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

PREFACE .................................................................................................................. 8

PART I. INTRODUCING FUSE TOOLING ................................................................. 9

CHAPTER 1. ABOUT RED HAT JBOSS FUSE TOOLING ........................................ 10

CHAPTER 2. FUSE INTEGRATION PERSPECTIVE ............................................... 12
  OVERVIEW .............................................................................................................. 12
  RELATED TOPICS .................................................................................................. 16

CHAPTER 3. DEBUG PERSPECTIVE ..................................................................... 17
  OVERVIEW .............................................................................................................. 17
  RELATED TOPICS .................................................................................................. 18

PART II. DEVELOPING APPLICATIONS ................................................................. 19

CHAPTER 4. CREATING A NEW FUSE INTEGRATION PROJECT ......................... 20
  OVERVIEW .............................................................................................................. 20
  CONFIGURING PROJECT BASICS .......................................................................... 20
  CREATING A NEW TARGET RUNTIME .................................................................... 22
  SETTING ADVANCED PROJECT OPTIONS ............................................................. 25
  RESOLVING MAVEN DEPENDENCY ERRORS ......................................................... 29

CHAPTER 5. CREATING A NEW CAMEL XML FILE ............................................. 31
  OVERVIEW .............................................................................................................. 31
  PROCEDURE .......................................................................................................... 31

CHAPTER 6. EDITING A ROUTING CONTEXT IN THE ROUTE EDITOR ..................... 33
  6.1. ADDING ROUTES TO THE ROUTING CONTEXT ............................................ 33
      Overview ............................................................................................................. 33
      Procedure .......................................................................................................... 33
  6.2. ADDING PATTERNS TO A ROUTE .................................................................. 34
      Procedure .......................................................................................................... 35
      Related topics .................................................................................................... 35
  6.3. CONNECTING PATTERNS TO MAKE A ROUTE ............................................ 35
      Overview ............................................................................................................. 36
      Manually connecting nodes .............................................................................. 36
  6.4. CONFIGURING A PATTERN .......................................................................... 36
      Overview ............................................................................................................. 36
      Procedure .......................................................................................................... 37
  6.5. REMOVING PATTERNS FROM A ROUTE ..................................................... 37
      Overview ............................................................................................................. 37
      Procedure .......................................................................................................... 37
      Related topics .................................................................................................... 37
  6.6. DELETING A ROUTE ....................................................................................... 37
      Overview ............................................................................................................. 38
      Procedure .......................................................................................................... 38
  6.7. MANUALLY ADDING BEANS AND CONFIGURATION .................................... 38
      Overview ............................................................................................................. 39
      Procedure .......................................................................................................... 39
      Related topics .................................................................................................... 40
  6.8. ADDING GLOBAL ENDPOINTS, DATA FORMATS OR BEANS ..................... 40
      Overview ............................................................................................................. 40
      Adding a global endpoint ................................................................................. 42
CHAPTER 11. USING THE JBOSS FUSE SAP TOOL SUITE ........................................... 88

11.1. INSTALLING THE JBOSS FUSE SAP TOOL SUITE .................................................. 88
    Overview .................................................................................................................. 88
    Platform restrictions for SAP tooling ........................................................................... 88
    Prerequisites ............................................................................................................. 88
    Procedure .................................................................................................................. 88

11.2. CREATE AND TEST SAP DESTINATION CONNECTION ............................................. 89
    Overview .................................................................................................................. 89
    Procedure .................................................................................................................. 89

11.3. CREATE AND TEST SAP SERVER CONNECTION ....................................................... 92
    Overview .................................................................................................................. 92
    Procedure .................................................................................................................. 92

11.4. DELETING DESTINATION AND SERVER CONNECTIONS ......................................... 93
    Overview .................................................................................................................. 93
    Procedure .................................................................................................................. 93

11.5. CREATE A NEW SAP ENDPOINT .............................................................................. 93
    Overview .................................................................................................................. 94
    Prerequisites ............................................................................................................. 94
    Procedure .................................................................................................................. 94

CHAPTER 12. GETTING STARTED WITH DATA TRANSFORMATION .................................... 96

12.1. FUSE TRANSFORMATION TOOLING ...................................................................... 96

12.2. DATA TRANSFORMATION TUTORIAL ..................................................................... 96
    Prerequisites ............................................................................................................. 96
    Importing the starter quickstart application ............................................................... 96
    Adding a data transformation node to the Camel route ................................................. 99
    Mapping source data items to target data items .......................................................... 103
    Creating the transformation test file and running the JUnit test .................................... 106
    Mapping a constant variable to a data item ............................................................... 107
    Mapping an expression to a data item ......................................................................... 109
    Adding a custom transformation to a mapped data item .............................................. 113
    Mapping a simple data item to a data item in a collection .......................................... 118
    Adding a built-in function to a mapped data item ....................................................... 120
    Publishing a Fuse Integration project with data transformation to a Red Hat JBoss Fuse ................................................................................................................. 123

CHAPTER 13. DEVELOPING EXTENSIONS FOR IGNITE INTEGRATIONS ............................. 125

PREREQUISITES ........................................................................................................ 125
    CREATING AN IGNITE EXTENSION PROJECT ......................................................... 125
    DEVELOPING THE STEPS FOR THE IGNITE EXTENSION ....................................... 126
    BUILDING THE IGNITE EXTENSION JAR FILE ....................................................... 128
    NEXT STEPS ............................................................................................................. 129

CHAPTER 14. CHANGING THE CAMEL VERSION .............................................................. 130

PART III. DEBUGGING ROUTING CONTEXTS ................................................................. 131

CHAPTER 15. SETTING BREAKPOINTS ........................................................................... 132
    OVERVIEW .............................................................................................................. 132
    SETTING UNCONDITIONAL BREAKPOINTS .............................................................. 132
    SETTING CONDITIONAL BREAKPOINTS .................................................................... 132
    DISABLING BREAKPOINTS ....................................................................................... 133
    DELETING BREAKPOINTS .......................................................................................... 134
    RELATED TOPICS .................................................................................................... 134

CHAPTER 16. RUNNING THE CAMEL DEBUGGER ............................................................... 135
16.1. DEBUGGING A LOCALLY RUNNING ROUTING CONTEXT ................................. 135
   Procedure 135
   Watching message exchanges progress through the routing context 137

16.2. DEBUGGING A REMOTELY RUNNING ROUTING CONTEXT ......................... 137
   Prerequisites 137
   Setting up and starting remote debugging 138
   Procedure 139

RELATED TOPICS 144

CHAPTER 17. STOPPING THE CAMEL DEBUGGER ............................................ 145
   OVERVIEW 145
   CLOSING THE CAMEL DEBUGGER 145
   RELATED TOPICS 145

CHAPTER 18. CHANGING VARIABLE VALUES ............................................... 146
   OVERVIEW 146
   PROCEDURE 146
   RELATED TOPICS 147

CHAPTER 19. ADDING VARIABLES TO THE WATCH LIST .................................... 148
   OVERVIEW 148
   PROCEDURE 148
   RELATED TOPICS 149

CHAPTER 20. DISABLING BREAKPOINTS IN A RUNNING CONTEXT ...................... 150
   OVERVIEW 150
   DISABLING AND ENABLING BREAKPOINTS IN BREAKPOINTS VIEW 150

PART IV. MONITORING AND TESTING APPLICATIONS ...................................... 152

CHAPTER 21. JMX NAVIGATOR ........................................................................ 153
   21.1. VIEWING PROCESSES IN JMX ......................................................... 154
         Overview 154
         Viewing processes in a local JMX server 154
         Viewing processes in alternate JMX servers 154
   21.2. ADDING A JMX SERVER .................................................................... 154
         Overview 154
         Procedure 154

CHAPTER 22. VIEWING A COMPONENT’S JMX STATISTICS ................................ 156
   OVERVIEW 156
   PROCEDURE 156

CHAPTER 23. BROWSING MESSAGES ................................................................. 158
   OVERVIEW 158
   PROCEDURE 158
   RELATED TOPICS 159

CHAPTER 24. TRACING ROUTES ...................................................................... 160
   24.1. CREATING TEST MESSAGES FOR ROUTE TRACING ......................... 160
         Overview 160
         Creating a new folder to store test messages 160
         Creating a test message 161
         Related topics 161
   24.2. ACTIVATING ROUTE TRACING ......................................................... 161
         Overview 161
Procedure

29.3. CONNECTING TO A RUNNING SERVER
  Overview
  Connecting to a running server in the Servers view
  Connecting to a running server in the JMX Navigator view
  Viewing bundles installed on the connected server

29.4. DISCONNECTING FROM A SERVER
  Overview
  Disconnecting from a server in the Servers view
  Disconnecting from a server in the JMX Navigator view

29.5. STOPPING A SERVER
  Overview
  Using the Servers view
  Using the remote console

29.6. DELETING A SERVER
  Overview
  Deleting a server
  Deleting the server's configuration

CHAPTER 30. PUBLISHING FUSE INTEGRATION PROJECTS TO A SERVER

OVERVIEW
PUBLISHING FUSE PROJECTS AUTOMATICALLY WHEN RESOURCES CHANGE
PUBLISHING FUSE PROJECTS MANUALLY
VERIFYING THE PROJECT WAS PUBLISHED TO THE SERVER

APPENDIX A. TOOLING PREFERENCES

A.1. EDITOR
PREFACE

Red Hat JBoss Fuse Tooling provides a set of developer tools that enable you to work with Fuse versions of Apache ActiveMQ, Apache Camel, and Apache CXF 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, debug routes, and deploy your project’s code to both Red Hat JBoss Fuse and Red Hat JBoss EAP containers.
PART I. INTRODUCING FUSE TOOLING

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, and an integrated testing platform that uses JMX to communicate with the containers in a small-scale test environment.

The tooling provides two main perspectives. You use Chapter 2, *Fuse Integration Perspective* to design, test and publish your Fuse Integration projects. You use Chapter 3, *Debug Perspective* to debug your routing context files.
CHAPTER 1. ABOUT RED HAT JBOSS FUSE TOOLING

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

NOTE

You can also develop Fuse projects using Maven (for details, see Developing and Deploying Applications).

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 routing 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.

See the 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
CHAPTER 2. FUSE INTEGRATION PERSPECTIVE

OVERVIEW

The Fuse Integration perspective, shown in Figure 2.1, “Fuse Integration perspective”, provides access to all of the tooling for designing, monitoring, testing, and publishing your integration application.

Figure 2.1. Fuse Integration perspective

NOTE

You can open the Fuse Integration perspective in two ways:

- Have the tooling switch to the Fuse Integration perspective when you create a new Fuse Integration project (see the section called “Setting advanced project options”).

- 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.

The Fuse Integration perspective consists of nine main areas:

- **Project Explorer** view – displays all projects known to the tooling. You can view all artifacts that make up each project. The Project Explorer view also displays all routing context .xml files for a project under its Camel Contexts node. This enables you to find and open a routing context file included in a project. Under each routing context .xml file, the Project Explorer view displays all routes defined within the context. For multiroute contexts, this lets you focus on a specific route on the canvas.
The route editor – The route editor provides the main design-time tooling and consists of three tabs:

- **Design** – Displays a large grid area on which routes are constructed and a palette from which Enterprise Integration Patterns (EIPs) and Camel components are selected and then connected on the canvas to form routes.
The canvas is the route editor’s workbench and where you do most of your work. It displays a graphical representation of one or more routes, which are made up of connected EIPs and Camel components (called nodes once they are placed on the canvas).

Selecting a node on the canvas populates the Properties view with the properties that apply to the selected node, so you can edit them.

The Palette contains all of the patterns and Camel components needed to construct a route and groups them according to function — Components, Routing, Control Flow, Transformation, and Miscellaneous.

- **Source** — Displays the .xml configuration corresponding to the routes constructed on the route editor’s canvas.
  You can edit the routing context in the Source tab as well as in the Design tab.
- **Configurations** – Provides an easy way to add shared configuration (global endpoints, data formats, beans) to a multiroute, routing context. For details see Section 6.8, “Adding global endpoints, data formats or beans”.

- **Properties view** – displays the properties of the node selected on the canvas.
**JMX Navigator** view — lists the JMX servers and the infrastructure they monitor. It enables you to browse JMX servers and the processes they are monitoring. It also identifies instances of Red Hat processes.

The **JMX Navigator** view drives all monitoring and testing activities in the **Fuse Integration** perspective. It determines which routes are displayed in the **Diagram View**, the **Properties** view, and the **Messages View**. It is also provides menu commands for activating route tracing, adding and deleting JMS destinations, and starting and suspending routes. It is also the target for dragging and dropping messages onto a route.

By default, the **JMX Navigator** view shows all Java processes that are running on your local machine. You can add JMX servers as needed to view infrastructure on other machines.

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

When you select a broker, the **Diagram View** displays up to three children: connections, topics, and queues. It also shows configured connections and destinations as grandchildren.

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

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

When a JMS destination is selected in the **JMX Navigator** view, the view lists all messages that are at the destination.

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

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

**Servers** view — displays a list of servers managed by the tooling. It displays their runtime status and provides controls for adding, starting and stopping them and for publishing projects to them.

**Terminal** view — displays the command console of the connected container. You can control the container by entering commands in the **Terminal** view.

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

**RELATED TOPICS**

- **Part II, “Developing Applications”**
- **Part IV, “Monitoring and Testing Applications”**
- **Part V, “Publishing Applications to a Container”**
CHAPTER 3. DEBUG PERSPECTIVE

OVERVIEW

The Debug perspective, shown in Figure 3.1, “Debug perspective”, 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 3.1. Debug perspective

- **Debug view**
  
  For the running routing context, the Debug view displays the debug stack.

  You can switch between breakpoints within the same message flow, listed under the Camel Context at service:jmx:rmi://jndi/rmi://localhost:1099/jmxrmi/camel entry, to review and compare variable values in the Variables view.

  Messages flows are identified by their unique breadcrumb ID, and the breadcrumb ID of each subsequent message flow is incremented by 2. For example, in Figure 3.1, “Debug perspective”, the breadcrumb ID for the first message flow is **ID-janemurpheysmbp-home-54620-1470949590275-0-1**, so the breadcrumb ID for the second message flow would be **ID-janemurpheysmbp-home-54620-1470949590275-0-3**.

