header → approvalCoc

Both **Transformations** panel and Details view 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”.

   Console view displays the json-formatted output data:

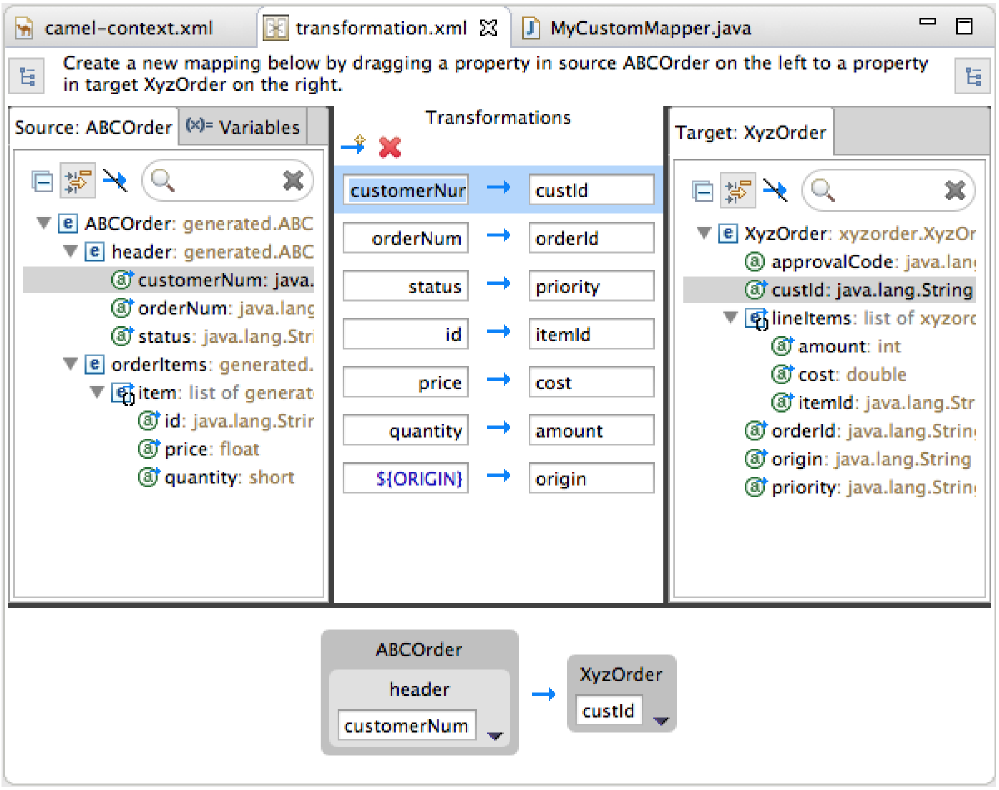
   ```
   {"custId":"ACME-
   123","priority":"GOLD","orderId":"ORDER1","origin":"Web",
   "approvalCode":"AUTO_OK","lineItems":
   [{"itemId":"PICKLE","amount":1000,"cost":2.25},
   {"itemId":"BANANA","amount":400,"cost":1.25}]}
   ```

Adding a custom function 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.

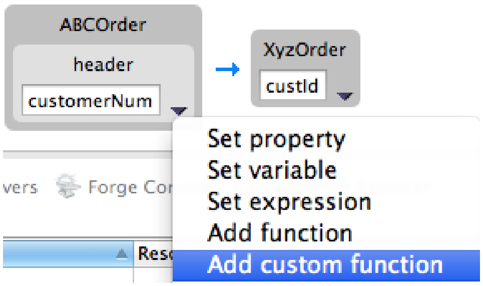
For example, to satisfy the target system's requirement that all customer ids be enclosed in brackets:

1. **Transformations** panel, select the customerNum mapping to populate Details view.

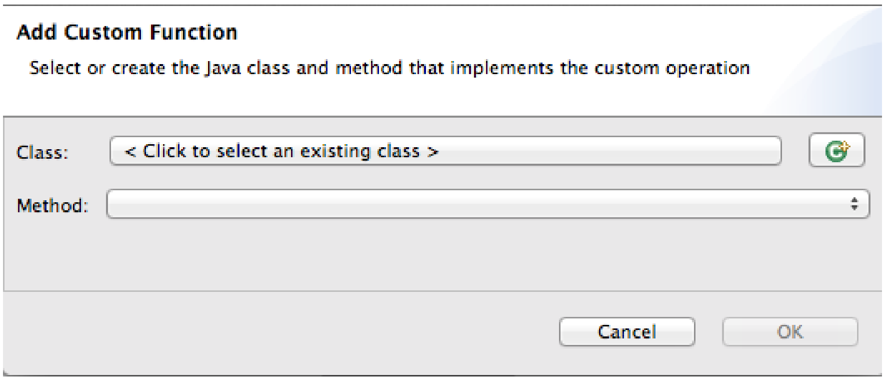


Create a new mapping below by dragging a property in source ABCOrder on the left to a property in target XyzOrder on the right.

2. In Details view, click ▼ on the **ABCOrder** source box to open the drop-down menu.



3. Select **Add custom function** to open the **Add Custom Function** page.



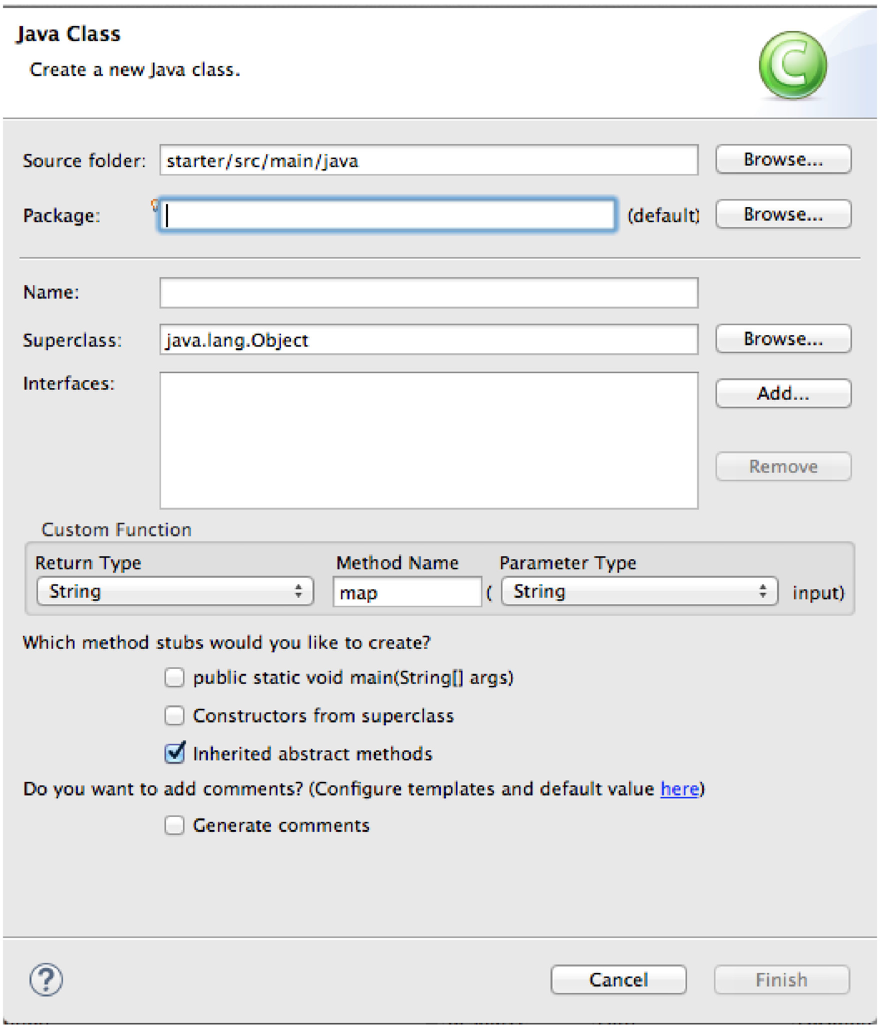
Add Custom Function

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

Class: < Click to select an existing class >

Method:

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



Java Class

Create a new Java class.

Source folder: starter/src/main/java

Package: (default)

Name:

Superclass: java.lang.Object

Interfaces:

Custom Function

Return Type: String

Method Name: map (Parameter Type: 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

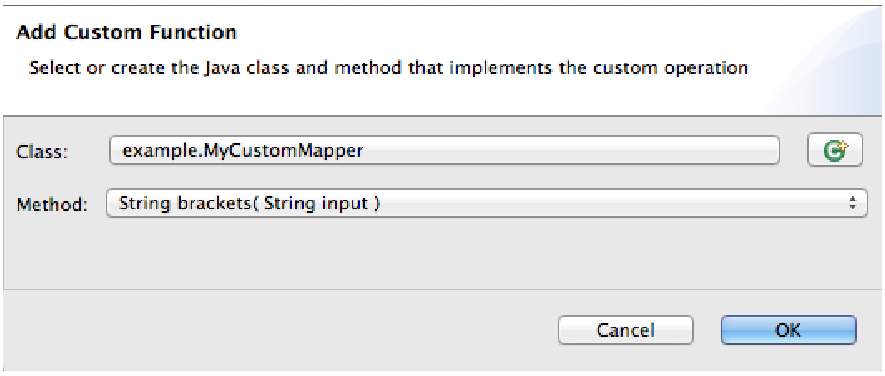
5. Modify the following fields:

   - **Package**—Enter **example**.
   - **Name**—Enter **MyCustomMapper**.
   - **Method Name**—Change map to **brackets**.

   Leave all other fields as is.

6. Click **Finish**.

   The **Add Custom Function** page opens with the **Class** and **Method** fields auto filled:



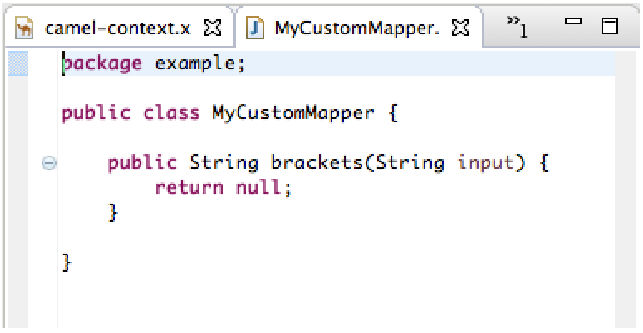
Add Custom Function

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

Class: example.MyCustomMapper

Method: String brackets(String input)

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

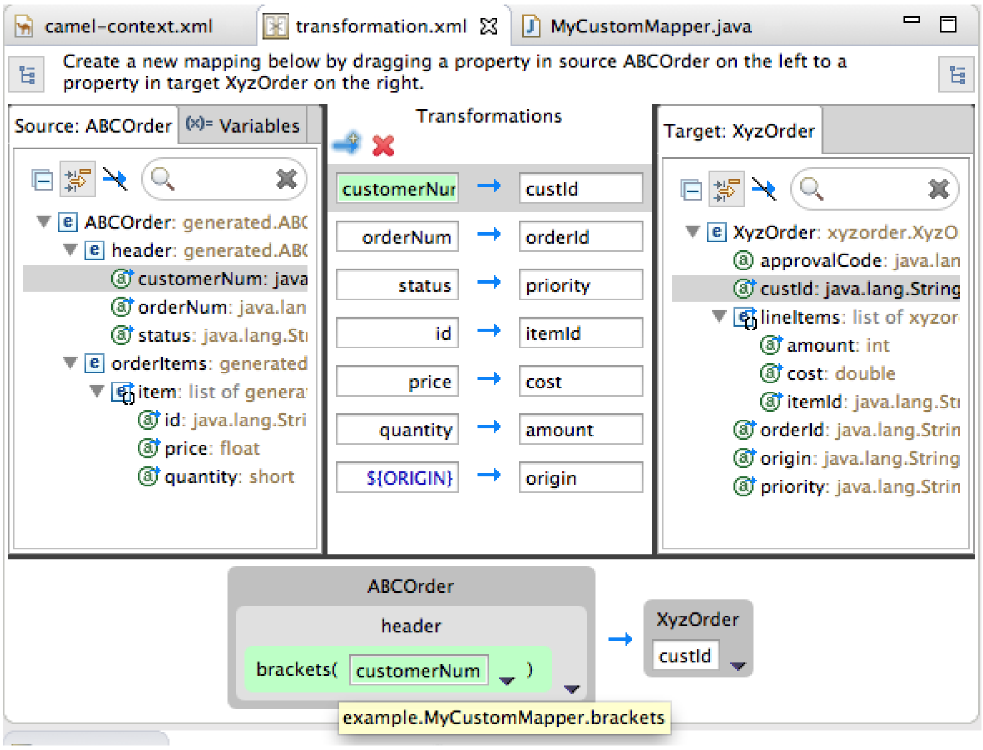


```
package example;

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.



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

Details view 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”.

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 ids in the Source to the itemIds 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.

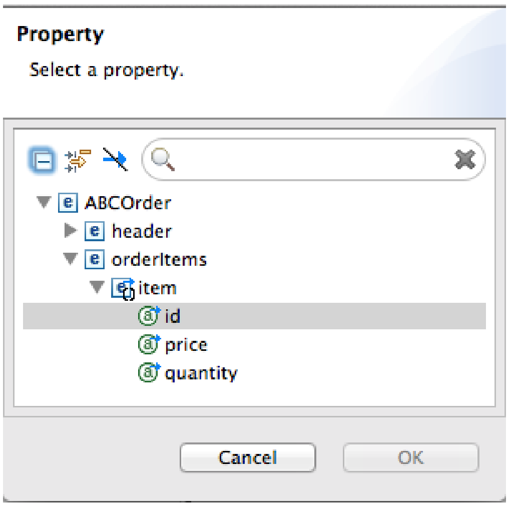
With this change, no ids in the Source will be mapped to itemIds in the Target.

NOTE

To discover which data items are in collections, click ⇄ at the top of both the **`Source`** and **`Target`** panels. The data type of each item is displayed next to it.

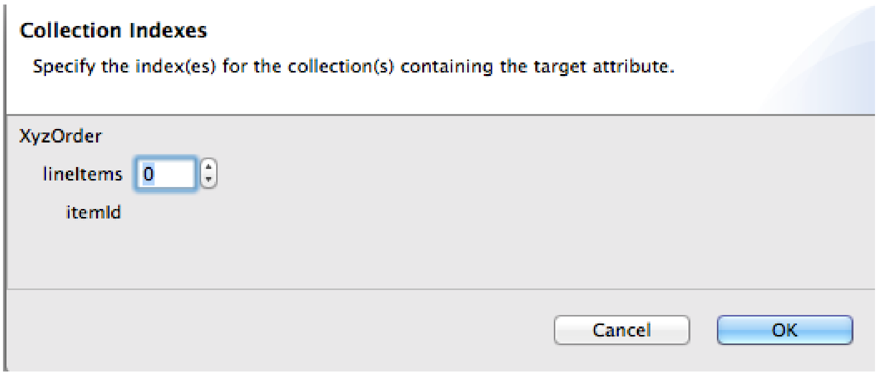
1. In the **`Transformations`** panel, select the mapping id —> itemId to display the mapping in Details view.

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.

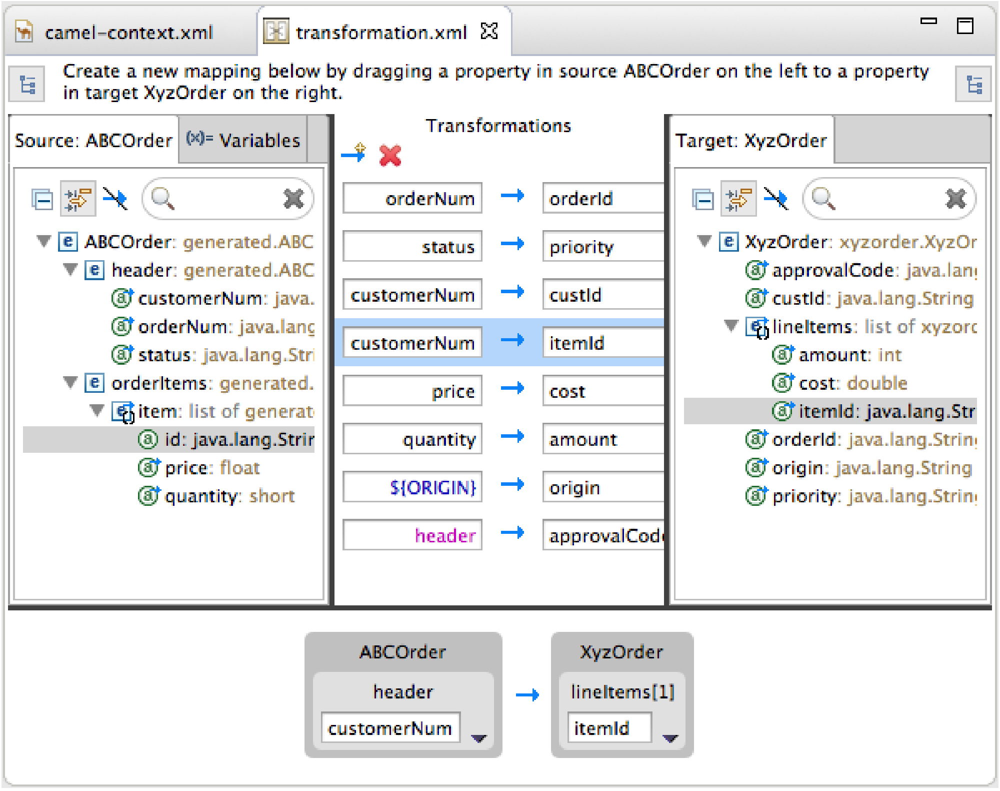
4. Click **`OK`** to save the change and open the **`Collection Indexes`** page.


Collection Indexes
Specify the index(es) for the collection(s) containing the target attribute.
XyzOrder
lineItems
0
itemId
Cancel
OK

5. In the **`Collection Indexes`** page, click the toggle button next to lineItems to increase its value to **`1`**.

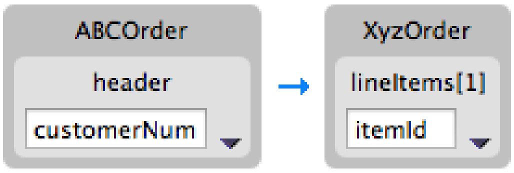
   Indexes are zero-based, so a value of **`1`** selects the second instance of itemId in the collection.

6. Click **`OK`** to save the change and open the **`transformation.xml`** file.



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

Notice that Details view shows customerNum mapped to the itemId of the second item in the lineItems collection.



7. 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”.

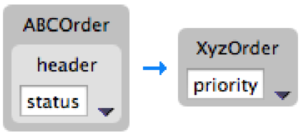
   Console view displays the json-formatted output data:

```
{"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}]}
```

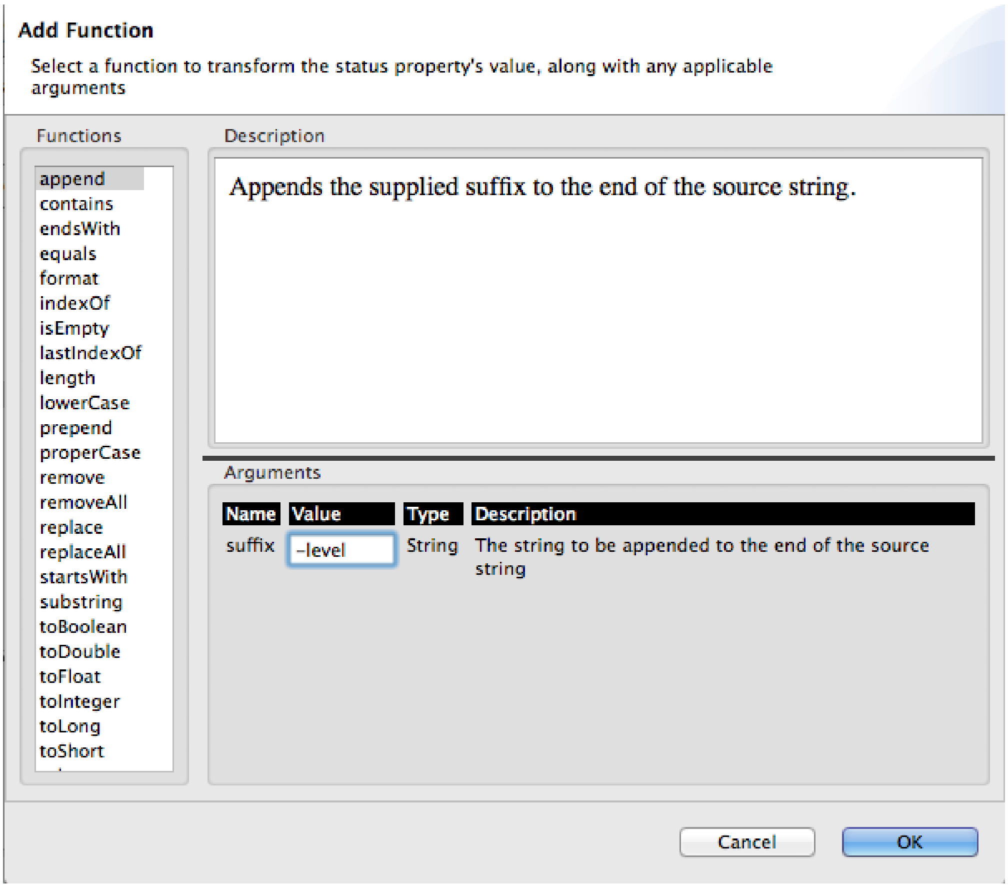
Adding a built-in function to a mapped data item

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 Details view.



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



Add Function

Select a function to transform the status property's value, along with any applicable arguments

Functions

append
contains
endsWith
equals
format
indexOf
isEmpty
lastIndexOf
length
lowerCase
prepend
properCase
remove
removeAll
replace
replaceAll
startsWith
substring
toBoolean
toDouble
toFloat
toInteger
toLong
toShort

Description

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

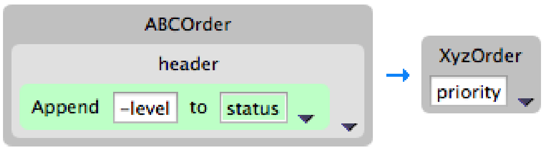
Arguments

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

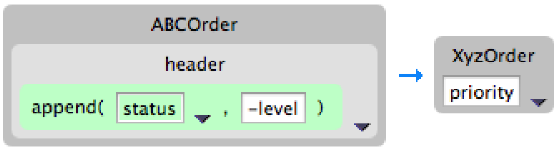
3. In the **Functions** pane, select **append**, and in the **Arguments** pane, enter **-level** for the value of ***suffix***.

   This **append** function adds the specified suffix to the end of thestatus string before mapping it to the target priority data item.

4. Click **OK**.



By default, Details view 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”.

   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}]}
```

CHAPTER 13. USING THE SWITCHYARD TOOLING

13.1. JBOSS INTEGRATION AND SOA DEVELOPMENT

JBoss Integration and SOA Development

The **JBoss SwitchYard Tooling** for JBoss Developer Studio supports integration and SOA development by providing these features:

- Creation of SwitchYard projects
- Configuration of SwitchYard capabilities
- Editing SwitchYard application configuration using a graphical editor
- Java2WSDL transformation
- XML catalog entries for SwitchYard configuration schema
- Support for workspace deployment of SwitchYard projects

The **JBoss SwitchYard Tooling** plug-in is provided by the JBoss Development Studio Integration Stack.

Installing JBoss Developer Studio Integration Stack

JBoss Developer Studio Integration Stack (JBDSIS) is packaged as a plug-in for JBoss Developer Studio installations. It contains numerous, individual plug-ins, including the Fuse and SwitchYard tooling. You can download JBDSIS through JBoss Central, and then install all of the plug-ins it contains or only a select few. For more details, see Product Documentation for Red Hat JBoss Developer Studio Integration Stack.

Once installed, you may need to complete additional configuration tasks before you can use the individual JBoss Developer Studio Integration Stack components.

Here are some general tips to follow:

- Honor all XML schema locations

  After installation, go to **XML** → **XML Files** → **Validation** → **Preferences** and disable **`Honor all XML schema locations`**. This prevents erroneous XML validation errors from appearing on **`switchyard.xml`** files.

- DTD warning

  Importing SwitchYard quickstarts into JBoss Developer Studio results in non-fatal warnings regarding **`log4j.dtd`**. You can safely ignore them. To stop receiving the warning, make sure that either the **`log4j.xml`** or **`log4j.properties`** file is present before starting a project.

- JavaSE-1.6 error message

  When commencing a project, the warning "Build path specifies execution environment JavaSE-1.6" may appear. To disable this warning, go to your Java preferences and ensure that JavaSE-1.7 or JavaSE-1.8 is checked to support JavaSE-1.6 environments.

13.2. RUNNING A SWITCHYARD PROJECT ON A JBOSS FUSE SERVER

To publish and run Fuse SwitchYard projects on an installed Fuse server, you must first add the server and its runtime definition to the tooling's **`Servers`** list. Once added to the list, the server appears in the **`Servers`** view, where you can configure it, start and stop it, and publish projects to it.

You can publish SwitchYard projects to JBoss Fuse 6.2.0 and newer servers that you have defined and added to the Servers list.

NOTE

If you haven't already installed a server, you can do so when you define and add it to the Servers list.

Adding a JBoss Fuse server

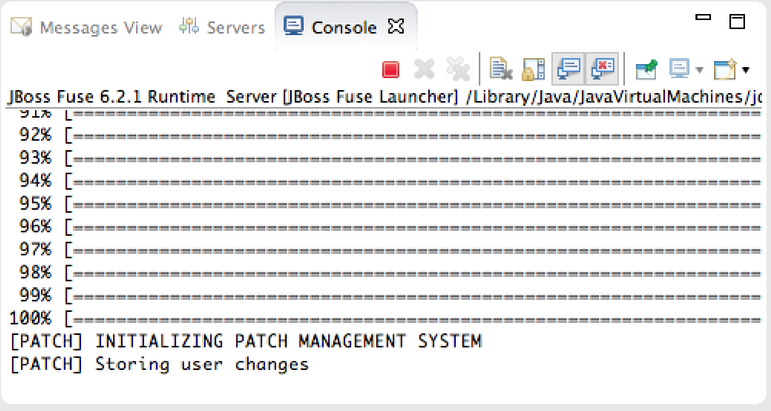
Follow the instructions in Section 28.1, “Adding a Server” to define and add an installed JBoss Fuse 6.2.1 server.

Installing the SwitchYard features on the server

JBoss Fuse is SwitchYard-ready, but you have to install the switchyard features.

1. In the **`Servers`** view, select the JBoss Fuse server you just added, and click ▶ on the view's menu bar to start it up.

   `Console` view opens and displays the server's startup progress:

   JBoss Fuse 6.2.1 Runtime Server [JBoss Fuse Launcher] /Library/Java/JavaVirtualMachines/jc

   ```
    91% [==============================================================
    92% [==============================================================
    93% [==============================================================
    94% [==============================================================
    95% [==============================================================
    96% [==============================================================
    97% [==============================================================
    98% [==============================================================
    99% [==============================================================
   100% [==============================================================
   [PATCH] INITIALIZING PATCH MANAGEMENT SYSTEM
   [PATCH] Storing user changes
   ```

   Wait until **`[PATCH] Storing user changes`** appears at the end of the output.

2. Switch to **`Shell`** view, and install the **`switchyard-*`** features, one after another, on the server:

   ```
   JBossFuse:admin@root> features:install switchyard-bean
   JBossFuse:admin@root> features:install switchyard-camel
   JBossFuse:admin@root> features:install switchyard-soap
   ```

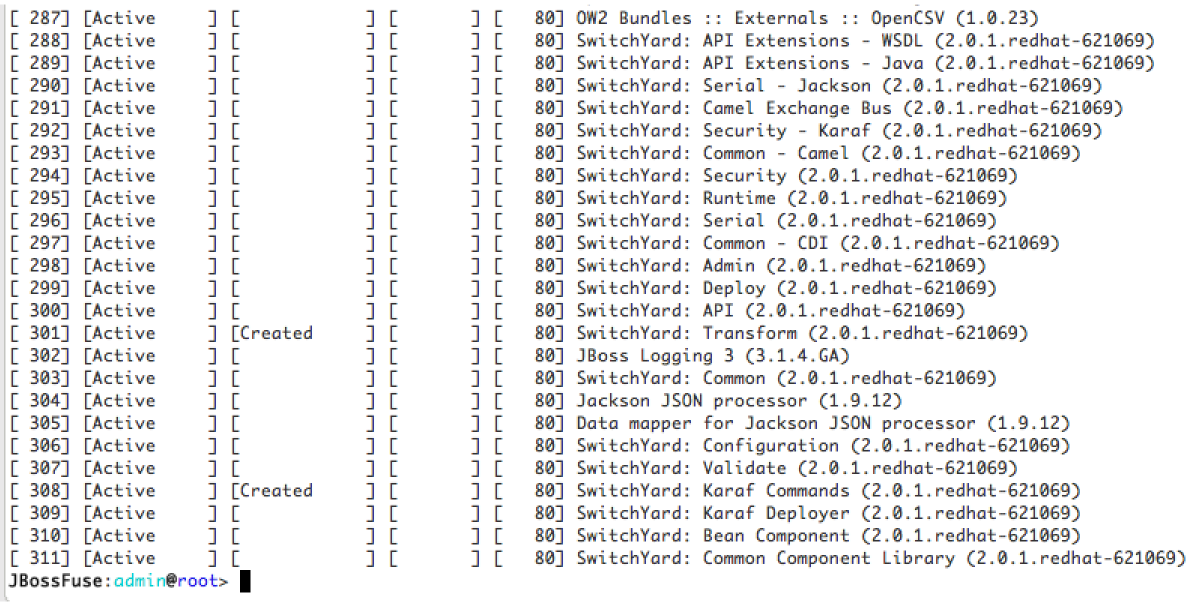
   ### NOTE

   To see a list of all available SwitchYard-related features, enter **`features:list | grep switchyard*`**.

3. Check that the **switchyard-*** features are installed and active:

```
JBossFuse:admin@root> osgi:list
```

The end of the output should show the switchyard components installed and active :

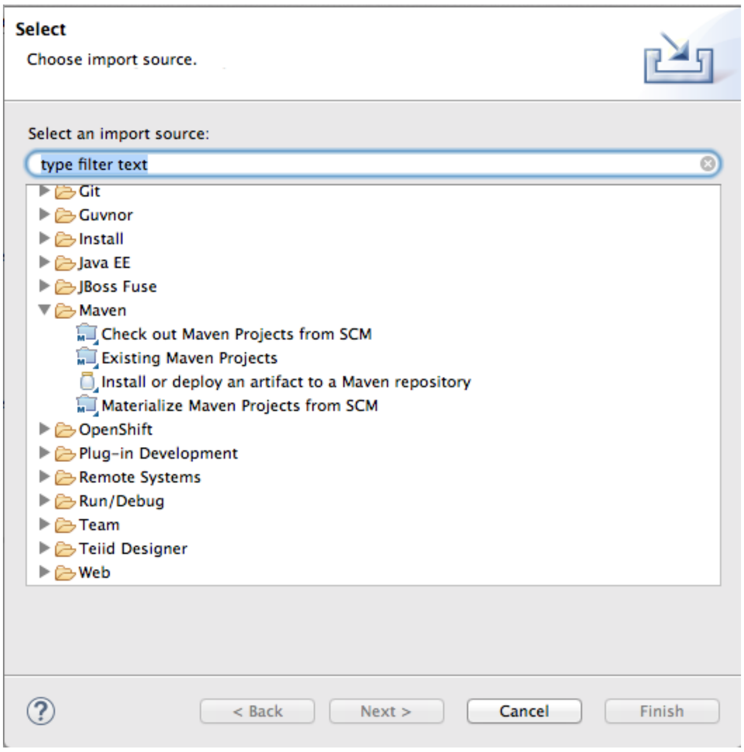


```
[ 287] [Active     ] [            ] [       ] [   80] OW2 Bundles :: Externals :: OpenCSV (1.0.23)
[ 288] [Active     ] [            ] [       ] [   80] SwitchYard: API Extensions - WSDL (2.0.1.redhat-621069)
[ 289] [Active     ] [            ] [       ] [   80] SwitchYard: API Extensions - Java (2.0.1.redhat-621069)
[ 290] [Active     ] [            ] [       ] [   80] SwitchYard: Serial - Jackson (2.0.1.redhat-621069)
[ 291] [Active     ] [            ] [       ] [   80] SwitchYard: Camel Exchange Bus (2.0.1.redhat-621069)
[ 292] [Active     ] [            ] [       ] [   80] SwitchYard: Security - Karaf (2.0.1.redhat-621069)
[ 293] [Active     ] [            ] [       ] [   80] SwitchYard: Common - Camel (2.0.1.redhat-621069)
[ 294] [Active     ] [            ] [       ] [   80] SwitchYard: Security (2.0.1.redhat-621069)
[ 295] [Active     ] [            ] [       ] [   80] SwitchYard: Runtime (2.0.1.redhat-621069)
[ 296] [Active     ] [            ] [       ] [   80] SwitchYard: Serial (2.0.1.redhat-621069)
[ 297] [Active     ] [            ] [       ] [   80] SwitchYard: Common - CDI (2.0.1.redhat-621069)
[ 298] [Active     ] [            ] [       ] [   80] SwitchYard: Admin (2.0.1.redhat-621069)
[ 299] [Active     ] [            ] [       ] [   80] SwitchYard: Deploy (2.0.1.redhat-621069)
[ 300] [Active     ] [            ] [       ] [   80] SwitchYard: API (2.0.1.redhat-621069)
[ 301] [Active     ] [Created     ] [       ] [   80] SwitchYard: Transform (2.0.1.redhat-621069)
[ 302] [Active     ] [            ] [       ] [   80] JBoss Logging 3 (3.1.4.GA)
[ 303] [Active     ] [            ] [       ] [   80] SwitchYard: Common (2.0.1.redhat-621069)
[ 304] [Active     ] [            ] [       ] [   80] Jackson JSON processor (1.9.12)
[ 305] [Active     ] [            ] [       ] [   80] Data mapper for Jackson JSON processor (1.9.12)
[ 306] [Active     ] [            ] [       ] [   80] SwitchYard: Configuration (2.0.1.redhat-621069)
[ 307] [Active     ] [            ] [       ] [   80] SwitchYard: Validate (2.0.1.redhat-621069)
[ 308] [Active     ] [Created     ] [       ] [   80] SwitchYard: Karaf Commands (2.0.1.redhat-621069)
[ 309] [Active     ] [            ] [       ] [   80] SwitchYard: Karaf Deployer (2.0.1.redhat-621069)
[ 310] [Active     ] [            ] [       ] [   80] SwitchYard: Bean Component (2.0.1.redhat-621069)
[ 311] [Active     ] [            ] [       ] [   80] SwitchYard: Common Component Library (2.0.1.redhat-621069)
JBossFuse:admin@root>
```

Importing a SwitchYard quickstart project

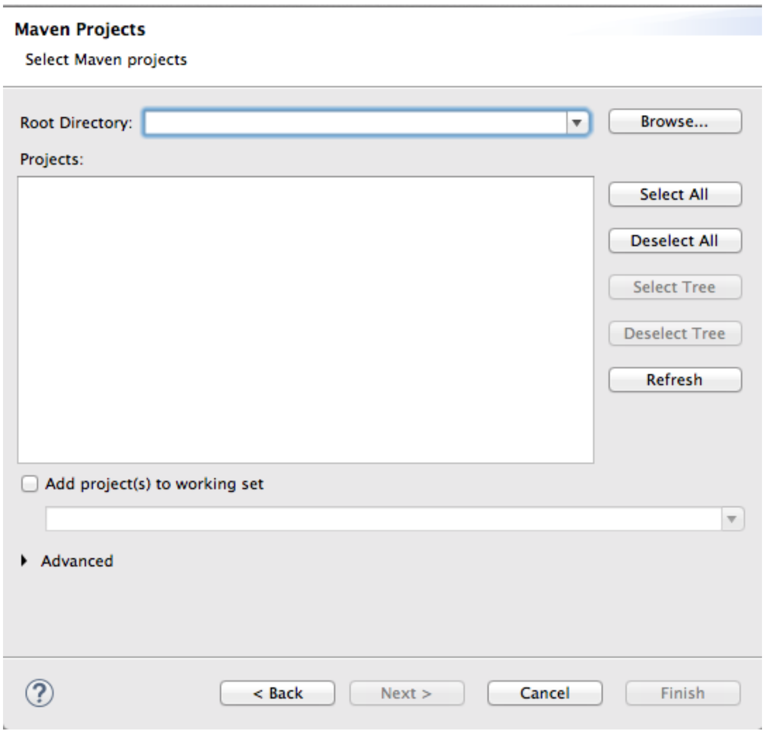
The SwitchYard quickstarts are installed in *$FUSE_HOME/***quickstarts/switchyard/**.

1. Right-click in **Project Explorer**, and select **Import** → **Import...** to open the **Choose Import source** dialog.

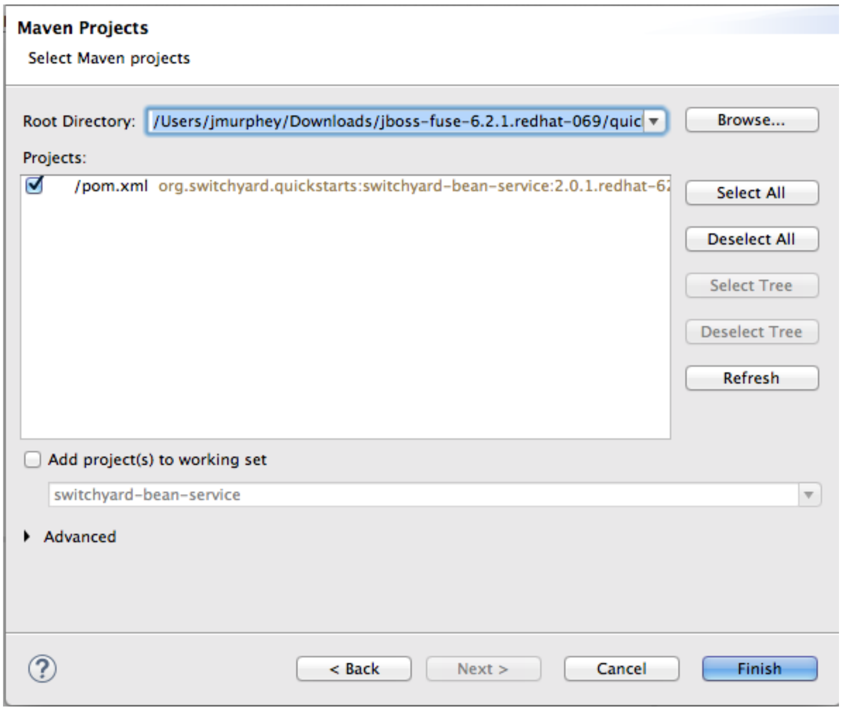


2. In the **Maven** folder, select **Existing Maven Projects**.

3. Click **Next** to open the **Select Maven projects** dialog.



4. Click the **Browse** button next to the **Root Directory** field to locate and select the ***$FUSE_HOME*/quickstarts/switchyard/bean-service**.



