Red Hat Fuse 7.1

Getting Started

Get started quickly with Red Hat Fuse!

Last Updated: 2019-06-25

Red Hat Fuse 7.1 Getting Started

Get started quickly with Red Hat Fuse!

Legal Notice

Copyright © 2019 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

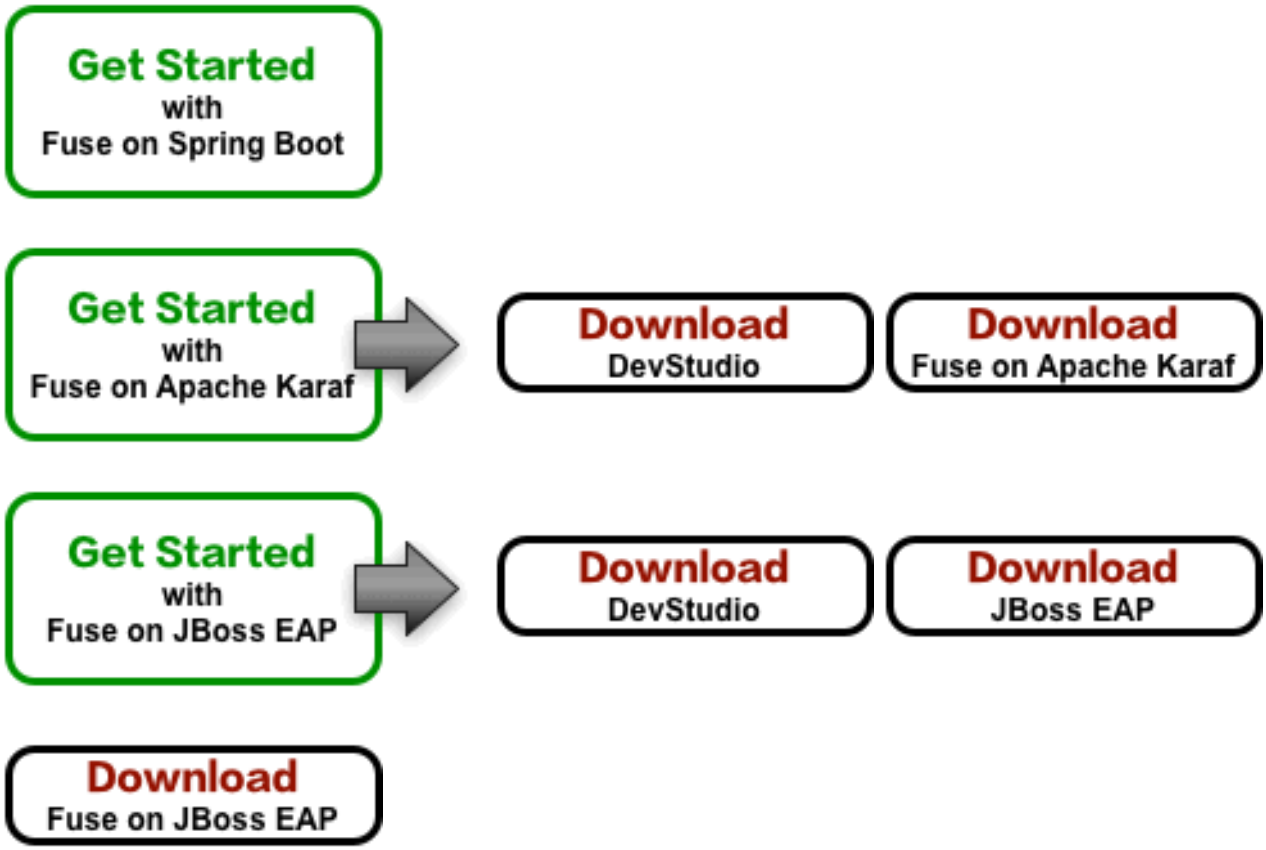
Get started with Fuse on Spring Boot, Fuse on Apache Karaf, Fuse on JBoss Enterprise Application Platform, and Fuse on JBoss Web Server

Table of Contents

CHAPTER 1. GETTING STARTED DASHBOARD

1.1. ALTERNATIVES FOR GETTING STARTED WITH FUSE STANDALONE

The following dashboard shows alternative paths for getting started with Fuse standalone, depending on which container type you prefer:



1.2. OVERVIEW OF THE FUSE CONTAINERS

To help you choose the right container for your project, the following sections give a brief overview of each container type.