- **Variables view**
  
  For each node in the routing context that has a breakpoint set, the Variables view displays the value of the available variables when the breakpoint is hit. Each variable who’s value changed since the preceding breakpoint is highlighted in yellow.

  You can change the value of editable variables to check whether such changes produce the expected results and to test the robustness of your routing context.

  You can also add variables to the watch list, so you can quickly and easily see whether their values change as expected at the expected point in the message flow.
Breakpoints view
Displays a list of the breakpoints set in the routing context, and shows whether they are enabled or disabled. You can enable and disable individual breakpoints by checking (enabling) or unchecking (disabling) them. This enables you to temporarily focus on nodes in your routing context that are behaving problematically.

The button skips over disabled breakpoints to jump to the next active breakpoint in the routing context. In contrast, the button jumps to the next node of execution in the routing context, regardless of breakpoints.

camel Context.xml view
Displays the running routing context file in graphical mode. For nodes set with breakpoints, it shows the type of breakpoint set and whether the breakpoint is enabled or disabled. When a breakpoint is hit, its corresponding node on the canvas is outlined in red.

To check a node's configuration, open the Properties view and then select the node on the canvas in camel Context.xml.

Console view
Displays the log output generated by the Camel debugger as it executes the routing context.

Properties view
Displays the properties set for the node selected on the canvas in CamelContext.xml.

RELATED TOPICS

- Part III, “Debugging Routing Contexts”
NOTE

You can also develop Fuse projects using Maven. For details, see Getting Started with Developing.
CHAPTER 4. CREATING A NEW FUSE INTEGRATION PROJECT

OVERVIEW

Creating a new Fuse Integration project involves three steps.

- Configuring project basics
- Creating a new target runtime or selecting an existing target runtime or Camel version on which to base the project
- Setting the advanced project options

Once the project has been configured, the tooling configures all of the required Maven dependencies and creates the POM file needed to run and publish the project. Configured templates, based on common use cases, also provide sample code and data to get you started quickly.

CONFIGURING PROJECT BASICS

1. Select File → New → Project → JBoss Fuse → Fuse Integration Project to open the New Fuse Integration Project wizard:

The wizard opens with the Use default workspace location option selected in the Location pane.

2. In Project Name, enter a name for the new project.

3. Specify the workspace location where the data for the project will be stored.
   a. To use the default workspace leave Use default workspace location enabled.
b. To use an alternative location clear **Use default workspace location** and specify a new location in the **Path** field. Click **Browse** to quickly find and select the alternate workspace.

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

You can base your new Fuse Integration project on a target runtime or on a specific version of Camel.

- **Target Runtime** – Specify the runtime to which you want to publish the new project. You can either select an existing configured runtime from the drop-down menu, or create a new one.

  In either case, the **Camel Version** pane displays the Camel version associated with the runtime once it is selected.

- **Camel Version** – You can specify the Camel version on which to base the new project, without creating or selecting a target runtime. The tooling supports the productised options available from the drop-down list, but allows you to type a different Camel version, if you want to experiment with non productised versions (which are not supported).

  **IMPORTANT**

  Camel versions 2.18.1.redhat-000012 and 2.18.1.redhat-000015 work only with Fuse Integration Services projects (for details, see [olink:RiderUG/RiderFisTools](#)). For Fuse Integration projects, select one of the other available productised Camel versions or type in a non productised version.

If you select an existing, configured target runtime or a specific Camel version on which to base your new Fuse Integration project, you need only set the project’s advanced options (See the section called [CHAPTER 4. CREATING A NEW FUSE INTEGRATION PROJECT](#)).
“Setting advanced project options”. Otherwise, you need to create a new target runtime before you set the project’s advanced options (see the section called “Creating a new target runtime”).

When you enter a non productised Camel version, you can click the Verify button to check whether the tooling can access the specified version. If not, a notification appears in the Select a Target Runtime header:

**Select a Target Runtime**

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

After you create, configure and save a project, it is possible to change the Camel version. See Chapter 14, *Changing the Camel version*.

**CREATING A NEW TARGET RUNTIME**

With the Select a Target Runtime page open:

1. In the Target Runtime pane, click New to open the New server runtime environment page:
2. Expand the Red Hat JBoss Middleware folder, and then select a JBoss Fuse runtime environment (for example JBoss Fuse 6.3).
Leave the Create a new local server option disabled. You can create the local server later when you are ready to publish your project to it (for details, see Section 29.1, “Adding a Server”).

**NOTE**
If you click the Create a new local server option, the New Fuse Integration Project wizard walks you through additional steps to define and configure the new JBoss Fuse 6.3 server runtime (as described in Section 29.1, “Adding a Server”). Then, when it builds the project, it also adds the server runtime to the Servers view in the Fuse Integration perspective.

3. Click Next to open the server's Runtime Definition page:
4. Specify the **Name**, **Home Directory**, and **Execution Environment** of the server runtime:

- **Name** — accept the default, or enter a new name for the runtime environment

- **Home Directory** — click the **Browse** button to locate and select the server runtime’s installation directory

  **NOTE**
  
  If the server is not already installed on your machine, you can install it now by clicking the 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.

- **Execution Environment** — accept the default, or select another JavaSE version from the drop-down menu
  
  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 6.3 requires either JRE version 1.8 (recommended) or 1.7.

- **Alternate JRE** — Leave as is
If you need to modify the configuration of an existing target runtime, you do so by selecting, from the menu bar, JBoss Developer Studio → Preferences → JBoss Fuse → Server → Runtime Environments to open the Server Runtime Environments page. (On Linux and Windows machines, you access Preferences through the Windows menu.) In the Name pane, select the runtime you want to modify, and click Edit to open the server’s Runtime Definition page. You can change the Name of the target runtime, its Home Directory, and its Execution Environment.

5. Click Finish to return to the Select a Target Runtime page:

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

Target Runtime
Select the target runtime you want to deploy your project to

JBoss Fuse 6.3 Runtime
New

Camel Version
Select the Camel version for your new project

2.17.0.redhat-630262
Verify

The newly configured target runtime appears in the Target Runtime pane’s drop-down menu, and the Camel version supported by the runtime appears in the Camel Version pane, grayed out.

To finish, set the advanced project options as described in the section called “Setting advanced project options”.

After you create and save a target runtime, it is possible to change the Camel version. See Chapter 14, Changing the Camel version.

SETTING ADVANCED PROJECT OPTIONS

With the completed Select a Target Runtime page open:
1. Click Next to open the Advanced Project Setup page. You can create an empty project, which creates a skeleton Camel context routing file based on one of the three supported Domain Specific Languages (DSLs), or you can use a predefined template, each of which is based on a common use case. Individual templates may not support all DSL options.

NOTE

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

- Start with an empty project:

Advanced Project Setup

Please select how you would like to setup your project.

What would you like to do?

- Start with an empty project
- Use a predefined template

Select the project type

- Blueprint DSL
- Spring DSL
- Java DSL

For an empty project, all DSL options are available.
Use a predefined template:

**Advanced Project Setup**

Please select how you would like to setup your project.

What would you like to do?

- [ ] Start with an empty project
- [x] Use a predefined template

Select the project type

- [x] Blueprint DSL
- [ ] Spring DSL
- [ ] Java DSL

The options are:

- **ActiveMQ** — Provides a sample Camel route that reads files from a specific location and routes them to a message queue. Another route, implementing a content-based router, pulls the messages off the queue and routes each to its target destination, according to message content.

**NOTE**

Before you can deploy this project, you must create, in your Red Hat JBoss Fuse installation, a pid file that configures the ActiveMQ connection properties. For details, open the README.txt file, listed under the project root in the Project Explorer view.
This template creates a project that runs on Red Hat JBoss Fuse servers, and it supports the Blueprint and Spring DSLs only.

- **Content Based Router** — Provides a sample Camel route that reads files from a specific location and routes them to different output folders according to message content. This template creates a project that runs on Red Hat JBoss Fuse servers, and it supports all three of the DSLs.

- **CXF code first** — Provides a sample Camel route that is started by a CXF web service call. This template creates a project that runs on Red Hat JBoss Fuse servers, and it supports the Spring and Java DSLs only.

- **Spring on EAP** — Provides a sample Camel route that calls into a bean service to say Hello. This template creates a project that runs on Red Hat EAP 6.4 servers, and it supports the Spring DSL only.

- **SpringBoot on OpenShift** — Demonstrates how to configure Camel routes in Spring Boot using a Spring XML configuration file. This template creates a Fuse Integration Services project and requires Camel version 2.18.1.redhat-000012 or 2.18.1.redhat-000015. This template creates a project that runs on OpenShift servers, and it supports the Spring DSL only. For details on using this template, see olink:RiderUG/RiderFisTools.

2. Select one of the project setup options and a DSL (as available), and then click Finish. The tooling starts building the project, adding it to the Project Explorer view. If you are not already in the Fuse Integration perspective, the tooling asks whether you want to switch to it now:

    ![Decision to switch perspective](image)

3. Click Yes to open the new project in the Fuse Integration perspective.
In this example, the CBRroute project appears in the Project Explorer view. CBRroute would have been the name entered in the Project Name field in \{config\ProjectName\}.

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

```
<camelContext id="cbr-example-context" xmlns="http://camel.apache.org/schema/blueprint">
  <route id="cbr-route">
    <from id="from1" uri="file:work/cbr/input"/>
    <log id="log1" message="Receiving Order $\{file:name\}"/>
    <choice id="chosen1">
      <when id="when1">
        <xpath>/order/order/order:customer/order:country = 'UK'</xpath>
        <log id="log2" message="Sending order $\{file:name\} to the UK"/>
        <to id="to1" uri="file:work/cbr/output/uk"/>
      </when>
      <when id="when2">
        <xpath>/order/order/order:customer/order:country = 'US'</xpath>
        <log id="log3" message="Sending order $\{file:name\} to the US"/>
        <to id="to2" uri="file:work/cbr/output/us"/>
      </when>
      <otherwise id="otherwise1">
        <log id="log4" message="Sending order $\{file:name\} to another country"/>
        <to id="to3" uri="file:work/cbr/output/others"/>
      </otherwise>
    </choice>
    <log id="log5" message="Done processing $\{file:name\}"/>
  </route>
</camelContext>
```

RESOLVING MAVEN DEPENDENCY ERRORS
You may encounter Maven dependency errors after you create a new Fuse Integration project.

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

You can often resolve these dependency errors by updating Maven dependencies this way:

1. In the Project Explorer view, 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:
   - Select the project you want to update, if more than one appears in the wizard's list.
   - Click the Force Update of Snapshots/Releases option to enable it.
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.
CHAPTER 5. 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 a new Apache Camel context file to your project:

1. Select File → New → Other → JBoss Fuse → Camel XML File from the main menu to open the Camel XML File wizard, as shown in Figure 5.1, “Camel XML File wizard”.

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

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

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

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

3. In File Name, enter a name for the new context file, or accept the default (camelContext.xml).
   The file’s name cannot contain spaces or special characters, and it must be unique within the JVM.

4. In Framework, accept the default, or select which framework the routes will use:
   - **Spring** – [default] for routes that will be deployed in Spring containers, non-OSGi containers, or as standalone applications
   - **OSGi Blueprint** – for routes that will be deployed in OSGi containers
   - **Routes** – for routes that you can load and add into existing camelContexts

5. Click Finish.
   The new context file is added to the project and opened in the route editor.
CHAPTER 6. EDITING A ROUTING CONTEXT IN THE ROUTE EDITOR

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

1. **Section 6.1, “Adding routes to the routing context”** one or more routes to the routing context.
2. **Section 6.2, “Adding patterns to a route”** a starting point pattern to a route.
3. **Section 6.2, “Adding patterns to a route”** one or more endpoint patterns to a route.
4. **Section 6.2, “Adding patterns to a route”** one or more processor patterns that represent how messages will be transformed and routed between the starting point and endpoints.
5. **Section 6.3, “Connecting patterns to make a route”** the patterns (referred to as nodes once they are placed on the canvas).
6. **Section 6.4, “Configuring a pattern”** the details for each of the endpoints and processors that make up the route.
7. **Section 6.7, “Manually adding beans and configuration”** any configuration beans or shared global components/data formats to the context.

6.1. 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, and each route, displayed on the canvas as a Route container node, maps to a route element in the generated camelContext element.

When you create a new Fuse Integration Project, the tooling creates an example camelContext.xml (Spring) or blueprint.xml (Blueprint) routing context file. You can view and edit the contents of the routing context file in the route editor's Source tab. In the Design tab, the route editor displays a Route container node, which represents the empty route element. You can drag patterns from the Palette and drop them into the Route container node on the canvas to create a route. The Camel tooling updates the empty route element with XML code generated from the patterns you dropped into the Route container node on the canvas.

The tooling provides two methods for adding a new route:

- In the Design tab, by dragging a Route pattern from the Palette’s Routing drawer and dropping it onto the canvas
- In the Source tab, by adding a <route/> element to the existing list within the camelContext element

**Procedure**

To add another route to the camelContext: element

1. Select one of the methods for adding a route.
   - With the Design tab selected, notice that the Properties view displays the list of the new route’s properties for you to edit.
2. In the Properties view, enter:
   - An ID (for example, Route2) for the new route in the route’s Id field
   
   **NOTE**

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

   - A description of the route in the Description field
   - A value for any other property as needed

3. 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 its entry under the project’s Camel Contexts folder in the Project Explorer view.

   ![Project Explorer](image)

   When you click the context file entry in the Project Explorer view, the canvas displays all routes in the context, as space allows.

### 6.2. ADDING PATTERNS TO A ROUTE

Routes consist of a sequence of connected patterns, referred to as nodes once they are placed on the canvas inside a Route container node. To add a pattern to a route you drag it from the Palette and drop it into a Route container on the canvas.

When patterns are dropped into a Route container on the canvas, they take on a color that indicates the type of node they are:

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

**Procedure**

To drag a pattern onto a route:

1. In the **Palette**, locate the pattern you want to add to the route.

2. Drag the pattern over the target **Route** container and drop it.

   Alternatively, you can drop a pattern on an existing node that has no outgoing connection, or on a connection existing between two nodes, to have the tooling automatically wire the connections between all nodes involved.