Root Directory: /Users/jmurphey/Downloads/jboss-fuse-6.2.1.redhat-069/quic

Projects:

☑ /pom.xml org.switchyard.quickstarts:switchyard-bean-service:2.0.1.redhat-62

switchyard-bean-service

5. In the **Projects** pane, make sure the box next to the **pom.xml** file entry for the **switchyard-bean-service** is checked to select it.

6. Click **Finish** to start the import.

   The **switchyard-bean-service** project appears in **Project Explorer**.

   Wait a moment for the import process to finish.

NOTE

See the section called “Resolving Maven dependency errors” for details on what to do if you encounter Maven dependency errors, caused by failure to download all of the project's dependencies from the Maven repositories.

7. In **`Project Explorer`**, right-click the **`switchyard-bean-service`** project to open the context menu, and select **Enable Fuse Camel Nature**.

   This option enables you to publish a switchyard project to the JBoss Fuse server, which you can do at any time now (for details, see the section called “Publishing a SwitchYard project to the server”).

Testing the SwitchYard quickstart project locally

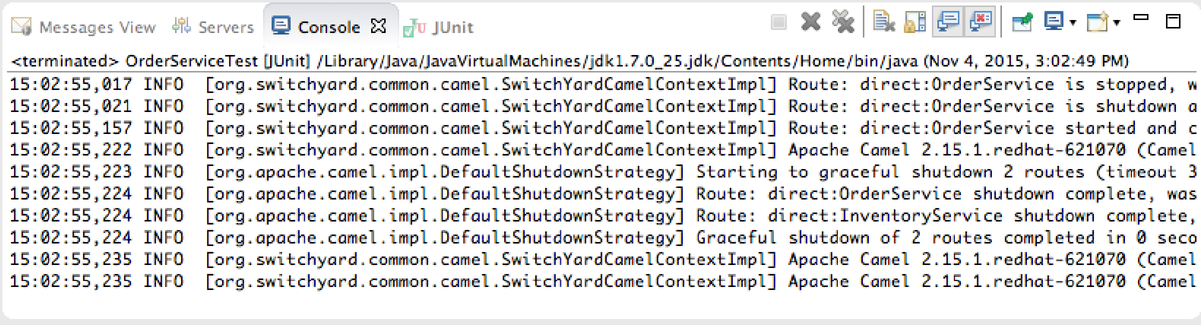
It's a good idea to run projects as a JUnit test to check that they successfully build and run locally. The provided test files included with each SwitchYard quickstart are complete and comprehensive and need no modification.

1. In **`Project Explorer`**, expand the **`switchyard-bean-service`** project to expose **`src/test/java/OrderServiceTest.java`** file.

   If you want to take a look at the test code, double-click the file to open it in the Java editor.

2. In **`Project Explorer`**, right-click the **`OrderServiceTest.java`** file to open the context menu, and select **Run As** → **JUnit Test**.

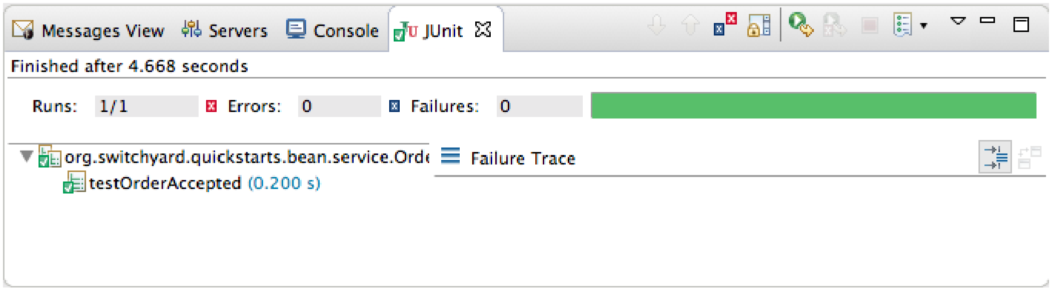
   `Console` view automatically opens and displays the log entries generated by the JUnit test.



```
<terminated> OrderServiceTest [JUnit] /Library/Java/JavaVirtualMachines/jdk1.7.0_25.jdk/Contents/Home/bin/java (Nov 4, 2015, 3:02:49 PM)
15:02:55,017 INFO  [org.switchyard.common.camel.SwitchYardCamelContextImpl] Route: direct:OrderService is stopped, w
15:02:55,021 INFO  [org.switchyard.common.camel.SwitchYardCamelContextImpl] Route: direct:OrderService is shutdown a
15:02:55,157 INFO  [org.switchyard.common.camel.SwitchYardCamelContextImpl] Route: direct:OrderService started and c
15:02:55,222 INFO  [org.switchyard.common.camel.SwitchYardCamelContextImpl] Apache Camel 2.15.1.redhat-621070 (Camel
15:02:55,223 INFO  [org.apache.camel.impl.DefaultShutdownStrategy] Starting to graceful shutdown 2 routes (timeout 3
15:02:55,224 INFO  [org.apache.camel.impl.DefaultShutdownStrategy] Route: direct:OrderService shutdown complete, was
15:02:55,224 INFO  [org.apache.camel.impl.DefaultShutdownStrategy] Route: direct:InventoryService shutdown complete,
15:02:55,224 INFO  [org.apache.camel.impl.DefaultShutdownStrategy] Graceful shutdown of 2 routes completed in 0 seco
15:02:55,235 INFO  [org.switchyard.common.camel.SwitchYardCamelContextImpl] Apache Camel 2.15.1.redhat-621070 (Camel
15:02:55,235 INFO  [org.switchyard.common.camel.SwitchYardCamelContextImpl] Apache Camel 2.15.1.redhat-621070 (Camel
```

   The last two log entries indicate that the test ran successfully against Apache Camel as expected.

3. Click the **`JUnit`** tab to open **`JUnit`** view.



Finished after 4.668 seconds

Runs: 1/1 Errors: 0 Failures: 0

org.switchyard.quickstarts.bean.service.Orde
testOrderAccepted (0.200 s)

This test proves that the **`switchyard-bean-service`** successfully builds and runs locally.

NOTE

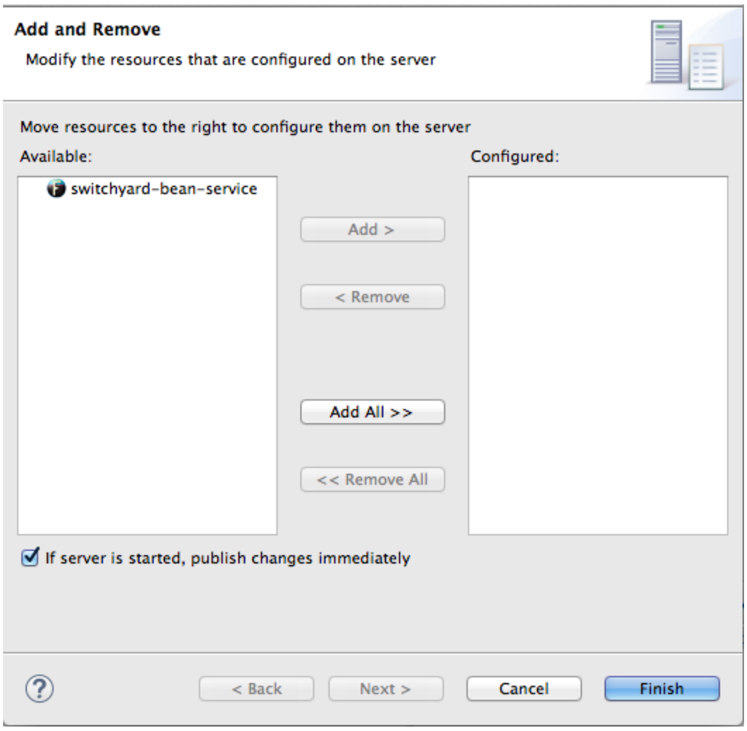
If you failed to set the **`Execution environment`** to JavaSE-1.7 or JavaSE-1.8 when you defined the JBoss Fuse server, this test will fail. You can easily reset it by right-clicking the **`JRE System Library`** node in **`Project Explorer`**, and selecting **Build Path** → **Configure Build Path**. In the **`Java Build Path`** dialog's **`Library`** tab, edit the current library to change the **`Execution environment`** setting to JavaSE-1.7 or JavaSE-1.8. (The library you select must be installed on your machine.) Then rerun the test.

Publishing a SwitchYard project to the server

You can publish a project to a supported server, defined and listed in the **`Servers`** view, whether it's running or not. A published project will run as scheduled according to the server's settings (for details, see Chapter 29, *Publishing Fuse Projects to a Server*).

1. In the **`Servers`** view, right-click the target server to open the context menu, and then select **Add and Remove**.

   When a project is ready for publishing, it appears in the **`Available`** column. For example:



Add and Remove

Modify the resources that are configured on the server

Move resources to the right to configure them on the server

Available:

switchyard-bean-service

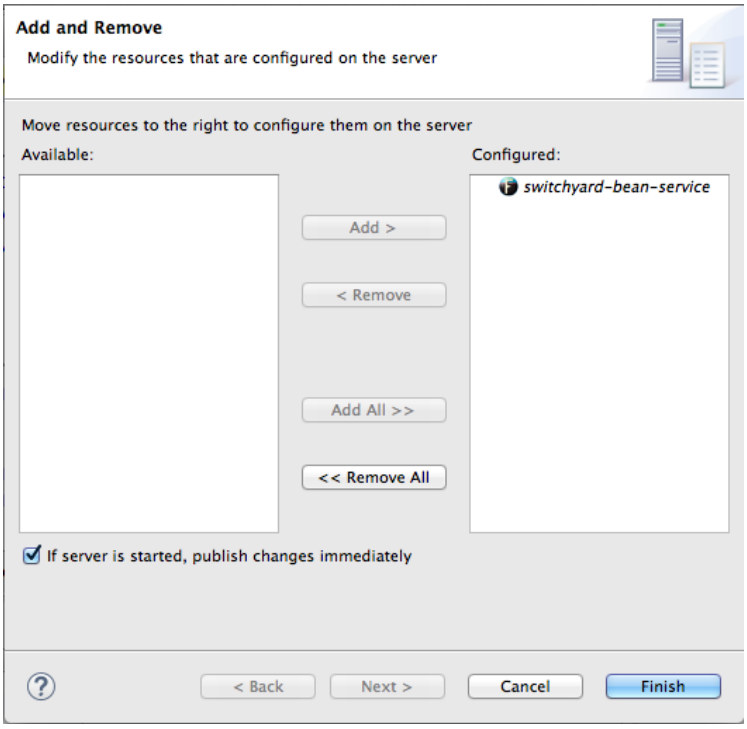
Configured:

☑ If server is started, publish changes immediately

NOTE

The option **`If server is started, publish changes immediately`** is enable by default. See the section called “Publishing Fuse projects automatically when resources change” for information on how this option works and on using other publishing options.

2. Double-click the project in the **`Available`** column to move it 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:

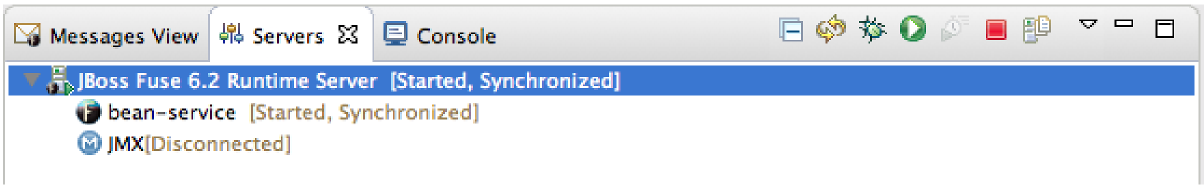
Configured:

switchyard-bean-service

☑ If server is started, publish changes immediately

3. Click **Finish**.

   Once publishing has finished, the project appears as a node under the server runtime node in the **Servers** view.



JBoss Fuse 6.2 Runtime Server [Started, Synchronized]

bean-service [Started, Synchronized]

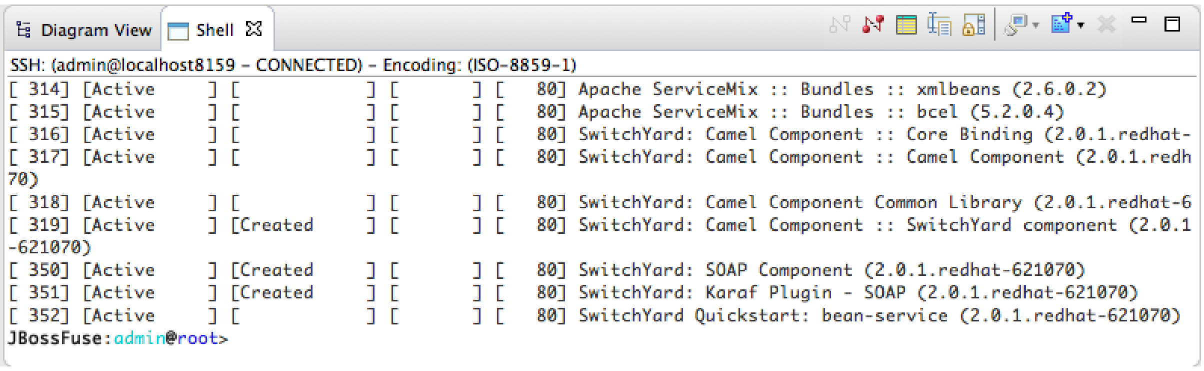
JMX[Disconnected]

   Servers view shows that both the server runtime and the project are started and synchronized.

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.

4. In **Shell** view, enter the command **osgi:list** to confirm that the **switchyard-bean-service** project is installed and active.

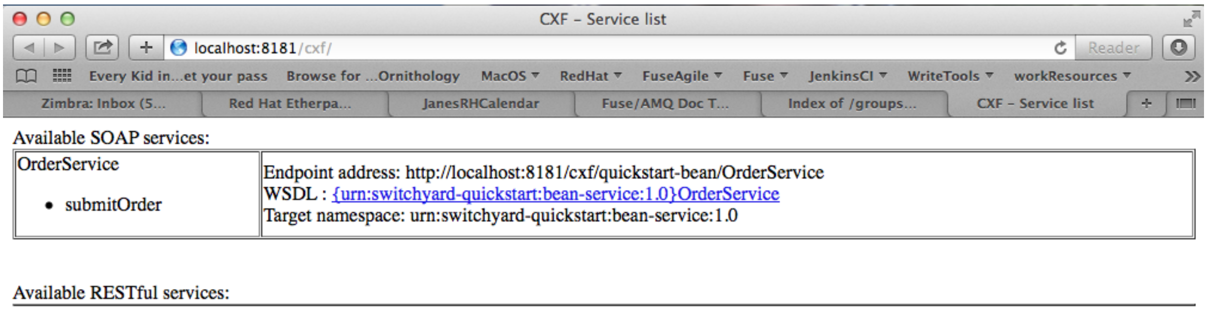


```
SSH: (admin@localhost8159 - CONNECTED) - Encoding: (ISO-8859-1)
[ 314] [Active     ] [            ] [       ] [   80] Apache ServiceMix :: Bundles :: xmlbeans (2.6.0.2)
[ 315] [Active     ] [            ] [       ] [   80] Apache ServiceMix :: Bundles :: bcel (5.2.0.4)
[ 316] [Active     ] [            ] [       ] [   80] SwitchYard: Camel Component :: Core Binding (2.0.1.redhat-
[ 317] [Active     ] [            ] [       ] [   80] SwitchYard: Camel Component :: Camel Component (2.0.1.redh
70)
[ 318] [Active     ] [            ] [       ] [   80] SwitchYard: Camel Component Common Library (2.0.1.redhat-6
[ 319] [Active     ] [Created     ] [       ] [   80] SwitchYard: Camel Component :: SwitchYard component (2.0.1
-621070)
[ 350] [Active     ] [Created     ] [       ] [   80] SwitchYard: SOAP Component (2.0.1.redhat-621070)
[ 351] [Active     ] [Created     ] [       ] [   80] SwitchYard: Karaf Plugin - SOAP (2.0.1.redhat-621070)
[ 352] [Active     ] [            ] [       ] [   80] SwitchYard Quickstart: bean-service (2.0.1.redhat-621070)
JBossFuse:admin@root>
```

Testing the published SwitchYard project

You can use the JBoss Web Service Tester tool to test your published web service.

1. Open a web browser, and go to **`http://localhost:8181/cxf/`** to see the available SOAP services with **`OrderService`** in the list.



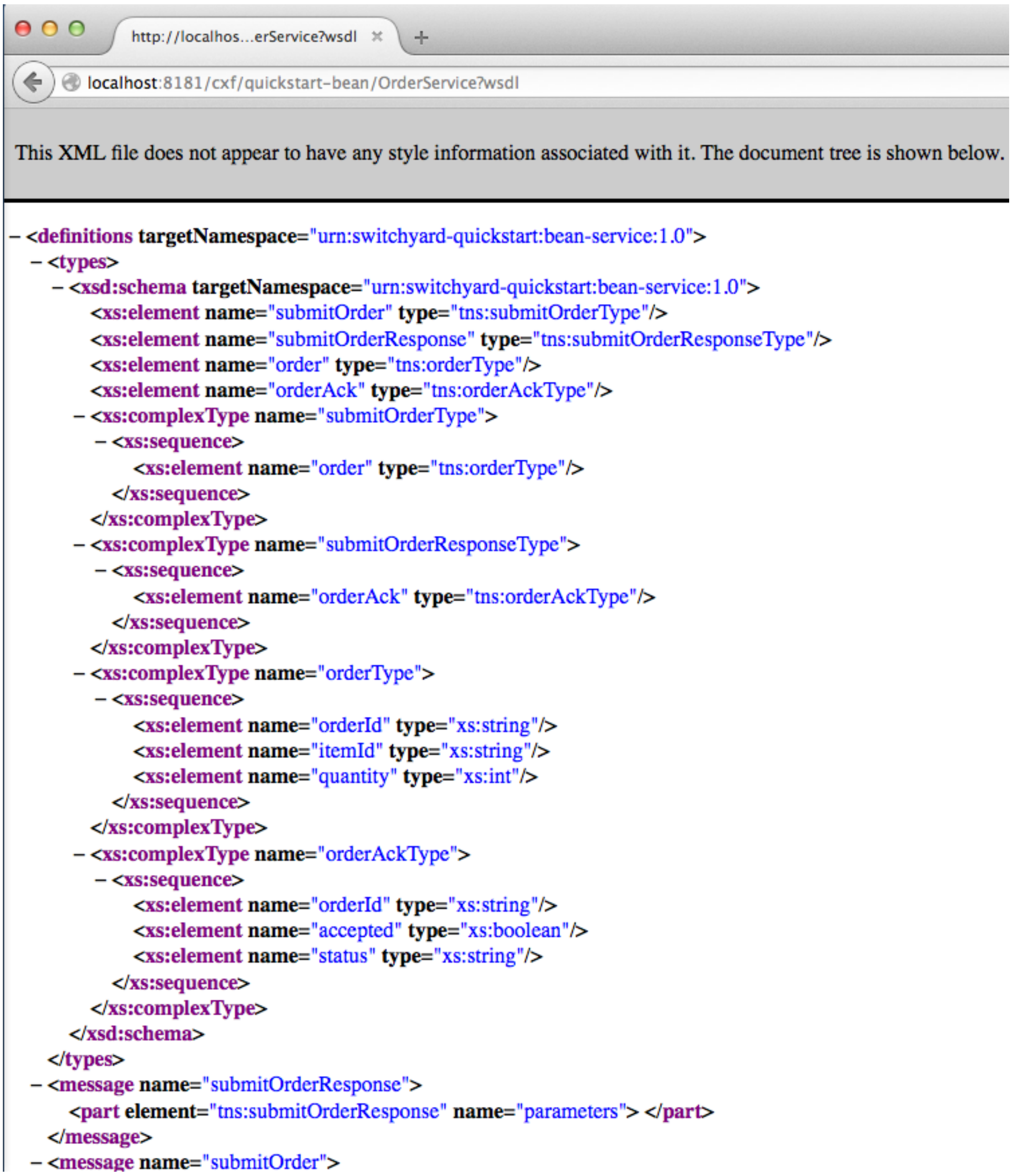
Available SOAP services:

OrderService • submitOrder	Endpoint address: http://localhost:8181/cxf/quickstart-bean/OrderService WSDL : {urn:switchyard-quickstart:bean-service:1.0}OrderService Target namespace: urn:switchyard-quickstart:bean-service:1.0

Available RESTful services:

   This site provides a link to the **`OrderService`**'s WSDL.

2. Click the link to the **`OrderService`**'s WSDL to open it in the browser.

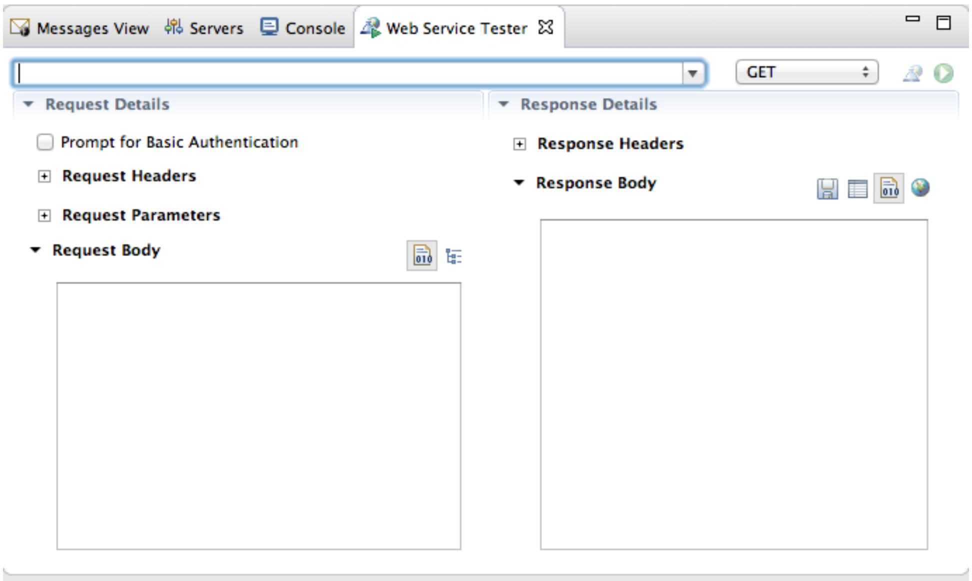

http://localhos...erService?wsdl
localhost:8181/cxf/quickstart-bean/OrderService?wsdl

This XML file does not appear to have any style information associated with it. The document tree is shown below.

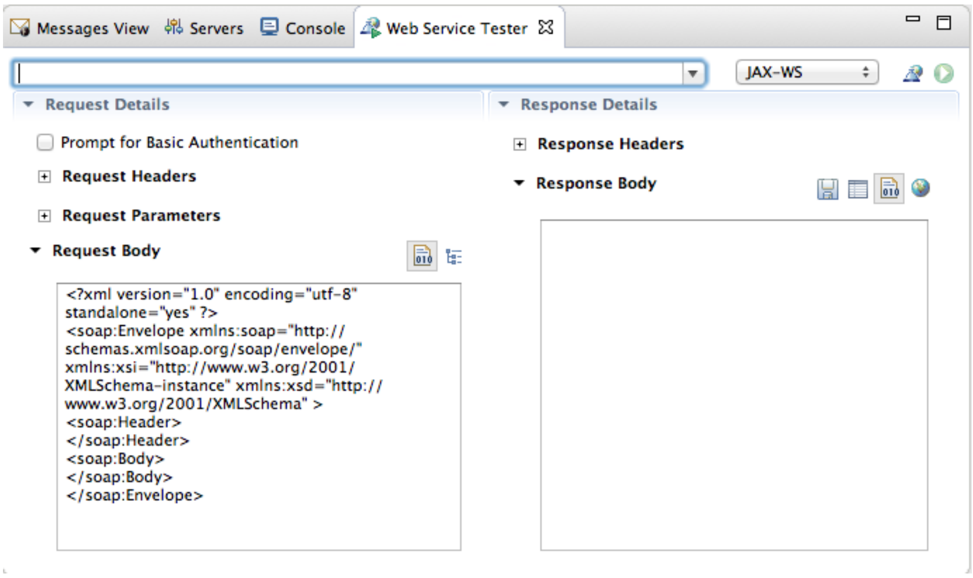
```
– <definitions targetNamespace="urn:switchyard-quickstart:bean-service:1.0">
  – <types>
    – <xsd:schema targetNamespace="urn:switchyard-quickstart:bean-service:1.0">
        <xs:element name="submitOrder" type="tns:submitOrderType"/>
        <xs:element name="submitOrderResponse" type="tns:submitOrderResponseType"/>
        <xs:element name="order" type="tns:orderType"/>
        <xs:element name="orderAck" type="tns:orderAckType"/>
      – <xs:complexType name="submitOrderType">
        – <xs:sequence>
            <xs:element name="order" type="tns:orderType"/>
          </xs:sequence>
        </xs:complexType>
      – <xs:complexType name="submitOrderResponseType">
        – <xs:sequence>
            <xs:element name="orderAck" type="tns:orderAckType"/>
          </xs:sequence>
        </xs:complexType>
      – <xs:complexType name="orderType">
        – <xs:sequence>
            <xs:element name="orderId" type="xs:string"/>
            <xs:element name="itemId" type="xs:string"/>
            <xs:element name="quantity" type="xs:int"/>
          </xs:sequence>
        </xs:complexType>
      – <xs:complexType name="orderAckType">
        – <xs:sequence>
            <xs:element name="orderId" type="xs:string"/>
            <xs:element name="accepted" type="xs:boolean"/>
            <xs:element name="status" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xsd:schema>
    </types>
  – <message name="submitOrderResponse">
      <part element="tns:submitOrderResponse" name="parameters"> </part>
    </message>
  – <message name="submitOrder">
```

3. Copy the URL displayed in the browser's address field.

4. In JBoss Developer Studio, click **Window** → **Show View** → **Other**, then scroll down to the **`JBoss Tools Web Services`** folder, and select **Web Service Tester**.



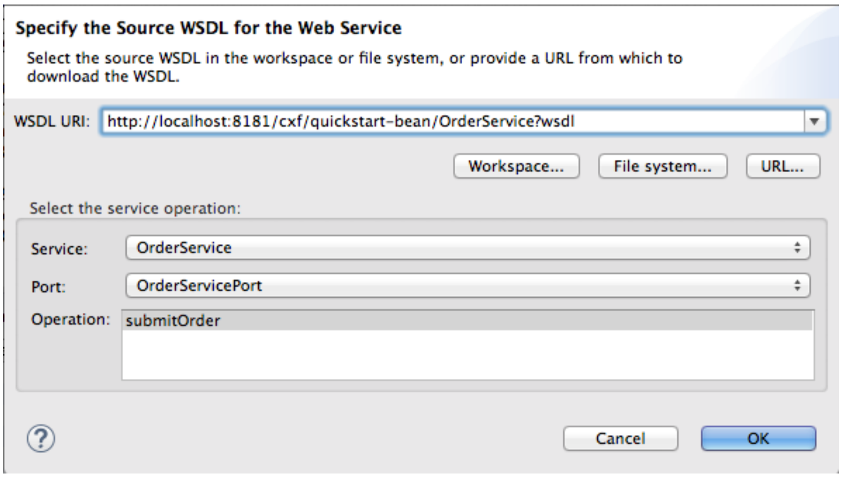
5. Click the drop-down menu labeled **`GET`**, and select **JAX-WS**.



```
<?xml version="1.0" encoding="utf-8"
standalone="yes" ?>
<soap:Envelope xmlns:soap="http://
schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema" >
<soap:Header>
</soap:Header>
<soap:Body>
</soap:Body>
</soap:Envelope>
```

The **`Request Body`** pane displays an example SOAP request message.

6. Click next to JAX-WS to open the **`Specify the Source WSDL for the Web Service`** dialog, and then paste the URl you copied in Step 3 into the **`WSDL URI`** field.



Specify the Source WSDL for the Web Service

Select the source WSDL in the workspace or file system, or provide a URL from which to download the WSDL.

WSDL URI: http://localhost:8181/cxf/quickstart-bean/OrderService?wsdl

Select the service operation:

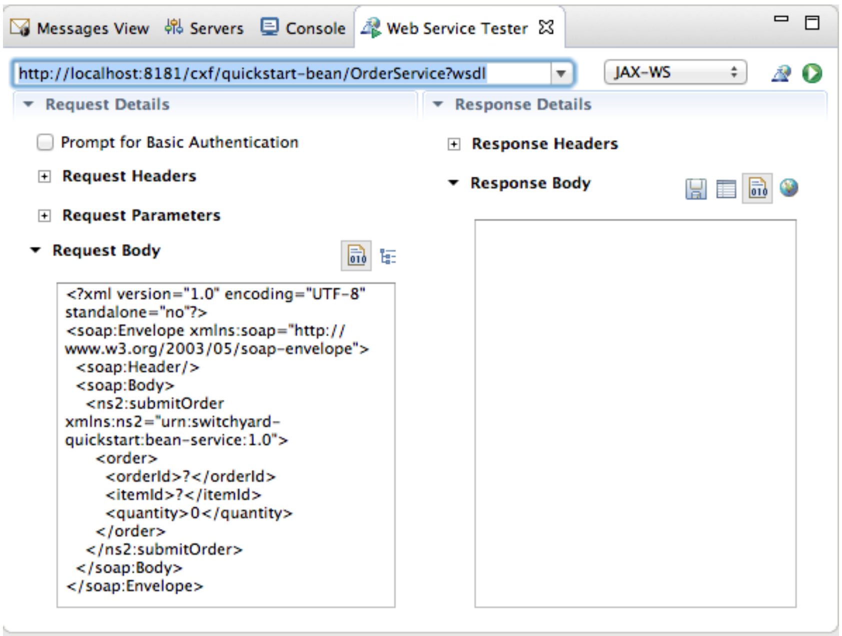
Service: OrderService

Port: OrderServicePort

Operation: submitOrder

The **Web Service Tester** auto fills the remaining fields with the data retrieved from the WSDL.

7. Click **OK**.



http://localhost:8181/cxf/quickstart-bean/OrderService?wsdl

JAX-WS

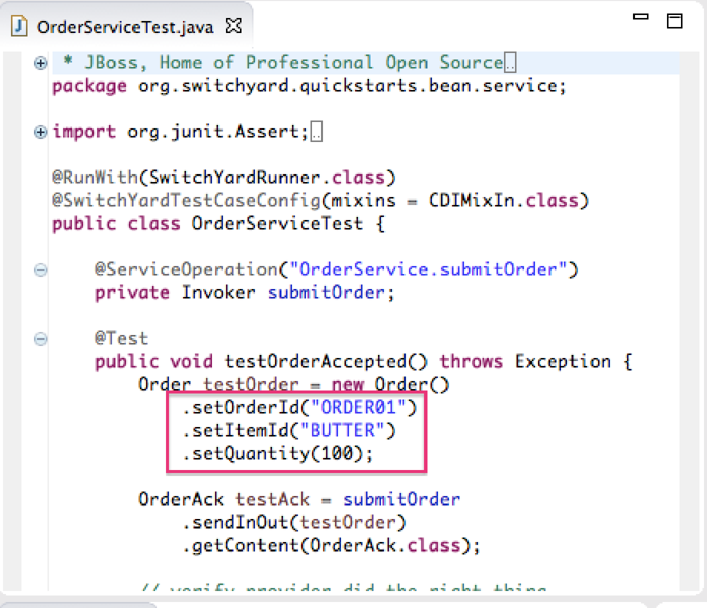
```
<?xml version="1.0" encoding="UTF-8"
standalone="no"?>
<soap:Envelope xmlns:soap="http://
www.w3.org/2003/05/soap-envelope">
  <soap:Header/>
  <soap:Body>
    <ns2:submitOrder
xmlns:ns2="urn:switchyard-
quickstart:bean-service:1.0">
      <order>
        <orderId>?</orderId>
        <itemId>?</itemId>
        <quantity>0</quantity>
      </order>
    </ns2:submitOrder>
  </soap:Body>
</soap:Envelope>
```

The **`Web Service Tester`** displays the XML request message retrieved from the WSDL in the **`Request Body`** pane.

You can enter values for the **`<order>`** items **`<orderId>`**, **`<itemId>`**, and **`<quantity>`** to test the project's request and response services.

NOTE

You can discover what the data types are for each order item. In **Project Explorer**, double-click **src/test/java/OrderServiceTest.java** file to open it in the Java editor:



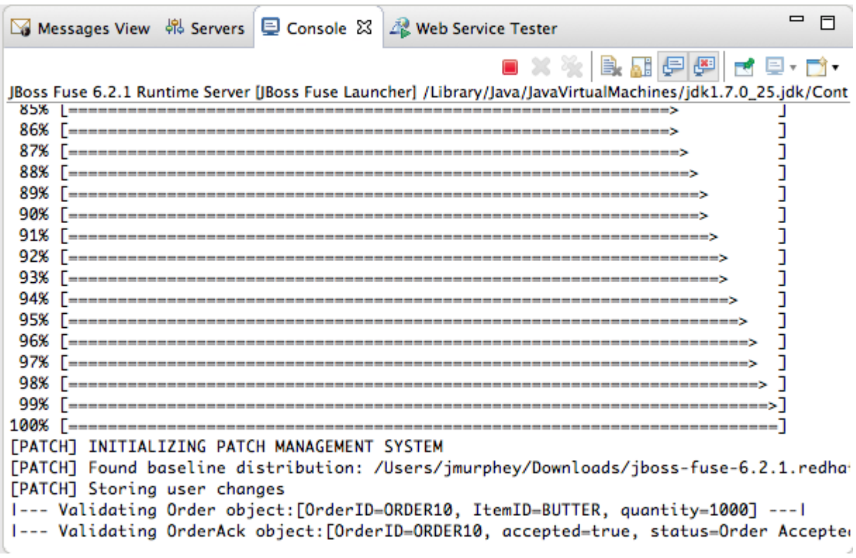
```
 * JBoss, Home of Professional Open Source
package org.switchyard.quickstarts.bean.service;

import org.junit.Assert;

@RunWith(SwitchYardRunner.class)
@SwitchYardTestCaseConfig(mixins = CDIMixIn.class)
public class OrderServiceTest {

    @ServiceOperation("OrderService.submitOrder")
    private Invoker submitOrder;

    @Test
    public void testOrderAccepted() throws Exception {
        Order testOrder = new Order()
            .setOrderId("ORDER01")
            .setItemId("BUTTER")
            .setQuantity(100);

        OrderAck testAck = submitOrder
            .sendInOut(testOrder)
            .getContent(OrderAck.class);

        // verify provider did the right thing
```

8. In the **Request Body** pane, click the value field of each order item and enter an appropriate value for it. For example:

   - For <orderID> replace **?** with *ORDER10*
   - For <itemId> replace **?** with *BUTTER*
   - For <quantity> replace **0** with *1000*

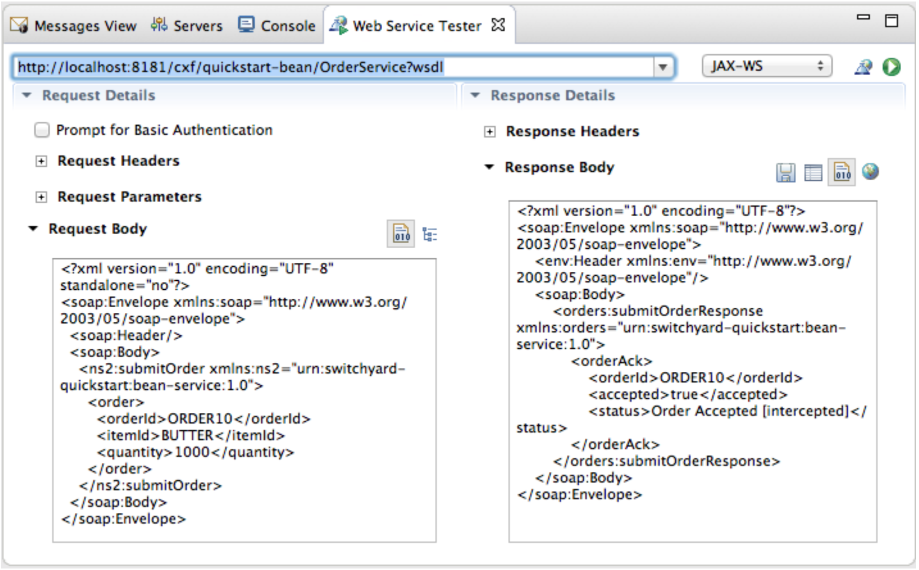
9. Click ▶ to the right of to invoke the **submitOrder** operation. and populate the **Response Body** pane with an example response message.