1.2.1. JBoss EAP

JBoss Enterprise Application Platform (EAP), based on Jakarta EE (previously, Java EE) technology from the Eclipse Foundation, was originally created to address the use cases for developing enterprise applications. Characterized by well-defined patterns for implementing services and standardized Java APIs (for accessing services such as persistence, messaging, security, and so on), in recent years this technology has evolved to be more lightweight, with the introduction of CDI for dependency injection and simplified annotations for enterprise Java beans.

Distinctive features of this container technology are:

- Particularly suited to running in standalone mode.
- Many standard services (for example, persistence, messaging, security, and so on) pre-configured and provided out of the box.

- Application WARs typically small and lightweight (since many dependencies are pre-installed in the container).
- Standardized, backward-compatible Java APIs.

1.2.2. Apache Karaf

Apache Karaf is based on the OSGi standard from the OSGi Alliance. OSGi originated in the telecommunications industry, where it was used to develop gateway servers that could be upgraded on the fly, without needing to shut down the server (a feature known as *hot code swapping*). Subsequently, OSGi container technology has found a variety of other uses and is popular for modularised applications (for example, the Eclipse IDE).

Distinctive features of this container technology are:

- Particularly suited to running in standalone mode.
- Strong support for modularisation (OSGi bundles), with sophisticated class-loading support.
- Multiple versions of a dependency can be deployed side by side in a container (but this requires some care in practice).
- Hot code swapping, enabling you to upgrade or replace a module without shutting down the container. This is a unique feature, but requires significant effort to make it work properly.

1.2.3. Spring Boot

Spring Boot is a recent evolution of the well-known Spring container. A distinctive quality of the Spring Boot container is that container functionality is divided up into small chunks, which can be deployed independently. This enables you to deploy a container with a small footprint, specialized for a particular kind of service, and this happens to be exactly what you need to fit the paradigm of a *microservices architecture*.

Distinctive features of this container technology are:

- Particularly suited to running on a scalable cloud platform (Kubernetes and OpenShift).
- Small footprint (ideal for microservices architecture).
- Optimized for *convention over configuration*.
- No application server required. You can run a Spring Boot application Jar directly in a JVM.

CHAPTER 2. GETTING STARTED WITH SPRING BOOT

2.1. OVERVIEW OF THE CIRCUIT BREAKER BOOSTER

The Netflix/Hystrix circuit breaker component enables distributed applications to cope with interruptions to network connectivity and temporary unavailability of backend services. The basic idea of the circuit breaker pattern is that the loss of a dependent service is detected automatically and an alternative behavior can be programmed, in case the backend service is temporarily unavailable.

The Fuse circuit breaker booster consists of two related services:

- A *name service*, which returns a name to greet, and
- A *greetings service*, which invokes the name service to get a name and then returns the string, **Hello, NAME**.

In this demonstration, the Hystrix circuit breaker is inserted between the greetings service and the name service. If the name service becomes unavailable, the greetings service can fall back to an alternative behavior and respond to the client immediately, instead of blocking while it waits for the name service to restart.

2.2. PREREQUISITES

To build and run the booster demonstration, install the following prerequisites:

- A supported version of the Java Developer Kit (JDK). See the Supported Configurations page for details.
- Apache Maven 3.3.x or later. See the Maven Download page. To learn more about Maven, see Appendix A, *Preparing to use Maven*.

2.3. GENERATE THE BOOSTER PROJECT

To generate the circuit breaker booster project, perform the following steps:

1. Navigate to https://developers.redhat.com/launch.
2. Click **START**.
   The launcher wizard prompts you to log in to your Red Hat account.
3. The launcher wizard prompts you to log in to your Red Hat account. Click the **Log in or register** button to log in.
4. On the **Launcher** page, click the **Deploy an Example Application** button.
5. On the **Create Example Application** page, type the name, **fuse-circuit-breaker**, in the **Create Example Application as** field.
6. Click **Select an Example**.
7. In the **Example** dialog, select the **Circuit Breaker** option. A **Select a Runtime** dropdown menu appears.
   a. From the **Select a Runtime** dropdown, select **Fuse**.

b. From the version dropdown menu, select **7.1.0 (Red Hat Fuse)** (do not select the **2.21.2 (Community)** version).

c. Click **Save**.

8. On the **Create Example Application** page, click **Download**.

9. When you see the **Your Application is Ready** dialog, click **Download.zip**. Your browser downloads the generated booster project (packaged as a ZIP file).