The tooling checks whether the resulting connection is valid and then either allows or prevents you from dropping the pattern on the drop target. For valid connections, the tooling:

- existing node — adds the new node to the target node’s outgoing side (beneath or to the right of it depending on how the editor preferences are set) and automatically wires the connection between them
- existing connection — inserts the new node between the two connected nodes and automatically rewires the connections between the three nodes

The new pattern appears on the canvas in the **Route** container and becomes the selected node. The **Properties** view displays a list of the new node’s properties for you to edit.

**NOTE**

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

**NOTE**

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

To access the route editor’s layout preference:

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

**Related topics**

- Section 6.4, “Configuring a pattern”
- Section 6.5, “Removing patterns from a route”

6.3. CONNECTING PATTERNS TO MAKE A ROUTE
Overview

A complete route typically consists of a starting endpoint, a string of processing nodes, and one or more destination endpoints. The tooling saves routes in the context file regardless of whether they are complete.

**IMPORTANT**

The tooling appends every new node added in the Design tab to its route element in the context file, and any that are not connected manually, using the connector arrows, are auto-connected when the context file is reopened, or when a route is reloaded in the route editor, by switching from the Source tab to the Design tab.

**NOTE**

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

Manually connecting nodes

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

To connect two nodes:

1. In the Route container on the canvas, select the source node to display its connector arrow.

2. 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 in the routing context file with the xml generated from the connection. You can view the xml in the Source tab.

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

6.4. CONFIGURING A PATTERN

Overview

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

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

- validating that all required properties have values
- validating that supplied values are the correct data type for the property
- drop-down lists for properties that have a fixed set of values
- drop-down lists that are populated with the available bean references from the Apache Camel Spring configuration

**Procedure**

To configure a pattern:

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

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

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

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

### 6.5. REMOVING PATTERNS FROM A ROUTE

**Overview**

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

**NOTE**

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

**Procedure**

To remove a node from a route:

1. Select the node you want to delete.

2. Click its icon.

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

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

**Related topics**

- Section 6.2, “Adding patterns to a route”

### 6.6. DELETING A ROUTE
Overview

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

**NOTE**

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

**IMPORTANT**

You cannot undo this operation.

Procedure

To delete a route:

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

2. On the canvas, click the Route container’s icon.

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

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

6.7. MANUALLY ADDING BEANS AND CONFIGURATION
Overview

Many routing patterns rely on references to Java objects (beans) for configuration or for implementation details. You manually add the beans into the routing context file by using the route editor's Source tab. However, this approach is not recommended because it is error prone. The recommendation is to use the Configurations tab. See Section 6.8, “Adding global endpoints, data formats or beans”.

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 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.

For example:

```xml
<!-- Configure the Inbound SAP Connections -->
<bean id="sap-configuration"
    class="org.fusesource.camel.component.sap.SapConnectionConfiguration"
    activation="eager">
    <property name="destinationDataStore">
        <map>
            <entry key="quickstartDest" value-ref="quickstartDestinationData" />
        </map>
    </property>
    <property name="serverDataStore">
        <map>
            <entry key="quickstartServer" value-ref="quickstartServerData" />
        </map>
    </property>
</bean>
...
<camelContext trace="false" id="sap-srfc-destination-fuse-context"
xmns="http://camel.apache.org/schema/blueprint">
    <route id="sap-srfc-destination-fuse-route">
        <from uri="file:work/camel-sap/input"/>
        <convertBodyTo type="java.lang.String"/>
        <log message="${body}" loggingLevel="INFO"/>
        <to uri="sap-srfc-destination:quickstartDest:BAPI_FLCUST_GETLIST"/>
    </route>
</camelContext>
```
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 the graphic display and the route diagram.

**Related topics**

- Chapter 7, *The Source Tab*
- Section 6.8, “Adding global endpoints, data formats or beans”
- *Apache Camel Development Guide*, Part IV Programming EIP Components

### 6.8. ADDING GLOBAL ENDPOINTS, DATA FORMATS OR BEANS

**Overview**

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

To add global elements to your routing context file:

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

2. At the bottom of the route editor, click the **Configurations** tab to display global configurations, if there are any.
3. Click Add to open the Create a new global element dialog.
The options are:

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

### Adding a global endpoint

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

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

NOTE

The Grouped by categories option groups components by type.

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

3. Click Finish.

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

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

NOTE

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

4. To complete the component’s uri, click Advanced → Path.

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

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

6. For example, click the Consumer (advanced) tab.

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

7. In the route editor, switch to the Source tab to see the code that the tooling added to the context file (in this example, a configured JMS endpoint), before the first route element.
8. When done, save your changes by selecting File → Save on the menu bar.

Adding a global data format

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

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

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

4. Click Finish.
5. In the Properties view, set property values as appropriate for your project, for example:
6. In the route editor, click the Source tab to see the code that the tooling added to the context file. In this example, a configured xmljson data format is before the first route element.

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

**Adding a global bean**

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

To add a global bean element:

1. In the Create a new global element window, select Bean and click OK to open the Bean Definition dialog.
2. In the Id field, enter an ID for the global bean, for example, **TransformBean**. The ID must be unique in the configuration.

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

   To fill the Class field, do one of the following:
Enter the name of a class that is in the project or in a referenced project.

Click … (Browse) to navigate to and select a class that is in the project or in a referenced project.

Click + to define a new bean class and add it as a global bean.

4. If the bean you are adding requires one or more arguments, in the Constructor Arguments section, for each argument:
   a. Click Add.
   b. Optionally, in the Type field, enter the type of the argument. The default is java.lang.String.
   c. In the Value field, enter the value of the argument.
   d. Click OK.

5. Optionally specify one or more properties that are accessible to the global bean. In the Bean Properties section, do the following for each property:
   a. Click Add.
   b. In the Name field, enter the name of the property.
   c. In the Value field, enter the value of the property.
   d. Click OK.

6. Click Finish to add the global bean to the configuration. The global bean ID you specified appears in the Configurations tab, for example:

   ![Blueprint XML](image)

   Click Add, Edit, or Delete.

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

   ![Blueprint XML](image)

   ```xml
   The root element for any OSGi Blueprint file is 'blueprint' - you also see the
   and the Camel namespaces.
   `<bean class='TransformationBean' id='TransformBean' scope='singleton'>
   `<argument type='string' value='UPPERCASE'/>
   `<property name='CaseStart' value='lower'/>
   </bean>
   ```

   The namespace for the camelContext element in Blueprint is 'https://camel.apache.org', and we can also define namespace prefixes we want to use them in the XPath expressions.
8. Click the Configurations tab to return to the list of global elements and select a global bean to display its standard properties in the Properties view, for example:

![Image of Configurations tab with TransformBean selected]

**NOTE**

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

9. Set global bean properties as needed:

- **Depends-on** is a string that you can use to identify a bean that must be created before this global bean. Specify the ID (name) of the depended on bean. For example, if you are adding the TransformBean and you set Depends-on to ChangeCaseBean then ChangeCaseBean must be created and then TransformBean can be created. When the beans are being destroyed then TransformBean is destroyed first.
- **Factory-method** is useful only when the global bean is a factory class. In this situation, specify or select a static factory method to be called when the bean is referenced.

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

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

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

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

### Deleting a global element

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

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

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

3. Switch to the **Source** tab to check that the tooling removed the XML code from the routing context.
4. When done, save your changes by selecting **File → Save** on the menu bar.

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

**Editing a global element**

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

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

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

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

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

4. In the route editor, click the Source tab and check that the tooling added the property concurrentConsumers=2 to the routing context:
5. When done, save your changes by selecting File → Save on the menu bar.

Related topics

- Section 6.7, “Manually adding beans and configuration”

**NOTE**

Manually adding a global element is possible but not recommended because doing so is error prone. The recommendation is to use the Configurations tab in the route editor.

6.9. CONFIGURING THE ROUTE EDITOR

**Overview**

Using JBoss Developer Studio preferences, you can specify aspects of the route editor’s behavior when configuring Enterprise Integration Patterns:

- 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

**Procedure**

To configure the route editor:

1. Click Windows → Preferences → Fuse Tooling → Editor on Linux and Windows machines, but JBoss Developer Studio → Preferences → Fuse Tooling → Editor on OS X, to open the Editor preferences dialog.

2. Select, from the drop-down list, the default language you want to use for expressions in EIP nodes. The default is Simple.

3. Click the checkbox next to If enabled the ID values will be used for labels if existing to enable or disable using node IDs as labels. The default is enabled.

4. Select, from the drop-down list, the direction in which you want the route editor to align the patterns in a route. The default is Down.
5. Click the checkbox next to Show diagram grid in Routes Editor to enable or disable displaying the grid overlay on the canvas. The default is enabled.

6. Click Apply to apply the changes to the Editor preferences.

7. When done, click OK to close the Preferences dialog. You can restore the route editor’s original defaults at any time by returning to the Editor preferences dialog and clicking Restore Defaults.

Related topics

- Section A.1, “Editor”
CHAPTER 7. THE SOURCE TAB

The tooling's Source tab displays the contents of a routing context .xml file.

Figure 7.1. Source tab

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

PREREQUISITES

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

- If you are replacing an existing JUnit test case, you need to delete it before you create a new one. See the section called “Deleting and existing JUnit test case”.

- If you are creating a new JUnit test case in a project that hasn't one, you need to first create the project_root/src/test/java folder for the test case that is included in the build path. See the section called “Creating and adding the src/test/java folder to the build path”.

DELETING AND EXISTING JUNIT TEST CASE

1. In the Project Explorer view, expand the project's root node to expose the <root_project>/src/test/java folder.

2. Locate the JUnit test case file in the /src/test/java folder. Depending on which DSL the project is based on, the JUnit test case file is named BlueprintXmlTest.java or CamelContextXmlTest.java.

3. Right-click the JUnit test case .java file to open the context menu, and then select Delete. The JUnit test case .java file disappears from the Project Explorer view.

   You can now create a new JUnit test case.

CREATING AND ADDING THE src/test/java FOLDER TO THE BUILD PATH

1. In the Project Explorer view, right-click the project's root to open the context menu.

2. Select New → Folder to open the Create a new folder resource wizard.

3. In the wizard's project tree pane, expand the project's root node and select the src folder. Make sure <project_root>/src appears in the Enter or select the parent folder field.

4. In Folder name, enter /test/java. This folder will store the new JUnit test case you create.
5. Click Finish. In the Project Explorer view, the new src/test/java folder appears under the src/main/resources folder. You can verify that this folder is on the class path by opening its context menu and selecting Build Path. If Remove from Build Path is a menu option, you know the src/test/java folder is on the class path.

You can now create a new JUnit test case.

CREATING A JUNIT TEST CASE

To create a new JUnit test case for your route:

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

2. Right-click it to open the context menu, and then select New → Camel Test Case to open the New Camel JUnit Test Case wizard, as shown in Figure 8.1, “New Camel JUnit Test Case wizard”.

Figure 8.1. New Camel JUnit Test Case wizard

Alternatively, you can open the wizard by selecting File → New → Other → JBoss Fuse → Camel Test Case from the menu bar.
3. In Source folder, accept the default location of the source code for the test case, or enter another location.
   You can click Browse... to search for a location.

4. In Package, accept the default package name for the generated test code, or enter another package name.
   You can click Browse... to search for a package.

5. In Camel XML file under test, accept the default pathname of the routing context file that contains the route you want to test, or enter another pathname.
   You can click Browse... to search for a context file.

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

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

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

9. Click Next > to open the Test Endpoints page. For example, Figure 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

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

11. Click Finish.
NOTE

If prompted, add JUnit to the build path.

The artifacts for the test are added to your project and appear in the Project Explorer view under src/test/java. The class implementing the test case opens in the Java editor.
CHAPTER 9. RUNNING ROUTES INSIDE RED HAT JBOSS FUSE TOOLING

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

- Section 9.1, “Running routes as a local Camel context”
- Section 9.2, “Running routes using Maven”

9.1. RUNNING ROUTES AS A LOCAL CAMEL CONTEXT

Overview

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

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

Procedure

To run a route as a local Camel context:

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

   NOTE

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

Result

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

Related topics

- Section 9.3.1, “Editing a Local Camel Context runtime profile”

9.2. RUNNING ROUTES USING MAVEN

Overview

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

Procedure
To run a route using Maven:

1. In the Project Explorer view, select the root of the project.

2. Right-click it to open the context menu, and then select Run As → Maven build.

   a. The first time you run the project using Maven, the Edit Configuration and launch editor opens, so you can create a Maven runtime profile.

      To create the runtime profile, on the Maven tab:

      i. Make sure the route directory of your Apache Camel project appears in the Base directory field.

         For example, on Linux the root of your project is similar to ~/workspace/simple-router.

      ii. In the Goals: field, enter camel:run.

         IMPORTANT

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

      iii. Click Apply and then Run.

   b. Subsequent Maven runs use this profile, unless you modify it between runs.

Results

The Console view displays the output from the Maven run.

Related topics

- Section 9.3.2, “Editing a Maven runtime profile”

9.3. 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.

9.3.1. Editing a Local Camel Context runtime profile

Overview

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

The runtime configuration editor for a Local Camel Context runtime profile contains the following tabs:
**Camel Context File** — specifies the name of the new configuration and the full path of the routing context file that contains your routes.

**JMX** — specifies JMX connection details, including the JMX URI and the user name and password (optional) to use to access it.

**Main** — specifies the fully qualified name of the project’s base directory, a few options for locating the base directory, any goals required to execute before running the route, and the version of the Maven runtime to use.

**JRE** — specifies the JRE and command line arguments to use when starting the JVM.

**Refresh** — specifies how Maven refreshes the project’s resource files after a run terminates.

**Environment** — specifies any environment variables that need to be set.

**Common** — specifies how the profile is stored and the output displayed.

The first time an Apache Camel route is run as a Local Camel Context, Red Hat 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 the Project Explorer view, select the Camel context file for which you want to edit or create a custom runtime profile.