   Console view automatically opens to show the status of the **submitOrder** operation:



```
JBoss Fuse 6.2.1 Runtime Server [JBoss Fuse Launcher] /Library/Java/JavaVirtualMachines/jdk1.7.0_25.jdk/Cont
 85% [=============================================================>          ]
 86% [=============================================================>          ]
 87% [==============================================================>         ]
 88% [===============================================================>        ]
 89% [================================================================>       ]
 90% [================================================================>       ]
 91% [=================================================================>      ]
 92% [==================================================================>     ]
 93% [==================================================================>     ]
 94% [===================================================================>    ]
 95% [====================================================================>   ]
 96% [=====================================================================>  ]
 97% [=====================================================================>  ]
 98% [======================================================================> ]
 99% [=======================================================================>]
100% [========================================================================]
[PATCH] INITIALIZING PATCH MANAGEMENT SYSTEM
[PATCH] Found baseline distribution: /Users/jmurphey/Downloads/jboss-fuse-6.2.1.redha
[PATCH] Storing user changes
|--- Validating Order object:[OrderID=ORDER10, ItemID=BUTTER, quantity=1000] ---|
|--- Validating OrderAck object:[OrderID=ORDER10, accepted=true, status=Order Accepte
```

10. Switch to the **`Web Service Tester`** tool and check the response message in the **`Response Body`** pane.



```
<?xml version="1.0" encoding="UTF-8"
standalone="no"?>
<soap:Envelope xmlns:soap="http://www.w3.org/
2003/05/soap-envelope">
  <soap:Header/>
  <soap:Body>
    <ns2:submitOrder xmlns:ns2="urn:switchyard-
quickstart:bean-service:1.0">
      <order>
        <orderId>ORDER10</orderId>
        <itemId>BUTTER</itemId>
        <quantity>1000</quantity>
      </order>
    </ns2:submitOrder>
  </soap:Body>
</soap:Envelope>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://www.w3.org/
2003/05/soap-envelope">
    <env:Header xmlns:env="http://www.w3.org/
2003/05/soap-envelope"/>
    <soap:Body>
        <orders:submitOrderResponse
xmlns:orders="urn:switchyard-quickstart:bean-
service:1.0">
            <orderAck>
                <orderId>ORDER10</orderId>
                <accepted>true</accepted>
                <status>Order Accepted [intercepted]</
status>
            </orderAck>
        </orders:submitOrderResponse>
    </soap:Body>
</soap:Envelope>
```

13.3. RUNNING A SWITCHYARD PROJECT ON A JBOSS ENTERPRISE APPLICATION SERVER

To publish and run Fuse SwitchYard projects on an installed EAP server, you must first:

- Install JBoss Fuse on the JBoss EAP server

  For details, see Installation of JBoss Fuse on JBoss EAP

- Add the server and its runtime definition to the tooling's **`Servers`** list

  Once added to the list, the server appears in the **`Servers`** view, where you can configure it, start and stop it, and publish projects to it.

You can publish SwitchYard projects to Red Hat JBoss Enterprise Platform 6.4 and newer servers that you have defined and added to the Servers list.

NOTE

If you haven't already installed a server, you can do so when you define and add it to the Servers list.

Adding a JBoss EAP server

You can add a new server to the **`Servers`** view in three ways:

- In **`Servers`** view, click the link **`No servers are available. Click this link to create a new server...`**.

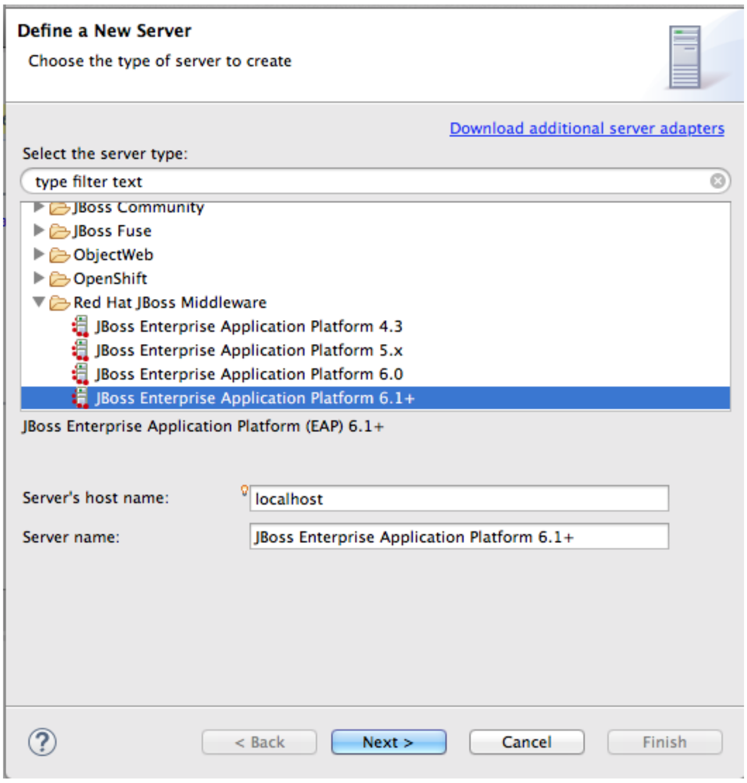
  #### NOTE

  This link appears in **`Servers`** view only when no server has been defined.

- Right-click in **`Servers`** view to open the context menu, then select **New** → **Server**.

- On the menu bar, select **File** → **New** → **Other** → **Server** → **Server**.

1. In the **`Define a New Server`** dialog, select **`Red Hat JBoss Enterprise Application Platform 6.1+`**.



Define a New Server

Choose the type of server to create

Download additional server adapters

Select the server type:

type filter text

JBoss Community
JBoss Fuse
ObjectWeb
OpenShift
Red Hat JBoss Middleware
JBoss Enterprise Application Platform 4.3
JBoss Enterprise Application Platform 5.x
JBoss Enterprise Application Platform 6.0
JBoss Enterprise Application Platform 6.1+

JBoss Enterprise Application Platform (EAP) 6.1+

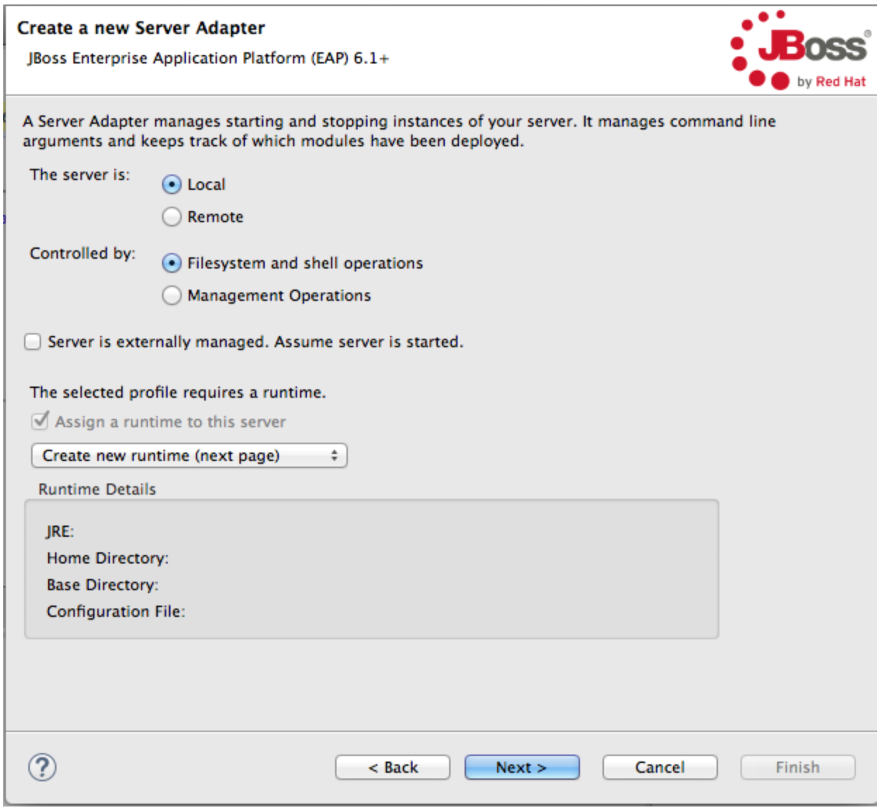
Server's host name: localhost

Server name: JBoss Enterprise Application Platform 6.1+

?
< Back
Next >
Cancel
Finish

`Server's host name` and **`Server name`** are auto filled by default. In **`Server name`**, you can enter a different name to use to identify the server in the **`Servers`** view.

2. Click **`Next`** to open the **`Create a New Server Adapter`** page.



Create a new Server Adapter

JBoss Enterprise Application Platform (EAP) 6.1+

A Server Adapter manages starting and stopping instances of your server. It manages command line arguments and keeps track of which modules have been deployed.

The server is: ● Local ○ Remote

Controlled by: ● Filesystem and shell operations ○ Management Operations

□ Server is externally managed. Assume server is started.

The selected profile requires a runtime.

☑ Assign a runtime to this server

Create new runtime (next page)

Runtime Details

JRE:

Home Directory:

Base Directory:

Configuration File:

To specify that the server life-cycle is managed from outside the tooling, check **`Server is externally managed. Assume server is started.`**.

If you have not configured a runtime before, **`Create new runtime (next page)`** appears on the drop-down menu bar under **`The selected profile requires a runtime`** (as shown in the figure above). Otherwise, a previously configured runtime appears on the drop-down menu bar.

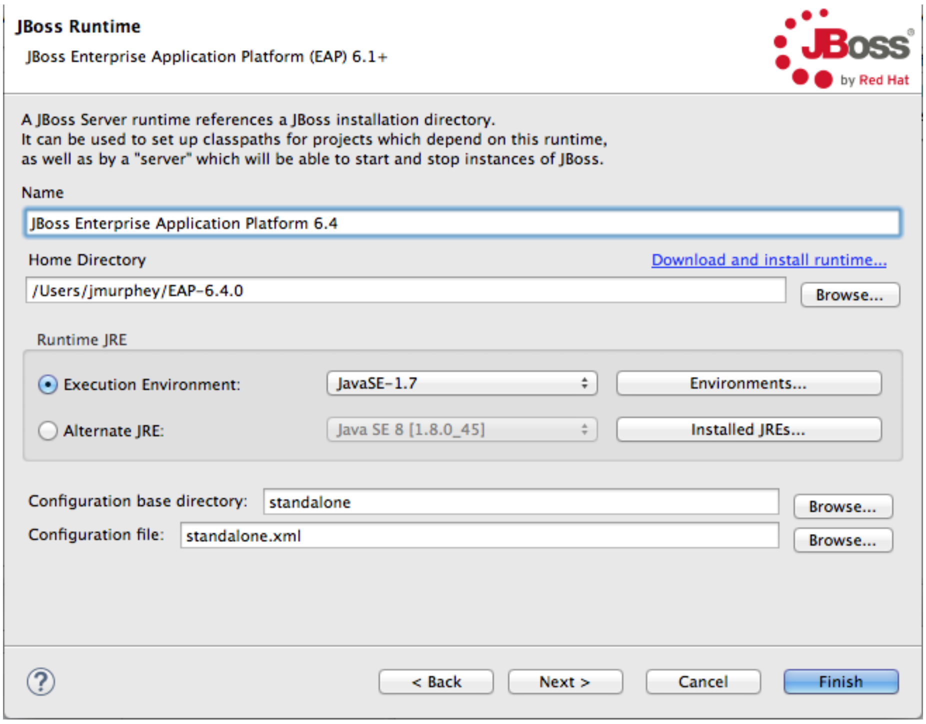
NOTE

You can assign the same runtime to multiple server instances, which enables you to quickly launch a server in debug mode or to configure other runtime settings.

3. To use a previously created runtime with this server, click the drop-down menu and select it from the list (skip to Step 6).

   To create a new runtime, select **`Create new runtime (next page)`** from the list.

4. Click **`Next`** to open the **`JBoss Runtime`** page.



JBoss Runtime

JBoss Enterprise Application Platform (EAP) 6.1+

A JBoss Server runtime references a JBoss installation directory.
It can be used to set up classpaths for projects which depend on this runtime,
as well as by a "server" which will be able to start and stop instances of JBoss.

Name: JBoss Enterprise Application Platform 6.4

Home Directory: /Users/jmurphey/EAP-6.4.0

Runtime JRE

● Execution Environment: JavaSE-1.7

○ Alternate JRE: Java SE 8 [1.8.0_45]

Configuration base directory: standalone

Configuration file: standalone.xml

NOTE

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

5. Configure the JBoss runtime:

   - Accept the default for **`Name`**, or enter a different name.
   - In **`Home Directory`**, enter the path where the server runtime is installed, or click **`Browse`** to find and select it.

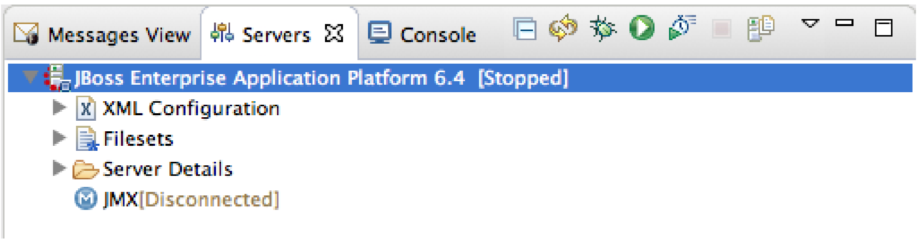
- Select the runtime JRE from the drop-down menu next to **`Execution Environment`**.

  If the version you want does not appear on the list, click **`Environments`**, and then select the version from the list. The JRE version you select must be installed on your machine.

- Accept the defaults for **`Configuration base directory`** and **`Configuration file`**, or browse to the directory where the **`.xml`** configuration file to use is located.

6. Click **`Finish`**.

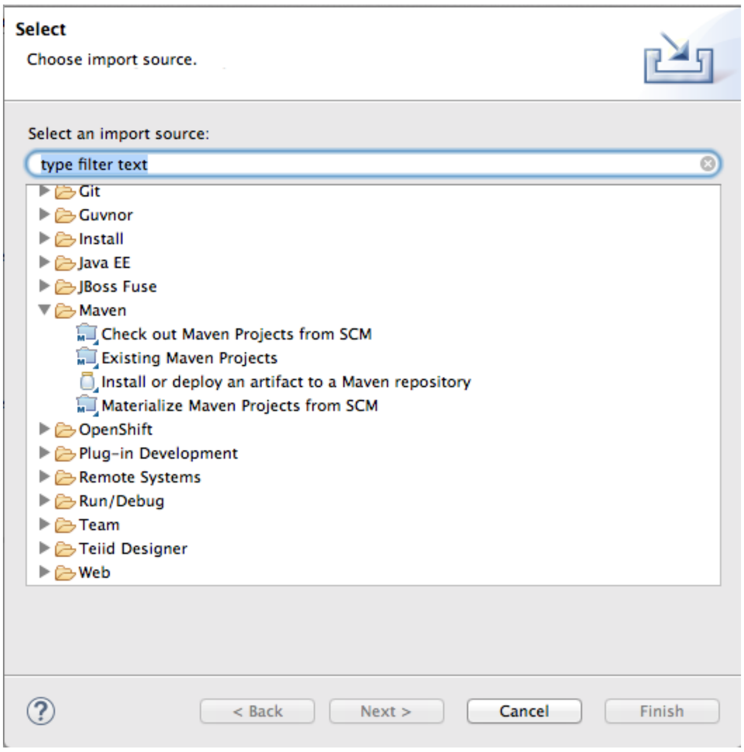
   The server appears in the **`Servers`** view.



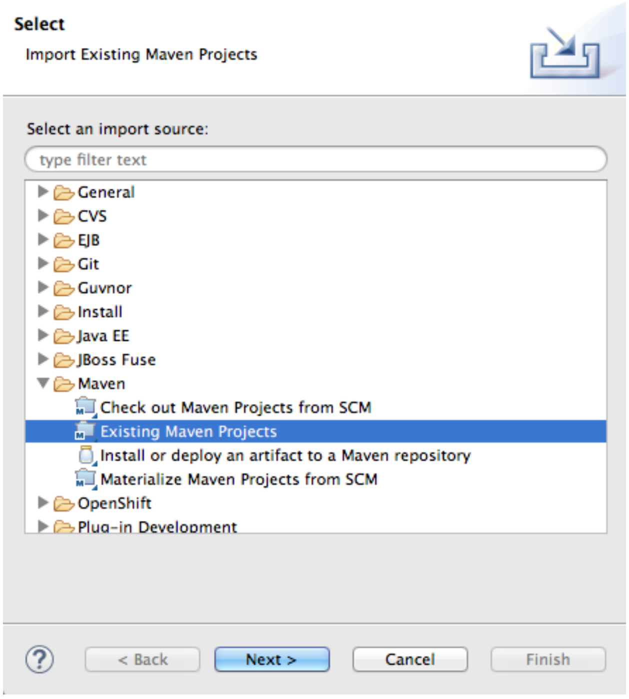
Importing a SwitchYard quickstart project

The JBoss Fuse on JBoss EAP installer installs the SwitchYard quickstarts in ***`$EAP_HOME`*****`/quickstarts/switchyard/`**.

1. Right-click in **`Project Explorer`**, and select **Import** → **Import...** to open the **`Choose Import source`** dialog.



2. In the **`Maven`** folder, select **`Existing Maven Projects`**.



Select

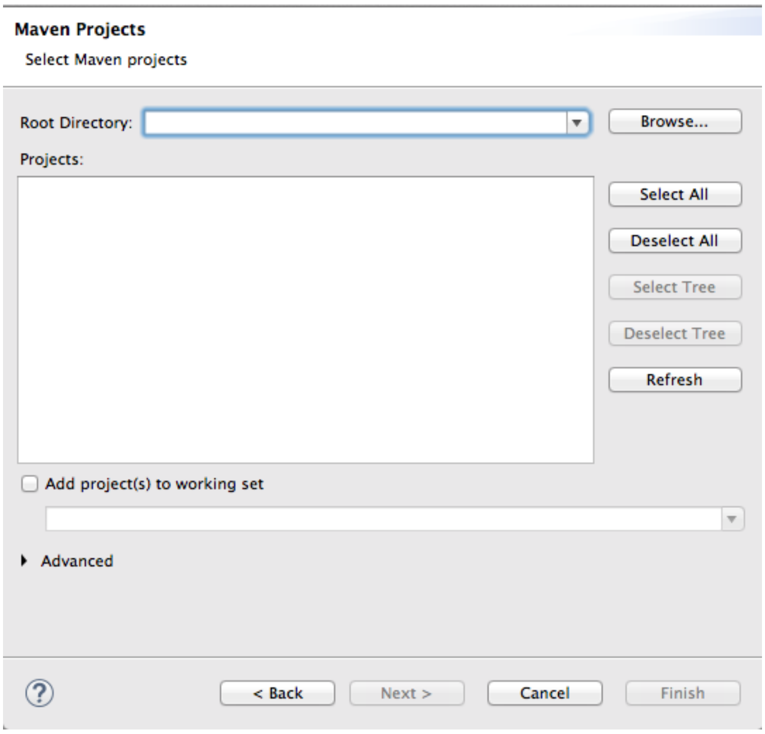
Import Existing Maven Projects

Select an import source:

type filter text

- General
- CVS
- EJB
- Git
- Guvnor
- Install
- Java EE
- JBoss Fuse
- Maven
  - Check out Maven Projects from SCM
  - Existing Maven Projects
  - Install or deploy an artifact to a Maven repository
  - Materialize Maven Projects from SCM
- OpenShift
- Plug-in Development

3. Click **Next** to open the **Select Maven projects** dialog.



Maven Projects

Select Maven projects

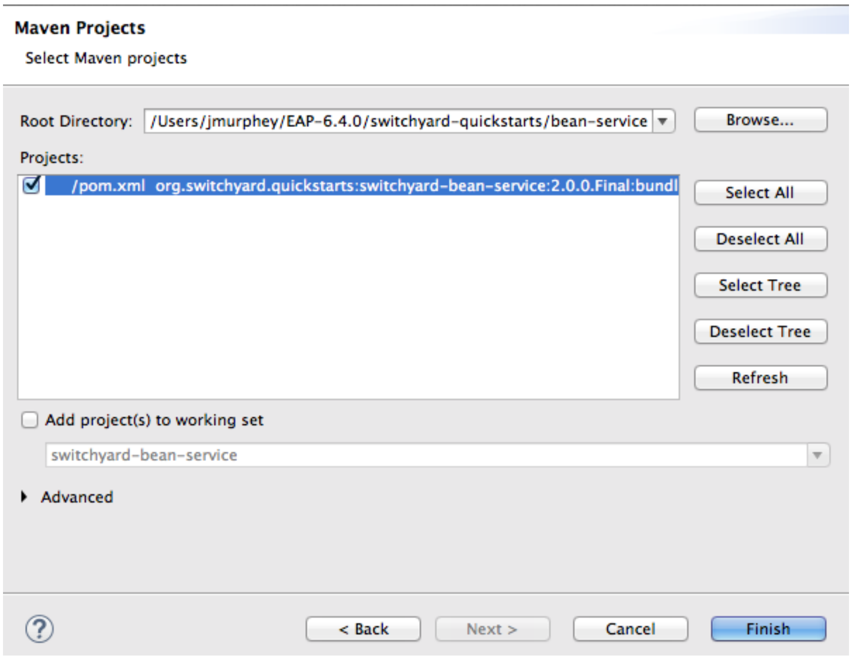
Root Directory:

Projects:

□ Add project(s) to working set

Advanced

4. Click the **Browse** button next to the **Root Directory** field to locate and select the ***$EAP_HOME*/quickstarts/switchyard/bean-service**.



Root Directory: /Users/jmurphey/EAP-6.4.0/switchyard-quickstarts/bean-service

Projects:

☑ /pom.xml org.switchyard.quickstarts:switchyard-bean-service:2.0.0.Final:bundl

□ Add project(s) to working set

switchyard-bean-service

5. In the **`Projects`** pane, make sure the **`pom.xml`** file entry for the **`switchyard-bean-service`** is selected.

6. Click **`Finish`** to start the import.

   The **`switchyard-bean-service`** project appears in **`Project Explorer`**.

   Wait a moment for the import process to finish.

Testing a SwitchYard project locally

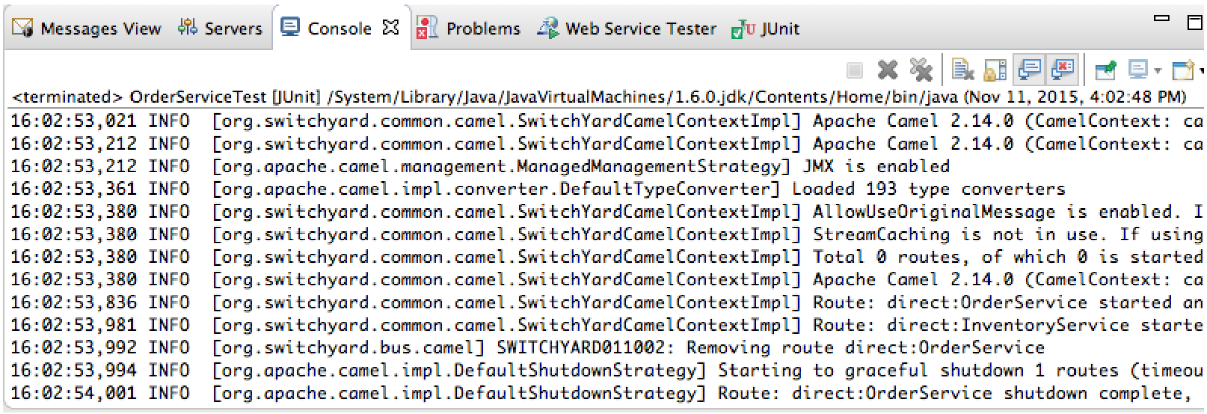
Before you try to publish your project to a server, it's useful to confirm that it builds and runs locally. The provided test files included with each SwitchYard quickstart are complete and comprehensive and need no modification.

1. In **`Project Explorer`**, expand the **`switchyard-bean-service`** project to expose **`src/test/java/OrderServiceTest.java`** file.

   If you want to take a look at the test code, double-click the file to open it in the Java editor.

2. In **`Project Explorer`**, right-click the **`OrderServiceTest.java`** file to open the context menu, and select **Run As → JUnit Test**.

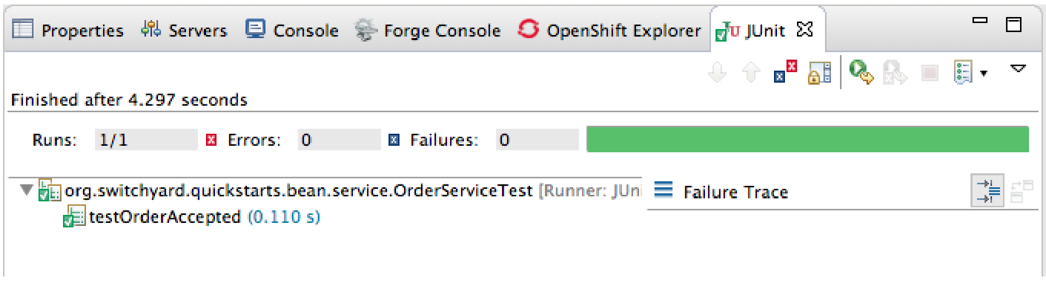
   `Console` view automatically opens and displays the log entries generated by the JUnit test.



```
<terminated> OrderServiceTest [JUnit] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (Nov 11, 2015, 4:02:48 PM)
16:02:53,021 INFO  [org.switchyard.common.camel.SwitchYardCamelContextImpl] Apache Camel 2.14.0 (CamelContext: ca
16:02:53,212 INFO  [org.switchyard.common.camel.SwitchYardCamelContextImpl] Apache Camel 2.14.0 (CamelContext: ca
16:02:53,212 INFO  [org.apache.camel.management.ManagedManagementStrategy] JMX is enabled
16:02:53,361 INFO  [org.apache.camel.impl.converter.DefaultTypeConverter] Loaded 193 type converters
16:02:53,380 INFO  [org.switchyard.common.camel.SwitchYardCamelContextImpl] AllowUseOriginalMessage is enabled. I
16:02:53,380 INFO  [org.switchyard.common.camel.SwitchYardCamelContextImpl] StreamCaching is not in use. If using
16:02:53,380 INFO  [org.switchyard.common.camel.SwitchYardCamelContextImpl] Total 0 routes, of which 0 is started
16:02:53,380 INFO  [org.switchyard.common.camel.SwitchYardCamelContextImpl] Apache Camel 2.14.0 (CamelContext: ca
16:02:53,836 INFO  [org.switchyard.common.camel.SwitchYardCamelContextImpl] Route: direct:OrderService started an
16:02:53,981 INFO  [org.switchyard.common.camel.SwitchYardCamelContextImpl] Route: direct:InventoryService starte
16:02:53,992 INFO  [org.switchyard.bus.camel] SWITCHYARD011002: Removing route direct:OrderService
16:02:53,994 INFO  [org.apache.camel.impl.DefaultShutdownStrategy] Starting to graceful shutdown 1 routes (timeou
16:02:54,001 INFO  [org.apache.camel.impl.DefaultShutdownStrategy] Route: direct:OrderService shutdown complete,
```

These log entries indicate that the test ran successfully against Apache Camel as expected.

3. Click the **`JUnit`** tab to open **`JUnit`** view.



Finished after 4.297 seconds

Runs: 1/1 Errors: 0 Failures: 0

org.switchyard.quickstarts.bean.service.OrderServiceTest [Runner: JUni
testOrderAccepted (0.110 s)

This test proves that the **`switchyard-bean-service`** successfully builds and runs locally.

Publishing a SwitchYard project to the server

You can publish a project to a supported EAP server, defined and listed in the **`Servers`** view, whether it's running or not. A published project will run as scheduled according to the server's settings (for details, see (Chapter 29, *Publishing Fuse Projects to a Server*).

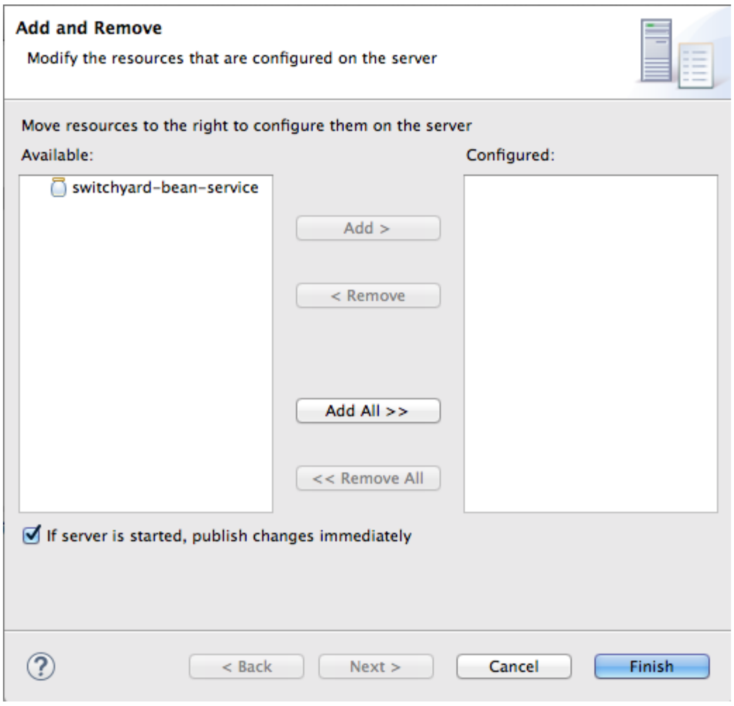
IMPORTANT

The EAP server must have JBoss Fuse on JBoss EAP installed on it. For details, see Installation of JBoss Fuse on JBoss EAP

1. In the **`Servers`** view, select the EAP 6.4 server you just added, and click ▶ on the menu bar to start it.

2. In the **`Servers`** view, right-click the server to open the context menu, and select **Add and Remove**.

   When a project is ready for publishing, it appears in the **`Available`** 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:
switchyard-bean-service

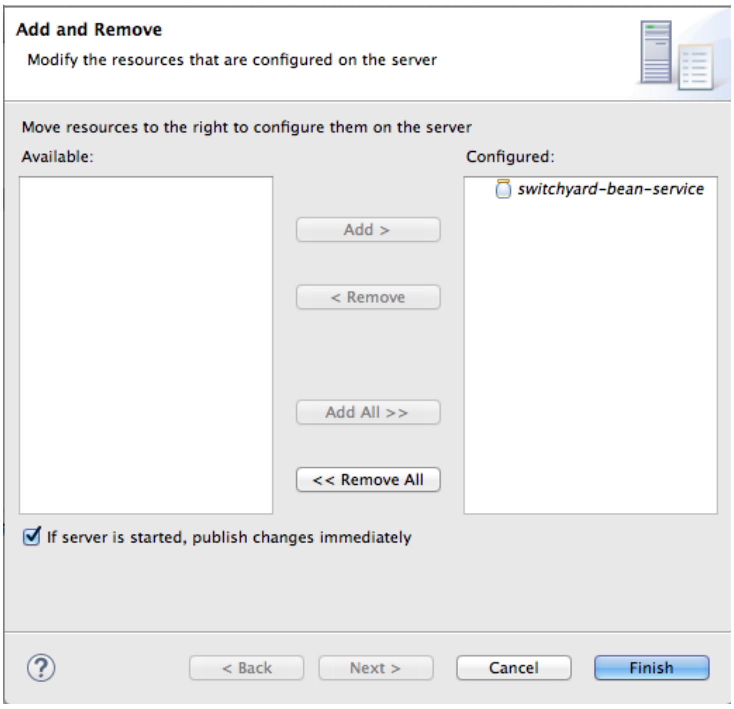
Configured:

☑ If server is started, publish changes immediately

NOTE

The option **If server is started, publish changes immediately** is enable by default. See the section called “Publishing Fuse projects automatically when resources change” for information on how this option works and on using other publishing options.

3. Double-click the project in the **Available** column to move it 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:

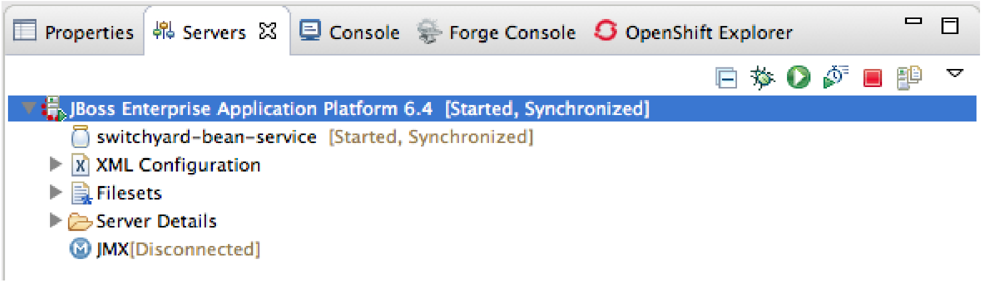
Configured:

switchyard-bean-service

☑ If server is started, publish changes immediately

4. Click **Finish**.

   Once publishing has finished, the project appears as a node under the server runtime node in the **Servers** view.



JBoss Enterprise Application Platform 6.4 [Started, Synchronized]

switchyard-bean-service [Started, Synchronized]

XML Configuration

Filesets

Server Details

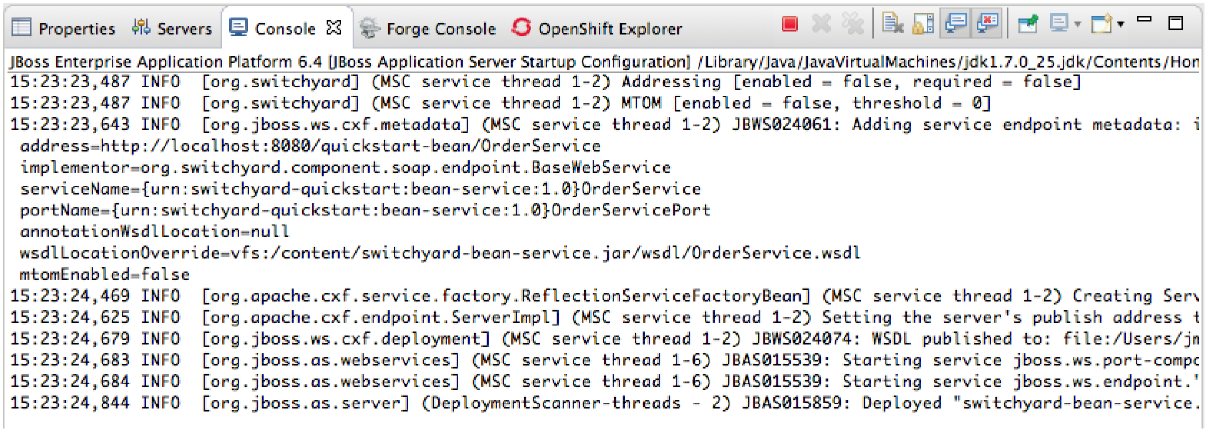
JMX[Disconnected]

Servers view shows that both the server runtime and the project are started and synchronized.

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.

5. Look at the log output in **Console** view to confirm that the **switchyard-bean-service** project is deployed and running:



```
JBoss Enterprise Application Platform 6.4 [JBoss Application Server Startup Configuration] /Library/Java/JavaVirtualMachines/jdk1.7.0_25.jdk/Contents/Hon
15:23:23,487 INFO  [org.switchyard] (MSC service thread 1-2) Addressing [enabled = false, required = false]
15:23:23,487 INFO  [org.switchyard] (MSC service thread 1-2) MTOM [enabled = false, threshold = 0]
15:23:23,643 INFO  [org.jboss.ws.cxf.metadata] (MSC service thread 1-2) JBWS024061: Adding service endpoint metadata: i
 address=http://localhost:8080/quickstart-bean/OrderService
 implementor=org.switchyard.component.soap.endpoint.BaseWebService
 serviceName={urn:switchyard-quickstart:bean-service:1.0}OrderService
 portName={urn:switchyard-quickstart:bean-service:1.0}OrderServicePort
 annotationWsdlLocation=null
 wsdlLocationOverride=vfs:/content/switchyard-bean-service.jar/wsdl/OrderService.wsdl
 mtomEnabled=false
15:23:24,469 INFO  [org.apache.cxf.service.factory.ReflectionServiceFactoryBean] (MSC service thread 1-2) Creating Serv
15:23:24,625 INFO  [org.apache.cxf.endpoint.ServerImpl] (MSC service thread 1-2) Setting the server's publish address t
15:23:24,679 INFO  [org.jboss.ws.cxf.deployment] (MSC service thread 1-2) JBWS024074: WSDL published to: file:/Users/jn
15:23:24,683 INFO  [org.jboss.as.webservices] (MSC service thread 1-6) JBAS015539: Starting service jboss.ws.port-compc
15:23:24,684 INFO  [org.jboss.as.webservices] (MSC service thread 1-6) JBAS015539: Starting service jboss.ws.endpoint.'
15:23:24,844 INFO  [org.jboss.as.server] (DeploymentScanner-threads - 2) JBAS015859: Deployed "switchyard-bean-service.
```

Testing the published SwitchYard project

You can use the JBoss Web Service Tester tool to test your published web service.

1. From the log output in **`Console`** view, copy the address of the **`bean-service`** project's **`OrderService`**.

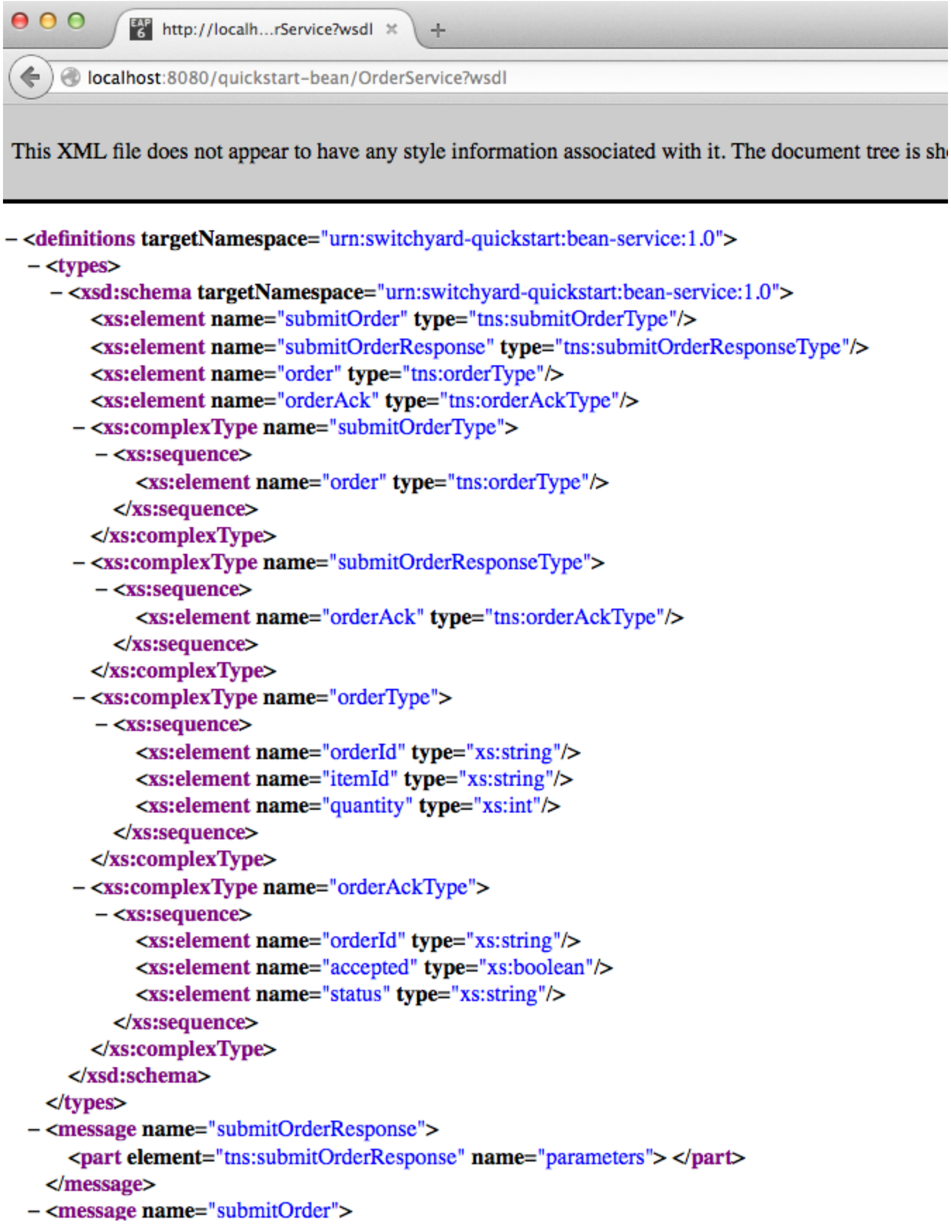
   In the example we are using, the address is **`http://localhost:8080/quickstart-bean/OrderService`**.