10. Your browser now commences downloading the generated booster project (packaged as a ZIP file).

11. After downloading the ZIP file, use an archive utility to extract the generated project to a convenient location on your local filesystem.

2.4. BUILD AND RUN THE BOOSTER

To build and run the booster project, perform the following steps:

1. Open a shell prompt and build the project from the command line, using Maven:

```
cd fuse-circuit-breaker
mvn clean package
```

2. Open a new shell prompt and start the name service, as follows:

```
cd name-service
mvn spring-boot:run -DskipTests -Dserver.port=8081
```

As Spring Boot starts up, you should see some output like the following:

```
...
2019-06-25 09:33:01.444  INFO 11066 --- [           main]
o.a.camel.spring.SpringCamelContext      : Route: route1 started and consuming from:
servlet:/name?httpMethodRestrict=GET
2019-06-25 09:33:01.445  INFO 11066 --- [           main]
o.a.camel.spring.SpringCamelContext      : Total 1 routes, of which 1 are started
2019-06-25 09:33:01.446  INFO 11066 --- [           main]
o.a.camel.spring.SpringCamelContext      : Apache Camel 2.21.0.fuse-710018-redhat-00001
(CamelContext: camel-1) started in 0.490 seconds
2019-06-25 09:33:01.460  INFO 11066 --- [           main] o.a.c.c.s.CamelHttpTransportServlet
: Initialized CamelHttpTransportServlet[name=CamelServlet, contextPath=]
2019-06-25 09:33:01.566  INFO 11066 --- [           main]
b.c.e.u.UndertowEmbeddedServletContainer : Undertow started on port(s) 8081 (http)
2019-06-25 09:33:01.572  INFO 11066 --- [           main]
com.redhat.fuse.boosters.cb.Application  : Started Application in 6.102 seconds (JVM
running for 9.772)
```

3. Open a new shell prompt and start the greetings service, as follows:

```
cd greetings-service
mvn spring-boot:run -DskipTests
```

As Spring Boot starts up, you should see some output like the following:

```
...
2019-06-25 09:35:00.028  INFO 11224 --- [           main] o.a.c.c.s.CamelHttpTransportServlet
: Initialized CamelHttpTransportServlet[name=CamelServlet, contextPath=]
2019-06-25 09:35:00.101  INFO 11224 --- [           main]
b.c.e.u.UndertowEmbeddedServletContainer : Undertow started on port(s) 8080 (http)
2019-06-25 09:35:00.108  INFO 11224 --- [           main]
com.redhat.fuse.boosters.cb.Application  : Started Application in 7.248 seconds (JVM
running for 11.143)
```

4. The greetings service exposes a REST endpoint at the URL, **http://localhost:8080/camel/greetings**. You can invoke the REST endpoint either from a Web browser or from a shell prompt, using the **curl** command, as follows:

```
$ curl http://localhost:8080/camel/greetings

{"greetings":"Hello, Jacopo"}
```

5. To demonstrate the circuit breaker functionality provided by Camel Hystrix, kill the backend name service by typing Ctrl-C in the window of the shell prompt where the name service is running.

6. Now that the name service is unavailable, the circuit breaker kicks in to prevent the greetings service from hanging when it is invoked. Invoke the greetings REST endpoint using the **curl** command, as follows:

```
$ curl http://localhost:8080/camel/greetings

{"greetings":"Hello, default fallback"}
```

The log in the window where the greetings service is running shows the following sequence of messages:

```
2019-06-25 09:42:42.766  INFO 11224 --- [-CamelHystrix-6] route2                   :
Try to call name Service
2019-06-25 09:42:42.771  INFO 11224 --- [-CamelHystrix-6]
o.a.c.httpclient.HttpMethodDirector      : I/O exception (java.net.ConnectException) caught
when processing request: Connection refused (Connection refused)
2019-06-25 09:42:42.771  INFO 11224 --- [-CamelHystrix-6]
o.a.c.httpclient.HttpMethodDirector      : Retrying request
2019-06-25 09:42:42.771  INFO 11224 --- [-CamelHystrix-6]
o.a.c.httpclient.HttpMethodDirector      : I/O exception (java.net.ConnectException) caught
when processing request: Connection refused (Connection refused)
2019-06-25 09:42:42.771  INFO 11224 --- [-CamelHystrix-6]
o.a.c.httpclient.HttpMethodDirector      : Retrying request
2019-06-25 09:42:42.772  INFO 11224 --- [-CamelHystrix-6]
o.a.c.httpclient.HttpMethodDirector      : I/O exception (java.net.ConnectException) caught
when processing request: Connection refused (Connection refused)
2019-06-25 09:42:42.772  INFO 11224 --- [-CamelHystrix-6]
o.a.c.httpclient.HttpMethodDirector      : Retrying request
2019-06-25 09:42:42.781  INFO 11224 --- [-CamelHystrix-6] route2                   :
We are falling back!!!!
```

7. For more information about this example, visit the Circuit Breaker Mission page at http://localhost:8080/ (while the **greetings-service** is running). This page provides a link to the Hystrix dashboard, which monitors the state of the circuit breaker.