2. Right-click it to open the context menu, and then select Run As → Run Configurations to open the Run Configurations dialog.

3. In the context selection pane, select Local Camel Context, and then click at the top, left of the context selection pane.

4. In the Name field, enter a new name for your runtime profile.
Setting the camel context file

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

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

Changing the command line options

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

```-fa context-file```

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

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

Changing where output is sent

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

1. Select the **Common** tab.

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

**Related topics**

- Section 9.1, “Running routes as a local Camel context”

### 9.3.2. Editing a Maven runtime profile

**Overview**

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

**IMPORTANT**

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

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

- **Main** — specifies the name of the new configuration, the fully qualified name of the project’s base directory, a few options for locating the base directory, any goals required to execute before running the route, and the version of the Maven runtime to use.

- **JRE** — specifies the JRE and command line arguments to use when starting the JVM.

- **Refresh** — specifies how Maven refreshes the project’s resource files after a run terminates.

- **Source** — specifies the location of any additional sources that the project requires.

- **Environment** — specifies any environment variables that need to be set.

- **Common** — specifies how the profile is stored and the output displayed.

**Accessing the Maven runtime configuration editor**

1. In the **Project Explorer** view, select the root of the project for which you want to edit or create a custom runtime profile.

2. Right-click it to open the context menu, and then select **Run As → Run Configurations** to open the **Run Configurations** dialog.

3. In the **context selection** pane, select **Maven Build**, and then click at the top, left of the context selection pane.
Changing the Maven goal

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

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

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

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

Changing the version of Maven

By default, Red Hat 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 Output File: field, and then enter the path to the file where you want to send the output. The Workspace…, File System…, and Variables… buttons facilitate building the path to the output file.

Related topics

- Section 9.2, “Running routes using Maven”
CHAPTER 10. GETTING STARTED WITH FUSE INTEGRATION SERVICES

OVERVIEW

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

IMPORTANT

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

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

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

IMPORTANT

JBoss Fuse Tooling works only with the s2i binary workflow and only with projects based on the Spring Boot framework.

NOTE

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

Creating and deploying your first Fuse FIS project involves:

- Adding the Container Development Kit (CDK) to Servers view
- Starting up the Container Development Environment (CDE)
- Creating a new OpenShift project
- Creating a new Fuse FIS project
- Deploying the Fuse FIS project to OpenShift
NOTE
You can also run a Fuse FIS project as a local Camel context, see Section 9.1, “Running routes as a local Camel context”, and then connect to it in the JMX Navigator view, where you can monitor and test the routing context. You can also run the Camel debugger on a Fuse FIS project (olink:RiderUG/RiderCamelDebugger) to expose and fix any logic errors in the routing context.

ADDING THE RED HAT CONTAINER DEVELOPMENT KIT SERVER

Add the Red Hat Container Development Kit to the Servers view:

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

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

2. In the Servers view, click the link No servers are available. Click this link to create a new server... to open the Define a New Server wizard. This link appears only when the Servers view contains no server entry.
   Otherwise, right-click in the view to open the context menu, and then select New → Server to open the Define a New Server wizard.
3. Select Red Hat JBoss Middleware → Red Hat Container Development Kit 3. Accept the defaults for:
   - **Server’s host name** – localhost
   - **Server name** – Container Development Environment

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

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

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

7. Set the credentials this way:
   - **Username** – Enter the name you use to log into your Red Hat account.
   - **Always prompt for password** – Leave as is (disabled).
   - **Password** – Enter the password you use to log into your Red Hat account.
8. Click OK to return to the **Red Hat Container Development Environment** page, which is now populated. For example:

![Red Hat Container Development Environment](image)

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

![Servers view](image)

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

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

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

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

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

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

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

CREATING A NEW OPENSHIFT PROJECT

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

1. In the OpenShift Explorer view, right-click the developer entry, to open the context menu.

2. Select New → Project to open the New OpenShift Project wizard.

3. Set the new project’s properties this way:
In the **Project Name** field, enter the name for the project’s namespace on the virtual OpenShift server. Only lower case letters, numbers, and dashes are valid.

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

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

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

   ![OpenShift Explorer](image)

   **NOTE**
   MyProject myproject is an initial example project included with OpenShift.
With New FIS Test newtest selected in the OpenShift Explorer view, the Properties view displays the project's details. For example:

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

NOTE

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

CREATING A NEW FUSE FIS PROJECT

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

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

1. In the Project Explorer view, right-click to open the context menu and then select New → Fuse Integration Project to open the wizard's Choose a project name page:
2. In the **Project Name** field, enter a name that is unique to the workspace you are using, for example, `myFISproject`.
   Accept the defaults for the other options.

3. Click **Next** to open the **Select a Target Runtime** page:
Leave the defaults for **Target Runtime** (*No Runtime selected*) and **Camel Version** (*2.18.1.redhat-000015*).

4. Click **Next** to open the **Advanced Project Setup** page:
5. Click Use a predefined template, and then expand the Fuse on OpenShift option:
6. Select **SpringBoot on OpenShift**, and then click **Finish**.

**NOTE**

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

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

When the build is done the **Fuse Integration** perspective displays the project, for example:
At this point, you can:

- **Deploy the project on OpenShift**
- **Section 9.1, “Running routes as a local Camel context”** to verify that the routing context runs successfully on your local machine
  
  Connecting to the running context in the JMX Navigator view (see the section called “Viewing processes in a local JMX server”), you can monitor route components and test whether the route performs as expected:
  
  - View a route component’s JMX statistics — see Chapter 22, *Viewing a component’s JMX statistics.*
  
  - Edit the running route — see Chapter 26, *Managing routing endpoints.*
  
  - Suspend/resume the running route — see Chapter 28, *Managing routing contexts*
  
  - Start/stop tracing on the running route — see Chapter 24, *Tracing Routes*

- **Run the Camel debugger on the project’s camel-context.xml file to discover and fix logic errors** — see olink:RiderUG/RiderCamelDebugger

**DEPLOYING THE FUSE FIS PROJECT TO OPENSHIFT**

1. In the Project Explorer view, right-click the project’s root (in this example, myFISproject) to open the context menu.

2. Select **Run As → Run Configurations** to open the Run Configurations wizard.

3. In the sidebar menu, select **Maven Build → Deploy <projectname → on OpenShift** (in this example, Deploy myFISproject on OpenShift) to open the project’s default run configuration:
Leave the default settings as they are on the Main tab.

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

```
-Dkubernetes.namespace=newtest
```

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

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

- `-Dkubernetes.trust.certificates=true`
When using the CDK, this argument is required. Leave it set to `true`.

- If you are using an OpenShift instance that has a valid SSL certificate, change the value of this argument to `false`.

6. Click **Apply** and then click **Run**.

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

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

   ![Console output showing successful deployment](image)

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

   ![Console output showing BUILD SUCCESS](image)

7. Switch to the **OpenShift Explorer** view and select **New FIS Test newtest**:
In the **Properties** view, the **Details** page displays all of the project’s property values.

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

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

Scroll through the other tabs to view other properties of the deployment configuration.
9. In the OpenShift Explorer view, select camel-ose-springboot-xml-1-mdmtd Pod Running, and then view the details of the running instance in the Properties view:

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Annotations</td>
<td>img/icons/camel.svg</td>
</tr>
<tr>
<td>fabric8.io/created-by</td>
<td>Kubernetes.io/created-by</td>
</tr>
<tr>
<td>kubernetes.io/created-by</td>
<td>(&quot;kind&quot;:&quot;SerializedReference&quot;,&quot;apiVersion&quot;:&quot;v1&quot;,&quot;reference&quot;:{&quot;kind&quot;:&quot;Rep...</td>
</tr>
<tr>
<td>openshift.io/deployment.yaml</td>
<td>camel-ose-springboot-xml-1</td>
</tr>
<tr>
<td>openshift.io/deployment-config.yaml</td>
<td>camel-ose-springboot-xml</td>
</tr>
<tr>
<td>basic</td>
<td></td>
</tr>
<tr>
<td>Creation Timestamp</td>
<td>2017-10-10T19:57:30Z</td>
</tr>
<tr>
<td>Kind</td>
<td>Pod</td>
</tr>
<tr>
<td>Name</td>
<td>camel-ose-springboot-xml-1-mdmtd</td>
</tr>
<tr>
<td>Namespace</td>
<td>newtest</td>
</tr>
<tr>
<td>Resource Version</td>
<td>3298</td>
</tr>
<tr>
<td>Labels</td>
<td></td>
</tr>
<tr>
<td>deployment</td>
<td>camel-ose-springboot-xml-1</td>
</tr>
<tr>
<td>deploymentconfig</td>
<td>camel-ose-springboot-xml</td>
</tr>
<tr>
<td>group</td>
<td>org.mycompany</td>
</tr>
<tr>
<td>project</td>
<td>camel-ose-springboot-xml</td>
</tr>
<tr>
<td>provider</td>
<td>fabric8</td>
</tr>
<tr>
<td>version</td>
<td>1.0.0-SNAPSHOT-ICT</td>
</tr>
<tr>
<td>Misc</td>
<td></td>
</tr>
<tr>
<td>Host</td>
<td>10.0.2.15</td>
</tr>
<tr>
<td>Image(s)</td>
<td>172.30.1.1:5000/newtest/camel-ose-springboot-xml@sha255:a3f473...</td>
</tr>
<tr>
<td>IP</td>
<td>172.17.0.3</td>
</tr>
<tr>
<td>Status</td>
<td>Running</td>
</tr>
</tbody>
</table>

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

**NOTE**

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

The Console view automatically opens, displaying the logs from the running pod:
Click × in the Console view’s menu bar to terminate the session and clear console output.

ACCESSING THE OPENSHIFT WEB CONSOLE

**NOTE**

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

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

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

- **Default developer role**
  Developer users can view only their own projects and the supplied OpenShift sample project, which demonstrates OpenShift v3 features. Developer users can create, edit and delete any project that they own that is deployed on OpenShift.

  - **Username**: developer
  - **Password**: developer

- **Default administrator role**
  An administrator user can view and access all projects on OpenShift (CDK). Administrator users can create, edit and delete, any project deployed on OpenShift.

  - **Username**: admin
  - **Password**: admin
For more information on using the OpenShift web console, see Getting Started Guide: Container Development Kit.
CHAPTER 11. USING THE JBOSS FUSE SAP TOOL SUITE

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). The SAP Tool Suite depends on the JCo and IDoc client libraries from SAP. To install and use these libraries, you must have an SAP Service Marketplace Account.

11.1. INSTALLING THE JBOSS FUSE SAP TOOL SUITE

Overview

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

Platform restrictions for SAP tooling

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

Prerequisites

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

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

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

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

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

Procedure

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

1. In JBoss Developer Studio, select menu:[File > Import] to open the Import wizard.

2. In the Select screen of the Import wizard, select menu:[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 SAP DESTINATION CONNECTION

**Overview**

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

**Procedure**

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

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

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

3. To create a new Destination Data Store, click the Add Destination tab. The Create Destination dialog appears.
4. Enter a name for the destination in the **Destination Name** field and click **Ok**.

5. In the **Properties** dialog,
   
a. Click the **Basic** tab to configure the basic properties required to connect to an SAP destination. In this tab, fill in the following property fields to configure the connection:
   
   - SAP Application Server
   - SAP System Number
   - SAP Client
   - Logon User
   - Logon Password
   - Logon Language

b. Click the **Connection** tab to add values required to connect to an SAP destination. Fill in the following property fields to configure the connection:
   
   - SAP System Number
   - SAP Router String
   - SAP Application Server
   - SAP Message Server
   - SAP Message Server Port
   - Gateway Host
   - Gateway Port
   - SAP System ID
   - SAP Application Server Group

c. Click the **Authenticate** tab to add values required to validate an SAP destination. Fill in the following property fields to configure the connection.
   
   - SAP Authentication type
   - SAP Client
   - Logon User
   - Logon User Alias
   - Logon Password
   - SAP SSO Logon Ticket
   - SAP X509 Login Ticket
   - Logon Language
d. Click the Special tab. In this tab, fill in the following property fields to configure the connection:

- Select CPIC Trace
- Initial Codepage

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

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

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

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

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

- Repository Destination
- Repository Logon User
- Repository Logon Password

**NOTE**

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

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

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

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

Otherwise, an error report appears in the status area.

8. Click **Close** to close the **Test Destination Connection** dialog.
Click Finish. The newly created SAP Destination Connection appears under SAP.

11.3. CREATE AND TEST SAP SERVER CONNECTION

Overview

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

Procedure

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

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

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

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

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

5. In the Properties dialog,
   a. Click the Mandatory tab to configure the basic properties required to connect to an SAP server. In this tab, fill in the following property fields to configure the connection:
      - Gateway Host
      - Gateway Port
      - Program ID
      - Repository Destination
      - Connection Count
   b. Click the Optional tab and fill in the following property fields to configure the connection:
      - SAP Router String
      - Worker Thread Count
      - Minimum Worker Thread Count
      - Maximum Startup Delay
      - Repository Map
   c. Click the SNC tab and fill in the following property fields to configure the connection.
6. You are now ready to test the server connection. In the Edit SAP Connection Configuration dialog, right-click on the server name and select Test. The Test Server Connection dialog opens.

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

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

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

8. Click Stop to shut down the Test Server.

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

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

### 11.4. DELETING DESTINATION AND SERVER CONNECTIONS

**Overview**

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

**Procedure**

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

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

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

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

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

### 11.5. CREATE A NEW SAP ENDPOINT
Overview

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

**NOTE**

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

Prerequisites

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

**NOTE**

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

Procedure

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

1. It is assumed that you already have a Fuse project and a Camel XML file to work with (which could either be in Blueprint XML or Spring XML format).

2. Open your Camel XML file in the route editor. If you have already installed the JBoss Fuse SAP Tool Suite, you should be able to see the SAP components under the **Components** palette in the route editor. The following SAP components are provided by the tool suite:

   - SAP IDoc Destination
   - SAP IDoc List Destination
   - SAP IDoc List Server
   - SAP qRFC Destination
   - SAP Queued IDoc Destination
   - SAP Queued IDoc List Destination
   - SAP sRFC Destination
   - SAP sRFC Server
   - SAP tRFC Destination
   - SAP tRFC Server

   In the **Design** tab of the route editor, drag one of these components onto the canvas to create a new SAP endpoint in the current **camelContext**.
NOTE
The SAP Netweaver component does not belong to the JBoss Fuse SAP Tool Suite. It is hosted in the Apache Camel project.

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

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

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

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

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

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

Importing the starter quickstart application

1. Right-click in the Project Explorer view to open the context menu.
2. Select menu: [Import > Import].
3. Expand the Maven folder, and select Existing Maven Projects.
4. Click Next to open the Maven Projects wizard.
5. Click **Browse** to find and select the root directory of the **starter** quickstart application. If the browse operation finds multiple projects, make sure you select the **starter** quickstart application. The full path to the **starter** quickstart application appears in the **Projects** pane.

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

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

8. Double-click **starter/src/main/resources/META-INF/spring/camel-context.xml** to open the route in the route editor's **Design** tab.
9. Click the **Source** tab to view the underlying XML.

You can see that an XML file is produced from a source endpoint and a JSON file is consumed by a target endpoint.

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

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

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

11. Remove the arrow that connects the source and target endpoints.
12. If the Console view is not already open, open it now by selecting **Window → Show View → Console**.

**NOTE**

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

### Adding a data transformation node to the Camel route

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

2. Drag a **Data Transformation** pattern over the canvas and drop it on the **Route_route1** container.

   The **New Transformation** wizard opens with the **Project**, **Dozer File Path**, and **Camel File Path** fields auto filled.

   ![New Transformation](image)

   **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**

   ![Types Transformed](image)

3. Fill in the remaining fields:

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

4. Click **Next**.

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

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

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

The Source Type (XML) definition page should now look like this:
8. Click Next to open the **Target Type (JSON)** definition page. This is where you specify the type definition for the target data.

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:
10. **Click Finish.**

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

The transformation editor is composed of three panels:

- **Source** — lists the available data items of the source
- **Mappings** — displays the mappings between the source and target data items
- **Target** — lists the available data items of the target
In addition, the editor's details pane, located just below the editor's three panels (once the first mapping has been made), graphically displays the hierarchical ancestors for both the mapped source and target data items currently selected. For example:

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

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

**Mapping source data items to target data items**

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

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

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

3. Continue dragging and dropping source data items onto their corresponding target data items until you have completed all basic mappings.

In the **starter** example, the remaining data items to map are:

<table>
<thead>
<tr>
<th>Source</th>
<th>Target +</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>orderNum</code></td>
<td><code>orderId</code></td>
</tr>
<tr>
<td><code>status</code></td>
<td><code>priority</code></td>
</tr>
<tr>
<td><code>id</code></td>
<td><code>itemId</code></td>
</tr>
<tr>
<td><code>price</code></td>
<td><code>cost</code></td>
</tr>
<tr>
<td><code>quantity</code></td>
<td><code>amount</code></td>
</tr>
</tbody>
</table>

**NOTE**

You can map collections (data items containing lists or sets) to noncollection data items and vice versa, but you cannot map collections to other collections.
4. Click on both the **Source** and **Target** panels to quickly determine whether all data items have been mapped.

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

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

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

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

7. Selecting the `file:src/data?file=` 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/messages?file=` node.
Connecting the nodes on the canvas creates a valid Camel route, which you can now save.

8. Click File → Save.

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