2. Open a browser, and paste the address of the **`OrderService`** in the browser's address field.

3. Add **`?wsdl`** to the end of the address, so it reads**`http://localhost:8080/quickstart-bean/OrderService?wsdl`**.

4. Press **`Enter`**.

   The browser displays the WSDL for the **`OrderService`**:



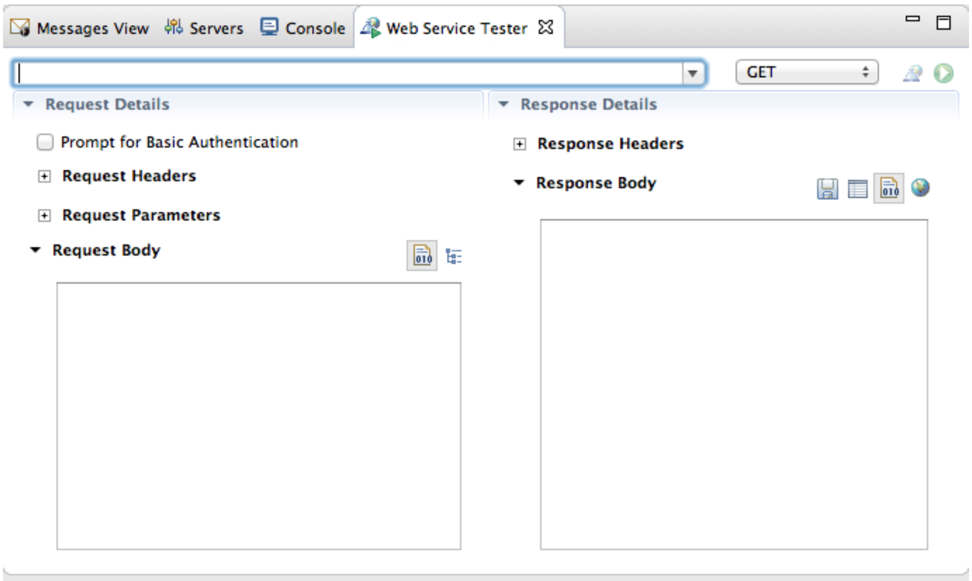
localhost:8080/quickstart-bean/OrderService?wsdl

This XML file does not appear to have any style information associated with it. The document tree is sh

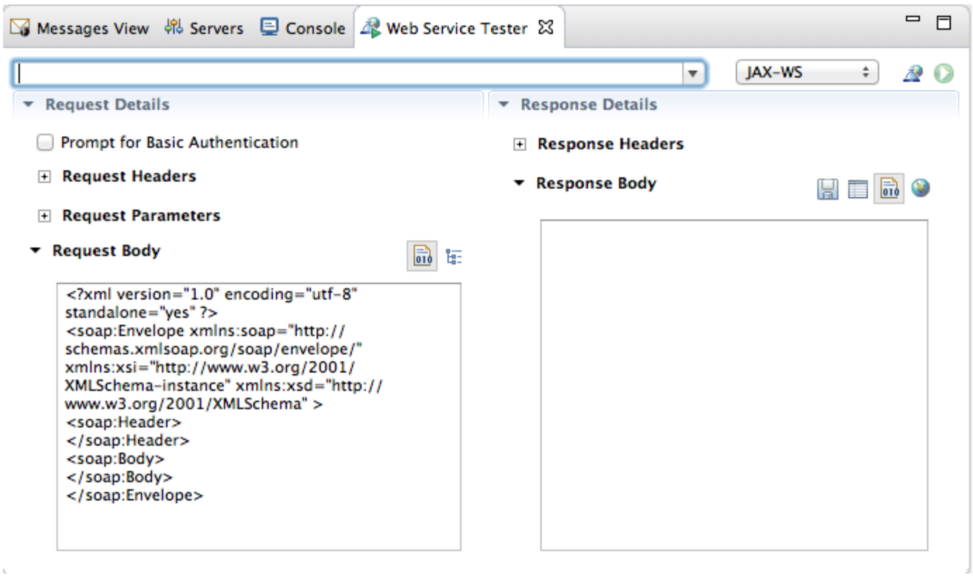
```
– <definitions targetNamespace="urn:switchyard-quickstart:bean-service:1.0">
  – <types>
    – <xsd:schema targetNamespace="urn:switchyard-quickstart:bean-service:1.0">
        <xs:element name="submitOrder" type="tns:submitOrderType"/>
        <xs:element name="submitOrderResponse" type="tns:submitOrderResponseType"/>
        <xs:element name="order" type="tns:orderType"/>
        <xs:element name="orderAck" type="tns:orderAckType"/>
      – <xs:complexType name="submitOrderType">
        – <xs:sequence>
            <xs:element name="order" type="tns:orderType"/>
          </xs:sequence>
        </xs:complexType>
      – <xs:complexType name="submitOrderResponseType">
        – <xs:sequence>
            <xs:element name="orderAck" type="tns:orderAckType"/>
          </xs:sequence>
        </xs:complexType>
      – <xs:complexType name="orderType">
        – <xs:sequence>
            <xs:element name="orderId" type="xs:string"/>
            <xs:element name="itemId" type="xs:string"/>
            <xs:element name="quantity" type="xs:int"/>
          </xs:sequence>
        </xs:complexType>
      – <xs:complexType name="orderAckType">
        – <xs:sequence>
            <xs:element name="orderId" type="xs:string"/>
            <xs:element name="accepted" type="xs:boolean"/>
            <xs:element name="status" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xsd:schema>
    </types>
  – <message name="submitOrderResponse">
      <part element="tns:submitOrderResponse" name="parameters"> </part>
    </message>
  – <message name="submitOrder">
```

5. Copy the URL in the browser's address field.

6. In JBoss Developer Studio, click **Window** → **Show View** → **Other**, then scroll down to the `JBoss Tools Web Services` folder, and select **Web Service Tester**.



7. Click the drop-down menu labeled **GET**, and select **JAX-WS**.

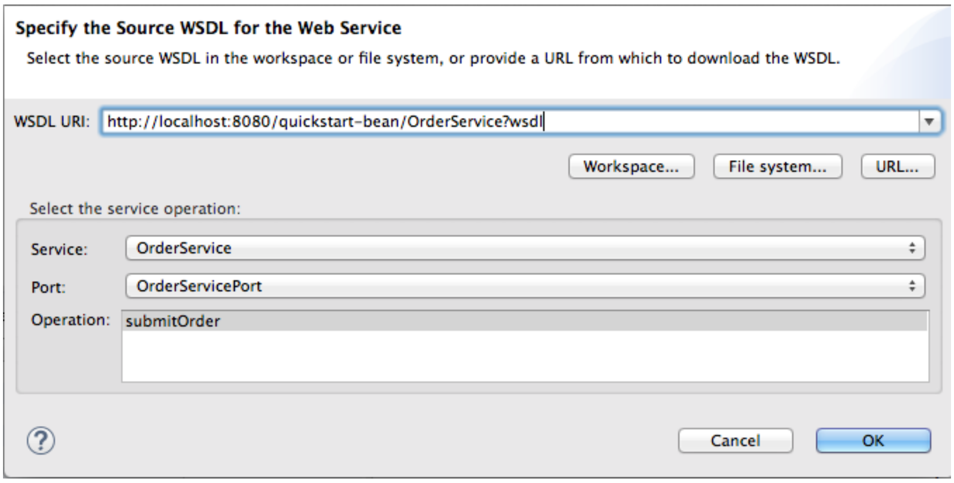


☐ Prompt for Basic Authentication

```
<?xml version="1.0" encoding="utf-8"
standalone="yes" ?>
<soap:Envelope xmlns:soap="http://
schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema" >
<soap:Header>
</soap:Header>
<soap:Body>
</soap:Body>
</soap:Envelope>
```

The **`Request Body`** pane displays an example SOAP request message.

8. Click next to JAX-WS to open the **`Specify the Source WSDL for the Web Service`** dialog, then paste the URL you copied in Step 5 into the **`WSDL URI`** field.



Specify the Source WSDL for the Web Service

Select the source WSDL in the workspace or file system, or provide a URL from which to download the WSDL.

WSDL URI: http://localhost:8080/quickstart-bean/OrderService?wsdl

Select the service operation:

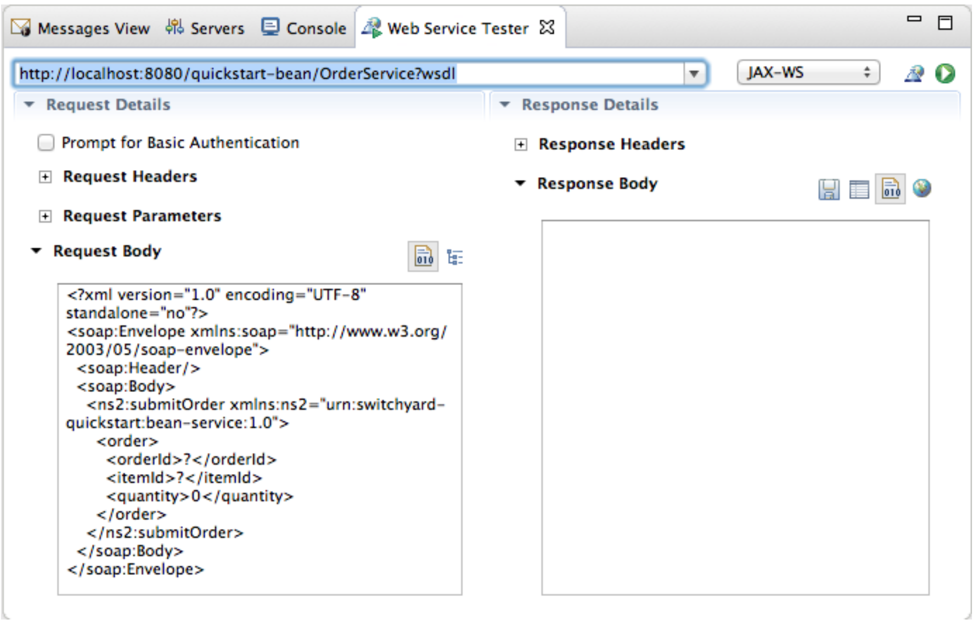
Service: OrderService

Port: OrderServicePort

Operation: submitOrder

The **`Web Service Tester`** auto fills the remaining fields with the data retrieved from the WSDL.

9. Click **`OK`**.



http://localhost:8080/quickstart-bean/OrderService?wsdl

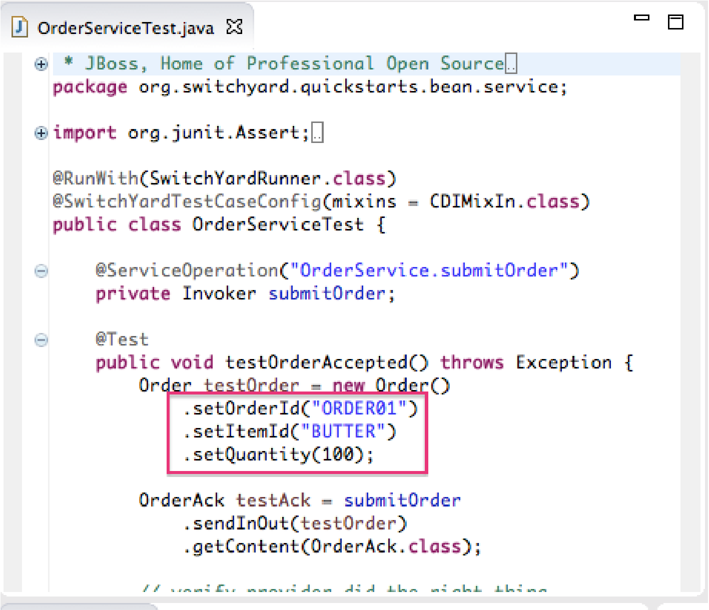
```
<?xml version="1.0" encoding="UTF-8"
standalone="no"?>
<soap:Envelope xmlns:soap="http://www.w3.org/
2003/05/soap-envelope">
  <soap:Header/>
  <soap:Body>
    <ns2:submitOrder xmlns:ns2="urn:switchyard-
quickstart:bean-service:1.0">
      <order>
        <orderId>?</orderId>
        <itemId>?</itemId>
        <quantity>0</quantity>
      </order>
    </ns2:submitOrder>
  </soap:Body>
</soap:Envelope>
```

The **Web Service Tester** displays the XML request message retrieved from the WSDL in the **Request Body** pane.

You can enter values for the **<order>** items **<orderId>**, **<itemId>**, and **<quantity>** to test the project's request and response services.

NOTE

You can discover what the data types are for each order item. In **Project Explorer**, double-click **src/test/java/OrderServiceTest.java** file to open it in the Java editor:



```
 * JBoss, Home of Professional Open Source
package org.switchyard.quickstarts.bean.service;

import org.junit.Assert;

@RunWith(SwitchYardRunner.class)
@SwitchYardTestCaseConfig(mixins = CDIMixIn.class)
public class OrderServiceTest {

    @ServiceOperation("OrderService.submitOrder")
    private Invoker submitOrder;

    @Test
    public void testOrderAccepted() throws Exception {
        Order testOrder = new Order()
            .setOrderId("ORDER01")
            .setItemId("BUTTER")
            .setQuantity(100);

        OrderAck testAck = submitOrder
            .sendInOut(testOrder)
            .getContent(OrderAck.class);

        // verify provider did the right thing
```

10. In the **Request Body** pane, click the value field of each order item and enter an appropriate value for it. For example:

    - For <orderID>, replace **?** with *ORDER10*

    - For <itemId>, replace **?** with *BUTTER*

- For <quantity>, replace **0** with *1000*

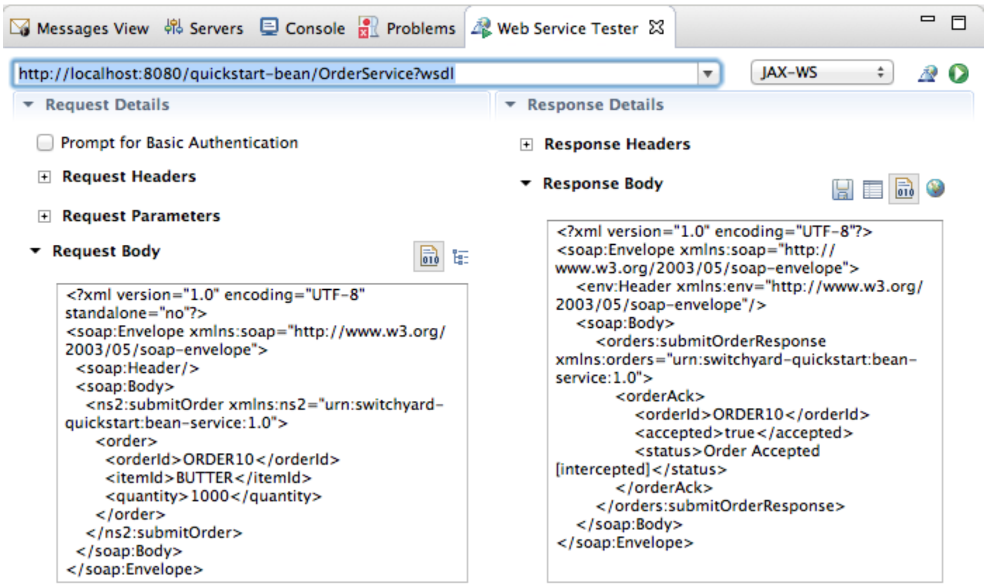
11. Click ▶ to the right of to invoke the **submitOrder** operation. and populate the **Response Body** pane with an example response message.

    Console view automatically opens to display the status of the **submitOrder** operation like this —

    ```
    ... INFO [stdout] (http-localhost/127.0.0.1:8080-2) |--- Validating
    Order object: [OrderID=ORDER10,
           ItemID=BUTTER, quantity=1000] ---|

    ... INFO [stdout] (http-localhost/127.0.0.1:8080-2) |--- Validating
    OrderAck object: [OrderID=ORDER10,
           accepted=true, status=Order Accepted [intercepted]] ---|
    ```

    —at the end of the **Console** output.

12. Switch to the **Web Service Tester** tool and check the response message in the **Response Body** pane.



http://localhost:8080/quickstart-bean/OrderService?wsdl

```
<?xml version="1.0" encoding="UTF-8"
standalone="no"?>
<soap:Envelope xmlns:soap="http://www.w3.org/
2003/05/soap-envelope">
  <soap:Header/>
  <soap:Body>
    <ns2:submitOrder xmlns:ns2="urn:switchyard-
quickstart:bean-service:1.0">
      <order>
        <orderId>ORDER10</orderId>
        <itemId>BUTTER</itemId>
        <quantity>1000</quantity>
      </order>
    </ns2:submitOrder>
  </soap:Body>
</soap:Envelope>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://
www.w3.org/2003/05/soap-envelope">
    <env:Header xmlns:env="http://www.w3.org/
2003/05/soap-envelope"/>
    <soap:Body>
        <orders:submitOrderResponse
xmlns:orders="urn:switchyard-quickstart:bean-
service:1.0">
            <orderAck>
                <orderId>ORDER10</orderId>
                <accepted>true</accepted>
                <status>Order Accepted
[intercepted]</status>
            </orderAck>
        </orders:submitOrderResponse>
    </soap:Body>
</soap:Envelope>
```

PART III. DEBUGGING ROUTING CONTEXTS

The Camel debugger includes many features for debugging routing contexts:

- Setting conditional and unconditional breakpoints on nodes in the route editor
- Autolaunching the debugger and switching to 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. Once that's done, 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 2, *Debug Perspective*).

CHAPTER 14. SETTING BREAKPOINTS

OVERVIEW

You can set breakpoints in either **JBoss** perspective or in **Fuse Integration** perspective, but **JBoss** perspective provides more room for expanding the route editor's canvas when building and modifying routes. To set breakpoints, your project's routing context **.xml** file must be open in the route editor's **Design View**.

The Camel debugger supports two types of breakpoints:

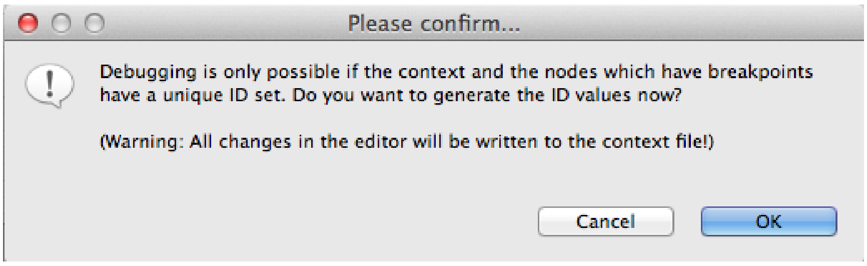
- Unconditional breakpoints—triggered whenever one is encountered during a debugging session
- Conditional breakpoints—triggered only when the breakpoint's specified condition is met during a debugging session

NOTE

You cannot set breakpoints on Consumer endpoints or on When or Otherwise nodes.

NOTE

The Camel debugger requires the **camelContext** element and each node for which a breakpoint is set to have a unique ID. You can set IDs manually in a node's **Properties** editor or directly in the CamelContext file in **Source** view, or you can let Fuse Tooling generate and apply them automatically. The **Please confirm...** dialog opens whenever you set a breakpoint on a node that does not already have a unique ID.



Please confirm...

Debugging is only possible if the context and the nodes which have breakpoints have a unique ID set. Do you want to generate the ID values now?

(Warning: All changes in the editor will be written to the context file!)

IMPORTANT

Make sure you save any changes you make to your routing context file before you run the Camel debugger on it. Otherwise, your changes will not be included in the debugging session.

SETTING UNCONDITIONAL BREAKPOINTS

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

1. Select a node whose state you want to examine during the debugging session.
2. Click its ● icon to set an unconditional breakpoint.
3. In the **Please Confirm...** dialog, click **OK** to have the Route Editor automatically generate and apply a unique ID to the <camelContext> element and to the node or nodes that need it, or click **Cancel** if you want to manually enter unique IDs in the <camelContext> element and to

each node that needs one.

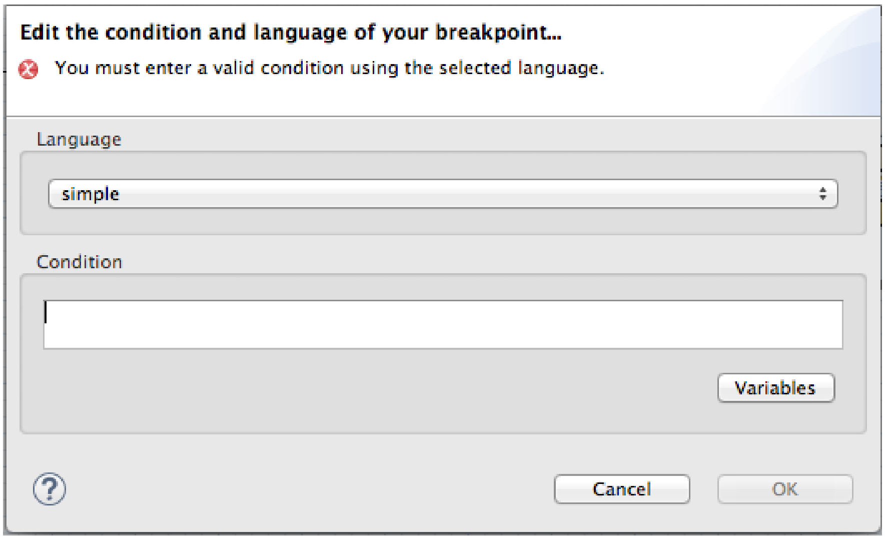
4. Repeat these steps for each node on which you want to set an unconditional breakpoint.

SETTING CONDITIONAL BREAKPOINTS

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

1. Select a node whose state you want to examine during the debugging session.

2. Click its ● icon to set a conditional breakpoint and 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 langauge to use to create the condition that will trigger the breakpoint.

   Fuse Tooling supports twenty expression languages from which to choose. Except for **`constant`**, **`method`**, **`property`**, and **`ref`**, each of these languages provides variables for creating conditional expressions.

4. Click the **`Variables`** drop-down menu and select in sequence one or more of the language's supported variables to create the condition for triggering the breakpoint. The variables you select appear in the **`Condition`** text box.

   Alternatively, you can manually enter the expression directly into the **`Condition`** text box.

5. Repeat steps Step 1 through Step 4 for each node on which you want to set a conditioanl 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 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 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.

- Individual breakpoints—on the canvas, select the node whose breakpoint you want to delete, and click its ⊗ icon.
- All breakpoints—right-click the canvas, and select ● **`Delete all breakpoints`**.

RELATED TOPICS

Chapter 15, *Starting the Camel Debugger*
chapter "To Debug a Routing Context" in "Tooling Tutorials"

CHAPTER 15. STARTING THE CAMEL DEBUGGER

OVERVIEW

NOTE

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

You run the Camel debugger on the routing context file, so be sure to save any changes you made to the file before you invoke the debugger. Otherwise your changes will not be included in the debugging session.

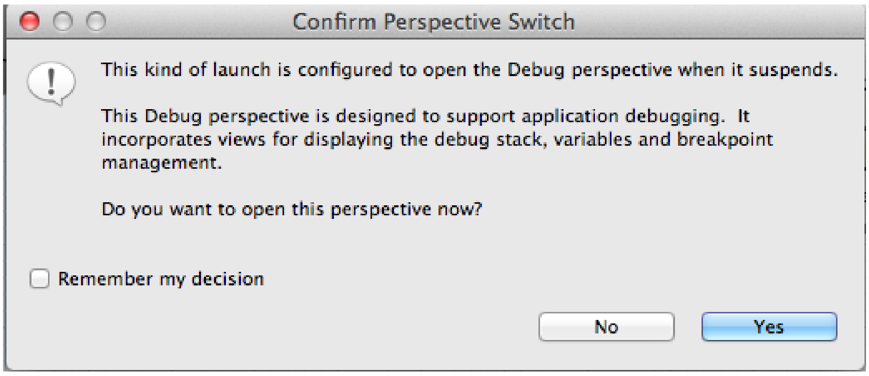
PROCEDURE

1. In `Project Explorer`, select the routing context file you want to debug.

2. Right-click it to open the context menu, and then select **Debug As... → Local Camel Context**.

   Fuse Tooling builds the Camel route, starts up 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 it encounters, and asks 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?

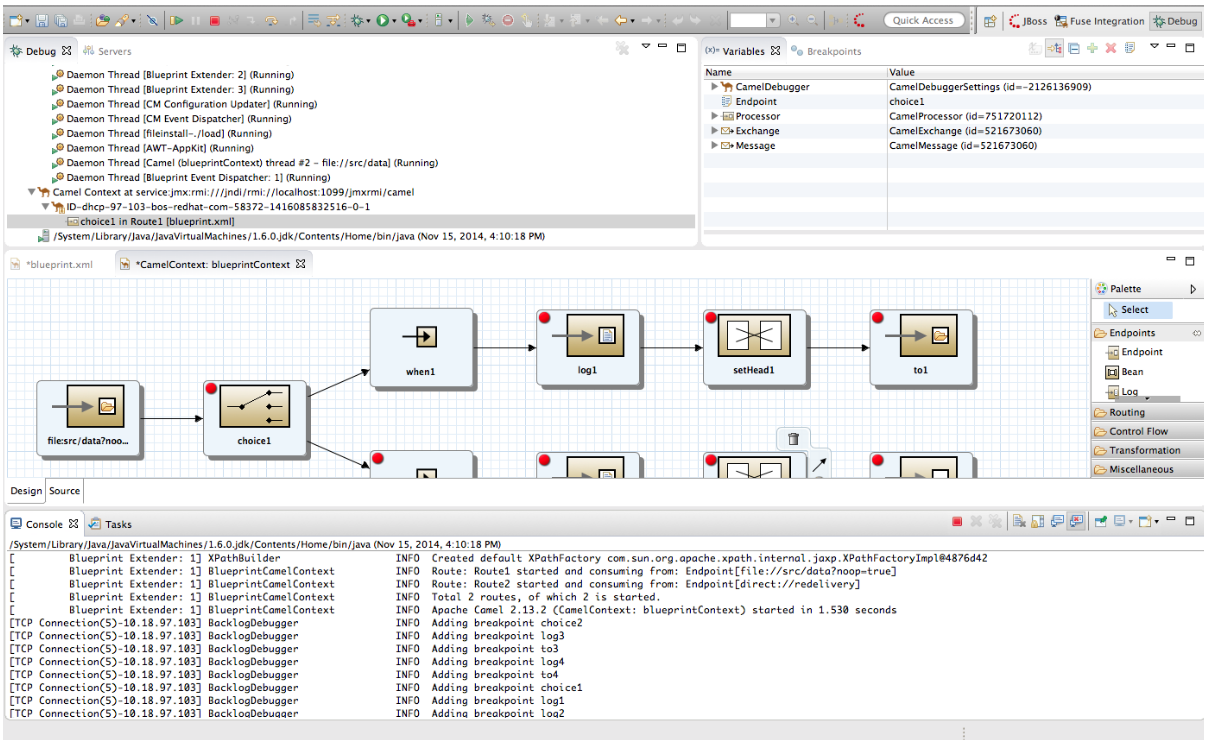
   □ Remember my decision

3. Click `Yes` to open `Debug` perspective.

   `Debug` perspective opens with the routing context suspended at the first breakpoint 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 if there are no more breakpoints.



```
Daemon Thread [Blueprint Extender: 2] (Running)
Daemon Thread [Blueprint Extender: 3] (Running)
Daemon Thread [CM Configuration Updater] (Running)
Daemon Thread [CM Event Dispatcher] (Running)
Daemon Thread [fileinstall-./load] (Running)
Daemon Thread [AWT-AppKit] (Running)
Daemon Thread [Camel (blueprintContext) thread #2 - file://src/data] (Running)
Daemon Thread [Blueprint Event Dispatcher: 1] (Running)
Camel Context at service:jmx:rmi:///jndi/rmi://localhost:1099/jmxrmi/camel
  ID-dhcp-97-103-bos-redhat-com-58372-1416085832516-0-1
    choice1 in Route1 [blueprint.xml]
/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (Nov 15, 2014, 4:10:18 PM)
```

Name	Value
CamelDebugger	CamelDebuggerSettings (id=-2126136909)
Endpoint	choice1
Processor	CamelProcessor (id=751720112)
Exchange	CamelExchange (id=521673060)
Message	CamelMessage (id=521673060)

```
/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (Nov 15, 2014, 4:10:18 PM)
[        Blueprint Extender: 1] XPathBuilder                   INFO  Created default XPathFactory com.sun.org.apache.xpath.internal.jaxp.XPathFactoryImpl@4876d42
[        Blueprint Extender: 1] BlueprintCamelContext          INFO  Route: Route1 started and consuming from: Endpoint[file://src/data?noop=true]
[        Blueprint Extender: 1] BlueprintCamelContext          INFO  Route: Route2 started and consuming from: Endpoint[direct://redelivery]
[        Blueprint Extender: 1] BlueprintCamelContext          INFO  Total 2 routes, of which 2 is started.
[        Blueprint Extender: 1] BlueprintCamelContext          INFO  Apache Camel 2.13.2 (CamelContext: blueprintContext) started in 1.530 seconds
[TCP Connection(5)-10.18.97.103] BacklogDebugger               INFO  Adding breakpoint choice2
[TCP Connection(5)-10.18.97.103] BacklogDebugger               INFO  Adding breakpoint log3
[TCP Connection(5)-10.18.97.103] BacklogDebugger               INFO  Adding breakpoint to3
[TCP Connection(5)-10.18.97.103] BacklogDebugger               INFO  Adding breakpoint log4
[TCP Connection(5)-10.18.97.103] BacklogDebugger               INFO  Adding breakpoint to4
[TCP Connection(5)-10.18.97.103] BacklogDebugger               INFO  Adding breakpoint choice1
[TCP Connection(5)-10.18.97.103] BacklogDebugger               INFO  Adding breakpoint log1
[TCP Connection(5)-10.18.97.103] BacklogDebugger               INFO  Adding breakpoint log2
```

NOTE

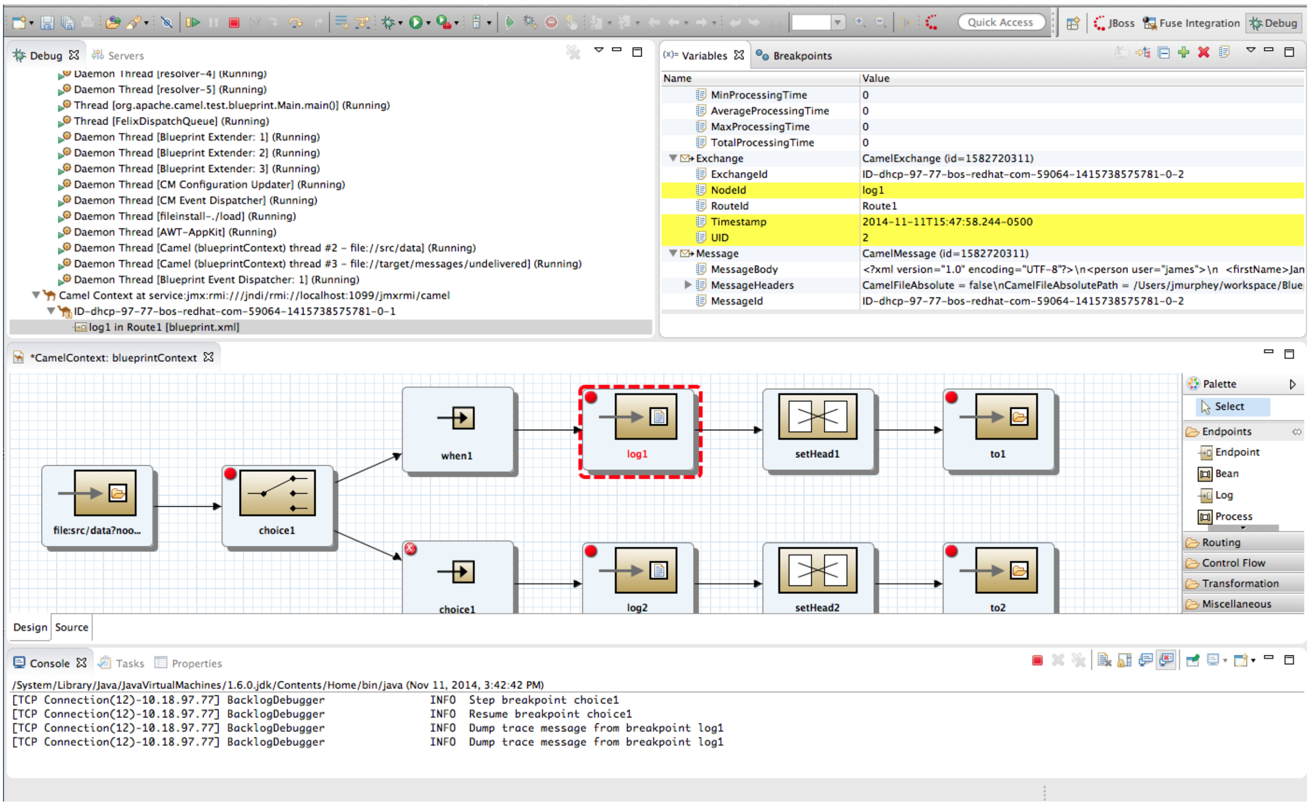
To see the console output, you'll need to open **Console** view if it was not open when you switched perspectives.