CHAPTER 3. GETTING STARTED WITH APACHE KARAF

3.1. LOG IN TO THE CUSTOMER PORTAL

Before you can download the required packages, you need an account on Red Hat's Customer Portal which has a Red Hat Fuse subscription. Using this account, log in to the portal at https://access.redhat.com/login.

3.2. DOWNLOAD THE REQUIRED PACKAGES

Click each of the **Download** buttons to get the required packages from the Customer Portal:

3.3. INSTALL AND CONFIGURE FUSE ON APACHE KARAF

To install and configure Fuse on Apache Karaf, perform the following steps:

1. Unpack the downloaded **.zip** archive file for Fuse on Apache Karaf to a convenient location on your file system, **FUSE_INSTALL**.
2. Add an administrator user to the Fuse runtime.
   a. Open the **FUSE_INSTALL/etc/users.properties** file in a text editor.
   b. Delete the **#** character at the start of the line that starts with **#admin = admin**.
   c. Delete the **#** character at the start of the line that starts with ***#g*\:admingroup**.
   d. Customize the username, **USERNAME**, and password, **PASSWORD**, of the user entry, so that you have a user entry and an admin group entry like the following (on consecutive lines):

```
USERNAME = PASSWORD,_g_:admingroup
_g_\:admingroup = group,admin,manager,viewer,systembundles,ssh
```

   e. Save the **etc/users.properties** file.

3.4. SET UP YOUR DEVELOPMENT ENVIRONMENT

To set up your development environment, perform the following steps:

1. Run the Developer Studio installer, as follows:

```
java -jar DOWNLOAD_LOCATION/devstudio-12.0.0.GA-installer-standalone.jar
```

2. During installation:

a. Accept the terms and conditions.

b. Choose your preferred installation path.

c. Select the Java 8 JVM.

d. At the **Select Platforms and Servers** step, configure the Fuse on Karaf runtime by clicking **Add** and browsing to the location of the **FUSE_INSTALL** directory (see Section 3.3, "Install and configure Fuse on Apache Karaf").

e. At the **Select Additional Features to Install** step, select **Red Hat Fuse Tooling**.

3. Developer Studio starts up. When the **Searching for runtimes** dialog appears, click **OK** to create the Fuse on Karaf runtime.

4. *(Optional)* In order to use Apache Maven from the command line, you need to install and configure Maven as described in Appendix A, *Preparing to use Maven*.

NOTE

If you are using Developer Studio exclusively, it is not strictly necessary to install Maven, because Developer Studio has Maven pre-installed and configured for you. But if you plan to invoke Maven from the command line, it is necessary to perform this step.

3.5. BUILD YOUR FIRST APPLICATION

To build your first application with Fuse on Karaf, perform the following steps:

1. In Developer Studio, create a new project, as follows:

a. Select **File→New→Fuse Integration Project**.

b. Enter **fuse-camel-cbr** in the **Project Name** field.

c. Click **Next**.

d. In the **Select a Target Environment** pane, choose the following settings:

- Select **Standalone** as the deployment platform.
- Select **Karaf/Fuse on Karaf** as the runtime environment and use the **Runtime (optional)** dropdown menu to select the **fuse-karaf-7.1.0.fuse-710023-redhat-00001 Runtime** server as the target runtime.

e. After selecting the target runtime, the **Camel Version** is automatically selected for you and the field is grayed out.

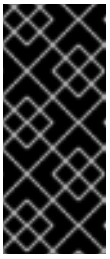
f. Click **Next**.

g. In the **Advanced Project Setup** pane, select the **Beginner→Content Based Router - Blueprint DSL** template.

h. Click **Finish**.

i. If prompted to open the associated Fuse Integration perspective, click **Yes**.

j. Wait while Developer Studio downloads required artifacts and builds the project in the background.