```xml
<?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

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

for the target JSON data.

Creating the transformation test file and running the JUnit test
1. Right-click the starter project in the Project Explorer view, and select menu: [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:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package</td>
<td>example</td>
</tr>
<tr>
<td>Transformation ID</td>
<td>json2xml</td>
</tr>
</tbody>
</table>

4. Click Finish.

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

6. Add the following code to the transform method:

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

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

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

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

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

**Mapping a constant variable to a data item**

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

In the starter example, the target data item origin does not have a corresponding source data item. To map the origin attribute to a constant variable:
1. In the Source panel, click the Variables view.

2. In the Variables view, click $0 to open the Enter a new variable name dialog.

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

4. Click OK.

   The newly created variable ORIGIN appears in the Variables view in the Name column and the default value ORIGIN in the Value column.

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

6. Press Enter.

7. Drag and drop the new variable ORIGIN onto the origin data item in the Target panel.
The new mapping of the variable $\text{(ORIGIN)}$ appears in the Mappings panel and in the details pane.

8. Run a JUnit test on your TransformationTest.java file. For details, see the section called “Creating the transformation test file and running the JUnit test”.

The Console view displays the JSON-formatted output data:

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

Mapping an expression to a data item

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

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

1. Click ➔ to add an empty transformation map to the Mappings panel.
2. From the Target panel, drag and drop the `approvalCode` data item to the target field of the newly created mapping in the Mappings panel.
The approvalCode data item also appears in the details pane's target box.

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

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

4. Select Set expression to open the Expression dialog.
5. In **Language**, select the expression language to use from the list of those available. Available options depend on the data item’s data type. For the **starter** example, select **Header**.

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

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

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

8. Run a JUnit test on your `TransformationTest.java` file. For details, see the section called “Creating the transformation test file and running the JUnit test”.

The Console view displays the JSON-formatted output data:

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

Adding a custom transformation to a mapped data item

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

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

1. In the Mappings panel, select the `customerNum` mapping to populate the details pane.
2. In the details pane, click on the ABCOrder source box to open the drop-down menu.

3. Select Add custom transformation to open the Add Custom Transformation page.
4. Click next to the Class field to open the Create a New Java Class wizard.
5. Modify the following fields:
   - **Package**—Enter example.
   - **Name**—Enter MyCustomMapper.
   - **Method Name**—Change map to brackets.
     Leave all other fields as is.

6. Click Finish.
   The Add Custom Transformation page opens with the **Class** and **Method** fields auto filled:
7. Click OK to open the MyCustomMapper.java file in the Java editor:

```
package example;

public class MyCustomMapper {
    public String brackets(String input) {
        return null;
    }
}
```

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

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

9. Click the `transformation.xml` tab to switch back to the transformation editor.
The details pane shows that the brackets method has been associated with the customerNum data item.

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

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

```json
{"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 itemId 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 itemId in the Target.

1. In the Mappings panel, select the mapping id –> itemId to display the mapping in the details pane.
2. On the Source box, click ▼ to open the drop-down menu, and select Set property.

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

4. The details pane now shows that XyzOrder has a lineItems field. Click the toggle button.
next to **lineItems** to increase its value to 1.

NOTE

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

![Diagram](image)

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

5. Run a JUnit test on your **TransformationTest.java** file. For details, see the section called “Creating the transformation test file and running the JUnit test”.

The **Console** view displays the JSON-formatted output data:

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

### Adding a built-in function to a mapped data item

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

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

![Diagram](image)

2. In the Source box, click ![Dropdown](image) to open the drop-down menu, and select **Add transformation**.
3. In the Transformations pane, select append, and in the Arguments pane, enter `-level` for the value of suffix.

This append function adds the specified suffix to the end of the `status` string before mapping it to the target `priority` data item.
4. Click OK.
By default, the details pane displays the results of adding the `append` function to the `status` data item in a user-friendly format. You can change this formatting by clicking the right-most ❯ on the Source box, and selecting `Show standard formatting`.

5. Run a JUnit test on your `TransformationTest.java` file. For details, see the section called “Creating the transformation test file and running the JUnit test”.

The Console view displays the JSON-formatted output data:

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

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

Before you publish your data transformation project to a JBoss Fuse server (see Chapter 30, *Publishing Fuse Integration Projects to a Server*), you need to install the following features in the JBoss Fuse runtime:

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

To install the required features on the JBoss Fuse runtime:

1. If not already there, switch to the **Fuse Integration** perspective.

2. If necessary, add the JBoss Fuse server to the **Servers** list (see Section 29.1, “Adding a Server”).

3. Start the JBoss Fuse Server (see Section 29.2, “Starting a Server”), and wait for the JBoss Fuse shell to appear in the **Terminal** view:
4. For each of the required camel-features, at the JBossFuse:admin@root> prompt type: features:install camel-<featureName>

Where featureName is one of dozer, jackson, or jaxb.

5. To verify that each of the features was successfully installed, at the JBossFuse:admin@root> prompt type: features:list --ordered --installed

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

```
[installed] [2.17.0.redhat-630159] camel-dozer
[installed] [2.17.0.redhat-630159] camel-exec
[installed] [2.17.0.redhat-630159] camel-flip
[installed] [2.17.0.redhat-630159] camel-jackson
[installed] [2.17.0.redhat-630159] camel-jospyt
[installed] [2.17.0.redhat-630159] camel-jaxb
```
CHAPTER 13. DEVELOPING EXTENSIONS FOR IGNITE INTEGRATIONS

Ignite is a Red Hat JBoss Fuse feature that provides a web interface for integrating applications. Without writing code, a business expert can use Ignite to connect to applications and optionally operate on data between connections to different applications. In Ignite, a data operation is referred to as a step in an integration.

Ignite provides steps for operations such as filtering and mapping data. To operate on data in ways that are not provided by Ignite built-in steps, you can develop an Ignite extension to define one or more custom steps.

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

1. In Red Hat JBoss Developer Studio, create an Ignite extension project.
2. Add routes, define actions, and specify any dependencies.
4. Provide the .jar file to the Ignite user.

The Ignite user uploads the .jar file to Ignite and can then add the custom step(s) defined in the extension to an integration. For information about Ignite and how to build integrations, see Integrating Applications with Ignite.

PREREQUISITES

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

- A description of the required functionality for the Ignite custom step (from the Ignite user).
- The version of each of the following components for the extension:
  - Spring Boot
  - Camel
  - Ignite
- You should be familiar with:
  - Spring Boot XML or Java
  - Apache Camel routes
  - Ignite

CREATING AN IGNITE EXTENSION PROJECT

To use the Fuse tooling to create an extension project, follow these steps.

1. In Red Hat JBoss Developer Studio, select File → New → Project → JBoss Fuse → Fuse Ignite Extension Project.
   The New Fuse Ignite Extension Project wizard opens.
2. Enter the name for the project, then click Next.

3. Specify the versions of the extension dependencies:
   - Spring Boot Version
   - Camel Version
   - Fuse Ignite Version

4. (Optional) Click Verify to confirm that the specified Camel version is available to the project.

5. Click Next.

6. Specify the following extension details:
   - Extension id - A value that you define and that is unique in the Ignite environment. This value will be visible in Ignite when the user imports the extension .jar file.
   - Version - The version of the extension. For example, if this is the initial version, you could enter 1.0. If you are updating a version, you could enter 1.1 or 2.0.
   - Name - The name of the extension. This value will be visible in Ignite as the extension’s name. In Ignite, on the Extensions tab, the user can see a list of the names and descriptions of extensions that have been uploaded to Ignite.
   - Description - An optional description of the custom steps that the extension defines.
   - Tags - An optional comma-separated list of values. Note that these tags are ignored in this release. They are reserved for future use.

7. Click Finish.

The new project appears in the Red Hat JBoss Developer Studio Project Explorer view. It includes the following files:

- A descriptor file: META-INF/syndesis/syndesis-extension-definition.json
  This is the file that you edit to add actions and their properties. An action in the .json file becomes a custom step in Ignite. A property becomes an action parameter in Ignite.

- A Camel context file: extension.xml
  This file contains a sample route with a log component. You customize the Camel routes in this file.

- A Maven Project Object Model file: pom.xml
  This file contains information about the project and configuration details used by Maven to build the project, including default extension dependencies. You edit this file to add custom dependencies or to edit the Extension Id, Name, Version, Tags, or Description values.

DEVELOPING THE STEPS FOR THE IGNITE EXTENSION

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

1. In the extension.xml file, create routes that address the purpose of the extension. The entrypoint of each route must match the entrypoint that you define in the syndesis-extension-definition.json file, as described in Step 2.
2. In the **syndesis-extension-definition.json** file, write the code that defines the actions and their properties. You need a new action for each entrypoint. Each property corresponds to an action parameter. In Ignite, when the user selects a custom step, Ignite prompts for values for action properties. For example, a custom log step might have a level property that indicates how much information to send to the log.

Each action that you create corresponds to a custom step in Ignite. You can use different types of code for each action. That is, you can use XML for one action and Spring Boot routes and beans for another action.

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

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

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

3. To edit the extension dependencies, open the `pom.xml` file in the editor. If you add a dependency, you must define its scope. The scope for any dependency that Red Hat ships is provided, for example:

```xml
<dependency>
    <groupId>io.syndesis.integration-runtime</groupId>
    <artifactId>runtime-api</artifactId>
    <version>${syndesis.version}</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.apache.camel</groupId>
```
To build the .jar file for the extension:

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

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

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

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

5. In the Project Explorer view, open the target folder to see the generated .jar file:

   For example: ignite-extension-1.0.0-SNAPSHOT

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