NOTE

By default, **Debug** perspective opens displaying **Outline** view, which provides the means to switch between separate routes in a routing context. If your routing context contains a single route, closing **Outline** view will provide more 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 [resolver-4] (Running)
Daemon Thread [resolver-5] (Running)
Thread [org.apache.camel.test.blueprint.Main.main()] (Running)
Thread [FelixDispatchQueue] (Running)
Daemon Thread [Blueprint Extender: 1] (Running)
Daemon Thread [Blueprint Extender: 2] (Running)
Daemon Thread [Blueprint Extender: 3] (Running)
Daemon Thread [CM Configuration Updater] (Running)
Daemon Thread [CM Event Dispatcher] (Running)
Daemon Thread [fileinstall-./load] (Running)
Daemon Thread [AWT-AppKit] (Running)
Daemon Thread [Camel (blueprintContext) thread #2 - file://src/data] (Running)
Daemon Thread [Camel (blueprintContext) thread #3 - file://target/messages/undelivered] (Running)
Daemon Thread [Blueprint Event Dispatcher: 1] (Running)
Camel Context at service:jmx:rmi:///jndi/rmi://localhost:1099/jmxrmi/camel
  ID-dhcp-97-77-bos-redhat-com-59064-1415738575781-0-1
    log1 in Route1 [blueprint.xml]
```

Name	Value
MinProcessingTime	0
AverageProcessingTime	0
MaxProcessingTime	0
TotalProcessingTime	0
Exchange	CamelExchange (id=1582720311)
ExchangeId	ID-dhcp-97-77-bos-redhat-com-59064-1415738575781-0-2
NodeId	log1
RouteId	Route1
Timestamp	2014-11-11T15:47:58.244-0500
UID	2
Message	CamelMessage (id=1582720311)
MessageBody	<?xml version="1.0" encoding="UTF-8"?>\n<person user="james">\n <firstName>Jan
MessageHeaders	CamelFileAbsolute = false\nCamelFileAbsolutePath = /Users/jmurphey/workspace/Blue
MessageId	ID-dhcp-97-77-bos-redhat-com-59064-1415738575781-0-2

```
/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (Nov 11, 2014, 3:42:42 PM)
[TCP Connection(12)-10.18.97.77] BacklogDebugger                INFO  Step breakpoint choice1
[TCP Connection(12)-10.18.97.77] BacklogDebugger                INFO  Resume breakpoint choice1
[TCP Connection(12)-10.18.97.77] BacklogDebugger                INFO  Dump trace message from breakpoint log1
[TCP Connection(12)-10.18.97.77] BacklogDebugger                INFO  Dump trace message from breakpoint log1
```

RELATED TOPICS

Chapter 16, *Stopping the Camel Debugger*

chapter "To Debug a Routing Context" in "Tooling Tutorials"

CHAPTER 16. STOPPING THE CAMEL DEBUGGER

OVERVIEW

To stop the Camel debugger, click 🟥 on the menu bar once if the debugging session has ended. Otherwise you need to click 🟥 twice. Once to terminate the currently running node thread and once to terminate the **`Camel Context`** thread (both displayed in **`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 **`Console`** view's menu bar.

TERMINATING THE CAMEL DEBUGGER BEFORE THE SESSION ENDS

Terminating the debugger before the debugging session ends may add extraneous entries to some of the project's folders in **`Project Explorer`**. You may see message lock files or **`target/.CamelContextInDebug_xxxxxxxx_temp/camel-context.xml`** lines in the root project's **`src/main/resources/`*`framework`*`-INF/`*`framework_name`*** folder. In**`Fuse Integration`** perspective, You may also see them in the project's **`Camel Contexts`** folder.

To clear them, right-click the root project, and then select **Refresh**.

RELATED TOPICS

Chapter 15, *Starting the Camel Debugger*

chapter "To Debug a Routing Context" in "Tooling Tutorials"

CHAPTER 17. CHANGING VARIABLE VALUES

OVERVIEW

When the Camel debugger hits a breakpoint, **`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.

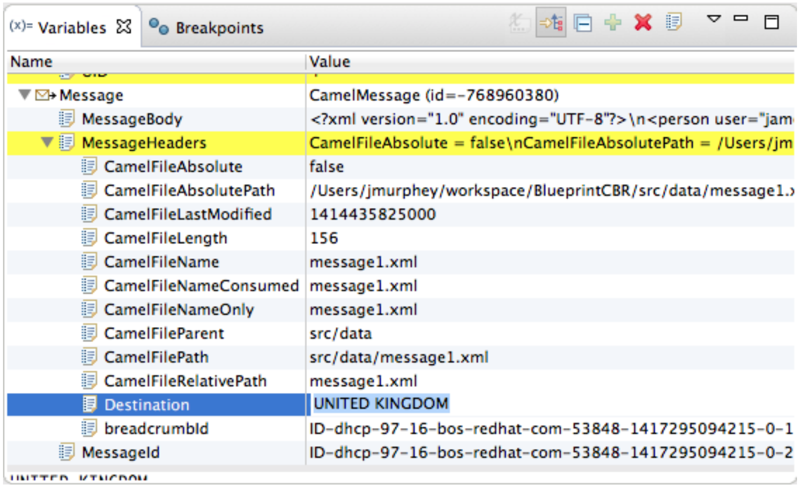
> **NOTE**
>
> Not all variables are editable. The context menu of those that are displays the option, **Change Value...**.

PROCEDURE

To change the value of a variable:

1. If necessary, start up the debugger. See Chapter 15, *Starting the Camel Debugger*.

2. In **`Variables`** view, select a variable whose value you want to change, and then click its **`Value`** field.

Name	Value
Message	CamelMessage (id=-768960380)
MessageBody	<?xml version="1.0" encoding="UTF-8"?>\n<person user="jam
MessageHeaders	CamelFileAbsolute = false\nCamelFileAbsolutePath = /Users/jm
CamelFileAbsolute	false
CamelFileAbsolutePath	/Users/jmurphey/workspace/BlueprintCBR/src/data/message1.›
CamelFileLastModified	1414435825000
CamelFileLength	156
CamelFileName	message1.xml
CamelFileNameConsumed	message1.xml
CamelFileNameOnly	message1.xml
CamelFileParent	src/data
CamelFilePath	src/data/message1.xml
CamelFileRelativePath	message1.xml
Destination	UNITED KINGDOM
breadcrumbId	ID-dhcp-97-16-bos-redhat-com-53848-1417295094215-0-1
MessageId	ID-dhcp-97-16-bos-redhat-com-53848-1417295094215-0-2

   The variable's value field turns a lighter shade of blue, indicating 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**.

   `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: ChangedTarget`**).

4. Continue stepping through the breakpoints and check whether the message is processed as expected. At each step, check **`Debug`** view, **`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 "To Debug a Routing Context" in "Tooling Tutorials"

CHAPTER 18. ADDING VARIABLES TO THE WATCH LIST

OVERVIEW

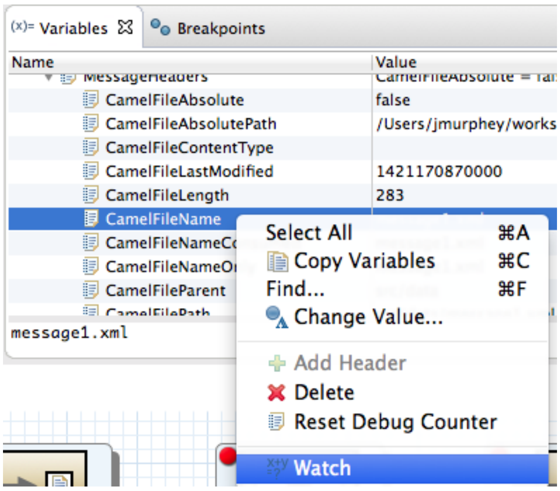
By adding them to the watch list, you can easily zero in on particular variables to see whether their values change as expected over the course of the routing context.

PROCEDURE

To add a variable to the watch list:

1. If necessary, start up the debugger. See Chapter 15, *Starting the Camel Debugger*.

2. In **`Variables`** view, right-click a variable you want to track to open the context menu.



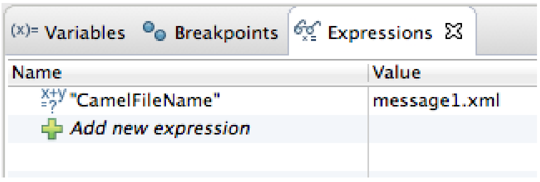
Name	Value
MessageHeaders	CamelFileAbsolute = fal
CamelFileAbsolute	false
CamelFileAbsolutePath	/Users/jmurphey/works
CamelFileContentType	
CamelFileLastModified	1421170870000
CamelFileLength	283
CamelFileName	
CamelFileNameC	
CamelFileNameO	
CamelFileParent	
CamelFilePath	

message1.xml

3. Select **Watch**.

   A new tab, **`Expressions`**, appears next to the **`Breakpoints`** tab.

4. Click the **`Expressions`** tab to open it.



Name	Value
"CamelFileName"	message1.xml
Add new expression	

   `Expressions` view displays the name of the added variable and its current value.

5. Repeat Step 2 through Step 4 to add additional variables to the Watch list.

   ### NOTE

   The variables you add will remain in the watch list until you remove them. To do so, right-click a variable in the list to open the context menu, and then click **Remove**.

6. With **`Expressions`** view open, step through the routing context to track how the value of each variable in the Watch list changes at each step in the routing context.

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 either **`Breakpoints`** view or on the canvas of the running **`CamelContext: <xxxxxx>`**. You can delete and reset a deleted breakpoint only on the canvas of the running **`CamelContext: <xxxxxx>`**.

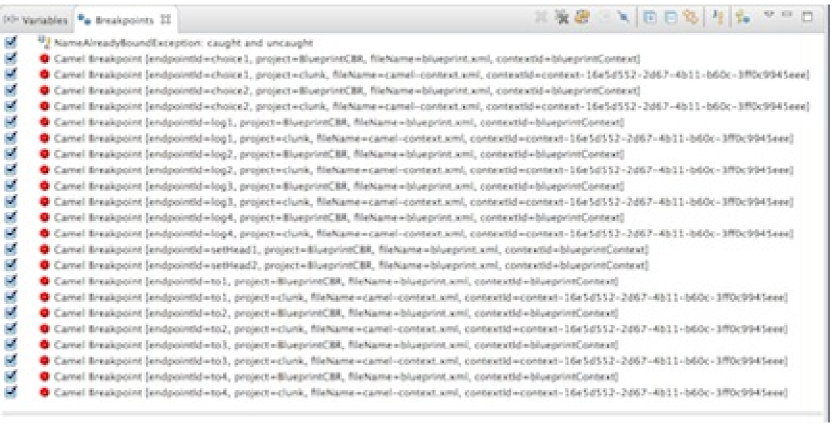
NOTE

The string *<xxxxxx>* represents the ID that you or the debugger assigned to the **`camelContext`** element in the project's routing context file.

When a breakpoint is disabled, the button causes the debugger to skip over it during the debugging session.

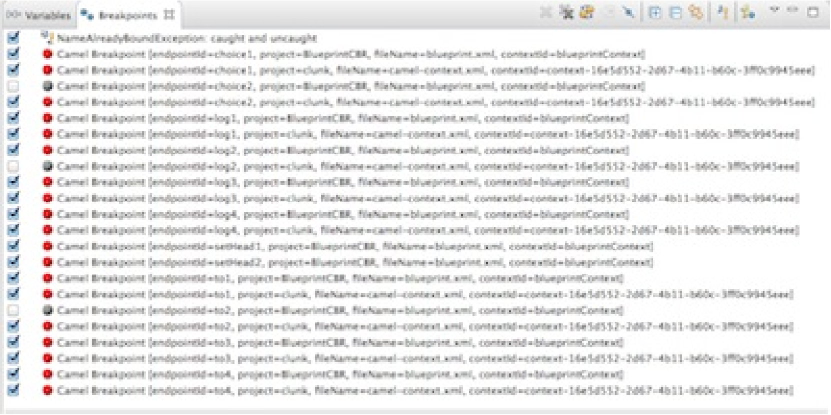
DISABLING AND ENABLING BREAKPOINTS IN BREAKPOINTS VIEW

`Breakpoints` view opens with all set breakpoints enabled.



```
☑ NameAlreadyBoundException: caught and uncaught
☑ Camel Breakpoint [endpointId=choice1, project=BlueprintCBR, fileName=blueprint.xml, contextId=blueprintContext]
☑ Camel Breakpoint [endpointId=choice1, project=clunk, fileName=camel-context.xml, contextId=context-16e5d552-2d67-4b11-b60c-3ff0c9945eee]
☑ Camel Breakpoint [endpointId=choice2, project=BlueprintCBR, fileName=blueprint.xml, contextId=blueprintContext]
☑ Camel Breakpoint [endpointId=choice2, project=clunk, fileName=camel-context.xml, contextId=context-16e5d552-2d67-4b11-b60c-3ff0c9945eee]
☑ Camel Breakpoint [endpointId=log1, project=BlueprintCBR, fileName=blueprint.xml, contextId=blueprintContext]
☑ Camel Breakpoint [endpointId=log1, project=clunk, fileName=camel-context.xml, contextId=context-16e5d552-2d67-4b11-b60c-3ff0c9945eee]
☑ Camel Breakpoint [endpointId=log2, project=BlueprintCBR, fileName=blueprint.xml, contextId=blueprintContext]
☑ Camel Breakpoint [endpointId=log2, project=clunk, fileName=camel-context.xml, contextId=context-16e5d552-2d67-4b11-b60c-3ff0c9945eee]
☑ Camel Breakpoint [endpointId=log3, project=BlueprintCBR, fileName=blueprint.xml, contextId=blueprintContext]
☑ Camel Breakpoint [endpointId=log3, project=clunk, fileName=camel-context.xml, contextId=context-16e5d552-2d67-4b11-b60c-3ff0c9945eee]
☑ Camel Breakpoint [endpointId=log4, project=BlueprintCBR, fileName=blueprint.xml, contextId=blueprintContext]
☑ Camel Breakpoint [endpointId=log4, project=clunk, fileName=camel-context.xml, contextId=context-16e5d552-2d67-4b11-b60c-3ff0c9945eee]
☑ Camel Breakpoint [endpointId=setHead1, project=BlueprintCBR, fileName=blueprint.xml, contextId=blueprintContext]
☑ Camel Breakpoint [endpointId=setHead2, project=BlueprintCBR, fileName=blueprint.xml, contextId=blueprintContext]
☑ Camel Breakpoint [endpointId=to1, project=BlueprintCBR, fileName=blueprint.xml, contextId=blueprintContext]
☑ Camel Breakpoint [endpointId=to1, project=clunk, fileName=camel-context.xml, contextId=context-16e5d552-2d67-4b11-b60c-3ff0c9945eee]
☑ Camel Breakpoint [endpointId=to2, project=BlueprintCBR, fileName=blueprint.xml, contextId=blueprintContext]
☑ Camel Breakpoint [endpointId=to2, project=clunk, fileName=camel-context.xml, contextId=context-16e5d552-2d67-4b11-b60c-3ff0c9945eee]
☑ Camel Breakpoint [endpointId=to3, project=BlueprintCBR, fileName=blueprint.xml, contextId=blueprintContext]
☑ Camel Breakpoint [endpointId=to3, project=clunk, fileName=camel-context.xml, contextId=context-16e5d552-2d67-4b11-b60c-3ff0c9945eee]
☑ Camel Breakpoint [endpointId=to4, project=BlueprintCBR, fileName=blueprint.xml, contextId=blueprintContext]
☑ Camel Breakpoint [endpointId=to4, project=clunk, fileName=camel-context.xml, contextId=context-16e5d552-2d67-4b11-b60c-3ff0c9945eee]
```

To disable a breakpoint, clear its check box.



```
☑ NameAlreadyBoundException: caught and uncaught
☑ Camel Breakpoint [endpointId=choice1, project=BlueprintCBR, fileName=blueprint.xml, contextId=blueprintContext]
☑ Camel Breakpoint [endpointId=choice1, project=clunk, fileName=camel-context.xml, contextId=context-16e5d552-2d67-4b11-b60c-3ff0c9945eee]
☐ Camel Breakpoint [endpointId=choice2, project=BlueprintCBR, fileName=blueprint.xml, contextId=blueprintContext]
☑ Camel Breakpoint [endpointId=choice2, project=clunk, fileName=camel-context.xml, contextId=context-16e5d552-2d67-4b11-b60c-3ff0c9945eee]
☑ Camel Breakpoint [endpointId=log1, project=BlueprintCBR, fileName=blueprint.xml, contextId=blueprintContext]
☑ Camel Breakpoint [endpointId=log1, project=clunk, fileName=camel-context.xml, contextId=context-16e5d552-2d67-4b11-b60c-3ff0c9945eee]
☑ Camel Breakpoint [endpointId=log2, project=BlueprintCBR, fileName=blueprint.xml, contextId=blueprintContext]
☐ Camel Breakpoint [endpointId=log2, project=clunk, fileName=camel-context.xml, contextId=context-16e5d552-2d67-4b11-b60c-3ff0c9945eee]
☑ Camel Breakpoint [endpointId=log3, project=BlueprintCBR, fileName=blueprint.xml, contextId=blueprintContext]
☑ Camel Breakpoint [endpointId=log3, project=clunk, fileName=camel-context.xml, contextId=context-16e5d552-2d67-4b11-b60c-3ff0c9945eee]
☑ Camel Breakpoint [endpointId=log4, project=BlueprintCBR, fileName=blueprint.xml, contextId=blueprintContext]
☑ Camel Breakpoint [endpointId=log4, project=clunk, fileName=camel-context.xml, contextId=context-16e5d552-2d67-4b11-b60c-3ff0c9945eee]
☑ Camel Breakpoint [endpointId=setHead1, project=BlueprintCBR, fileName=blueprint.xml, contextId=blueprintContext]
☑ Camel Breakpoint [endpointId=setHead2, project=BlueprintCBR, fileName=blueprint.xml, contextId=blueprintContext]
☑ Camel Breakpoint [endpointId=to1, project=BlueprintCBR, fileName=blueprint.xml, contextId=blueprintContext]
☑ Camel Breakpoint [endpointId=to1, project=clunk, fileName=camel-context.xml, contextId=context-16e5d552-2d67-4b11-b60c-3ff0c9945eee]
☐ Camel Breakpoint [endpointId=to2, project=BlueprintCBR, fileName=blueprint.xml, contextId=blueprintContext]
☑ Camel Breakpoint [endpointId=to2, project=clunk, fileName=camel-context.xml, contextId=context-16e5d552-2d67-4b11-b60c-3ff0c9945eee]
☑ Camel Breakpoint [endpointId=to3, project=BlueprintCBR, fileName=blueprint.xml, contextId=blueprintContext]
☑ Camel Breakpoint [endpointId=to3, project=clunk, fileName=camel-context.xml, contextId=context-16e5d552-2d67-4b11-b60c-3ff0c9945eee]
☑ Camel Breakpoint [endpointId=to4, project=BlueprintCBR, fileName=blueprint.xml, contextId=blueprintContext]
☑ Camel Breakpoint [endpointId=to4, project=clunk, fileName=camel-context.xml, contextId=context-16e5d552-2d67-4b11-b60c-3ff0c9945eee]
```

For each breakpoint you disable, **`Console`** view displays an INFO level log entry noting that it has been disabled (for example, **`Removing breakpoint to3`**). Likewise, for each breakpoint you re-enable, **`Console`** view displays an INFO level log entry noting that it has been enabled (for example, **`Adding breakpoint to3`**).

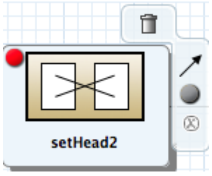
NOTE

To re-enable a disabled breakpoint, click its check box. **`Console`** view displays an INFO level log entry noting that the breakpoint has been added to the selected node.

DISABLING AND ENABLING BREAKPOINTS ON THE RUNNING CAMELCONTEXT: CANVAS

Make sure the routing context running in memory, **`CamelContext: <xxxxxx>`**, is active on the canvas.

1. On the canvas, select the node with the breakpoint you want to disable; for example:

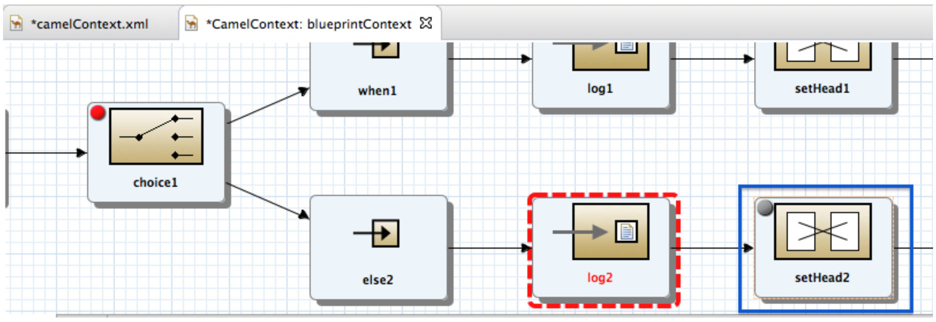


NOTE

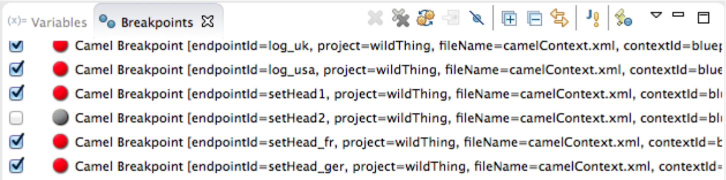
The 🔴 icon denotes an unconditional breakpoint. A 🟠 icon denotes a conditional breakpoint.

2. Click the ● icon in the node's side bar.

   The ● icon replaces the 🔴 icon in the node's upper left corner, indicating that the breakpoint is now disabled:

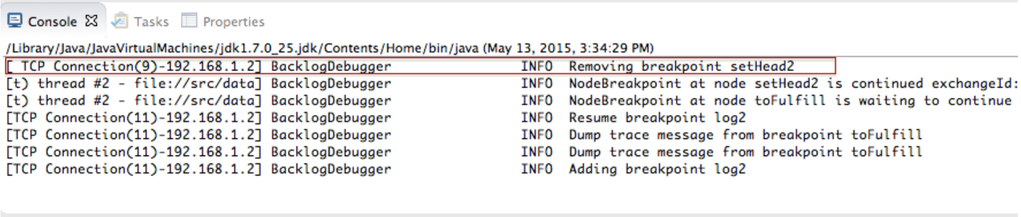


   The node's check box in **`Breakpoints`** view is clear, also indicating that the breakpoint is disabled:



☑ 🔴 Camel Breakpoint [endpointId=log_uk, project=wildThing, fileName=camelContext.xml, contextId=bluep
☑ 🔴 Camel Breakpoint [endpointId=log_usa, project=wildThing, fileName=camelContext.xml, contextId=blue
☑ 🔴 Camel Breakpoint [endpointId=setHead1, project=wildThing, fileName=camelContext.xml, contextId=bl
□ ● Camel Breakpoint [endpointId=setHead2, project=wildThing, fileName=camelContext.xml, contextId=bl
☑ 🔴 Camel Breakpoint [endpointId=setHead_fr, project=wildThing, fileName=camelContext.xml, contextId=b
☑ 🔴 Camel Breakpoint [endpointId=setHead_ger, project=wildThing, fileName=camelContext.xml, contextId=

   `Console` view contains a log of the event:



```
/Library/Java/JavaVirtualMachines/jdk1.7.0_25.jdk/Contents/Home/bin/java (May 13, 2015, 3:34:29 PM)
[ TCP Connection(9)-192.168.1.2] BacklogDebugger               INFO  Removing breakpoint setHead2
[t) thread #2 - file://src/data] BacklogDebugger               INFO  NodeBreakpoint at node setHead2 is continued exchangeId:
[t) thread #2 - file://src/data] BacklogDebugger               INFO  NodeBreakpoint at node toFulfill is waiting to continue
[TCP Connection(11)-192.168.1.2] BacklogDebugger               INFO  Resume breakpoint log2
[TCP Connection(11)-192.168.1.2] BacklogDebugger               INFO  Dump trace message from breakpoint toFulfill
[TCP Connection(11)-192.168.1.2] BacklogDebugger               INFO  Dump trace message from breakpoint toFulfill
[TCP Connection(11)-192.168.1.2] BacklogDebugger               INFO  Adding breakpoint log2
```

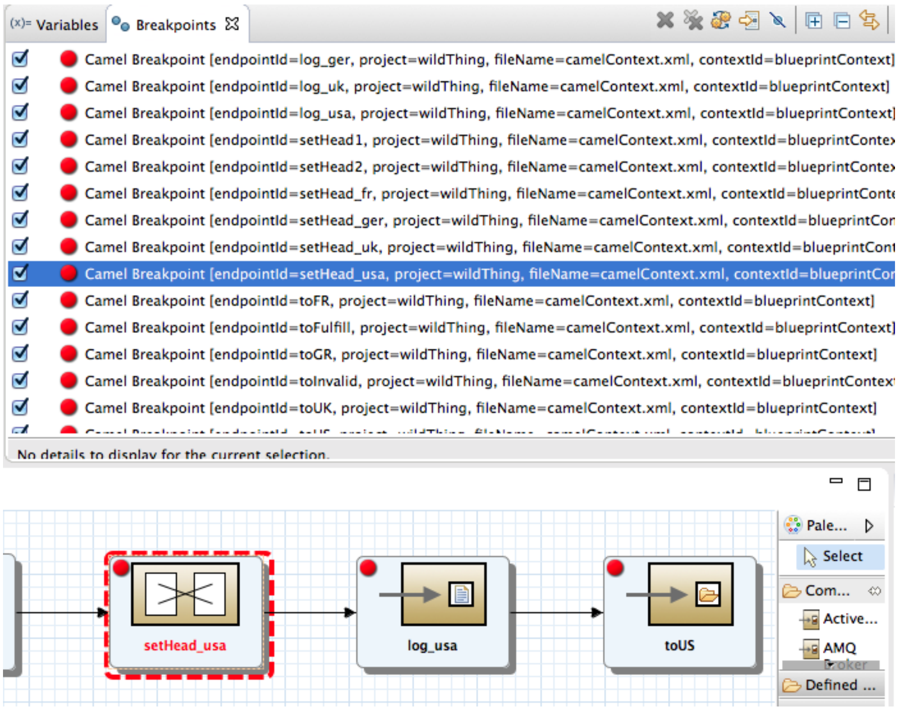
NOTE

To re-enable the breakpoint, select the node, and then click the ● icon in its side bar.

DELETING AND RESETTING BREAKPOINTS ON THE RUNNING CAMELCONTEXT: CANVAS

You can delete and reset deleted breakpoints only on the running **`CamelContext: <xxxxxx>`** canvas.

1. On the **`CamelContext: <xxxxxx>`** canvas, select the node whose breakpoint you want to delete; for example, **`setHead_usa`**:



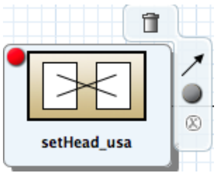
```
☑ ● Camel Breakpoint [endpointId=log_ger, project=wildThing, fileName=camelContext.xml, contextId=blueprintContext]
☑ ● Camel Breakpoint [endpointId=log_uk, project=wildThing, fileName=camelContext.xml, contextId=blueprintContext]
☑ ● Camel Breakpoint [endpointId=log_usa, project=wildThing, fileName=camelContext.xml, contextId=blueprintContext]
☑ ● Camel Breakpoint [endpointId=setHead1, project=wildThing, fileName=camelContext.xml, contextId=blueprintConte
☑ ● Camel Breakpoint [endpointId=setHead2, project=wildThing, fileName=camelContext.xml, contextId=blueprintConte
☑ ● Camel Breakpoint [endpointId=setHead_fr, project=wildThing, fileName=camelContext.xml, contextId=blueprintConte
☑ ● Camel Breakpoint [endpointId=setHead_ger, project=wildThing, fileName=camelContext.xml, contextId=blueprintCor
☑ ● Camel Breakpoint [endpointId=setHead_uk, project=wildThing, fileName=camelContext.xml, contextId=blueprintCont
☑ ● Camel Breakpoint [endpointId=setHead_usa, project=wildThing, fileName=camelContext.xml, contextId=blueprintCor
☑ ● Camel Breakpoint [endpointId=toFR, project=wildThing, fileName=camelContext.xml, contextId=blueprintContext]
☑ ● Camel Breakpoint [endpointId=toFulfill, project=wildThing, fileName=camelContext.xml, contextId=blueprintContext]
☑ ● Camel Breakpoint [endpointId=toGR, project=wildThing, fileName=camelContext.xml, contextId=blueprintContext]
☑ ● Camel Breakpoint [endpointId=toInvalid, project=wildThing, fileName=camelContext.xml, contextId=blueprintContex
☑ ● Camel Breakpoint [endpointId=toUK, project=wildThing, fileName=camelContext.xml, contextId=blueprintContext]
☑ ● Camel Breakpoint [endpointId=toUS, project=wildThing, fileName=camelContext.xml, contextId=blueprintContext]
No details to display for the current selection.
```

Note that the node's breakpoint is enabled in **`Breakpoints`** view.

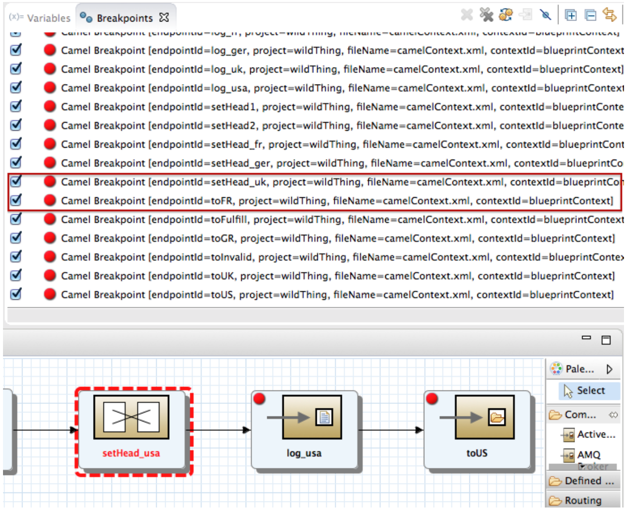
NOTE

In the above graphic, the dotted red line surrounding the **`setHead_usa`** node identifies the breakpoint on which the debugger is currently suspended.

2. Click the ⊗ icon in the node's side bar:



The breakpoint is removed from the node on the canvas and also from the list of breakpoints in **`Breakpoints`** view:



```
☑ ● Camel Breakpoint [endpointId=log_fr, project=wildThing, fileName=camelContext.xml, contextId=blueprintContext]
☑ ● Camel Breakpoint [endpointId=log_ger, project=wildThing, fileName=camelContext.xml, contextId=blueprintContext
☑ ● Camel Breakpoint [endpointId=log_uk, project=wildThing, fileName=camelContext.xml, contextId=blueprintContext]
☑ ● Camel Breakpoint [endpointId=log_usa, project=wildThing, fileName=camelContext.xml, contextId=blueprintContext
☑ ● Camel Breakpoint [endpointId=setHead1, project=wildThing, fileName=camelContext.xml, contextId=blueprintConte
☑ ● Camel Breakpoint [endpointId=setHead2, project=wildThing, fileName=camelContext.xml, contextId=blueprintConte
☑ ● Camel Breakpoint [endpointId=setHead_fr, project=wildThing, fileName=camelContext.xml, contextId=blueprintCont
☑ ● Camel Breakpoint [endpointId=setHead_ger, project=wildThing, fileName=camelContext.xml, contextId=blueprintCo
☑ ● Camel Breakpoint [endpointId=setHead_uk, project=wildThing, fileName=camelContext.xml, contextId=blueprintCo
☑ ● Camel Breakpoint [endpointId=toFR, project=wildThing, fileName=camelContext.xml, contextId=blueprintContext]
☑ ● Camel Breakpoint [endpointId=toFulfill, project=wildThing, fileName=camelContext.xml, contextId=blueprintContext
☑ ● Camel Breakpoint [endpointId=toGR, project=wildThing, fileName=camelContext.xml, contextId=blueprintContext]
☑ ● Camel Breakpoint [endpointId=toInvalid, project=wildThing, fileName=camelContext.xml, contextId=blueprintContex
☑ ● Camel Breakpoint [endpointId=toUK, project=wildThing, fileName=camelContext.xml, contextId=blueprintContext]
☑ ● Camel Breakpoint [endpointId=toUS, project=wildThing, fileName=camelContext.xml, contextId=blueprintContext]
```

NOTE

To reset the breakpoint, select the node, and then click the ● or ● icon in its side bar, depending on whether you want to set an unconditional or conditional breakpoint.

RELATED TOPICS

Chapter 14, *Setting Breakpoints*

the section called “Disabling breakpoints”

the section called “Deleting breakpoints”

chapter "To Debug a Routing Context" in "Tooling Tutorials"

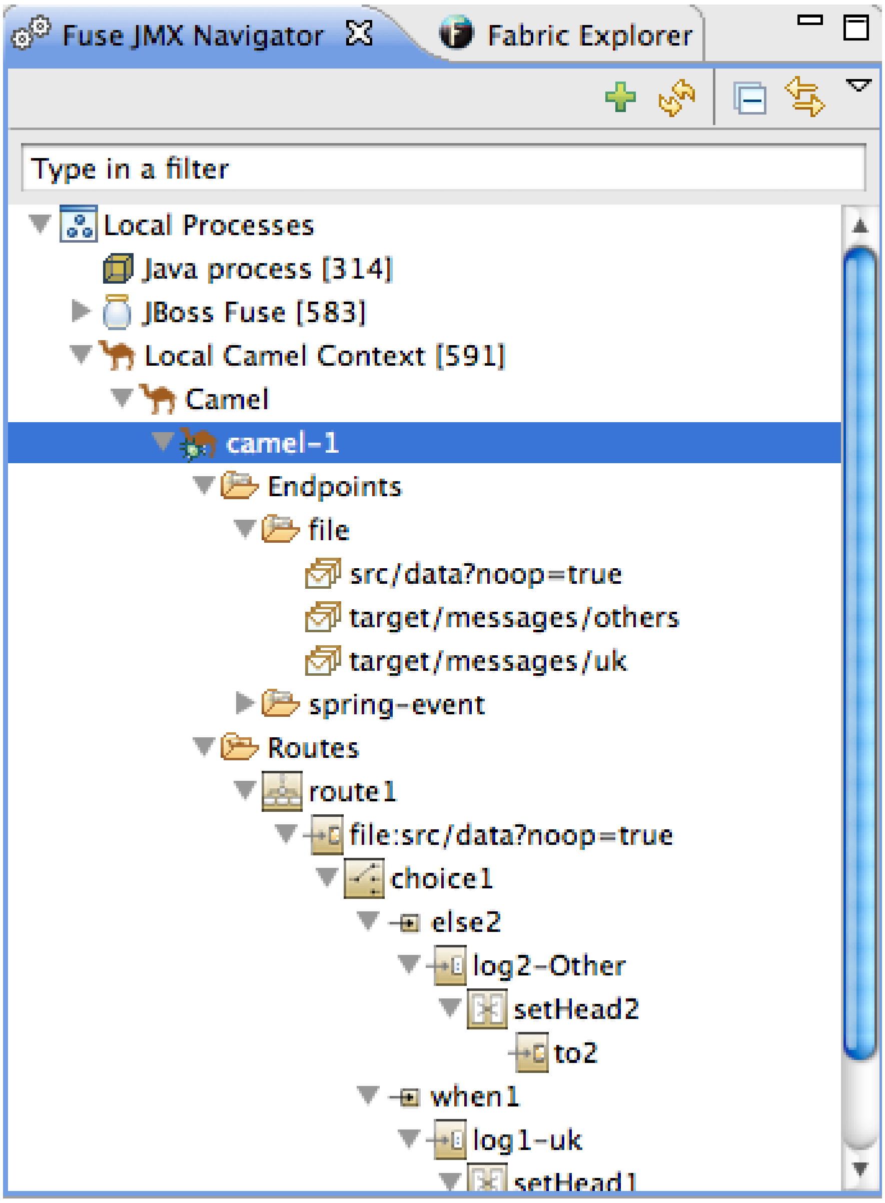
PART IV. MONITORING AND TESTING APPLICATIONS

To open `Fuse Integration` perspective (Chapter 3, *Fuse Integration Perspective*), click on the right side of the JBoss 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.

CHAPTER 20. JMX NAVIGATOR

`JMX Navigator`, shown in Figure 20.1, displays all of the processes running in your application and 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 JMX Navigator. JMX Navigator's context menu also provides the commands needed to activate route tracing and to add JMS destinations.

Figure 20.1. JMX Navigator