IMPORTANT

If this is the first time you are building a Fuse project in Developer Studio, it will take *several minutes* for the wizard to finish generating the project, as it downloads dependencies from remote Maven repositories. Do not attempt to interrupt the wizard or close Developer Studio while the project is building in the background.

2. Deploy the project to the server, as follows:

a. In the **Servers** view (bottom left corner of the Fuse Integration perspective), if the server is not already started, select the **fuse-karaf-7.1.0.fuse-710023-redhat-00001 Runtime Server** server and click the green arrow to start it.

NOTE

If you see the dialog, **Warning: The authenticity of host 'localhost' can't be established.**, click **Yes** to connect to the server and access the Karaf console.

b. Wait until you see a message like the following in the **Console** view:

```
Karaf started in 1s. Bundle stats: 12 active, 12 total
```

c. After the server has started, switch back to the **Servers** view, right-click on the server and select **Add and Remove** from the context menu.

d. In the **Add and Remove** dialog, select the **fuse-camel-cbr** project and click the **Add >** button.

e. Click **Finish**.

f. You can check whether the project's OSGi bundle has started up by going to the **Terminal** view and entering **bundle:list | tail**. You should see some output like the following:

```
...
228 │ Active │  80 │ 1.0.0.201505202023          │ org.osgi:org.osgi.service.j
232 │ Active │  80 │ 1.0.0.SNAPSHOT              │ Fuse CBR Quickstart
```

3.5.1. Verify the project

As soon as the Camel route starts up, it will create a directory, **work/cbr/input**, in your Fuse installation (*not* in the **fuse-camel-cbr** project).

Now you can test your Camel route and see it in action.

Copy the files you find in the project's **fuse-camel-cbr/src/main/data** directory (under the Eclipse workspace directory) to the newly created **work/cbr/input** directory. You can do this in your system file browser (outside of Eclipse).

Wait a few moments and you will find the same files organized by country under the **work/cbr/output** directory:

1. **order1.xml** in **work/cbr/output/others**
2. **order2.xml** and **order4.xml** in **work/cbr/output/uk**
3. **order3.xml** and **order5.xml** in **work/cbr/output/us**

Enter **log:display** in the Karaf console (**Terminal** view) to see the application output in the log, for example:

```
...
15:46:14.859 INFO [Camel (cbr-example-context) thread #3 - file://work/cbr/input] Receiving order
order2.xml
15:46:14.888 INFO [Camel (cbr-example-context) thread #3 - file://work/cbr/input] Sending order
order2.xml to the UK
15:46:14.891 INFO [Camel (cbr-example-context) thread #3 - file://work/cbr/input] Done processing
order2.xml
15:46:14.895 INFO [Camel (cbr-example-context) thread #3 - file://work/cbr/input] Receiving order
order3.xml
```

3.5.2. Undeploy the project

Undeploy the project, as follows:

1. In the **Servers** view, select the **fuse-karaf-7.1.0.fuse-710023-redhat-00001 Runtime Server** server.
2. Right-click on the server and select **Add and Remove** from the context menu.
3. In the **Add and Remove** dialog, select your **fuse-camel-cbr** project and click the **< Remove** button.
4. Click **Finish**.

CHAPTER 4. GETTING STARTED WITH FUSE ON JBOSS ENTERPRISE APPLICATION PLATFORM

4.1. LOG IN TO THE CUSTOMER PORTAL

Before you can download the required packages, you need an account on Red Hat's Customer Portal which has a Red Hat Fuse subscription. Using this account, log in to the portal at https://access.redhat.com/login.

4.2. DOWNLOAD THE REQUIRED PACKAGES

Click each of the **Download** buttons to get the required packages from the Customer Portal:

4.3. INSTALL AND CONFIGURE FUSE ON JBOSS EAP

To install and configure Fuse on JBoss EAP, perform the following steps:

1. Run the JBoss EAP installer from a shell prompt, as follows:

```
java -jar DOWNLOAD_LOCATION/jboss-eap-7.1.0-installer.jar
```

2. During installation:
   a. Accept the terms and conditions.
   b. Choose your preferred installation path, **EAP_INSTALL**, for the JBoss EAP runtime.
   c. Create an administrative user and make a careful note of these administrative user credentials for later.
   d. You can accept the default settings on the remaining screens.
3. Open a shell prompt and change directory to **EAP_INSTALL**.
4. From the **EAP_INSTALL** directory, run the Fuse on EAP installer, as follows:

```
java -jar DOWNLOAD_LOCATION/fuse-eap-installer-7.1.0.fuse-710018-redhat-00001.jar
```