```
{
  "name" : "Syndesis Extension (XML)",
  "description" : "A simple Syndesis Extension (XML)",
  "extensionId" : "com.mycompany:syndesis-extension-example-xml",
  "version" : "1.0.0-SNAPSHOT",
  "tags" : [ "experimental", "feature" ],
  "actions" : [ {
    "id" : "log-body",
    "name" : "simple-log-body",
    "description" : "A simple xml based logging extension",
    "descriptor" : {
      "kind" : "ENDPOINT",
      "entrypoint" : "direct:log-body",
      "inputDataShape" : {
        "kind" : "any"
      },
      "outputDataShape" : {
        "kind" : "any"
      },
      "propertyDefinitionSteps" : [ ]
    },
    "tags" : [ "xml" ],
    "actionType" : "extension"
  }
```
NEXT STEPS

Provide the following to the Ignite user:

- The .jar file

- A document that describes the extension, including information about data shapes that each action in the extension requires as input or provides as output (for data mapping)

In Ignite, the user uploads the .jar file as described in Integrating Applications with Ignite.
CHAPTER 14. CHANGING THE CAMEL VERSION

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

To change the Camel version that a project uses:

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

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

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

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

You can check the project’s Camel version in its pom.xml file, in the <camel.version> element.
PART III. DEBUGGING ROUTING CONTEXTS

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

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

**NOTE**

Before you can run the Camel debugger, you must set breakpoints on the nodes of interest displayed on the route editor's canvas. Then you can run the Camel debugger on a project's routing context `.xml` file to find the logic errors in it and fix them. Invoking the Camel debugger runs the routing context in debug mode and opens the **Debug** perspective ([olink:RiderUG/DebugPerspective]).
CHAPTER 15. SETTING BREAKPOINTS

OVERVIEW

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

The Camel debugger supports two types of breakpoints:

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

NOTE

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

SETTING UNCONDITIONAL BREAKPOINTS

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

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

2. Click its icon to set an unconditional breakpoint.

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

SETTING CONDITIONAL BREAKPOINTS

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

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

2. Click its icon to set a conditional breakpoint and to open the Edit the condition and language of your breakpoint... dialog:
3. Click the Language drop-down menu and select the expression language to use to create the condition that will trigger the breakpoint. Fuse Tooling supports twenty-four expression languages. Some of these languages provide variables for creating conditional expressions, while others do not.

4. Click Variables to display a list of the selected language’s supported variables. If a list appears, select in sequence one or more of the variables to create the condition for triggering the breakpoint. The variables you select appear in the Condition text box. If <nothing available> appears, enter the expression directly into the Condition text box.

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

DISABLING BREAKPOINTS

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

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

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

You can also disable and re-enable breakpoints during debugging sessions. For details, see Chapter 20, Disabling Breakpoints in a Running Context.

DELETING BREAKPOINTS

You can delete individual breakpoints or all breakpoints.

- To delete individual breakpoints – in a route container, select the node whose breakpoint you want to delete, and click its icon.

- To delete all breakpoints in a particular route – right-click the target route’s container, and select Delete all breakpoints.

- To delete all breakpoints in all routes – right-click the canvas, and select Delete all breakpoints.

RELATED TOPICS

- Chapter 16, Running the Camel Debugger
CHAPTER 16. RUNNING THE CAMEL DEBUGGER

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

- **Local debugging** — runs the debugger on the routing context running in the same JVM with Fuse Tooling. This mode is activated by selecting the project's routing context in the Project Explorer view and selecting **Debug As → Local Camel Context** from the context menu.

- **Remote debugging** — runs the debugger on a routing context running in a separate JVM either on the local machine or on a remote machine. This mode requires a supported runtime server installed on the local machine or on an accessible remote machine. It is activated by creating and running a debug launch configuration that specifies the remote runtime's connection details.

**NOTE**

If your project contains Java code, you can use the standard Eclipse Java debugging tools to debug it. For local debugging, it is automatically available. For remote debugging, you need to use the **Remote Camel Context and Java** launch option (see Section 16.2, “Debugging a remotely running routing context”).

**NOTE**

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

16.1. DEBUGGING A LOCALLY RUNNING ROUTING CONTEXT

**Procedure**

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

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

   - Fuse Tooling builds the Camel route, starts Apache Camel, starts the routing context, enables JMX, starts the route(s) in the routing context, adds the breakpoints to the nodes, and enables the Camel debugger.

   The Camel debugger suspends execution of the routing context at the first breakpoint hit (received a message), and prompts you to indicate whether you want it to open the **Debug perspective**.
3. Click Yes to open the Debug perspective. The Debug perspective opens with the routing context suspended at the first breakpoint encountered in the running routing context.

**IMPORTANT**

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

**NOTE**

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

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

Watching message exchanges progress through the routing context

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

16.2. DEBUGGING A REMOTELY RUNNING ROUTING CONTEXT

Prerequisites

- Access to a JBoss Fuse runtime in one of the following ways:
  - Install a JBoss Fuse server on your local machine and edit its FUSE_HOME/etc/users.properties file to activate the admin user. For details, see olink:RiderUG/FIDIEAddServer.
  - Be able to access a JBoss Fuse server that is installed on a remote machine. You must know its connection details, including the credentials of the admin user.

- Create a new Fuse Integration project, see olink:RiderUG/nfpProjBasics. In the Fuse Integration project:
  - Select the Camel version that matches the version of the JBoss Fuse server runtime.
  - Create an empty Blueprint DSL project or use one of the JBoss Fuse pre-defined templates
with Blueprint DSL.

**NOTE**

Though not a requirement, it is a good idea to verify that you can run the project's routing context successfully as a Local Camel Context. For details, see olink:RiderUG/RiderRunningProfile.

**NOTE**

The examples in the section called “Setting up and starting remote debugging” are based on the JBoss Fuse → Content Based Router predefined template and a Red Hat JBoss Fuse 6.3.0 runtime.

- In Red Hat JBoss Developer Studio:
  - Add the JBoss Fuse server to the Servers view. For details, see olink:RiderUG/FIDEAddServer.
  - Start the JBoss Fuse server. For details, see olink:RiderUG/FIDEStartServer.
  - Publish your project to the JBoss Fuse Runtime server. For details, see olink:RiderUG/autoPublish.

**NOTE**

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

- Verify that the project's bundle is deployed and active. For details, see olink:RiderUG/verifyPublished.
- Stop the JBoss Fuse Runtime server. For details, see olink:RiderUG/FIDEStopServer.

### Setting up and starting remote debugging

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

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

To set up and start remote debugging:

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

   $ [FUSE_HOME]/bin/fuse debug

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

3. In the tooling’s route editor, with the project’s routing context open in the Design tab, set breakpoints on the nodes of interest. For details, see olink:RiderUG/setBreakpoints.

4. In the Project Explorer view, right-click the project’s root and select menu:[Debug As > Debug Configurations] to open the Debug Configurations wizard:

   ![Debug Configurations wizard](image)

   5. In the configuration type pane, select either Remote Camel Context or Remote Camel Context and Java, and then click 🍃:
For both the Remote Camel Context and Remote Camel Context and Java options, you need to specify configuration details on the Camel and JMX tabs.

For the Remote Camel Context and Java option only, you also need to specify configuration details on the Connect tab.

**NOTE**

Unless your project contains Java code that you want to debug using the standard Eclipse Java debugging tools, select the Remote Camel Context option.

6. In the Name field, enter a name for the new launch configuration.

7. On the Camel tab, click the Browse button to locate the project’s routing context .xml file in the Open Resources dialog:
NOTE

When you select a file in the Matching items pane, the tooling displays the file’s location, relative to the project root, at the bottom of the pane.

8. In the Matching items pane, select your project’s routing context file from the list, and then click OK.
The tooling inserts the file’s path into the Select Camel Context file field:

9. Click the JMX tab:

Edit the JMX connection details as follows:

- JMX Uri – change :9011/jmxrmi to :1099/karaf-root
If the JBoss Fuse server is running on a remote host, replace `localhost` with the DNS name or IP address of the remote host.

- **JMX User** — enter `admin`.
- **JMX Password** — enter `admin`.

**IMPORTANT**

The values shown for **JMX User** and **JMX Password** are the JBoss Fuse admin user defaults, stored in the `FUSE_HOME/etc/users.properties` file. If your setup is different, enter the values specific to it.

If you are creating a Remote Camel Context debug launch configuration, you are done. Skip to [debugCfgGo].

10. Click the **Connect** tab:

Change the **Port** value from 8000 to 5005. Leave each of the other properties as is.

11. Click **Apply** and then click **Debug**.

12. In the **JMX Navigator** view, double-click **JBoss Fuse [xxx] [Disconnected]** to connect to it, and then expand its tree:
13. In the Project Explorer view, drag a test message from `src/test/resources/data` and drop it on the `cbr-example-context/Endpoints/file/work/cbr/input` folder in the JMX Navigator view. When the message hits the first breakpoint that is set in the routing context, the tooling asks you to switch to the Debug perspective:

14. Click Yes.
At this point, you can use any of the Camel debugger’s tools to debug your routing context.

**NOTE**

In remote debugging sessions, the Console view does not display log output.

**NOTE**

When one message reaches the end of the routing context, the debugger is suspended. To continue debugging, switch back to the Fuse Integration perspective and drop another message on the input node in the JMX Navigator view. Each time you do so, the tooling asks you to confirm the switch to the Debug perspective.

**RELATED TOPICS**

- Chapter 17, Stopping the Camel Debugger
CHAPTER 17. STOPPING THE CAMEL DEBUGGER

OVERVIEW

The way to stop the Camel debugger depends on the mode in which it is running:

- **Local debugging** ([link](RiderUG/localDebug))

  To stop the Camel debugger, click on the menu bar once if the debugging session has ended. Otherwise, click twice: once to terminate the currently running node thread and once to terminate the Camel Context thread (both displayed in the Debug view).

  **NOTE**
  Terminating the Camel debugger also terminates the console but does not clear its output. To clear the output, click (Clear Console) on the Console view’s menu bar.

- **Remote debugging** ([link](RiderUG/remoteDebug))

  To stop the Camel debugger, select the [Remote Camel Context] thread or the [Remote Camel Context and Java] thread in the Debug view, and then click on the menu bar.

  **NOTE**
  Terminating the Camel debugger does not close the connection to the remote runtime server in the Servers view nor in the JMX Navigator view.

CLOSING THE CAMEL DEBUGGER

After you have finished debugging your project, you may want to close the Debug perspective to make more space for your workbench.

To do so, right-click on the right side of the JBoss Developer Studio toolbar, and then select Close.

RELATED TOPICS

- Chapter 16, Running the Camel Debugger
CHAPTER 18. CHANGING VARIABLE VALUES

OVERVIEW

When the Camel debugger hits a breakpoint, the Variables view displays the values of all variables available at that point in the routing context. Some variables are editable, allowing you to change their value. This enables you to see how the application handles changes in program state.

NOTE

Not all variables are editable. The context menu of editable variables displays the Change Value… option.

PROCEDURE

To change the value of a variable:

1. If necessary, start the debugger. See Chapter 16, Running the Camel Debugger.

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

The variable’s value field turns a lighter shade of blue, indicating that it is in edit mode.

NOTE

Alternatively, you can right-click the variable to open its context menu, and select Change Value… to edit its value.

3. Enter the new value and then click Enter.

The Console view displays an INFO level log entry noting the change in the variable’s value.
(for example, Breakpoint at node to1 is updating message header on exchangeId: ID-dhcp-97-16-bos-redhat-com-52574-1417298894070-0-2 with header: Destination and value: UNITED KINGDOM).

4. Continue stepping through the breakpoints and check whether the message is processed as expected. At each step, check the Debug view, the Variables view, and the Console output.

RELATED TOPICS

- Chapter 20, Disabling Breakpoints in a Running Context
- Chapter 19, Adding Variables to the Watch List
CHAPTER 19. ADDING VARIABLES TO THE WATCH LIST

OVERVIEW

By adding variables to the watch list, you can focus on particular variables to see whether their values change as expected as they flow through the routing context.

PROCEDURE

To add a variable to the watch list:

1. If necessary, start the debugger. See Chapter 16, Running the Camel Debugger.

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

3. Select Watch.

   A new view, Expressions, opens next to the Breakpoints view. The Expressions view displays the name of the variable being watched and its current value, for example:
4. Repeat [watch1] and [watch2] to add additional variables to the watch list.

**NOTE**

The variables you add remain in the watch list until you remove them. To stop watching a variable, right-click it in the list to open the context menu, and then click Remove.

5. With the Expressions view open, step through the routing context to track how the value of each variable in the watch list changes as it reaches each step in the route.

**RELATED TOPICS**

- Chapter 18, *Changing Variable Values*
CHAPTER 20. DISABLING BREAKPOINTS IN A RUNNING CONTEXT

OVERVIEW

You can disable and re-enable breakpoints in a running routing context in the Breakpoints view.

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

DISABLING AND ENABLING BREAKPOINTS IN BREAKPOINTS VIEW

The Breakpoints view opens with all set breakpoints enabled.

To disable a breakpoint, clear its check box.
For each breakpoint you disable, the Console view displays an **INFO** level log entry noting that it has been disabled (for example, *Removing breakpoint log2*). Likewise, for each breakpoint you re-enable, the Console view displays an **INFO** level log entry noting that it has been enabled (for example, *Adding breakpoint log2*).

**NOTE**

To re-enable a disabled breakpoint, click its check box. The Console view displays an **INFO** level log entry noting that the breakpoint has been added to the selected node.
PART IV. MONITORING AND TESTING APPLICATIONS

The *JMX Navigator^ view provides numerous ways to monitor and test your Fuse applications.
CHAPTER 21. JMX NAVIGATOR

The JMX Navigator view, shown in Figure 21.1, “JMX Navigator view”, displays all processes that are running in your application and it drives all interactions with the monitoring and testing features. Other areas of the Fuse Integration perspective adapt to display information related to the node selected in the JMX Navigator view. In the JMX Navigator view, its context menu provides the commands needed to activate route tracing and to add JMS destinations.

Figure 21.1. JMX Navigator view

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

- Local Processes
- Server Connections
21.1. VIEWING PROCESSES IN JMX

Overview
The JMX Navigator view lists all known processes in a series of trees. The root for each tree is a JMX server.

The first tree in the list is a special Local Processes tree that contains all JMX servers that are running on the local machine. You must connect to one of the JMX servers to see the processes it contains.

Viewing processes in a local JMX server
To view information about processes in a local JMX server:

1. In the JMX Navigator view, expand Local Processes.
2. Under Local Processes, double-click one of the top-level entries to connect to it.
3. Click the icon that appears next to the entry to display a list of its components that are running in the JVM.

Viewing processes in alternate JMX servers
To view information about processes in an alternate JMX server:

1. Section 21.2, “Adding a JMX server” the JMX server to the JMX Navigator view.
2. In the JMX Navigator view, expand the server’s entry by using the icon that appears next to the entry. This displays a list of that JMX server’s components that are running in the JVM.

21.2. ADDING A JMX SERVER

Overview
In the JMX Navigator view, under the Local Processes branch of the tree, you can see a list of all local JMX servers. You may need to connect to specific JMX servers to see components deployed on other machines.

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

Procedure
To add a JMX server to the JMX Navigator view:

1. In the JMX Navigator view, click New Connection.
2. In the Create a new JMX connection wizard, select Default JMX Connection.

3. Click Next. 

4. Select the Advanced tab.

5. In the Name field, enter a name for the JMX server.
   The name can be any string. It is used to label the entry in the JMX Navigator tree.

6. In the JMX URL field, enter the JMX URL of the server.

7. If the JMX server requires authentication, enter your user name and password in the Username and Password fields.

8. Click Finish.
   The new JMX server appears as a branch in the User-Defined Connections tree.
CHAPTER 22. VIEWING A COMPONENT’S JMX STATISTICS

OVERVIEW

The tooling collects all JMX statistics reported by Fuse components and displays them in the Properties view. This statistical information can provide significant insight into what is happening in your integration application.

JMX statistics are grouped into three categories: Properties, Processor, and Profile.

PROCEDURE

To see a Fuse component’s statistics:

1. In the JMX Navigator view, locate the node for the component. You may have to expand nodes on the tree to locate low-level components.

2. Select the node of the Fuse component whose statistics you want to review.

3. Open the Properties view.

4. The Properties page displays the JMX properties for the selected component:

5. Click Processors to check exchange metrics for the selected component:

6. Click Profile to check message metrics for the selected node and its subnodes:
# Viewing a Component's JMX Statistics

## Profile Table

<table>
<thead>
<tr>
<th>ID</th>
<th>Total Time (m Self Time s Count)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-e: When /order/orderLine/quantity/text() &gt; /order/orderLine/maxAllowed/text()</td>
<td>4 0 2</td>
</tr>
<tr>
<td>-@ Log log1</td>
<td>4 0 2</td>
</tr>
<tr>
<td>-# SetHeader setHead1</td>
<td>4 0 2</td>
</tr>
<tr>
<td>-# file:target/messages/invalidOrders</td>
<td>4 4 2</td>
</tr>
<tr>
<td>-e: Otherwise</td>
<td>45 4 4</td>
</tr>
<tr>
<td>-@ Log log2</td>
<td>45 0 4</td>
</tr>
<tr>
<td>-# SetHeader setHead2</td>
<td>45 0 4</td>
</tr>
<tr>
<td>-# direct:OrderFulfillment</td>
<td>45 45 4</td>
</tr>
</tbody>
</table>
CHAPTER 23. BROWSING MESSAGES

OVERVIEW

A key tool in debugging applications in a distributed environment is seeing all of the messages stored in the JMS destinations and route endpoints in the application. The tooling can browse the following:

- JMS destinations
- JMS routing endpoints
- Apache Camel routing endpoints
- SEDA routing endpoints
- Browse routing endpoints
- Mock routing endpoints
- VM routing endpoints
- DataSet routing endpoints

PROCEDURE

To browse messages:

1. In the JMX Navigator view, select the JMS destination or endpoint you want to browse. The list of messages appears in the Messages View.

2. In the Messages View, select an individual message to inspect.

Message details and content are displayed in the Properties view:
RELATED TOPICS

- Section 24.3, “Tracing messages through a routing context”
CHAPTER 24. TRACING ROUTES

Debugging a route often involves solving one of two problems:

- A message was improperly transformed.
- A message failed to reach its destination endpoint.

Tracing one or more test messages through the route is the easiest way to discover the source of such problems.

The tooling’s route tracing feature enables you to monitor the path a message takes through a route and see how the message is transformed as it passes from processor to processor.

The Diagram View displays a graphical representation of the route, which enables you to see the path a message takes through it. For each processor in a route, it also displays the average processing time, in milliseconds, for all messages processed since route start-up and the number of messages processed since route start-up.

The Messages View displays the messages processed by a JMS destination or route endpoint selected in the JMX Navigator tree. Selecting an individual message trace in the Messages View displays the full details and content of the message in the Properties view and highlights the corresponding node in the Diagram View.

Tracing messages through a route involves the following steps:

1. Section 24.1, “Creating test messages for route tracing”
2. Section 24.2, “Activating route tracing”
3. Section 24.3, “Tracing messages through a routing context”
4. Section 24.4, “Deactivating route tracing”

24.1. CREATING TEST MESSAGES FOR ROUTE TRACING

Overview

Route tracing works with any kind of message structure. The Fuse Message wizard creates an empty .xml message, leaving the structuring of the message entirely up to you.

NOTE

If the folder where you want to store the test messages does not exist, you need to create it before you create the messages.

Creating a new folder to store test messages

To create a new folder:

1. In the Project Explorer view, right-click the project root to open the context menu.
2. Select New → Folder to open the New Folder wizard.
   The project root appears in the Enter or select the parent folder field.
3. Expand the nodes in the graphical representation of the project's hierarchy, and select the node you want to be the parent folder.

4. In the **Folder name** field, enter a name for the new folder.

5. Click **Finish**.
   The new folder appears in the **Project Explorer** view, under the selected parent folder.

   **NOTE**
   If the new folder does not appear, right-click the parent folder and select **Refresh**.

**Creating a test message**

To create a test message:

1. In the **Project Explorer** view, right-click the project to open the context menu.

2. Select **New** → **Fuse Message** to open the **New File** wizard.

3. Expand the nodes in the graphical representation of the project’s hierarchy, and select the folder in which you want to store the new test message.

4. In the **File name** field, enter a name for the message, or accept the default (*message.xml*).

5. Click **Finish**.
   The new message opens in the XML editor.

6. Enter the message contents, both body and header text.

   **NOTE**
   You may see the warning, **No grammar constraints (DTD or XML Schema) referenced in the document**, depending on the header text you entered. You can safely ignore this warning.

**Related topics**

- *Section 24.3, “Tracing messages through a routing context”*

**24.2. ACTIVATING ROUTE TRACING**

**Overview**

You must activate route tracing for the routing context before you can trace messages through that routing context.

**Procedure**

To activate tracing on a routing context:
1. In the JMX Navigator view, select the running routing context on which you want to start tracing.

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

2. Right-click the selected routing context to open the context menu, and then select Start Tracing to start the trace.
   If Stop Tracing Context is enabled on the context menu, then tracing is already active.

Related topics
- Section 24.3, “Tracing messages through a routing context”
- Section 24.4, “Deactivating route tracing”

24.3. TRACING MESSAGES THROUGH A ROUTING CONTEXT

Overview
The best way to see what is happening in a routing context is to watch what happens to a message at each stop along the way. The tooling provides a mechanism for dropping messages into a running routing context and tracing the path the messages take through it.

Procedure
To trace messages through a routing context:

1. Create one or more test messages as described in Section 24.1, “Creating test messages for route tracing”.

2. In the Project Explorer view, right-click the project’s Camel context file to open the context menu, and select Run As → Local Camel Context (without Tests).

   NOTE
   Do not run it as Local Camel Context unless you have created a comprehensive JUnit test for the project.

3. Activate tracing for the running routing context as described in Section 24.2, “Activating route tracing”.

4. Drag one of the test messages from the Project Explorer view onto the routing context’s starting point in the JMX Navigator view.

5. In the JMX Navigator view, select the routing context being traced.
   The tooling populates the Messages View with message instances that represent the message at each stage in the traced context.

   The Diagram View displays a graphical representation of the selected routing context.

6. In the Messages View, select one of the message instances.
The Properties view displays the details and content of the message instance.

In the Diagram View, the route step corresponding to the selected message instance is highlighted. If the route step is a processing step, the tooling tags the exiting path with timing and processing metrics.

7. Repeat this procedure as needed.

Related topics

- Section 24.1, “Creating test messages for route tracing”
- Section 24.2, “Activating route tracing”
- Section 24.4, “Deactivating route tracing”

24.4. DEACTIVATING ROUTE TRACING

Overview

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

IMPORTANT

Deactivating tracing stops tracing and flushes the trace data for all of the routes in the routing context. This means that you cannot review any past tracing sessions.

Procedure

To stop tracing for a routing context:

1. In the JMX Navigator view, select the running routing context for which you want to deactivate tracing.

   NOTE

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

2. Right-click the selected routing context to open the context menu, and then select Stop Tracing Context. If Start Tracing appears on the context menu, tracing is not activated for the routing context.

Related topics

- Section 24.2, “Activating route tracing”
- Section 24.3, “Tracing messages through a routing context”
CHAPTER 25. MANAGING JMS DESTINATIONS

The JMX Navigator view lets you add or delete JMS destinations in a running instance of Red Hat JBoss Fuse.

IMPORTANT

These changes are not persistent across broker restarts.

25.1. ADDING A JMS DESTINATION

Overview

When testing a new scenario, it is convenient to add a new JMS destination to one of your brokers.

Procedure

To add a JMS destination to a broker:

1. In the JMX Navigator view, under the broker node for which you want to add a destination, select either the Queues child or the Topics child.

2. Right-click the selected node to open the context menu, and then select either Create Queue or Create Topic.

3. In either the Create Queue or Create Topic dialog, enter a name for the new destination.

4. Click OK.

5. Right-click either the Queues or the Topics child, and then select Refresh. The new destination appears in the JMX Navigator view under the Queues child or the Topics child.

Related topics

- Section 25.2, “Deleting a JMS destination”

25.2. DELETING A JMS DESTINATION

Overview

When testing failover scenarios or other scenarios that involve handling failures, it is helpful to be able to easily remove a JMS destination.

Procedure

To delete a JMS destination:

1. In the JMX Navigator view, under the Queues child or the Topics child, select the JMS destination you want to delete.
2. Right-click the selected destination to open the context menu, and then select **Delete Queue/Topic**.

**Related topics**

- Section 25.1, “Adding a JMS destination”
CHAPTER 26. MANAGING ROUTING ENDPOINTS

The JMX Navigator view lets you add or delete routing endpoints.

**IMPORTANT**

These changes are not persistent across routing context restarts.

26.1. ADDING A ROUTING ENDPOINT

**Overview**

When testing a new scenario, you might want to add a new endpoint to a routing context.

**Procedure**

To add an endpoint to a routing context:

1. In the JMX Navigator view, under the routing context node, select the Endpoints child to which you want to add an endpoint.

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

3. In the Create Endpoint dialog, enter a URL that defines the new endpoint, for example, `file://target/messages/validOrders`.

4. Click OK.

5. Right-click the routing context node, and select Refresh. The new destination appears in the JMX Navigator view under the Endpoints node, in a folder that corresponds to the type of endpoint it is, for example, file.

**Related topics**

- Section 26.2, “Deleting a routing endpoint”

26.2. DELETING A ROUTING ENDPOINT

**Overview**

When testing failover scenarios or other scenarios that involve handling failures, it is helpful to be able to remove an endpoint from a routing context.

**Procedure**

To delete a routing endpoint:

1. In the JMX Navigator view, select the endpoint you want delete.

2. Right-click the selected endpoint to open the context menu, and then select Delete Endpoint. The tooling deletes the endpoint.
3. To remove the deleted endpoint from the view, right-click the **Endpoints** node, and select **Refresh**.
   The endpoint disappears from the **JMX Navigator** view.

![NOTE]

To remove the endpoint’s node from the **Project Explorer** view without rerunning the project, you need to explicitly delete it by right-clicking the node and selecting **Delete**. To remove it from view, refresh the project display.

**Related topics**

- Section 26.1, “Adding a routing endpoint”
CHAPTER 27. EDITING RUNNING ROUTES

OVERVIEW

You can experiment with changes to a running route without changing your project's routing context.

To do so:

- In the JMX Navigator view, enable the Edit Routes option on the running routing context. This opens an in-memory model of it — Remote CamelContext:<camelContextId> — in the route editor.

- In the route editor, make your changes to the in-memory model of the routing context. At the same time, you can set breakpoints on relevant nodes to use the Camel debugger and all of its features. You can edit the in-memory model to add, remove, or rearrange nodes; to add or remove properties of existing nodes; and to modify property values set on existing nodes. You must save changes made to the in-memory model to update the running context and to see results in the Debug perspective if you set breakpoints.

- In the JMX Navigator view, drop messages on the running routing context, or wait for messages to arrive from a timer, ActiveMQ, file, or other continuous input node.

- In the Debug perspective, evaluate results and use the Camel debugger to gain deeper insight into your routing context.

MODIFYING A RUNNING ROUTE AND EVALUATING RESULTS

1. In the JMX Navigator view, select the routing context that contains the routes you want to edit.

2. Right-click the selected routing context to open the context menu, and select Edit Routes. The route editor opens an in-memory model of the routing context, Remote CamelContext: <contextId>, and displays all routes in the context, for example:
3. Edit the route as described in Chapter 6, Editing a routing context in the route editor, then select File → Save to save the changes you made to the in-memory model and to update the running routing context.

4. Set breakpoints on the relevant nodes as described in Chapter 15, Setting Breakpoints.

5. In the JMX Navigator view, drop a message on the running routing context’s input node. If your project does not include test messages, you can create them as described in Section 24.1, “Creating test messages for route tracing”.

6. Click Yes to confirm the switch to the Debug perspective.

7. In the Camel debugger, step the message through the breakpoints as you normally would (see Chapter 16, Running the Camel Debugger) to see the results your changes generated. The Camel debugger behaves the same in Edit Routes mode as in normal debug mode, so you can use any of the Camel debugger’s features while a message is transiting the routing context.

NOTE
When a message reaches the end of the routing context, the debugger is suspended. To continue debugging, switch back to the Fuse Integration perspective and drop another message on the input node in the JMX Navigator view. Each time you do so, the tooling asks you to confirm the switch to the Debug perspective.

NOTE
During a route editing session, it is possible to lose the connection to the running routing context. If this happens, then in the JMX Navigator view, you would see something like this: Local Processes → maven[xxxx[Disconnected]]. To continue the session, you must reconnect to the running routing context, select it in the JMX Navigator view, and then re-select Edit Routes.

TERMINATING THE ROUTE EDITING SESSION
1. In the Debug perspective’s Debug view, select the Remote Camel Debug - camelContext--<contextId>--xxxxxxxxxxxxxxxxxx.xml [Remote Camel Context] thread, and then click on the menu bar to terminate the debugging session.

2. On Console view’s menu bar, click to terminate the routing context.

3. If you want to clear console output, click on the Console view’s menu bar.

4. Switch to the Fuse Integration perspective, and in the route editor, click on the Remote CamelContext:<contextId> tab to close the in-memory model of the routing context file.
RELATED TOPICS

- Chapter 6, *Editing a routing context in the route editor*
- Part III, “Debugging Routing Contexts”
CHAPTER 28. MANAGING ROUTING CONTEXTS

The JMX Navigator view lets you suspend and resume running routing contexts.

28.1. SUSPENDING OPERATION OF A ROUTING CONTEXT

Overview

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

Procedure

To suspend operation of a routing context:

1. In the JMX Navigator view, expand the project’s Camel Contexts node, and select the routing context whose operation you want to suspend.

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

NOTE

If Resume Camel Context appears on the context menu, operation of the context is already suspended.

Related topics

- Section 28.2, “Resuming operation of a routing context”

28.2. RESUMING OPERATION OF A ROUTING CONTEXT

Overview

The tooling lets you resume operation of a suspended routing context in the JMX Navigator view. Resuming operation of a context restarts all of the routes in it so that they can process messages.

Procedure

To resume operation of a routing context:

1. In the JMX Navigator view, expand the project’s Camel Contexts node, and select the routing context whose operation you want to resume.

2. Right-click the selected context to open the context menu, and then select Resume Camel Context.

NOTE

If Suspend Camel Context appears in the context menu, the context and its routes are running.
Related topics

- Section 28.1, “Suspending operation of a routing context”
PART V. PUBLISHING APPLICATIONS TO A CONTAINER

To publish Fuse Integration projects to a server container, you must first add the server and its runtime definition to the tooling's Servers list. Then you can assign projects to the server runtime and set the publishing options for it.
CHAPTER 29. MANAGING SERVERS

The Servers view lets you run and manage servers in your Red Hat JBoss Developer Studio environment. The supported servers are:

- Red Hat JBoss Fuse Server – versions 6.2+ and 6.3
- Red Hat JBoss Enterprise Application Platform – version 6.4

NOTE

For step by step instructions on how to publish a Camel project to Red Hat JBoss Fuse, see.

29.1. ADDING A SERVER

Overview

For the tooling to manage a server, you need to add the server to the Servers list. Once added, the server appears in the Servers view, where you can connect to it and publish your Fuse Integration projects.

NOTE

If adding a Red Hat JBoss Fuse 6.2+ or 6.3 server, it is recommended that you edit its installDir/etc/users.properties file and add user information, in the form of user=password, role, to enable the tooling to establish an SSH connection to the server.

Procedure

There are three ways to add a new server to the Servers view:

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

NOTE

This link appears in the Servers view only when no server has been defined. If you defined and added a server when you first created your project, the Servers view displays that server.

- In the Servers view, right-click to open the context menu and select New → Server.

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

In the Define a New Server dialog, to add a new server:

1. Expand the Red Hat JBoss Middleware node to expose the list of available server options:
Choose the type of server to create

Select the server type:

- OpenShift
- Red Hat JBoss Middleware
  - Red Hat Container Development Kit 2.x
  - Red Hat Container Development Kit 3
  - Red Hat JBoss Enterprise Application Platform 4.3
  - Red Hat JBoss Enterprise Application Platform 5.x
  - Red Hat JBoss Enterprise Application Platform 6.0
  - Red Hat JBoss Enterprise Application Platform 6.1+
  - Red Hat JBoss Enterprise Application Platform 7.0
  - Red Hat JBoss Enterprise Application Platform 7.1
  - Red Hat JBoss Fuse 6.0 Server
  - Red Hat JBoss Fuse 6.1 Server
  - Red Hat JBoss Fuse 6.2 Server
  - Red Hat JBoss Fuse 6.3 Server
- Resin

Server Definition of Red Hat JBoss Fuse 6.3

Server’s host name: localhost

Server name: Red Hat JBoss Fuse 6.3 Runtime Server

< Back  Next >  Cancel  Finish
2. Click the server you want to add.

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

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

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

5. Click Next to open the server’s Runtime definition page:

   ![Runtime definition page]

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

6. Accept the default for the installation Name.

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

8. Next to Execution Environment, select the runtime JRE from the drop-down menu.
   
   If the version you want does not appear in the list, click Environments and select the version from the list that appears. The JRE version you select must be installed on your machine.
NOTE

JBoss Fuse 6.3 requires either JRE version 1.8 or 1.7.

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

10. Click **Next** to save the server’s runtime definition and open its **Configuration details** page:

![Configuration details page](image)

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

   The runtime uses the SSH port to connect to the server’s Karaf shell. If this default is incorrect for your setup, you can discover the correct port number by looking in the server’s `installDir/etc/org.apache.karaf.shell.cfg` file.

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

   For Red Hat 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 the **User Name** and **Password** fields with the default user’s name and password, as shown in `[servCnfigDetails]`. The default user is already activated in JBoss Fuse 6.2, but not in JBoss Fuse 6.0 or 6.1.

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

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

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

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

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

The server runtime appears in the **Servers** view. For example:

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

### 29.2. STARTING A SERVER

#### Overview

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

#### Procedure

To start a server:

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

2. Click ![Run](https://example.com/run-icon.png).
   - The **Console** view opens and displays a message asking you to wait while the container is starting, for example:
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 29.1, “Adding a Server”.

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

  ![Terminal View]

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

  `JBoss Fuse (6.3.0.redhat-187)`

  `http://www.redhat.com/products/jbossenterprisemiddleware/fuse/`

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

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

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

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

  `JBossFuse:admin@root`

- The running server appears in the Servers view:
The running server also appears in the JMX Navigator view under Server Connections:

NOTE
If the server is running on the same machine as the tooling, the server also has an entry under Local Processes.

29.3. CONNECTING TO A RUNNING SERVER

Overview

After you start a configured server, it appears in the Servers view and in the JMX Navigator view under the Server Connections node. You may need to expand the Server Connections node to see the server.

To publish and test your Fuse project application on the running server, you must first connect to it. You can connect to a running server either in the Servers view or in the JMX Navigator view.

NOTE
The Servers view and the JMX Navigator view are synchronized with regards to server connections. That is, connecting to a server in the Servers view also connects it in the JMX Navigator view, and vice versa.

Connecting to a running server in the Servers view

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

2. Double-click the JMX[Disconnected] node:
Connecting to a running server in the JMX Navigator view

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

2. Double-click the selected server:

Viewing bundles installed on the connected server

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

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

Using the Properties view’s Search tool, you can search for bundles by their Symbolic Name or by their Identifier, if you know it. As you type the symbolic name or the identifier, the list updates, showing only the bundles that match the current search string.
NOTE

Alternatively, you can issue the `osgi:list` command in the Terminal view to see a generated list of bundles installed on the JBoss Fuse server runtime. The tooling uses a different naming scheme for OSGi bundles displayed by the `osgi:list` command.

In the `<build>` section of project's `pom.xml` file, you can find the bundle's symbolic name and its bundle name (OSGi) listed in the `maven-bundle-plugin` entry. For more details, see the section called “Verifying the project was published to the server”.

29.4. DISCONNECTING FROM A SERVER

Overview

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

NOTE

The Servers view and the JMX Navigator view are synchronized with regards to server connections. That is, disconnecting from a server in the Servers view also disconnects it in the JMX Navigator view, and vice versa.

Disconnecting from a server in the Servers view

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

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

Disconnecting from a server in the JMX Navigator view

1. In the JMX Navigator view, under Server Connections, select the server from which you want to disconnect.

2. Right-click the selected server to open the context menu, and then select Disconnect.
29.5. STOPPING A SERVER

Overview
You can shut down a server in the Servers view or in the server’s remote console in the Terminal view.

Using the Servers view
To stop a server:

1. In the Servers view, select the server you want to stop.
2. Click 

Using the remote console
To stop a server:

1. Open the Terminal view that is hosting the server’s remote console.
2. Type CTRL+D.

29.6. DELETING A SERVER

Overview
When you are finished with a configured server, or if you misconfigure a server, you can delete it and its configuration.

First, delete the server from the Servers view or from the JMX Navigator view. Next, delete the server’s configuration.
Deleting a server

1. In the **Servers** view, right-click the server you want to delete to open the context menu.

2. Select **Delete**.

3. Click **OK**.

Deleting the server's configuration

1. On the menu bar, select **JBoss Developer Studio → Preferences → Server**.

   NOTE

   On Linux and Windows machines, select menu: [Window > Preferences].

2. Expand the **Server** folder, and then select **Runtime Environments** to open the **Server Runtime Environments** page.

3. From the list, select the runtime environment of the server that you previously deleted from the **Servers** view, and then click **Remove**.

4. Click **OK**.
CHAPTER 30. PUBLISHING FUSE INTEGRATION PROJECTS TO A SERVER

You deploy Fuse Integration projects into a server runtime using the Eclipse publishing mechanism. To do so, you must have defined and added the server to the Servers view in the Fuse Integration perspective. For a step-by-step demonstration, see .

OVERVIEW

You can set up supported servers to publish assigned Fuse projects automatically or to publish them only when you manually invoke the publish command.

Each server runtime added to the Servers view has its own Overview page that contains its configuration, connection, and publishing details:

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 Automotively publish when resources change.

1. If necessary, start the server runtime to which you want to publish a Fuse project. For details, see Section 29.2, “Starting a Server”.

2. In the Servers view, double-click the server runtime to open its Overview page.

3. Expand Publishing, and then select Automatically publish when resources change.

4. To increase or decrease the interval between publishing cycles, click the radio button next to Publishing interval (in seconds) up or down, as appropriate.

5. In the Servers view, right-click the server runtime to open the context menu, and then select Add and Remove.

6. To assign a resource (in this case, the CBRroute Fuse project) to the server runtime:

All resources available for publishing appear in the Available column.
- Double-click it, or
- Select it, and click Add.

The selected resource moves to the Configured column:

At this stage, the time at which the assigned resource would actually be published depends on whether the server runtime was running and on the Publishing interval setting. However, if the server was stopped, you would have to manually publish the project after you started the server (for details, see the section called “Publishing Fuse projects manually”).

7. Click the If server started, publish changes immediately option to enable it:
This option ensures that the configured project is published immediately once you click Finish. The Automatically publish when resources change option on the server runtime Overview page ensures that the configured project is republished whenever changes made to the local project are saved.

8. **Click Finish.**

The project appears in the **Servers** view under the server runtime node, and the server runtime status reports `[Started, Publishing...]`.

When publishing is done, the status of both the server runtime and the project report is `[Started, Synchronized]`.
NOTE

For a server runtime, Synchronized means that all published resources on the server are identical to their local counterparts. For a published resource, Synchronized means that it is identical to its local counterpart.

PUBLISHING FUSE PROJECTS MANUALLY

1. If necessary, start the server runtime to which you want to publish a Fuse project. For details, see Section 29.2, “Starting a Server”.

2. In the Servers view, double-click the server runtime to open its Overview page.

3. Expand Publishing, and then select Never publish automatically.

4. Click File → Save to save the publishing option changes.

5. If the Fuse project has already been assigned to the server runtime, make sure this option is disabled: If server started, publish changes immediately:
   a. In the Servers view, right-click the server runtime to open the context menu.
   b. Click Add and Remove… to open the server’s Add and Remove page.
   c. If the following option is enabled, disable it: If server started, publish changes immediately.
   d. Skip to [finish].

6. If the Fuse project has not been assigned to the server runtime, assign it now:
   a. Follow [startAssignResource] through [stopAssignResource] in the section called “Publishing Fuse projects automatically when resources change”.
   b. Do not enable the If server started, publish changes immediately option.

7. Click Finish.
   The project appears in the Servers view under the server runtime node, and the server runtime status reports [Started]:

8. In the Servers view, right-click the project’s node. In this example, select the CBRroute Fuse project to open the context menu:
9. Select Full Publish.
   During the publishing operation, the status of both the server runtime and the project report
   [Started, Republish].

   When publishing is done, the status of both the server runtime and the project report
   [Started, Synchronized]:

   **NOTE**
   The tooling does not support the Incremental Publish option. Clicking
   Incremental Publish results in a full publish.

**VERIFYING THE PROJECT WAS PUBLISHED TO THE SERVER**

After you have published a Fuse project to a server runtime, you can connect to the server and check
that the project’s bundle was installed on it.

1. Connect to the server runtime. For details see the section called “Connecting to a running
   server in the Servers view”.

2. In the Servers view, expand the server runtime tree to expose the Bundles node and select it.
   The tooling populates the Properties view with a list of bundles that are installed on the
   server:
To find your project’s bundle, either scroll down to the bottom of the list, or start typing the bundle’s Symbolic Name in the Properties view’s Search box. The bundle’s Symbolic Name is the name you gave your project when you created it.

NOTE

Alternatively, you can issue the osgi:list command in the Terminal view to see a generated list of bundles installed on the JBoss Fuse server runtime. The tooling uses a different naming scheme for OSGi bundles displayed by the osgi:list command.

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