```
Local Processes
    Java process [314]
    JBoss Fuse [583]
    Local Camel Context [591]
        Camel
            camel-1
                Endpoints
                    file
                        src/data?noop=true
                        target/messages/others
                        target/messages/uk
                    spring-event
                Routes
                    route1
                        file:src/data?noop=true
                            choice1
                                else2
                                    log2-Other
                                        setHead2
                                            to2
                                when1
                                    log1-uk
                                        setHead1
```

By default, **`JMX Navigator`** discovers all of the JMX servers running on the local machine and lists them under the lists them following categories:

- Local Processes
- Server Connections
- User-defined connections

You can add other JMX servers using a server's JMX URL.

20.1. VIEWING PROCESSES IN JMX

Overview

`JMX Navigator` 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 of the JMX servers 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. Expand the **`Local Processes`** entry in **`JMX Navigator`**.
2. Double-click one of the top-level entries under **`Local Processes`** to connect to the JMX server.
3. Double-click it to open a connection.
4. Click the ▶ icon that appears next to the entry to display the list of all components running in the JVM.

Viewing processes in alternate JMX servers

To view information about processes in an alternate JMX server:

1. Add the JMX server to **`JMX Navigator`**.
2. Expand the server's entry in **`JMX Navigator`** using the ▶ icon that appears next to the entry to display the list of all components running in the JVM.

20.2. ADDING A JMX SERVER

Overview

`JMX Navigator` lists all of the local JMX servers under the **`Local Processes`** branch of the tree. You may need to connect to specific JMX servers to see components deployed on other machines.

To add a JMX server you need to know the JMX URL of the server you want to add.

Procedure

To add a JMX server to the **`JMX Navigator`**:

1. Click **`New Connection`** on the **`JMX Navigator`** tab .
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 is displayed as a branch in the **`JMX Navigator`** tree.

CHAPTER 21. VIEWING A COMPONENT'S PROPERTIES

OVERVIEW

The tooling collects and displays all of the JMX properties reported by Fuse components. This information can provide significant insight into what is happening in your integration application.

PROCEDURE

To see a Fuse component's properties:

1. Locate the node for the component in the Fuse JMX Navigator.

   You may have to expand nodes on the tree to locate low-level components.

2. Select the Fuse component's node from the tree.

3. Open the **`Properties`** view.

   The selected component's properties will be displayed in table format.

   You can use the **`Search`** box in **`Properties`** view to locate specific properties based on value.

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. Select the JMS destination or endpoint from Fuse JMX Navigator.

   The list of messages is displayed in the **`Messages View`**.

2. Select an individual message to inspect from the list of messages in the **`Messages View`**.

   Message details and content are displayed in the **`Properties`** view.

RELATED TOPICS

Section 23.3, “Tracing messages through a route”

CHAPTER 23. TRACING ROUTES

Debugging a route usually 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 these problems.

The route tracing feature of the tooling 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.

`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.

`Messages View` displays the messages processed by a JMS destination or route endpoint selected in the **`JMX Navigator`** tree. Selecting an individual message in **`Messages View`** displays the full details and content of the message in **`Properties`** view.

Tracing messages through a route involves the following steps:

1. Creating test messages for route tracing
2. Activating route tracing
3. Tracing messages through a route
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 message, leaving the structuring of the message entirely up to you.

Procedure

To create a test message:

1. In **`Project Explorer`**, right-click the project to open the context menu.
2. Select **New** → **Fuse Message** to open the **`New File`** wizard.
3. Enter or select the folder in which you want to store the message.
4. In the **`File name`** field, enter a name for the message.
5. Click **`Finish`**.

   The new message opens in the XML editor.

6. Fill in the contents (body and header text) of the message.

7. On the menu bar, click **File** → **Save** to save the test message.

Related topics

Section 23.3, “Tracing messages through a route”
New Fuse Message
chapter "To Trace a Message Through a Route" in "Tooling Tutorials"

23.2. ACTIVATING ROUTE TRACING

Overview

Before you can trace messages through a route, you must activate route tracing for the route's routing context.

Procedure

To start tracing on a routing context:

1. In the Fuse JMX Navigator tree, select the routing context on which you want to start tracing.

   You can select any route in the context to start tracing on the entire context.

2. Right-click it 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 route”
Section 23.4, “Deactivating route tracing”

23.3. TRACING MESSAGES THROUGH A ROUTE

Overview

The best way to see what's happening in a route is to watch what happens to a message at each stop along the route. The tooling provides a mechanism for dropping messages into a route and tracing the message's path through it.

Procedure

To trace messages through a route:

1. Create one or more test messages as described in Section 23.1, “Creating test messages for route tracing”.

2. Activate tracing for the route's routing context as described in Section 23.2, “Activating route tracing”.

3. Drag one of the test messages onto the route's starting point.

4. In the Fuse JMX Navigator tree, select the route being traced.

   The **`Messages View`** is populated with a list of messages representing the message at each stage in the traced route.

5. Open **`Diagram View`** to see a graphical representation of the selected route.

6. In the **`Messages View`**, select one of the messages.

   In **`Diagram View`**, the route step in which the selected message is sitting is highlighted. The **`Properties`** view displays the full details and content of the test message.

You can repeat this process 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”
chapter "To Trace a Message Through a Route" in "Tooling Tutorials"

23.4. DEACTIVATING ROUTE TRACING

Overview

When you are finished debugging the routes in a routing context, you should deactivate tracing for the routing context.

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 Fuse JMX Navigator tree, select the routing context for which you want to deactivate tracing.

   You can select any route in the context to stop tracing for the context.

2. Right-click it 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 route”

CHAPTER 24. MANAGING JMS DESTINATIONS

The tooling's **`Fuse JMX Navigator`** allows you to easily add or delete JMS destinations from a running Red Hat JBoss Fuse A-MQ.

> **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. The tooling provides a menu choice for adding a destination with a single click.

Procedure

To add a JMS destination to a broker:

1. In the **`Fuse JMX Navigator`** tree, select either the **`Queues`** child or the **`Topics`** child in the broker node where you want to add a destination.

2. Right-click it to open the context menu, and then select **Create Queue**/**Topic**.

3. In the naming dialog, enter a name for the destination.

4. Click **`OK`**.

   The new destination appears in the tree.

Related topics

Section 24.2, “Deleting a JMS destination”

24.2. DELETING A JMS DESTINATION

Overview

When testing fail over scenarios or other scenarios that involve failure handling, it's convenient to be able to easily remove a JMS destination. The tooling has a menu option that allows you to remove a destination with a single click.

Procedure

To delete a JMS destination:

1. In the **`Fuse JMX Navigator`** tree, select the JMS destination you want to delete.

2. Right-click it 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

`JMX Navigator` allows you to easily add or delete routing endpoints.

IMPORTANT

These changes are not persistent across router restarts.

25.1. ADDING A ROUTING ENDPOINT

Overview

When testing a new scenario, it is convenient to add a new endpoint to a routing context. The tooling provides a menu choice for adding a routing endpoint with a single click.

Procedure

To add an endpoint to a routing context:

1. In the **`JMX Navigator`** tree, select the **`Endpoints`** child in the routing context node where you want to add an endpoint.

2. Right-click it to open the context menu, and then select **Create Endpoint**.

3. In the naming dialog, enter a URL defining the new endpoint.

4. Click **`OK`**.

   The new destination appears in the tree.

Related topics

Section 25.2, “Deleting a routing endpoint”

25.2. DELETING A ROUTING ENDPOINT

Overview

When testing fail over scenarios or other scenarios that involve failure handling, it is convenient to be able to easily remove an endpoint from a routing context. The tooling has a menu option that allows you to remove an endpoint with a single click.

Procedure

To delete a routing endpoint:

1. In the **`JMX Navigator`** tree, select the endpoint you want delete.

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

Related topics

Section 25.1, “Adding a routing endpoint”

CHAPTER 26. EDITING RUNNING ROUTES

OVERVIEW

If you want to experiment with changes to a running route, you can make the edits directly from the Fuse Integration perspective. The tooling opens the context file containing the route in the route editor. When your changes are saved, they take effect immediately.

The changes are made to the in-memory model of the running context, but are not persisted as part of their original project or any other project.

NOTE

If you want to save the changes permanently, you must use the **Save As** menu option.

PROCEDURE

To edit a running route:

1. In the `JMX Navigator` tree, select the context containing the route(s) you want to edit.
2. Right-click to open the context menu, and select **Edit Routes**.

   Route editor displays the context and all of its routes.
3. Edit the route(s) as described in Part II, “Developing Applications”.
4. When done, save the edited route.
   - Select **File** → **Save** to update the in-memory version of the routing context.

     Changes take effect immediately.
   - >Select **File** → **Save As** to save the edited routing context in a new context file.

RELATED TOPICS

Section 7.1, “Adding routes to the routing context”
Section 7.2, “Adding patterns to a route”
Section 7.4, “Configuring a pattern”
Section 7.3, “Connecting patterns to make a route”
Section 7.5, “Rearranging patterns on the canvas”
Section 7.6, “Removing patterns from a route”
Section 7.7, “Disconnecting two patterns”

Section 7.8, “Deleting a route”

CHAPTER 27. MANAGING ROUTING CONTEXTS

`JMX Navigator` enables you to easily suspend and resume routing contexts.

27.1. SUSPENDING A ROUTING CONTEXT

Overview

The tooling enables you to suspend the operation of a routing context from `JMX Navigator`. Suspending a context gracefully shuts down all of the routes in the context, but keeps them loaded in memory so that they can easily be resumed.

Procedure

To suspend a routing context:

1. In the `JMX Navigator` tree, expand the project's `Camel Contexts` node, and select the routing context you want to suspend.

2. Right-click it to open the context menu, and then select **Suspend Camel Context**.

> **NOTE**
>
> If **Resume Camel Context** appears on the context menu, the context is already suspended.

Related topics

Section 27.2, “Resuming a routing context”

27.2. RESUMING A ROUTING CONTEXT

Overview

The tooling enables you to resume a suspended routing context from `JMX Navigator`. Resuming a context restarts all of the routes in it, so that they can process messages.

Procedure

To resume a routing context:

1. In the `JMX Navigator` tree, expand the project's `Camel Contexts` node, and select the routing context you want to resume.

2. Right-click it 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 a routing context”

PART V. PUBLISHING APPLICATIONS TO A CONTAINER

To publish Fuse 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 Fuse projects to the server runtime and set the publishing options for it.

You manage servers and publish Fuse projects to them in **`Fuse Integration`** perspective (for an overview, see Chapter 3, *Fuse Integration Perspective*). To open **`Fuse Integration`** perspective, click on the right side of the JBoss Developer Studio tool bar. If the icon does not appear on the tool bar, click and then select **Fuse Integration** from the list of available perspectives.

CHAPTER 28. MANAGING SERVERS

The **Servers** panel enables you to run and manage servers in your Red Hat Developer Studio environment. The supported servers are:

- Red Hat JBoss Fuse Server—versions 6.x
- Apache Karaf Server—versions 2.x, 3.0
- Apache ServiceMix Server—versions 4.5, 5.x

NOTE

For step by step instructions on how to publish a Camel project to Red Hat JBoss Fuse, see Red Hat JBoss Fuse Tooling Tutorials, To Publish a Fuse Project to Red Hat JBoss Fuse.

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 **Servers** panel, where you can connect to it and publish your Fuse projects.

NOTE

If adding a Red Hat JBoss Fuse 6.0 or 6.1 server, it's 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

You can add a new server to the **Servers** panel in three ways:

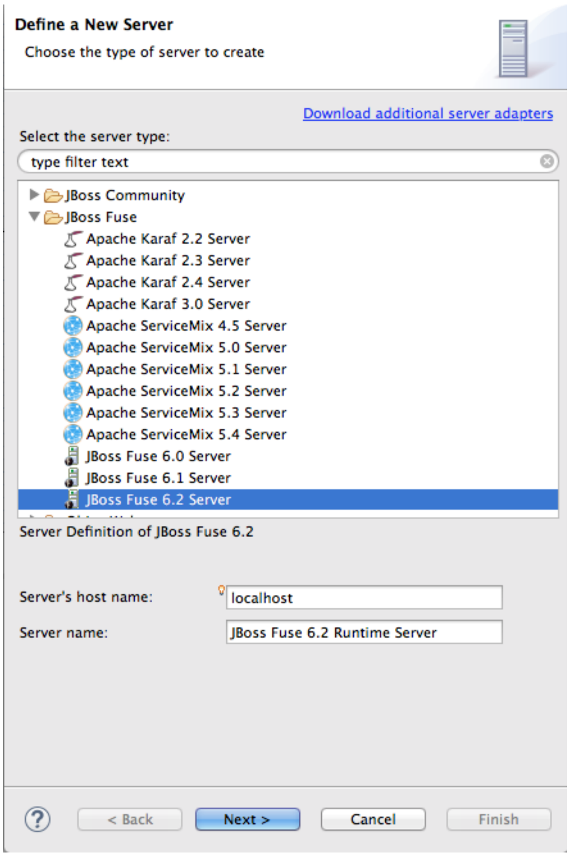
- In **Servers** panel, click the link **No servers are available. Click this link to create a new server....**

  #### NOTE

  This link appears in **Servers** panel only when no server has been defined.

- Right-click in **Servers** panel to open the context menu, then select **New** → **Server**.
- On the menu bar, select **File** → **New** → **Other** → **Server** → **Server**.

1. In the **Define a New Server** dialog, expand the **JBoss Fuse** node to expose the list of available server options:



Define a New Server

Choose the type of server to create

Select the server type:

type filter text

- JBoss Community
- JBoss Fuse
  - Apache Karaf 2.2 Server
  - Apache Karaf 2.3 Server
  - Apache Karaf 2.4 Server
  - Apache Karaf 3.0 Server
  - Apache ServiceMix 4.5 Server
  - Apache ServiceMix 5.0 Server
  - Apache ServiceMix 5.1 Server
  - Apache ServiceMix 5.2 Server
  - Apache ServiceMix 5.3 Server
  - Apache ServiceMix 5.4 Server
  - JBoss Fuse 6.0 Server
  - JBoss Fuse 6.1 Server
  - JBoss Fuse 6.2 Server

Server Definition of JBoss Fuse 6.2

Server's host name: localhost

Server name: JBoss Fuse 6.2 Runtime Server

2. Click the server you want to add.

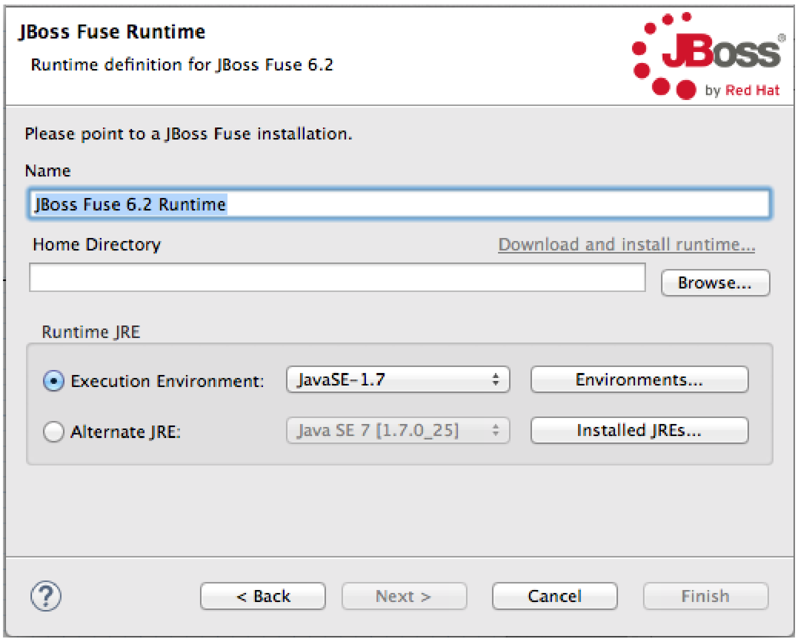
3. In **Server's host name**, accept the default (**localhost**).

NOTE

The address of **localhost** is **0.0.0.0**.

4. In **Server name**, accept the default, or enter a different name for the runtime server.

5. Click **Next >** to open the server's **Runtime definition** page:



JBoss Fuse Runtime

Runtime definition for JBoss Fuse 6.2

Please point to a JBoss Fuse installation.

Name: JBoss Fuse 6.2 Runtime

Home Directory:

Runtime JRE

● Execution Environment: JavaSE-1.7

○ Alternate JRE: Java SE 7 [1.7.0_25]

NOTE

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

6. Accept the default for **`Name`**.

7. In **`Home Directory`**, enter the path where the server runtime is installed, or click **`Browse`** to find and select it.

8. Select the runtime JRE from the drop-down menu next to **`Execution Environment`**.

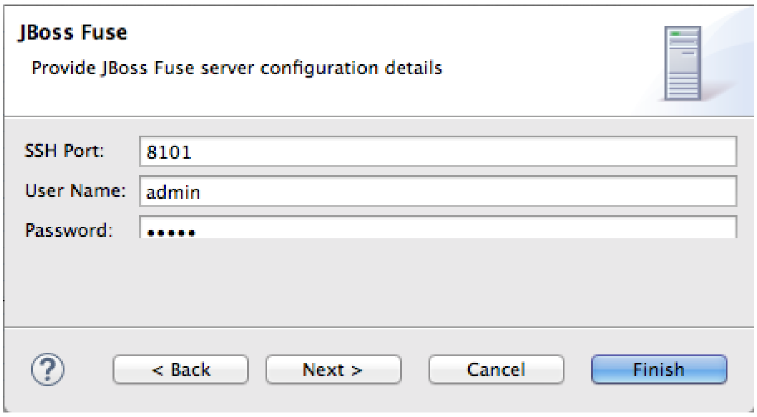
   If the version you want does not appear on the list, click the **`Environments`** button and select the version from that list. The JRE version you select must be installed on your machine.

NOTE

JBoss Fuse 6.2 requires either JRE versions 1.7 or 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:



JBoss Fuse

Provide JBoss Fuse server configuration details

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

    In the case of Red Hat JBoss Fuse, this is a user name stored in the Red Hat JBoss Fuse *installDir***`/etc/users.properties`** file.

> **NOTE**
>
> If the default user has been activated (uncommented) in the **/etc/users.properties** file, the tooling autofills **User Name** and **Password** with the default user's name and password, as shown in Step 10. The default user is already activated in JBoss Fuse 6.2, but not in JBoss Fuse 6.0 or 6.1.

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

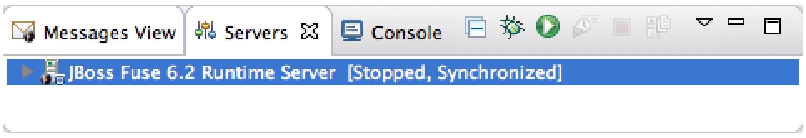
- **jaas:realms**—to list the realms
- **jaas:manage --index 1**—to edit the first (server) realm
- **jaas:useradd <username> <password>**—to add a user and associated password
- **jaas:roleadd <username> Administrator**—to specify the new user's role
- **jaas:update**—to update the realm with the new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

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

13. In **Password**, enter the password required for **User Name** to log into the server.

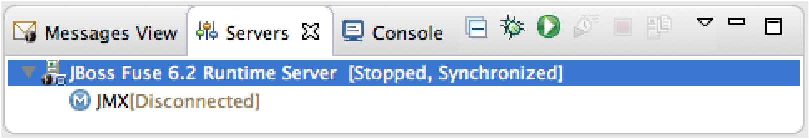
14. Click **Finish** to save the server's configuration details.

    The server runtime will appear in **Servers** panel. For example:



JBoss Fuse 6.2 Runtime Server [Stopped, Synchronized]

Expanding the server node exposes the server's JMX node:



JBoss Fuse 6.2 Runtime Server [Stopped, Synchronized]
JMX[Disconnected]

Related topics

Define a New Server
New Server Configuration
Add and Remove

28.2. STARTING A SERVER

Overview

When you start a configured server, the tooling opens the server's remote management console in a **`Shell`** panel. This allows you to easily manage the container while testing your application.

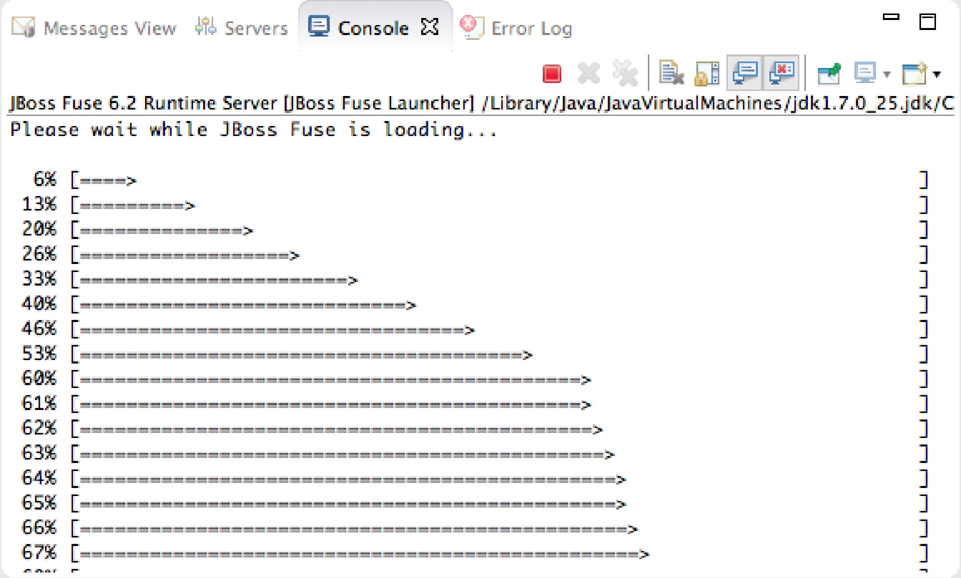
Procedure

To start a server:

1. In the **`Servers`** panel, select the server you want to start.

2. Click ▶ .

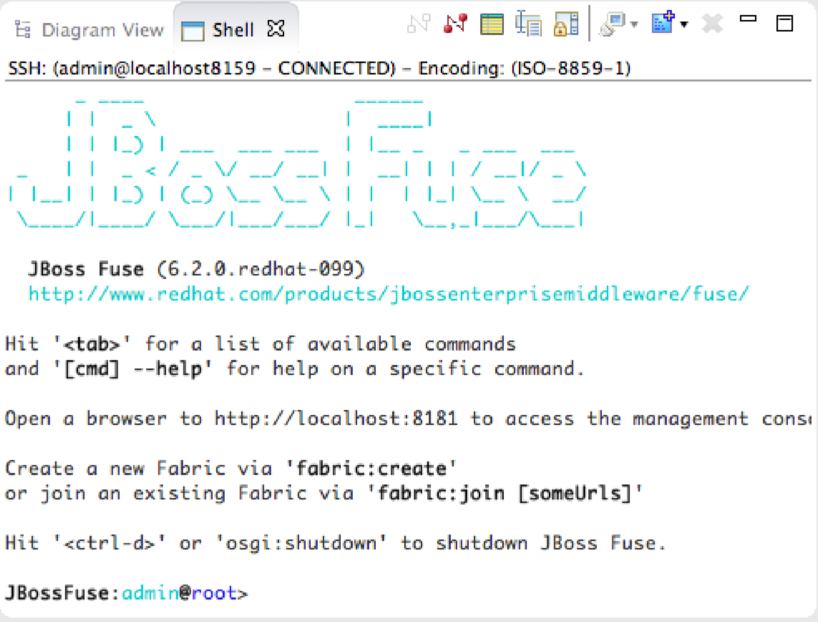
   - **`Console`** view opens and displays a message asking you to wait while the container is starting; for example:



```
JBoss Fuse 6.2 Runtime Server [JBoss Fuse Launcher] /Library/Java/JavaVirtualMachines/jdk1.7.0_25.jdk/C
Please wait while JBoss Fuse is loading...

  6% [====>                                                                          ]
 13% [=========>                                                                     ]
 20% [==============>                                                                ]
 26% [==================>                                                            ]
 33% [=======================>                                                       ]
 40% [============================>                                                  ]
 46% [=================================>                                             ]
 53% [======================================>                                        ]
 60% [===========================================>                                   ]
 61% [===========================================>                                   ]
 62% [============================================>                                  ]
 63% [=============================================>                                 ]
 64% [==============================================>                                ]
 65% [==============================================>                                ]
 66% [===============================================>                               ]
 67% [================================================>                              ]
```

> **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”.

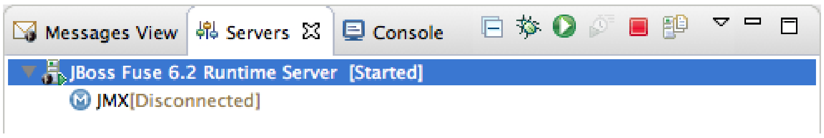
   - After the container has started up, **`Shell`** view opens to display the container's management console; for example:



SSH: (admin@localhost8159 – CONNECTED) – Encoding: (ISO-8859-1)

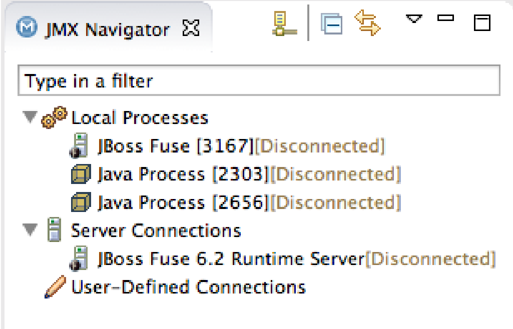
```
      _ ____                  ______
     | |  _ \                |  ____|
     | | |_) | ___  ___ ___  | |__ _   _ ___  ___
 _   | |  _ < / _ \/ __/ __| |  __| | | / __|/ _ \
| |__| | |_) | (_) \__ \__ \ | |  | |_| \__ \  __/
 \____/|____/ \___/|___/___/ |_|   \__,_|___/\___|

  JBoss Fuse (6.2.0.redhat-099)
  http://www.redhat.com/products/jbossenterprisemiddleware/fuse/

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

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

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

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

JBossFuse:admin@root>
```

- The running server appears in **Servers** panel:



```
JBoss Fuse 6.2 Runtime Server  [Started]
    JMX[Disconnected]
```

- The running server also appears in **JMX Navigator** under **Server Connections**:



```
Local Processes
    JBoss Fuse [3167][Disconnected]
    Java Process [2303][Disconnected]
    Java Process [2656][Disconnected]
Server Connections
    JBoss Fuse 6.2 Runtime Server[Disconnected]
User-Defined Connections
```

28.3. CONNECTING TO A RUNNING SERVER

Overview

After you start up a configured server, it appears in **Servers** panel and in **JMX Navigator**, under the **ServerConnections** 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 **Servers** panel or in **JMX Navigator**.

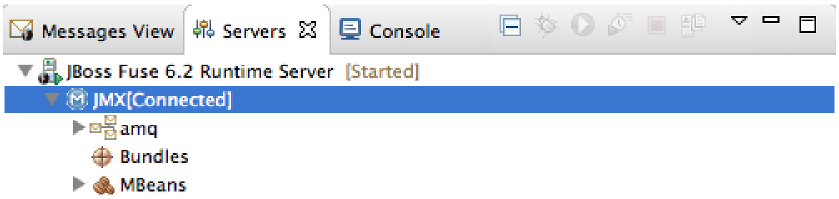
NOTE

Servers panel and **JMX Navigator** are synchronized with regards to server connections, so connecting to a server in **Servers** panel also connects it in **JMX Navigator**, and vice versa.

Connecting to a running server in Servers panel

1. In **Servers** panel, expand the server runtime to expose its **JMX[Disconnected]** node.

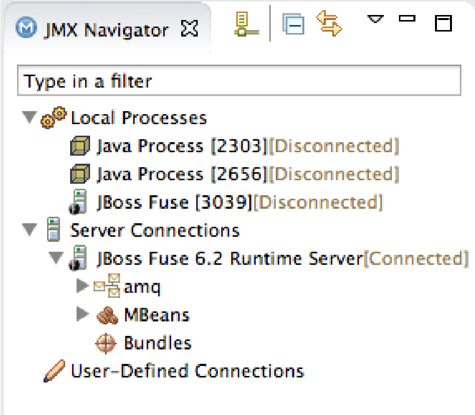
2. Double-click the **JMX[Disconnected]** node:



Connecting to a running server in JMX Navigator

1. In **JMX Navigator**, select the server to which you want to connect under the **Server Connections** node.

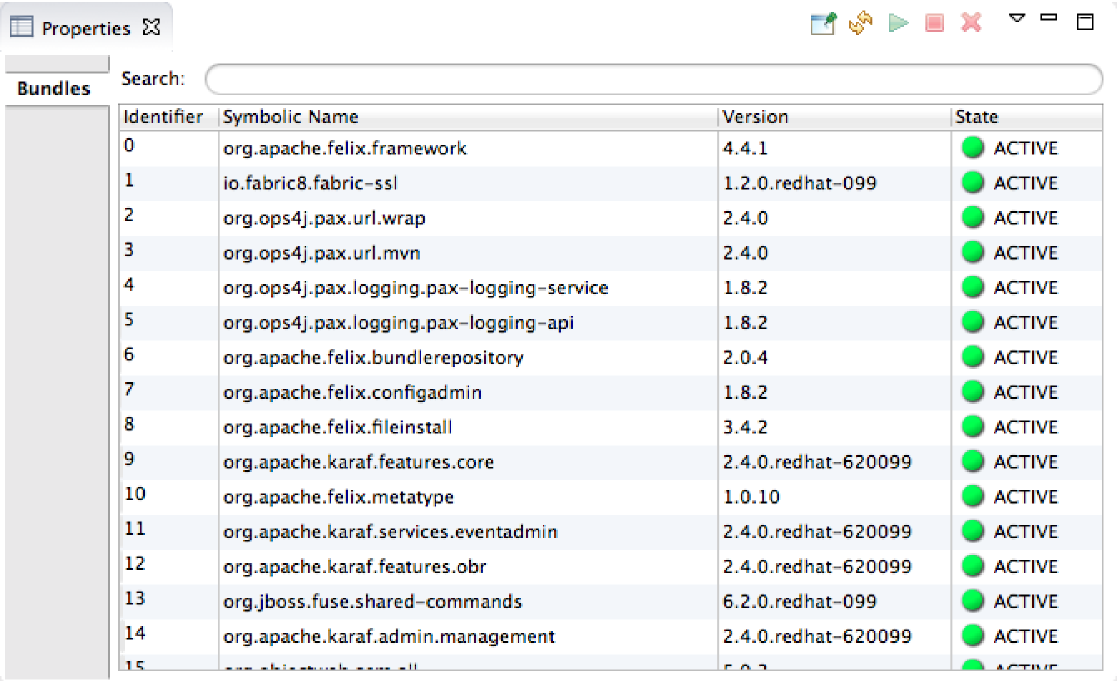
2. Double-click it:


JMX Navigator
Type in a filter
Local Processes
Java Process [2303][Disconnected]
Java Process [2656][Disconnected]
JBoss Fuse [3039][Disconnected]
Server Connections
JBoss Fuse 6.2 Runtime Server[Connected]
amq
MBeans
Bundles
User-Defined Connections

Viewing bundles installed on the connected server

1. In either the **Servers** panel or **JMX Navigator**, expand the server runtime tree to expose the **Bundles** node, and select it.

2. The tooling populates **Properties** view with a list of bundles that are installed on the server:



Identifier	Symbolic Name	Version	State
0	org.apache.felix.framework	4.4.1	● ACTIVE
1	io.fabric8.fabric-ssl	1.2.0.redhat-099	● ACTIVE
2	org.ops4j.pax.url.wrap	2.4.0	● ACTIVE
3	org.ops4j.pax.url.mvn	2.4.0	● ACTIVE
4	org.ops4j.pax.logging.pax-logging-service	1.8.2	● ACTIVE
5	org.ops4j.pax.logging.pax-logging-api	1.8.2	● ACTIVE
6	org.apache.felix.bundlerepository	2.0.4	● ACTIVE
7	org.apache.felix.configadmin	1.8.2	● ACTIVE
8	org.apache.felix.fileinstall	3.4.2	● ACTIVE
9	org.apache.karaf.features.core	2.4.0.redhat-620099	● ACTIVE
10	org.apache.felix.metatype	1.0.10	● ACTIVE
11	org.apache.karaf.services.eventadmin	2.4.0.redhat-620099	● ACTIVE
12	org.apache.karaf.features.obr	2.4.0.redhat-620099	● ACTIVE
13	org.jboss.fuse.shared-commands	6.2.0.redhat-099	● ACTIVE
14	org.apache.karaf.admin.management	2.4.0.redhat-620099	● ACTIVE
15	org.objectweb.asm.all	5.0.3	● ACTIVE

Using **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 in the symbolic name or the identifier, the list updates, showing only the bundles that match the current search string.

NOTE

The **Symbolic Name** is the artifactId that you gave the project when you first created it. You can find the artifactId in the project's **pom.xml**file.

28.4. DISCONNECTING FROM A SERVER

Overview

When you are done testing your application, you can disconnect from the server without stopping it.

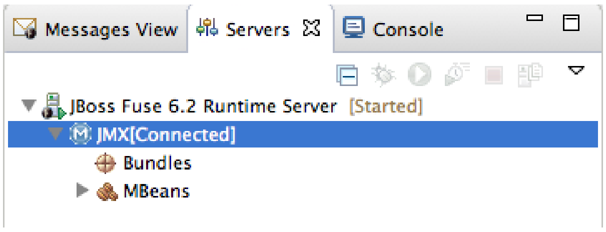
NOTE

Servers panel and **JMX Navigator** are synchronized with regards to server connections, so disconnecting from a server in **Servers** panel also disconnects it in **JMX Navigator**, and vice versa.

Disconnecting from a server in Servers panel

1. In **Servers** panel, expand the server runtime to expose its **JMX[Connected]** node.

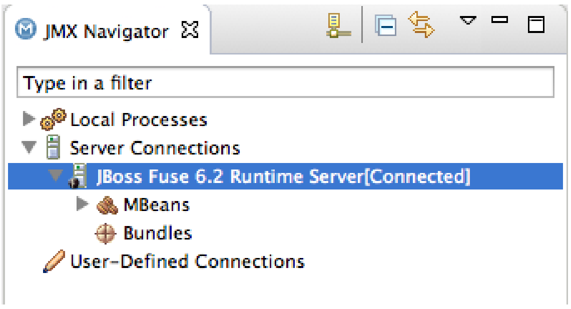
2. Right-click the **JMX[Connected]** node to open the context menu, and then select**Disconnect**.



Disconnecting from a server in JMX Navigator

1. In **`JMX Navigator`**, select the server from which you want to disconnect.

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



28.5. STOPPING A SERVER

Overview

You can shut down a server in one of two ways:

- from **`Servers`** panel

- from the server's remote console in **`Shell`** view

Using Servers panel

To stop a server:

1. In **`Servers`** panel, select the server you want to stop.

2. Click 🟥 .

Using the remote console

To stop a server:

1. Open the **`Shell`** view hosting the server's remote console.

2. Type **`CTRL+D`**.

28.6. DELETING A SERVER

Overview

When you are finished with a configured server, or if you misconfigure a server, it is easy to delete it and it's configuration.

Deleting a server

1. In **`Servers`** panel, 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, click **JBoss Developer Studio** → **Preferences** → **Server**.

   > **NOTE**
   >
   > On Linux and Windows machines, you access **`Preferences`** through the **`Window`** menu.

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 **`Servers`** panel, and then click **`Remove`**.

4. Click **`OK`**.

CHAPTER 29. PUBLISHING FUSE PROJECTS TO A SERVER

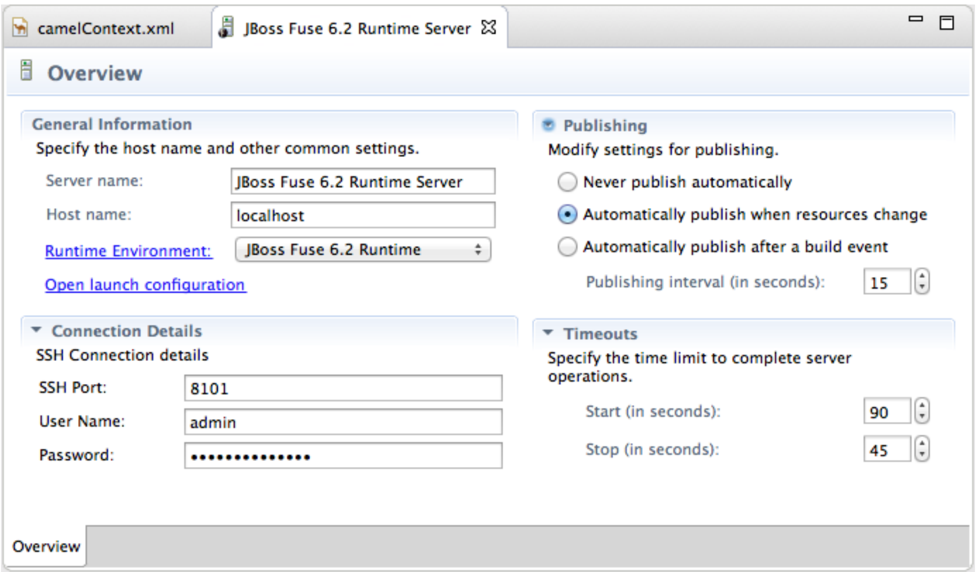
Abstract

You deploy Fuse projects into a server runtime using the Eclipse publishing mechanism. To do so, you must have defined and added the server to the **Servers** panel in **Fuse Integration** perspective (for details, see Chapter 28, *Managing servers*. For a step-by-step demonstration, see Red Hat JBoss Fuse Tooling Tutorials, *To Publish a Fuse Project to Red Hat JBoss Fuse*.

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** panel has its own **Overview** page that contains its configuration, connection, and publishing details:



General Information
Specify the host name and other common settings.
Server name: JBoss Fuse 6.2 Runtime Server
Host name: localhost
Runtime Environment: JBoss Fuse 6.2 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 may 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`**.

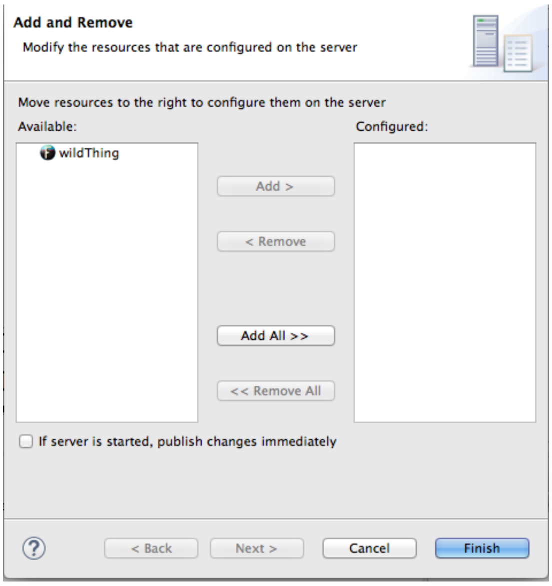
1. If necessary, start up the server runtime to which you want to publish a Fuse project. For details, see Section 28.2, “Starting a Server”.

2. In the **`Servers`** panel, 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`** panel, 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: wildThing

   Configured:

   ☐ If server is started, publish changes immediately

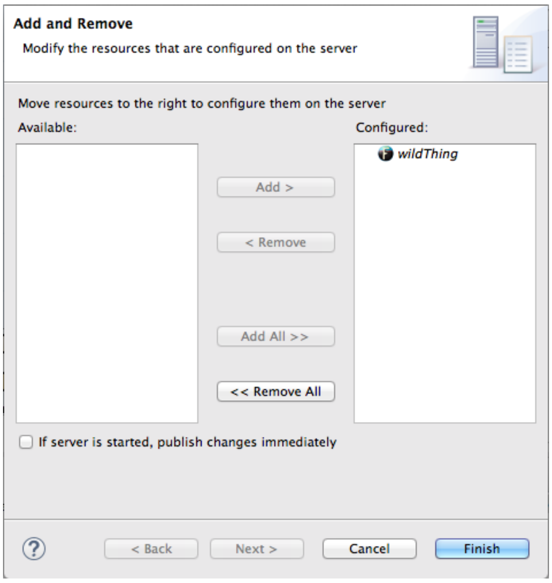
   All resources available for publishing appear in the **`Available`** column.

6. To assign a resource (in this case, the **`wildThing`** 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:

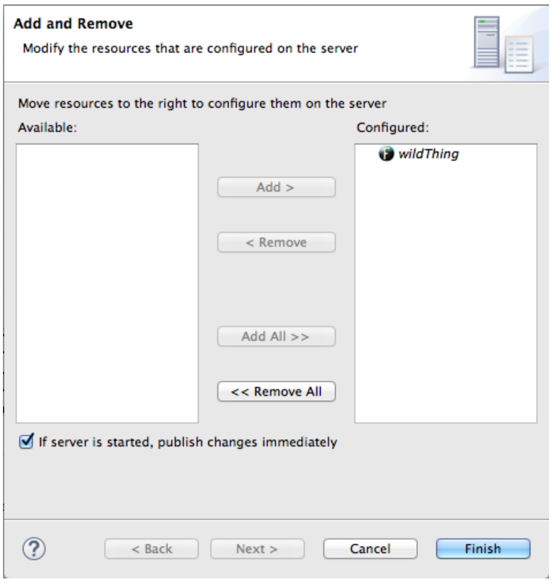
Configured:

wildThing

□ If server is started, publish changes immediately

At this stage, the time at which the assigned resource would actually be published would depend 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:

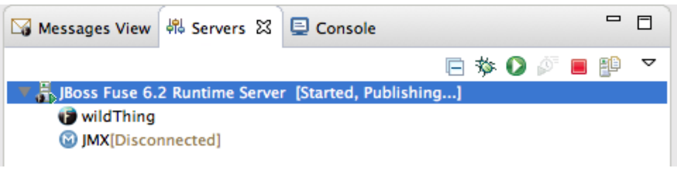
wildThing

☑ If server is started, publish changes immediately

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** panel under the server runtime node, and the server runtime status reports **[Started,Publishing...]**:

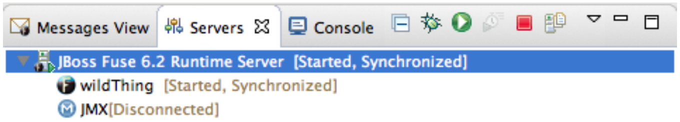


JBoss Fuse 6.2 Runtime Server [Started, Publishing...]

wildThing

JMX[Disconnected]

When publishing is done, the status of both the server runtime and the project report **`[Started,Synchronized]`**:



JBoss Fuse 6.2 Runtime Server [Started, Synchronized]
wildThing [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 up 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`** panel, double-click the server runtime to open its **`Overview`** page.
3. Expand **`Publishing`**, and then select **`Never publish automatically`**.
4. If the Fuse project has already been assigned to the server runtime, make sure the **`If server started, publish changes immediately`** option is disabled.

   a. In the **`Servers`** panel, right-click the server runtime to open the context menu.

   b. Click **Add and Remove...** to open the server's **`Add and Remove`** page.

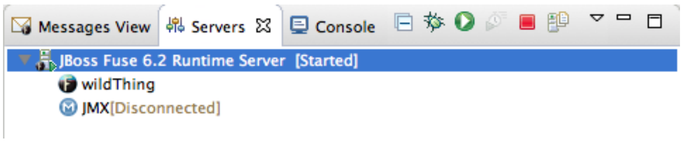
   c. If the **`If server started, publish changes immediately`** option is enabled, disable it.

   d. Skip to Step 6.
5. If the Fuse project has not been assigned to the server runtime, assign it now.

   a. Follow Step 5 through Step 6 in the section called “Publishing Fuse projects automatically when resources change”.

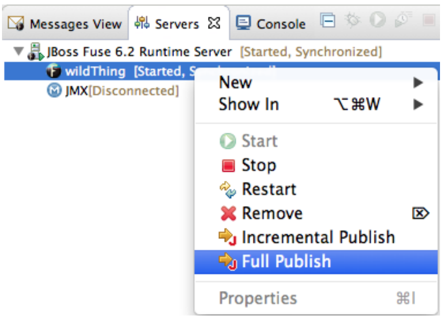
   b. Do not enable the **`If server started, publish changes immediately`** option.
6. Click **`Finish`**.

   The project appears in the **`Servers`** panel under the server runtime node, and the server runtime status reports **`[Started]`**:



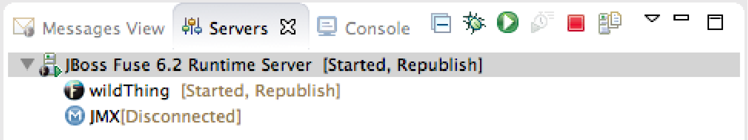
JBoss Fuse 6.2 Runtime Server [Started]
wildThing
JMX[Disconnected]

7. In the **`Servers`** panel, right-click the project's node (in this case, the Fuse project**`wildThing`**) to open the context menu:



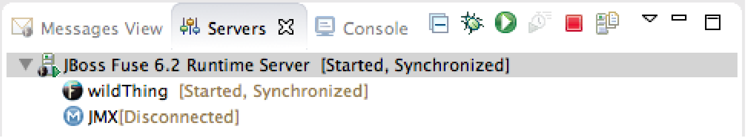
8. Select **Full Publish**.

   During the publishing operation, the status of both the server runtime and the project report **`[Started,Republish]`**:



```
JBoss Fuse 6.2 Runtime Server  [Started, Republish]
    wildThing  [Started, Republish]
    JMX[Disconnected]
```

   When publishing is done, the status of both the server runtime and the project report **`[Started,Synchronized]`**:



```
JBoss Fuse 6.2 Runtime Server  [Started, Synchronized]
    wildThing  [Started, Synchronized]
    JMX[Disconnected]
```

NOTE

The tooling does not support the **Incremental Publish** option. Clicking **Incremental Publish** results in a full publish.

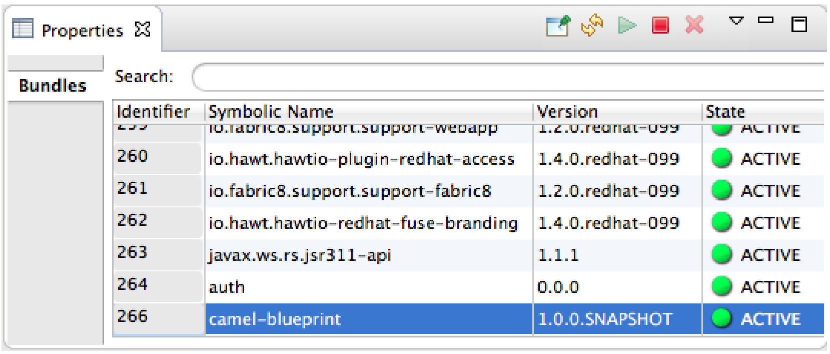
VERIFYING THE PROJECT WAS PUBLISHED TO THE SERVER

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 Servers panel”.

2. In the **`Servers`** panel, expand the server runtime tree to expose the **`Bundles`** node and select it.

   The tooling populates **`Properties`** view with a list of bundles that are installed on the server:



Identifier	Symbolic Name	Version	State
[illegible]	io.fabric8.support.support-webapp	1.2.0.redhat-099	● ACTIVE
260	io.hawt.hawtio-plugin-redhat-access	1.4.0.redhat-099	● ACTIVE
261	io.fabric8.support.support-fabric8	1.2.0.redhat-099	● ACTIVE
262	io.hawt.hawtio-redhat-fuse-branding	1.4.0.redhat-099	● ACTIVE
263	javax.ws.rs.jsr311-api	1.1.1	● ACTIVE
264	auth	0.0.0	● ACTIVE
266	camel-blueprint	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 **`Properties`** view's **`Search`** box.

   The bundle's symbolic name is the artifactId you gave the project when you created it. You can find the artifactId at the very beginning of the project's **`pom.xml`** file. To open it in the tooling's XML editor, click **`pom.xml`** in **`Project Explorer`**, and then click the **`pom.xml`** tab in the XML editor.

RELATED TOPICS

Red Hat JBoss Fuse Tooling Tutorials, To Publish a Fuse Project to Red Hat JBoss Fuse

PART VI. WORKING WITH FABRICS

A fabric is a distributed configuration, provisioning, and clustering mechanism for developing, testing, and running Red Hat JBoss Fuse Tooling projects on multiple computers in an on-premises or public Cloud. You can view and edit fabrics, and the containers, versions, and profiles of which they consist. You can also create and delete containers and profiles, but you can only create versions.

For in-depth information on Fabric8, see the Fabric Guide and Cloud Computing with Fabric on the Red Hat Customer Portal

To open **`Fabric8`** perspective (Chapter 4, *Fabric8 Perspective*), click ⊞ on the right side of the JBoss Developer Studio tool bar, and then select **Fabric8** from the list of available perspectives.

> ⚠ **WARNING**
>
> The Fabric8 Tooling feature has now been deprecated. Red Hat support for this feature will be discontinued from JBoss Fuse version 6.2. However, this feature will be removed completely after JBoss Fuse 7.0 release.

CHAPTER 30. SETTING UP A FABRIC ENVIRONMENT

The tooling provides an environment for developing and debugging fabric applications. Use the tooling to build a mock-up of your actual fabric into which you can deploy your application to test and debug it.

OVERVIEW

To set up a fabric environment, you must have access to an existing fabric.

NOTE

With Red Hat JBoss Fuse and the JBoss Server Extensions feature installed, you can start up Red Hat JBoss Fuse and create a fabric from inside the tooling. See Chapter 28, *Managing servers*

Setting up a fabric environment involves several basic steps:

- Providing the details for connecting to an existing fabric
- Connecting to the fabric
- Creating the containers that will host the various components of your distributed integration application
- Assigning each container one or more profiles, which provision containers by deploying and installing specified applications.

NOTE

For cloud applications, unless the cloud already has a fabric registry agent running on it, you must create one yourself before you can include it to your fabric environment. For details, see Chapter 35, *Creating a Fabric in the Cloud*.

OPENING THE FABRIC PERSPECTIVE

To set up a fabric environment, you must switch to the Fabric8 perspective, by selecting **Window → Open Perspective → Other... → Fabric8**, or clicking ⊞ in the Perspectives tab, and selecting **Fabric8**.

RELATED TOPICS

Chapter 31, *Specifying and Connecting to a Fabric*

CHAPTER 31. SPECIFYING AND CONNECTING TO A FABRIC

Before you can view and edit your fabric, you must add it to the `Fabric Explorer` view. You can add multiple fabrics in Fabric8 perspective and connect to them concurrently.

31.1. ADDING FABRIC DETAILS

Overview

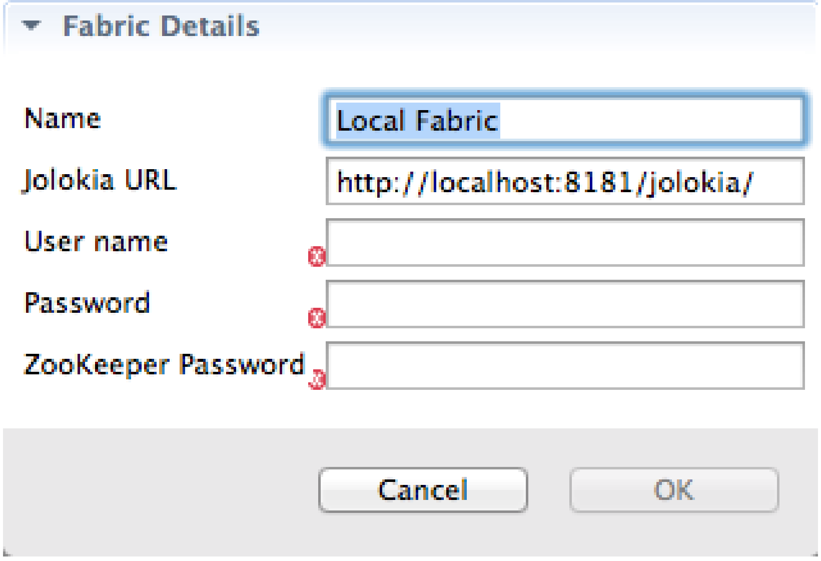
For the tooling to connect to a fabric, you must provide certain details about the fabric.

Procedure

To specify the details for connecting to a fabric:

1. In **`Fabric Explorer`**, right-click **Fabrics** to open the context menu, and then select **Add Fabric details** to open the **`Fabric Details`** wizard, as shown in Figure 31.1.

   Figure 31.1. Fabric Details wizard

   Fabric Details

   Name: Local Fabric

   Jolokia URL: http://localhost:8181/jolokia/

   User name:

   Password:

   ZooKeeper Password:

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

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

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

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

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

   This is the user name that was either specified when the fabric was created or is defined in the Red Hat JBoss Fuse *installDir*/**`etc/users.properties`** file. In that file, user information is specified using this format: **`user=password,role`** (for example, **`admin=admin,Administrator`**).

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

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

   This is the password that was either specified for **`User name`** when the fabric was created or is defined in the Red Hat JBoss Fuse *installDir*/**`etc/users.properties`** file.

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

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

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

7. Click **`OK`**.

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

NOTE

Before you can view or modify a fabric's containers, profiles or versions, you must connect to the fabric. For details see Section 31.2, “Connecting to a fabric”

Related topics

Section 31.5, “Deleting a fabric's details”

31.2. CONNECTING TO A FABRIC

Overview

The tooling connects to a fabric using the connection information you specified for it in the **`Fabric Details`** dialog.

Procedure

To connect to a fabric:

1. In **`Fabric Explorer`**, select the fabric to which you want to connect, then right-click it to open the context menu.

2. Select **Connect**.

   An expand arrow (▶) appears next to the fabric you selected in **`Fabric Explorer`**.

3. Click (▶) to expand the fabric's tree and browse its containers, profiles, and versions.

   From here, you can create and deploy new containers or modify existing ones by assigning them new profiles or modified versions of existing profiles.

Related topics

Section 31.1, “Adding fabric details”

31.3. DISCONNECTING FROM A FABRIC

Overview

Typically, you'd disconnect from a fabric without stopping its containers since they usually run application services. In this case, the containers will continue to run, unless stopped using the tooling or the Red Hat JBoss Fuse console command line.

Procedure

1. In **`Fabric Explorer`**, select the fabric from which you want to disconnect, then right-click it to open the context menu.
2. Select **Disconnect**.

   The tooling closes the connection with the selected fabric.

31.4. EDITING A FABRIC'S DETAILS

Overview

When any of the details for connecting to your fabric change, you need to update the details in the tooling.

IMPORTANT

Before you edit a fabric's details, disconnect from the fabric.

Procedure

1. In **`Fabric Explorer`**, select the fabric whose details you want to edit, and right-click it to open the context menu.
2. Select **Edit Fabric details** to open the **`Fabric Details`** dialog, as shown in Figure 31.1, “Fabric Details wizard”.
3. In **`Name`**, enter a new name for the fabric, if necessary.
4. In **`Jolokia URL`**, enter the new url of the fabric, if necessary.
5. In **`User name`**, enter a name with which the new user will log into the selected fabric, if necessary.
6. In **`Password`**, enter the new password to use for logging into the selected fabric, if necessary.
7. Click **`OK`**.

31.5. DELETING A FABRIC'S DETAILS

Overview

To remove a fabric from **`Fabric Explorer`**, you delete its details.

Procedure

1. In **`Fabric Explorer`**, select the fabric whose details you want to delete, then right-click it to open the context menu.

2. Select **Delete Fabric details**.

   The tooling closes the connection with the selected fabric, and then deletes the fabric's details.

Related topics

Section 31.1, “Adding fabric details”

CHAPTER 32. WORKING WITH FABRIC CONTAINERS

32.1. CREATING A NEW CHILD CONTAINER

Overview

A child container is created on the server where its parent is.

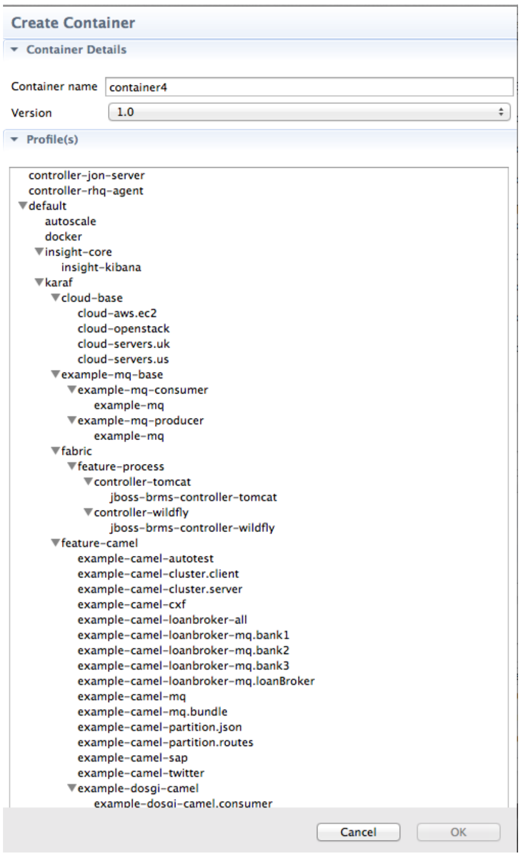
Procedure

To create a new child container:

1. If necessary, in **`Fabric Explorer`**, expand the tree of the selected fabric in which you want to create the new child container.

2. Right-click the target parent container to open the context menu, and then select **Create a new child container**.

   Figure 32.1. Create Child Container wizard



Create Container

Container Details

Container name: container4

Version: 1.0

3. In **`Container name`**, enter a name for the new child container.

   We recommend that you replace the default name with a meaningful name that identifies the container's role or function.

4. If multiple versions exist, you can select one from the **`Version`** drop-down list. Otherwise, accept the default value.

5. Under **`Profiles`**, click the checkbox next to the profile or profiles you want to assign to the new child container.

Profiles determine the function of containers; what applications they run. You can assign multiple profiles to a container, as long as the applications they install do not conflict with one another.

IMPORTANT

Do not assign a *base profile* (one ending in **`-base`**) to a container.

6. Click **`OK`**.

   The new child container appears in **`Fabric Explorer`** as a node under **`Containers`**

NOTE

Selecting the new container in **`Fabric Explorer`** populates its **`Profiles`** and **`Profile Details`** pages in **`Properties`** viewer.

Related topics

Section 33.1, “Creating a new profile”
Section 34.1, “Creating a new version of a profile”

32.2. CREATING A CONTAINER ON A REMOTE HOST

Overview

To create a container on a remote host, the remote host machine must be ssh-enabled.

NOTE

Creating a container via ssh on a remote Windows machine is not supported.

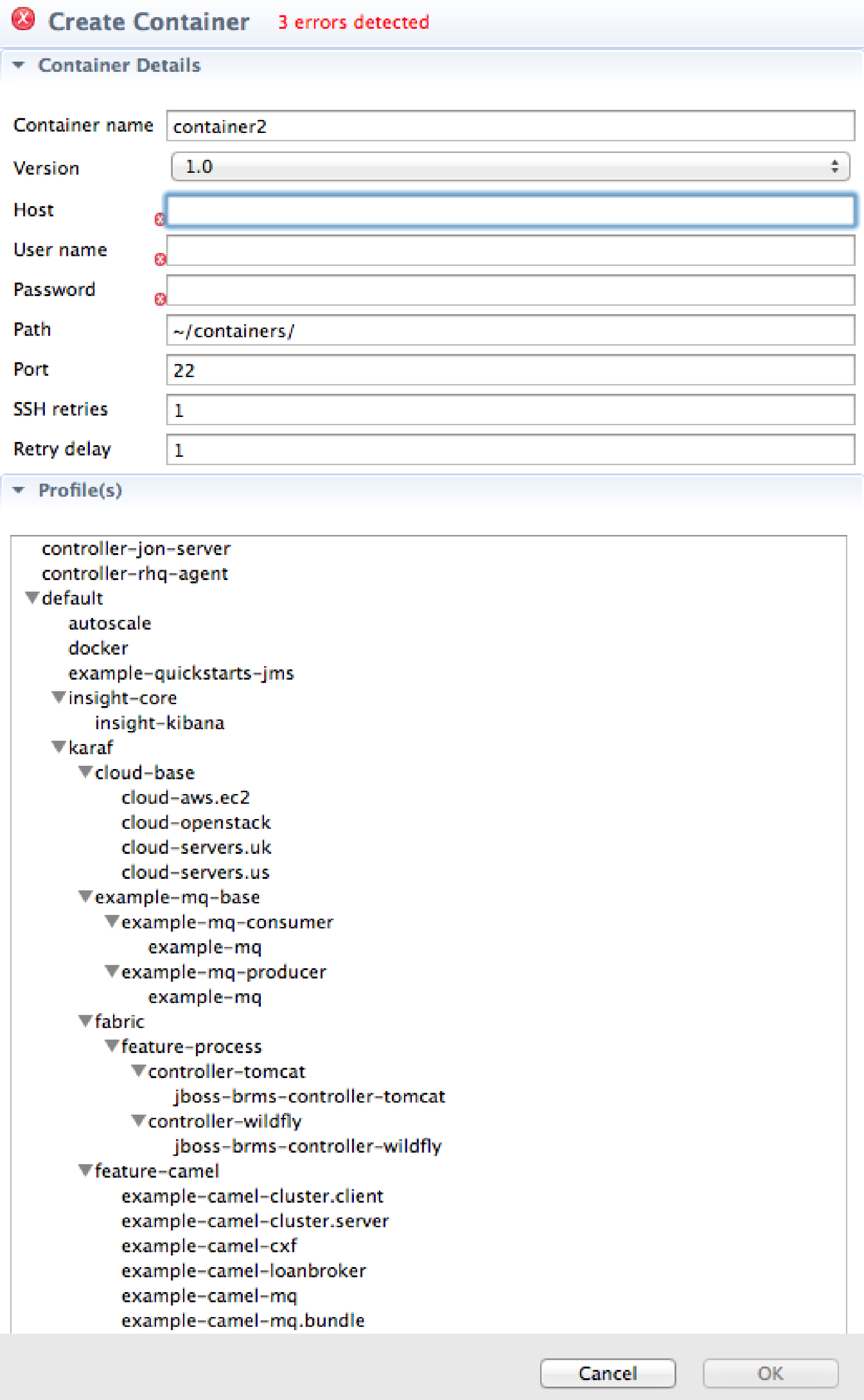
Procedure

To create a new container on a remote machine:

1. If necessary, in **`Fabric Explorer`**, expand the tree of the fabric for which you want to create the new remote container.

2. Right-click **`Containers`** to open the context menu, and then select **Create a new container via SSH**.

Figure 32.2. Create Container via SSH wizard



Create Container 3 errors detected

Container Details

Container name	container2
Version	1.0
Host	⊗
User name	⊗
Password	⊗
Path	~/containers/
Port	22
SSH retries	1
Retry delay	1

Profile(s)

```
controller-jon-server
controller-rhq-agent
▼default
    autoscale
    docker
    example-quickstarts-jms
  ▼insight-core
      insight-kibana
  ▼karaf
    ▼cloud-base
        cloud-aws.ec2
        cloud-openstack
        cloud-servers.uk
        cloud-servers.us
    ▼example-mq-base
      ▼example-mq-consumer
          example-mq
      ▼example-mq-producer
          example-mq
    ▼fabric
      ▼feature-process
        ▼controller-tomcat
            jboss-brms-controller-tomcat
        ▼controller-wildfly
            jboss-brms-controller-wildfly
    ▼feature-camel
        example-camel-cluster.client
        example-camel-cluster.server
        example-camel-cxf
        example-camel-loanbroker
        example-camel-mq
        example-camel-mq.bundle
```

NOTE

An error icon (⊗) marks required fields that lack a valid value, and the dialog banner displays the number of errors detected.

3. In **`Container name`**, enter a name for the new container.

   We recommend that you replace the default name with a meaningful name that identifies the container's role or function.

4. If multiple versions exist, you can select one from the **Version** drop-down list. Otherwise, accept the default value.

5. In **Host**, enter the name or the IP address of the remote host.

6. In **User name**, enter the name of a user authorized to log into the remote host.

7. In **Password**, enter the password for **User name**.

8. In **Path**, accept the default (*~/containers/*), or enter the path of the new container's location on the remote host.

9. In **Port**, accept the default port number (*22*), or enter a new port number to use for establishing an ssh connection on the remote host.

10. In **SSH retries**, accept the default (*1*), or enter the maximum number of retries to attempt at establishing an ssh connection on the remote host.

11. In **Retry delay**, accept the default (*1*), or enter the delay, in milliseconds, between retry attempts.

12. Under **Profiles**, click the checkbox next to the profile or profiles you want to assign to the container.

    Profiles determine the function of a container; what applications it runs. You can assign multiple profiles to a container, as long as the applications they install do not conflict with one another.

    IMPORTANT

    Do not assign a *base profile* (one ending in **-base**) to a container.

13. Click **OK**.

    The new container appears in **Fabric Explorer** as a container node under **Containers**

    Selecting the new container in **Fabric Explorer**s populates its **Profiles** and **Profile Details** pages in **Properties** viewer.

Related topics

Section 33.1, “Creating a new profile”
Section 34.1, “Creating a new version of a profile”

32.3. CREATING A NEW CONTAINER ON A CLOUD

Overview

To create a container on a cloud, you need to specify the cloud's connection details and then create the container.

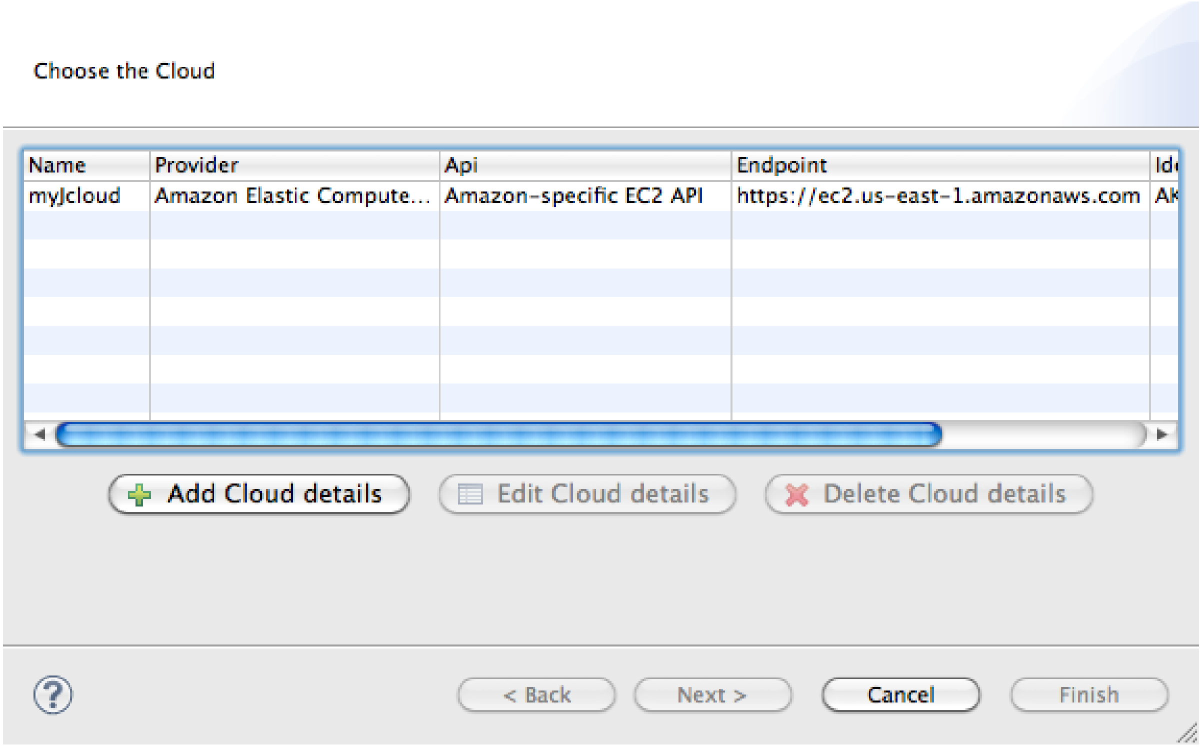
Procedure

To create a new container on a cloud:

1. In **Fabric Explorer**, if necessary, expand the tree of the fabric for which you want to create the new container on the cloud. Right-click **Containers** to open the context menu.

2. Select **Create a new container on the cloud**to open the **Create Container in a Cloud** wizard.

   Figure 32.3. Create Container in a Cloud/Choose the Cloud page

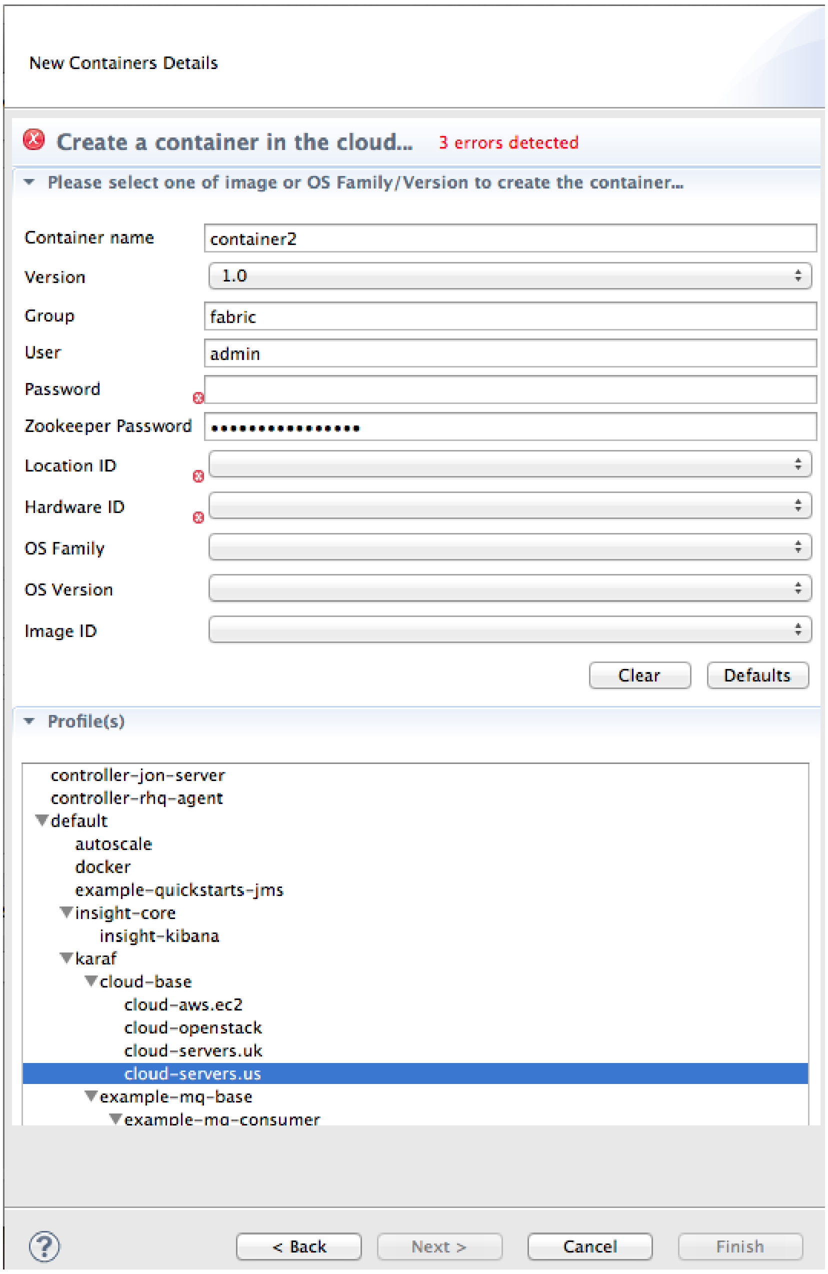
Name	Provider	Api	Endpoint	Id
myJcloud	Amazon Elastic Compute...	Amazon-specific EC2 API	https://ec2.us-east-1.amazonaws.com	AK

   ### NOTE

   Unless the connection details for a cloud have already been added to the tooling, this dialog opens devoid of selections. You can add a cloud's connection details now by clicking **Add Cloud details** and following the procedure in Section 35.1, “Adding cloud details”.

3. Select a cloud from the list, and click **Next** to open the **New Containers Details** page.

Figure 32.4. New Containers Details page



New Containers Details

⊗ Create a container in the cloud... 3 errors detected

▾ Please select one of image or OS Family/Version to create the container...

Container name container2
Version 1.0
Group fabric
User admin
Password ⊗
Zookeeper Password ••••••••••••••••
Location ID ⊗
Hardware ID ⊗
OS Family
OS Version
Image ID

▾ Profile(s)