```xml
<build>
  <defaultGoal>install</defaultGoal>
  <plugins>
    <plugin>
      <groupId>org.apache.felix</groupId>
      <artifactId>maven-bundle-plugin</artifactId>
      <version>${version.maven-bundle-plugin}</version>
      <extensions>true</extensions>
      <configuration>
        <instructions>
          <Bundle-SymbolicName>CBRroute</Bundle-SymbolicName>
          <Bundle-Name>Empty Camel Blueprint Example [CBRroute]</Bundle-Name>
        </configuration>
      </plugin>
  </plugins>
</build>
```
A.1. EDITOR

Sets the default value for some route editor properties. Table A.1, “Editor Properties” describes the properties you can specify.

Table A.1. Editor Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>Simple</td>
<td>Specifies the default expression language used by the route editor.</td>
</tr>
<tr>
<td>ID values</td>
<td>Enabled</td>
<td>Determines whether ID values are used to label EIP nodes on the route editor's canvas.</td>
</tr>
<tr>
<td>Layout direction</td>
<td>Right</td>
<td>Specifies the direction in which routes are laid out on the route editor's canvas.</td>
</tr>
<tr>
<td>Diagram grid</td>
<td>Enabled</td>
<td>Determines whether a grid overly is displayed in the background on the route editor’s canvas.</td>
</tr>
</tbody>
</table>

Related topics * Section 6.9, “Configuring the route editor”