```
  controller-jon-server
  controller-rhq-agent
▼default
    autoscale
    docker
    example-quickstarts-jms
  ▼insight-core
      insight-kibana
  ▼karaf
    ▼cloud-base
        cloud-aws.ec2
        cloud-openstack
        cloud-servers.uk
        cloud-servers.us
    ▼example-mq-base
      ▼example-mq-consumer
```

NOTE

An error icon (⊗) marks required fields that lack a valid value, and the dialog banner displays the number of errors detected.

4. In **`Container name`**, enter a name for the new container.

   We recommend that you replace the default name with a meaningful name that identifies the container's role or function.

5. If multiple versions exist, you can select one from the **`Version`** drop-down list. Otherwise, accept the default version.

6. In **`Group`**, accept the default (*fabric*), or enter the group in which to start up the new container.

7. In **`User`**, accept the default (*admin*), or enter the name of another user who is authorized to log into the new cloud container.

This is the user that was either specified when the fabric was created or is defined in the Red Hat JBoss Fuse *installDir*/**`etc/users.properties`** file (see Section 31.1, “Adding fabric details”).

8. In **`Password`**, enter the pasword required for **`User`** to log into the new cloud container.

   This is the password that was specified for **`User`** when the fabric was created or is defined for the user in the Red Hat JBoss Fuse *installDir*/**`etc/users.properties`** file.

9. In **`Zookeeper Password`**, accept the default generated password for logging into the cloud container's zookeeper registry, or enter the zookeeper password that was used when the fabric was created.

   You can discover the zookeeper password by issuing the commmand **`JBossFuse:karaf@root>fabric:ensemble-password`**.

10. In **`Hardware ID`**, select from the drop-down list the ID of the hardware archetype to use.

    Hardware archetypes specify the amount of memory and compute power available for running applications on machine images.

    ### IMPORTANT

    The next three fields provide two methods for specifying the software configuration (operating system, application server, and applications) you want to use. Either select an OS Family (and optionally a version), or select an Image ID.

11. In **`OS Family`**, select from the drop-down list the family name of the operating system you want to use.

    If you select an **`OS Family`**, do not also select an **`Image ID`**.

12. In **`OS Version`**, optionally select from the drop-down list the version for the operating system you selected.

13. If you selected an **`OS Family`**, skip this step. Otherwise, in **`Image ID`**, select from the drop-down list the ID of the image template to use for the software configuration.

    ### NOTE

    The tooling starts an instance of the image in the cloud, and then installs a fabric agent and provisions the new agent with the selected profiles. Depending on the cloud provider, instantiation and provisioning could take some time.

14. Under **`Profiles`**, select the profile or profiles you want to assign to the container.

    Profiles determine the function of containers—what applications they run. You can assign a container multiple profiles, as long as the applications they install do not conflict with one another.

    ### IMPORTANT

    Do not assign a *base profile* (one ending in **`-base`**) to a container.

15. Click **OK**.

    The new container appears in **Fabric Explorer** as a container node under **Containers**

    NOTE

    Selecting the new container in **Fabric Explorer** populates its **Profiles** and **Profile Details** pages in **Properties** viewer.

Related topics

Chapter 35, *Creating a Fabric in the Cloud*

Section 33.1, “Creating a new profile”

Section 34.1, “Creating a new version of a profile”

32.4. STARTING A CONTAINER

Overview

When you create a container, it starts up automatically and continues to run until stopped by some event, planned or otherwise. You can manually restart a stopped container.

Procedure

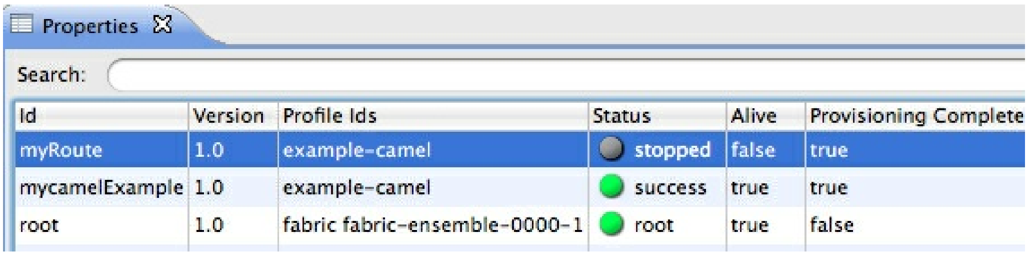
To start a container:

1. If necessary, in **Fabric Explorer**, expand the tree of the fabric whose container you want to start.
2. Select the container and right-click it to open the context menu.
3. Click **Start Container**.

    NOTE

    To check whether the tooling started the container, click **Containers** in **Fabric Explorer**, and open the **Properties** view. You should see something similar to Figure 32.5, where **mycamelExample** was started and provisioned successfully.

Figure 32.5. Container status in Properties view



Id	Version	Profile Ids	Status	Alive	Provisioning Complete
myRoute	1.0	example-camel	● stopped	false	true
mycamelExample	1.0	example-camel	● success	true	true
root	1.0	fabric fabric-ensemble-0000-1	● root	true	false

The status indicators in the Status column of the **`Properties`** view indicate the startup state of the container, as shown Table 32.1:

Table 32.1. Container startup status

Icon	Meaning
●	Container stopped
🟠	Container started; analysis and provisioning in progress
🟢	Container started and provisioned successfully
🔴	Failure occurred during startup or provisioning

Related topics

Section 32.6, “Stopping a container”

32.5. CHANGING A CONTAINER'S PROFILES

Overview

A container's profiles determine what the container is hosting. If you want to change what is deployed in a container or how it is configured, you change the profiles deployed to it.

IMPORTANT

The profile or profiles you select will override the entire list of profiles currently deployed in the selected container.

Procedure

To change the profiles deployed to a container:

1. Select the container.
2. In the **`Properties`** view, select the **`Profiles`** tab.
3. From the list, select the profile or profiles you want to deploy to the container by highlighting them.

   NOTE

   To select multiple profiles, hold down the **`CTRL`** key while you select them.

32.6. STOPPING A CONTAINER

Overview

There are times when you might want to stop then restart a container; for example, to recycle a hung container, force a container to reprovision itself, or to reclaim CPU cycles.

Procedure

To stop a container:

1. If necessary, in **`Fabric Explorer`**, expand the tree of the fabric whose container you want to stop.

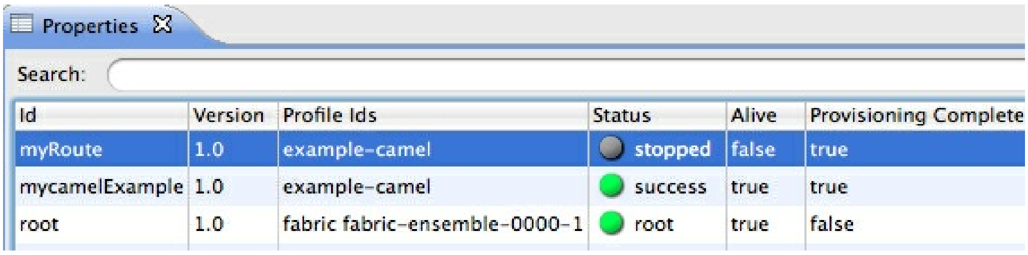
2. Select the container and right-click it to open the context menu.

3. Click **Stop Container**.

NOTE

To check whether the tooling stopped the container, click **`Containers`** in **`Fabric Explorer`**, and open the **`Properties`** view. You should see something similar to Figure 32.6, where **`myRoute`** was stopped successfully.

Figure 32.6. Container status in Properties view



Id	Version	Profile Ids	Status	Alive	Provisioning Complete
myRoute	1.0	example-camel	● stopped	false	true
mycamelExample	1.0	example-camel	● success	true	true
root	1.0	fabric fabric-ensemble-0000-1	● root	true	false

For information on the meaning of the status indicators that might appear in the Status column of the **`Properties`** view, see Table 32.1, “Container startup status”.

Related topics

Section 32.4, “Starting a container”
Section 32.7, “Deleting a container”

32.7. DELETING A CONTAINER

Overview

When a container becomes obsolete—no longer used in the fabric integration project—you may want to delete it. Before you do so, make sure no other container in the project depends on it. Deleting a container terminates the container and its processes and removes its configuration from the fabric.

NOTE

To delete a container, it must be running in the fabric.

Procedure

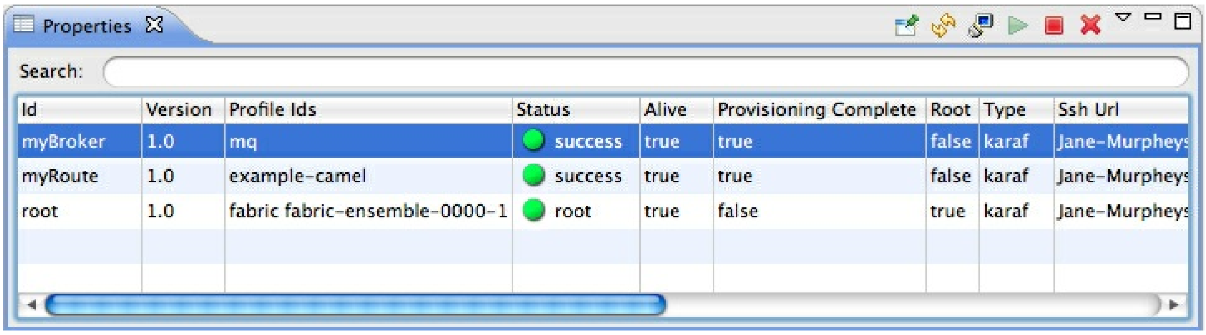
To delete a container:

1. If necessary, in **Fabric Explorer**, expand the tree of the fabric whose container you want to delete.

2. Click **Containers** to populate **Properties** view with a list of the fabric's containers.

3. In **Properties** view, select the container you want to delete, and then click ✖ on the right-hand side of the toolbar.

   Figure 32.7. Container selected for deletion in Properties view

Id	Version	Profile Ids	Status	Alive	Provisioning Complete	Root	Type	Ssh Url
myBroker	1.0	mq	● success	true	true	false	karaf	Jane-Murpheys
myRoute	1.0	example-camel	● success	true	true	false	karaf	Jane-Murpheys
root	1.0	fabric fabric-ensemble-0000-1	● root	true	false	true	karaf	Jane-Murpheys

4. In the **Destroy Container(s)** dialog, click **OK** to delete the selected container.

   The tooling displays a message that removal is in progress. In **Properties** view, the container's status briefly changes to **stopped**, and then the container disappears from the list.

5. In **Fabric Explorer**, open the fabric's context menu, and click**Refresh** to update **Containers** and remove the deleted container from the tree.

NOTE

If you have the Red Hat JBoss Fuse console open in **Shell** view, you can verify that the container and its configuration have been removed from the fabric by entering at the command line **fabric:container-list**.

Related topics

Section 32.6, “Stopping a container”
Section 32.1, “Creating a new child container”
Section 32.2, “Creating a container on a remote host”
Section 32.3, “Creating a new container on a cloud”

CHAPTER 33. WORKING WITH FABRIC PROFILES

Profiles are applied to fabric containers, and they determine the function and behavior of their containers. A profile consists of code and configurations, which specify the type of service or application a container runs. The code may come from OSGi bundles and Karaf features pulled from repositories, also specified in the profile. Defined hierarchically, profiles can be set up to inherit from other profiles, building on their parents' definitions.

33.1. CREATING A NEW PROFILE

Overview

The tooling provides access to numerous profiles, each of which imparts specific functionality and behaviors to the containers to which it's assigned. But you may want to create new profiles that inherit from these base profiles, so you can override, for example, configuration details. You may also want to create your own profiles to deploy your own code. To do so, you create a new profile based on an existing one, and then change its configuration and properties to generate the specific behaviors you want to deploy to the target containers.

NOTE

This procedure creates a brand new profile. If you want to create a new version of an existing profile to incrementally test changes to deployed containers, see Chapter 34, *Working with Versions*

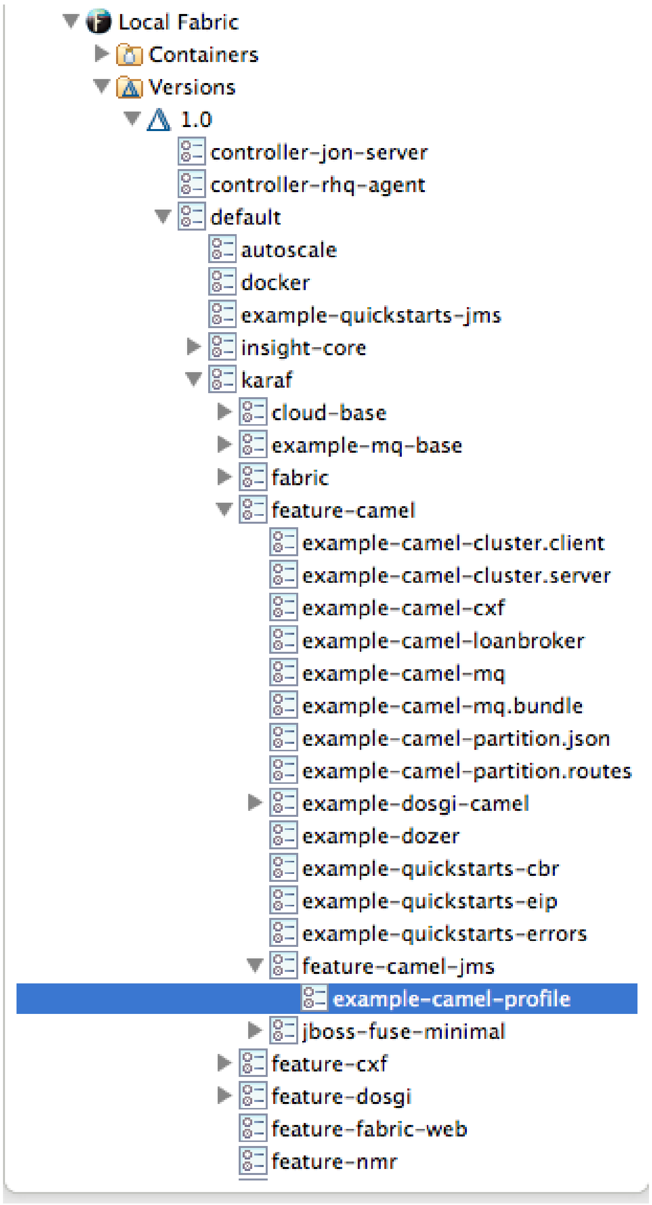
Procedure

To create a new profile:

1. In **`Fabric Explorer`**, expand the fabric in which you want to create a new profile.

2. Expand the fabric's tree to the level of the profiles, as shown in Figure 33.1.

Figure 33.1. Accessing the profiles



- Local Fabric
  - Containers
  - Versions
    - 1.0
      - controller-jon-server
      - controller-rhq-agent
      - default
        - autoscale
        - docker
        - example-quickstarts-jms
        - insight-core
        - karaf
          - cloud-base
          - example-mq-base
          - fabric
          - feature-camel
            - example-camel-cluster.client
            - example-camel-cluster.server
            - example-camel-cxf
            - example-camel-loanbroker
            - example-camel-mq
            - example-camel-mq.bundle
            - example-camel-partition.json
            - example-camel-partition.routes
            - example-dosgi-camel
            - example-dozer
            - example-quickstarts-cbr
            - example-quickstarts-eip
            - example-quickstarts-errors
            - feature-camel-jms
              - example-camel-profile
            - jboss-fuse-minimal
          - feature-cxf
          - feature-dosgi
          - feature-fabric-web
          - feature-nmr

3. Select the profile on which you want to base the new profile, and right-click it to open the context menu.

IMPORTANT

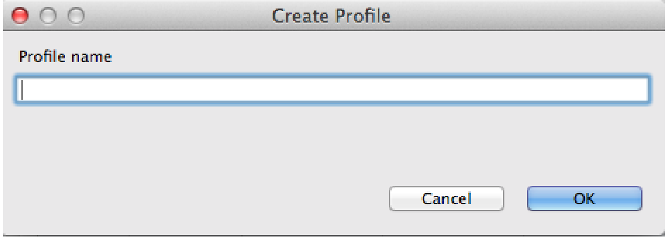
Because the selected profile is the parent of the new profile, the new profile inherits the selected profile's configurations, properties, features, repositories and bundles as well as those of the selected profile's ancestors. However, a child's configurations override an ancestor's when their configuration files have the same name.

NOTE

Neither configuration files nor properties are visible within the tooling, but you can view them in the Red Hat JBoss Fuse Management Console. For details, see the Red Hat JBoss Fuse Management Console documentation at https://access.redhat.com/documentation/en-US/Red_Hat_JBoss_Fuse/.

4. Select **Create a new Profile**.

Figure 33.2. Create Profile dialog



Profile name

5. In **Profile name**, enter a name for the new profile.

   Use a name that is descriptive of the new profile's function.

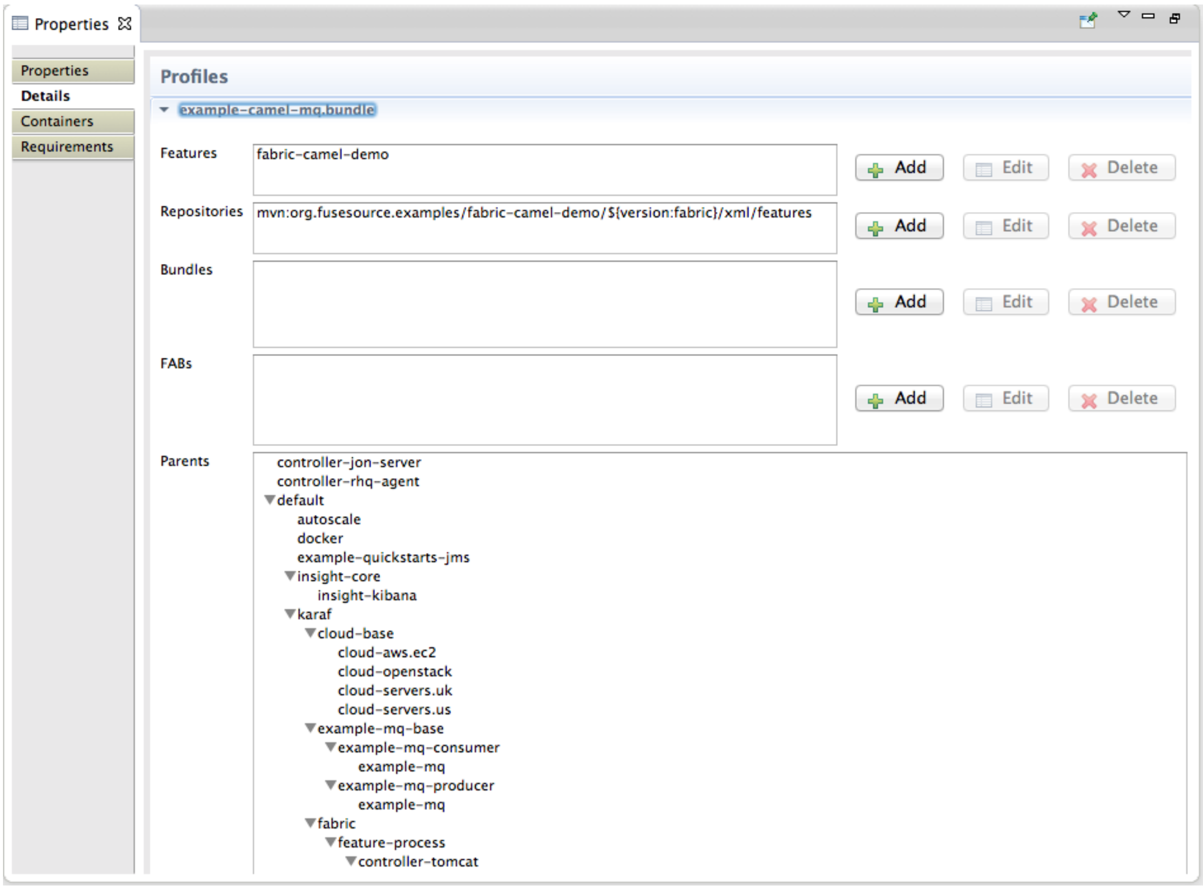
6. Click **OK** to create and save the new profile.

   The new profile is added to the list of profiles in **Fabric Explorer**, under its parent profile.

7. In **Fabric Explorer**, select the new profile to populate the **Properties** editor with the new profile's properties information and profile details.

8. In the **Properties** editor, click the **Details** tab.

Figure 33.3. Profiles page



example-camel-mq.bundle

Features: fabric-camel-demo

Repositories: mvn:org.fusesource.examples/fabric-camel-demo/${version:fabric}/xml/features

Bundles

FABs

Parents:

```
  controller-jon-server
  controller-rhq-agent
▼default
    autoscale
    docker
    example-quickstarts-jms
  ▼insight-core
      insight-kibana
  ▼karaf
    ▼cloud-base
        cloud-aws.ec2
        cloud-openstack
        cloud-servers.uk
        cloud-servers.us
    ▼example-mq-base
      ▼example-mq-consumer
          example-mq
      ▼example-mq-producer
          example-mq
    ▼fabric
      ▼feature-process
        ▼controller-tomcat
```

On the **Profiles** page, you can add, delete, or edit features, the repositories from which features are downloaded, and bundles. You can also add or remove parent profiles. Profiles can have multiple parents, and they inherit the qualities of each

Features are files that aggregate references to interdependent or related bundles that together define a feature. Typically, you'd use bundles only if the feature they define is not already included in a features file. For more information on bundles and features, see Deploying

Features.

Child profiles inherit everything (configurations, properties, features, repositories, and bundles) from their immediate parents and from all of their parents' ancestors. When the child's and any ancestor's configuration file have the same name, the child's configuration file replaces the ancestor's file. Otherwise, configurations are cumulative.

NOTE

The **`Profiles`** page displays only the features, repositories, and bundles added by the child profile, not those contributed by the profile's parents or other ancestors. The **`Parents`** pane shows (highlighted) the profile's immediate parents only.

9. Make the changes you want to the new profile to configure it.

   Changes are automatically saved as you make them.

NOTE

In **`Fuse Integration`** perspective, you can easily add a Fuse project to the new profile by dragging it from **`Project Explorer`** and dropping it onto the new profile in **`Fabric Explorer`**. When you do, the fabric tooling builds the project, runs all tests, installs the project in the new profile, and uploads the profile into the fabric's internal Maven repository. Because profiles are stored in a fabric's internal repository, they cannot be shared across other fabrics.

IMPORTANT

If you drop your project on a fabric container, rather than on a profile, the next time that container is provisioned, your project is wiped out.

You can now assign the new profile to a container.

10. In **`Fabric Explorer`**, select the container to which you want to assign the new profile.

    The fabric tooling populates the **`Properties`** editor with the container's properties information and profile details.

11. Open the container's **`Details`** tab, and then select the new profile in the **`Profiles`** list.

    Repeat Step 10 and Step 11 for each container to which you want to assign the new profile.

Related topics

Section 34.1, “Creating a new version of a profile”

33.2. DELETING A PROFILE

Overview

When a profile is inactive—no longer assigned to any container in the fabric—you can safely delete it.

Procedure

To delete an inactive profile:

1. In `Fabric Explorer`, expand the fabric which contains the profile you want to delete.

2. Expand the fabric's tree to the level of versions, and then expand the version that contains the target profile.

3. Select the target profile, and right-click it to open the context menu.

4. Select **Delete Profile**.

CHAPTER 34. WORKING WITH VERSIONS

Versioning provides the means for safely rolling out upgrades of the profiles deployed and running in containers across a fabric. Using versioning, you can make incremental updates to existing profiles, and then test them out on small groups of containers. When you've completed and tested the final versions, you can confidently deploy them to their target containers in the fabric.

34.1. CREATING A NEW VERSION OF A PROFILE

Overview

A version is simply the collection of all profiles associated with a fabric. When you create a new version, you are copying all of the profiles from the most recent version—the base version—into the new version. Once that's done, you can upgrade the profiles in the new version by modifying them.

Creating a new version

To create a new version:

1. In **`Fabric Explorer`**, select the fabric in which you want to create a new version.

2. Expand the fabric's tree to the level of Versions, shown in Section 33.1, “Creating a new profile”.

3. Select **`Versions`**, and then right-click it to open the context menu.

4. Select **Create Version**.

5. In the **`Create Version`** wizard, enter a new version or accept the default value.

   The default version is always a minor increment of the highest-numbered version. So, if the highest-numbered version is 1.3, the new version will default to 1.4, and all of the profiles in version 1.3 will be copied into version 1.4.

   IMPORTANT

   Version names are important! The tooling sorts version names based on the numeric version string, according to major.minor numbering, to determine the version on which to base a new one. You can safely add a text description to a version name as long as you append it to the end of the generated default name like this: **`1.3<`*****`.description`*****`>`**. If you abandon the default naming convention and use a textual name instead (for example, **`Patch051312`**), the next version you create will be based, not on the last version (**`Patch051312`**), but on the highest-numbered version determined by dot syntax.

6. Click **OK**.

   The tooling copies all of the base version's profiles into the new version and displays the new version in **`Fabric Explorer`**, directly under the base version.

Modifying profiles in the new version

To modify a profile in the new version:

1. In **`Fabric Explorer`**, expand the newly created version.
2. Select a profile that you want to modify.

   The tooling populates the **`Properties`** editor with the profile's details for you to edit.
3. In the **`Properties`** editor, click the **`Details`** tab to access the **`Profiles`** page.
4. Make the changes you want to the new profile to configure it.

   Changes are automatically saved as you make them.

Related topics

Section 33.1, “Creating a new profile”

34.2. SETTING A CONTAINER'S VERSION

Overview

After you've created a new version, you can assign it to selected containers. This feature enables you to easily roll out upgraded profiles in a controlled and orderly way.

IMPORTANT

Setting a container's version immediately provisions the container with the profiles from the selected version.

Procedure

To set a container's version:

1. In **`Fabric Explorer`**, select the container whose profiles you want to change.
2. Right-click it to open the context menu, and then select **Select Version**.

   The **Select Version** submenu lists all available versions.
3. Click the version you want to assign the selected container.

   The container is immediately provisioned with the new version of any profile assigned to it.

CHAPTER 35. CREATING A FABRIC IN THE CLOUD

To create a fabric on a cloud, you need to

- Have established an account with a cloud provider
- Provide the tooling the account details for connecting to the cloud
- Specify the details required for creating the fabric and the machine image on which it will run

35.1. ADDING CLOUD DETAILS

Overview

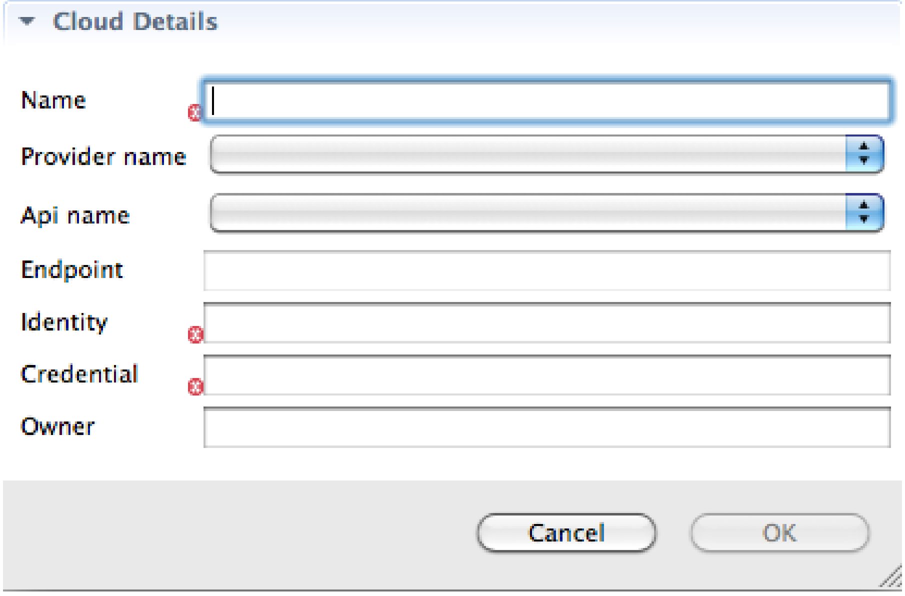
To connect to a cloud, the tooling needs the relevant access and authorization information. This information was provided to you or a system administrator by the cloud provider, when the account was set up.

Procedure

To add a cloud's access details:

1. In **`Fabric Explorer`**, right-click **`Clouds`** to open the context menu, and then select **Add Cloud details**.

   Figure 35.1. Cloud Details wizard

   NOTE

   An error icon (✖) marks required fields that lack a valid value, and the dialog banner displays the number of errors detected.

2. In **`Name`**, enter a name for the cloud.

IMPORTANT

The next three fields provide two methods for specifying the provider's information. You select either a provider's name, or you select both an API and its endpoint.

3. In **Provider name**, select the name of the cloud provider from the drop-down list.

   If you select a provider name, the tooling autofills **Api name** and **Endpoint** with the selected provider's information, so you can skip to Step 6.

4. In **Api name**, select from the drop-down list the name of the API you want to use.

   This list displays only APIs that have been installed on your local machine or environment.

5. In **Endpoint**, enter the uri of the web service that exposes the API you selected in Step 4.

   If you have an on-premises, private cloud installed, you need to get this uri from the cloud administrator.

6. In **Identity**, enter the account identifier supplied by the cloud provider.

   For example, for Amazon Elastic Compute Cloud (EC2) accounts, you'd enter the Access Key ID.

7. In **Credential**, enter the account password supplied by the cloud provider.

   For example, for Amazon Elastic Compute Cloud (EC2) accounts, you'd enter the Secret Access Key.

8. In **Owner**, for EC2 only, enter the id assigned to a custom image.

   This option is EC2-specific and applies only to custom images. To query or use a custom image, you must enter the owner id assigned to it by EC2; otherwise, the image will not appear in the list of instances.

If you entered valid access details, the tooling connects to the cloud whose access details you specified and loads the provider's images. Once that's done, the name you entered in Step 2 appears as a node under **Clouds** in **Fabric Explorer**.

Related topics

Section 32.3, “Creating a new container on a cloud”

35.2. SPECIFYING FABRIC DETAILS

After you've added the access information that the tooling needs to connect to a cloud, you can create a fabric on the cloud. This procedure assumes that you have already added the cloud details.

Overview

Creating a fabric on a cloud involves creating on the cloud a container that contains a fabric registry and specifying the machine image on which you want to create and run it.

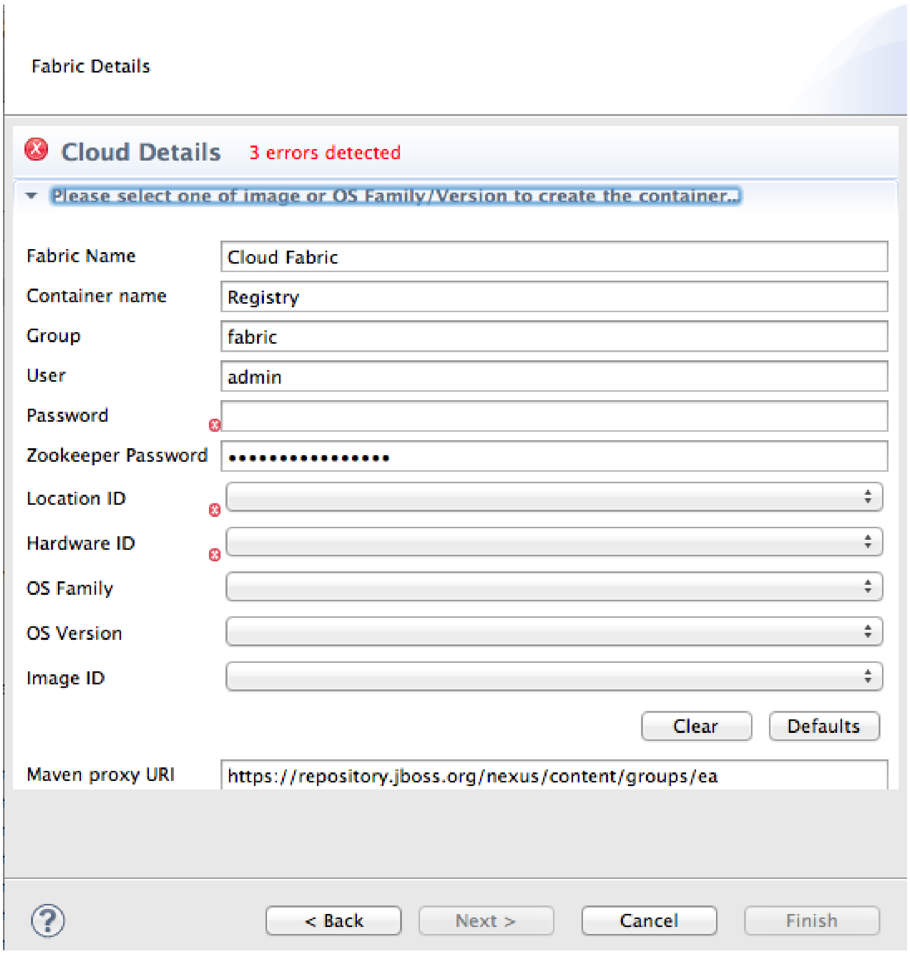
Procedure

To create a fabric on a cloud:

1. In `Fabric Explorer`, right-click **Fabrics** to open the context menu, and then select **Create Fabric in the cloud**.

2. On the **Create Fabric in the cloud** wizard's `Choose the Cloud` page, select the cloud from the list, and then click `Next` to open the wizard's `Fabric Details` page.

   Figure 35.2. Fabric Details page

   Fabric Details

   Cloud Details 3 errors detected

   Please select one of image or OS Family/Version to create the container.

   Fabric Name: Cloud Fabric
   Container name: Registry
   Group: fabric
   User: admin
   Password:
   Zookeeper Password: ••••••••••••••••
   Location ID:
   Hardware ID:
   OS Family:
   OS Version:
   Image ID:
   Maven proxy URI: https://repository.jboss.org/nexus/content/groups/ea

   ### NOTE

   An error icon (⊗) marks required fields that lack a valid value, and the dialog banner displays the number of errors detected.

   You may have to wait a few seconds while the tooling retrieves the location, hardware, and image IDs from your cloud provider to populate the wizard's corresponding fields.

3. In `Fabric Name`, accept the default name, or enter a different name for the new fabric.

4. In `Container name`, accept the default name, or enter a different name for the new container that will host the fabric registry.

5. In `Group`, accept the default name, or enter a different name for the group in which to start up the new fabric registry container.

6. In `User`, accept the default user name, or enter a different user name to use for logging into the new fabric registry container.

7. In `Password`, enter the password required for `User` to log into the specified fabric.

8. In `Zookeeper Password`, accept the default, generated password for logging into the cloud fabric's zookeeper registry, or enter a different password.

9. In `Location ID`, select from the drop-down list the ID of the region in which the machines you want to use are located.

10. In `Hardware ID`, select from the drop-down list the ID of the hardware archetype you want to use.

    Hardware archetypes specify the amount of memory and compute power available for running applications on machine images.

    ### IMPORTANT

    The next three fields provide two methods for specifying the software configuration (operating system, application server, and applications) you want to use. Either select an OS Family (and optionally a version), or select an Image ID.

11. In `OS Family`, select from the drop-down list the family name of the operating system you want to use.

    If you select an `OS Family`, do not also select an `Image ID`.

12. In `OS Version`, optionally select from the drop-down list the version for the operating system you selected.

13. If you selected an `OS Family`, skip this step. Otherwise, in `Image ID`, select from the drop-down list the ID of the image template to use for the software configuration.

14. In `Maven proxy URI`, accept the default URI, or enter the URI to the Maven proxy repository you want to use.

15. Click `Finish`.

    The tooling creates the container, then provisions it with all of the bundles, features, and profiles required to create a fabric registry. This process can take up to ten minutes. When it's done, the container specified in Step 4 appears in `Fabric Explorer` as a node under `Fabrics`.

    ### NOTE

    It's a good idea to access your cloud account and verify that the new fabric registry container is up and running.

Related topics

Section 35.1, “Adding cloud details”
Section 32.3, “Creating a new container on a cloud”

APPENDIX A. WIZARD FIELD REFERENCES

NAME

New Camel XML File — Creates a new XML file for editing routes with the route editor

PROPERTIES

Table A.1 describes the properties you can specify.

Table A.1. New Camel XML File properties

Name	Default	Description
RouteContainer		Specifies the folder into which the new file will be placed.
File Name	camelContext.xml	Specifies the name of the new routing context file.
Framework	Spring	Specifies whether the context file is intended to be used in a Spring container or in an OSGi container that supports Blueprint or to be added to into existing **`camelContext`**s.

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`** folder:

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

RELATED TOPICS

Chapter 6, *Creating a New Camel XML file*

NAME

New Fuse Project — Creates a new Fuse project

PROPERTIES

Table A.2 describes the properties you can specify.

Table A.2. New Fuse IDE project: select project location properties

Name	Default	Description
Project Name		Specifies the name of the new project.
Use default Workspace location	Enabled	Specifies whether to create the new project in the default workspace.
Location		Specifies the full path to an alternate workspace when the default workspace location option is disabled.
Add project(s) to working set	Disabled	Specifies whether to add the new project to one or more established working sets.
Working set		Specifies the working sets in which to add the new project when the working set option is enabled.

RELATED TOPICS

Chapter 5, *Creating a New Fuse Project*

NAME

New Fuse Project — Creates a new Fuse project

PROPERTIES

Table A.3 describes the properties you can specify.

Table A.3. New Fuse project: select project archetype and specify its details properties

Name	Default	Description
Archetype		Specifies the Maven archetype to use to create the new project.
Group ID		Specifies the group ID Maven uses when generating the new project. This ID is mapped to the POM file's **`groupId`** element.

Name	Default	Description
Artifact ID		Specifies the artifact ID Maven uses when generating the new project. This ID is mapped to the POM file's **`artifactId`** element.
Version	*1.0.0-SNAPSHOT*	Specifies the version name Maven uses when generating the new project. You can accept the default version name, or enter a new one. This value is mapped to the POM file's **`version`** element.
Package	*<groupID>.<artifactID>*	Specifies the package name for the generated artifacts. You can accept the default package name, or enter a new one.

RELATED TOPICS

Chapter 5, *Creating a New Fuse Project*

NAME

New Fuse Message — Creates a new message to use in route tracing.

PROPERTIES

Table A.4 describes the properties you can specify.

Table A.4. Fuse Message File properties

Name	Default	Description
Parent Folder	The selected folder	Specifies the folder into which the new message file will be placed.
File Name	message.xml	Specifies the name of the new message file.

RELATED TOPICS

Section 23.1, “Creating test messages for route tracing”

NAME

New Camel JUnit Test Case — Configures the JUnit artifacts created to test a Apache Camel route.

PROPERTIES

Table A.5 describes the properties you can specify.

Table A.5. New Camel JUnit properties

Name	Default	Description
Source Folder	*project*/src/test/java	Specifies the folder into which the JUnit code is placed.
Package	Name of the package when the project was created	Specifies the package name for the JUnit code.
Camel XML file under test	Selected context file, or blank if other than a context file is selected	Specifies the Camel XML file containing the routes to be tested.
Name	classname based on the selected context file (for example, CamelContextXMLTest), or empty if none selected	Specifies the name of the JUnit class.
Method stubs		Specifies the JUnit method stubs to include in the generated test class.
Generate comments		Specifies whether to generate comments in the new test class.

RELATED TOPICS

Chapter 8, *Creating a new Apache Camel JUnit test case*

NAME

Test Endpoints — Lists the endpoints that can be tested by a JUnit test case.

PROPERTIES

Table A.6 describes the properties you can specify.

Table A.6. Test Endpoint properties

Name	Default	Description

Name	Default	Description
Available endpoints	all	Specifies the endpoints to test in the Camel JUnit test case.

RELATED TOPICS

Chapter 8, *Creating a new Apache Camel JUnit test case*

NAME

Define a New Server — Defines a new server instance.

PROPERTIES

Table A.7 describes the properties you can specify.

Table A.7. New Server properties

Name	Default	Description
Server type		Specifies the type of server to define.
Server's host name		Specifies the name of the machine running the server.
Server name		Specifies the name of the server.
Server runtime environment	Selected server's runtime	Specifies the runtime environment of the server.

RELATED TOPICS

Section 28.1, “Adding a Server”

NAME

New Server Configuration — Configures access details for the server.

PROPERTIES

Table A.8 describes the properties you can specify.

Table A.8. Server Configuration details properties

Name	Default	Description
Host Name	For localhost, `0.0.0.0`	Specifies the address of the machine running the server.
Port Name	8101	Specifies the port used to contact the server.
User Name		[Optional] Specifies the name of a user authorized to access the server remotely.
Password		[Optional] Specifies the password the authorized user must enter to access the server remotely.

RELATED TOPICS

Section 28.1, “Adding a Server”

NAME

Add and Remove — Specifies the resources available to a server.

PROPERTIES

Table A.9 describes the properties you can specify.

Table A.9. Add and Remove properties

Name	Description
Available	Specifies the resources that are available to the server.
Configured	Specifies the resources that the server is configured to use.

RELATED TOPICS

Section 28.1, “Adding a Server”

NAME

Fabric Details — Specifies information needed to connect to a fabric.

PROPERTIES

Table A.10 describes the properties you can specify.

Table A.10. Fabric Details properties

Name	Default	Description
Name	*Local Fabric*	Specifies the name of the fabric to connect to.
Jolokia URL	*https://localhost:8181/jolokia*	Specifies the url, in the form of https://hostname:port/jolokia, of the specified fabric.
User name		Specifies the name of a user that has login privileges.
Password		Specifies the password of the specified user.
Zookeeper Password		Specifies the password for accessing the fabric registry.

RELATED TOPICS

Section 31.1, “Adding fabric details”

Section 31.4, “Editing a fabric's details”

NAME

Create Child Container — Specifies information needed to create a container on the local host.

PROPERTIES

Table A.11 describes the properties you can specify.

Table A.11. Create Child Container properties

Name	Default	Description
Container name	Container*#*	Specifies the name of the new container.
Version	The highest-numbered version according to major.minor numbering	Specifies the version of the profiles to use.

Name	Default	Description
Profiles		Specifies the profile(s) to install in the new container.

RELATED TOPICS

Section 32.1, “Creating a new child container”

NAME

Create Container via SSH — Specifies information needed to create a container on a remote host.

PROPERTIES

Table A.12 describes the properties you can specify.

Table A.12. Create Container via SSH properties

Name	Default	Description
Container name	Container*#*	Specifies the name of the new container.
Version	The highest-numbered version according to major.minor numbering	Specifies the version of the profiles to use.
Host		Specifies the name or the IP address of the remote host.
User name		Specifies the name of a user authorized to login to the remote host.
Password		Specifies the password of the specified user.
Path	*~/containers*	Specifies the path of the new container's location on the remote host.
Port	*22*	Specifies the port number to use for establishing an ssh connection on the remote host.

Name	Default	Description
SSH retries	*1*	Specifies the maximum number of retries to attempt at establishing an ssh connection on the remote host.
Retry delay	*1*	Specifies the delay, in milliseconds, between retry attempts.
Profiles		Specifies the profile(s) to install in the new container.

RELATED TOPICS

Section 32.2, “Creating a container on a remote host”

NAME

Create Container in a Cloud — Specifies information needed to create a container in a cloud.

PROPERTIES

Table A.13 describes the properties you can specify.

Table A.13. Create Container in a Cloud:New Containers details properties

Name	Default	Description
Container name	Container#	Specifies the name of the new container.
Version	The highest-numbered version according to major.minor numbering	Specifies the version of the profiles to use.
Group	*fabric*	Specifies the name of the group in which to start up the new container.
User	*admin*	Specifies the name of a user authorized to login to the new container.
Location ID		Selects the region in which the machines you want to use are located.

Name	Default	Description
Hardware ID		Selects the hardware archetype to use.
OS Family		Selects the general operating system to use.
OS Version		Selects the version of the operating system you selected.
Image ID		Selects a specific image template to use for the software configuration.
Profiles		Specifies the profile(s) to install in the new container.

RELATED TOPICS

Section 32.3, “Creating a new container on a cloud”

NAME

Create Version — Specifies the identifier for the new version of the fabric's profiles.

PROPERTIES

Table A.14 describes the properties you can specify.

Table A.14. Create Version properties

Name	Default	Description
Create a new version for rolling upgrades to Profiles	*The highest-numbered version according to major.minor numbering*	Specifies the identifier for the new version of fabric's profiles.

RELATED TOPICS

Section 34.1, “Creating a new version of a profile”

NAME

Cloud Details — Specifies information needed to connect to a cloud.

PROPERTIES

Table A.15 describes the properties you can specify.

Table A.15. Cloud Details properties

Name	Default	Description
Name		Specifies the name of the cloud to connect to.
Provider name		Selects the cloud provider.
Api name		Selects the API to use on the specified cloud.
Endpoint		Specifies the cloud endpoint uri.
Identity		Specifies the account identifier (supplied by the cloud provider).
Credential		Specifies the account password (supplied by the cloud provider).
Owner		Specifies the name of the account owner.

RELATED TOPICS

Section 35.1, “Adding cloud details”

NAME

Create Fabric in the Cloud — Specifies information needed to create a fabric in a cloud.

PROPERTIES

Table A.16 describes the properties you can specify.

Table A.16. Create Fabric in a Cloud:Fabric details properties

Name	Default	Description
Fabric name	*Cloud Fabric*	Specifies the name of the new fabric.
Container name	*Registry*	Specifies the name of the fabric registry container.

Name	Default	Description
Group	*fabric*	Specifies the name of the group in which to start up the fabric registry container.
User	*admin*	Specifies the name of a user authorized to login to the fabric registry container.
Password		Specifies the user's password for logging into the the fabric.
Zookeeper Password	****************	Specifies the zookeeper password for logging into the cloud fabric's zookeeper registry.
Location ID		Selects the region in which the machines you want to use are located.
Hardware ID		Selects the hardware archetype to use.
OS Family		Selects the general operating system to use.
OS Version		Selects the version of the operating system you selected.
Image ID		Selects a specific image template to use for the software configuration.
Maven proxy URI	*http://repo.fusesource.com/nexus/content/groups/ea*	Specifies the Maven proxy repository to use.

RELATED TOPICS

Section 35.2, “Specifying fabric details”

APPENDIX B. TOOLING PREFERENCES

NAME

Editor — Sets the default value for some route editor properties.

PROPERTIES

Table B.1 describes the properties you can specify.

Table B.1. Editor Properties

Name	Default	Description
Language	Simple	Specifies the default expression language used by the route editor.
ID values	Enabled	Determines whether ID values are used to label EIP nodes on the route editor's canvas.
Layout direction	Right	Specifies the direction in which routes are laid out on the route editor's canvas.
Diagram grid	Enabled	Determines whether a grid overly is displayed in the background on the route editor's canvas.

RELATED TOPICS

Section 7.10, “Configuring the route editor”

NAME

Colors — Sets the default colors for diagrammatic components of Route editor.

PROPERTIES

Table B.2 describes the properties you can specify.

Table B.2. Colors Properties

Name	Description
Route Editor Diagram Grid Color	Specifies the default color for the diagram grid.
Route Editor Diagram Connection Color	Specifies the default color for the connection in the diagram.

Name	Description
Route Editor Diagram Text Color	Specifies the default color for the text in the diagram.
Route Editor Diagram Figure Background	Specifies the default color of EIP node background.
Route Editor Diagram Figure Foreground	Specifies the default color of EIP node foreground.

RELATED TOPICS

Section 7.10, “Configuring the route editor”