4.4. SET UP YOUR DEVELOPMENT ENVIRONMENT

To set up your development environment, perform the following steps:

1. Run the Developer Studio installer, as follows:

```
java -jar DOWNLOAD_LOCATION/devstudio-12.0.0.GA-installer-standalone.jar
```

2. During installation:
   a. Accept the terms and conditions.
   b. Choose your preferred installation path.
   c. Select the Java 8 JVM.
   d. At the **Select Platforms and Servers** step, configure the JBoss EAP runtime by clicking **Add** and browsing to the location of the **EAP_INSTALL** directory (see Section 4.3, "Install and configure Fuse on JBoss EAP").
   e. At the **Select Additional Features to Install** step, select **Red Hat Fuse Tooling**.
3. Developer Studio starts up. When the **Searching for runtimes** dialog appears, click **OK** to create the JBoss EAP runtime.
4. *(Optional)* In order to use Apache Maven from the command line, you need to install and configure Maven as described in Appendix A, *Preparing to use Maven*.

NOTE

If you are using Developer Studio exclusively, it is not strictly necessary to install Maven, because Developer Studio has Maven pre-installed and configured for you. But if you plan to invoke Maven from the command line, it is necessary to perform this step.

4.5. BUILD YOUR FIRST APPLICATION

To build your first application with Fuse on JBoss EAP, perform the following steps:

1. In Developer Studio, create a new project, as follows:
   a. Select **File→New→Fuse Integration Project**.
   b. Enter **eap-camel** in the **Project Name** field.
   c. Click **Next**.
   d. In the **Select a Target Environment** pane, choose the following settings:
      - Select **Standalone** as the deployment platform.
      - Select **Wildfly/Fuse on EAP** as the runtime environment and use the **Runtime (optional)** dropdown menu to select the **JBoss EAP 7.x Runtime** server as the target runtime.
   e. After selecting the target runtime, the **Camel Version** is automatically selected for you and the field is grayed out.

f. Click **Next**.

g. In the **Advanced Project Setup** pane, select the **Spring Bean - Spring DSL** template.

h. Click **Finish**.

i. If prompted to open the associated Fuse Integration perspective, click **Yes**.

j. Wait while Developer Studio downloads required artifacts and builds the project in the background.

IMPORTANT

If this is the first time you are building a Fuse project in Developer Studio, it will take *several minutes* for the wizard to finish generating the project, as it downloads dependencies from remote Maven repositories. Do not attempt to interrupt the wizard or close Developer Studio while the project is building in the background.

2. Deploy the project to the server, as follows:

a. In the **Servers** view (bottom left corner of the Fuse Integration perspective), if the server is not already started, select the **Red Hat JBoss EAP 7.1 Runtime** server and click the green arrow to start it.

b. Wait until you see a message like the following in the **Console** view:

```
12:02:25,467 INFO  [org.jboss.as] (Controller Boot Thread) JBAS015874: JBoss EAP
6.4.0.GA (AS 7.5.0.Final-redhat-21) started in 9494ms - Started 480 of 518 services (69
services are lazy, passive or on-demand)
```

c. After the server has started, switch back to the **Servers** view, right-click on the server and select **Add and Remove** from the context menu.

d. In the **Add and Remove** dialog, select the **eap-camel** project and click the **Add >** button.

e. Click **Finish**.

3. Verify that the project is working, as follows:

a. Browse to the following URL to access the service running in the **eap-camel** project: http://localhost:8080/camel-test-spring?name=Kermit

b. The browser window should show the response **Hello Kermit**.

4. Undeploy the project, as follows:

a. In the **Servers** view, select the **Red Hat JBoss EAP 7.1 Runtime** server.

b. Right-click on the server and select **Add and Remove** from the context menu.

c. In the **Add and Remove** dialog, select your **eap-camel** project and click the **< Remove** button.

d. Click **Finish**.

APPENDIX A. PREPARING TO USE MAVEN

A.1. OVERVIEW

This section gives a brief overview of how to prepare Maven for building Red Hat Fuse projects and introduces the concept of Maven coordinates, which are used to locate Maven artifacts.

A.2. PREREQUISITES

In order to build a project using Maven, you must have the following prerequisites:

- **Maven installation** – Maven is a free, open source build tool from Apache. You can download the latest version from the Maven download page .
- **Network connection** – whilst performing a build, Maven dynamically searches external repositories and downloads the required artifacts on the fly. By default, Maven looks for repositories that are accessed over the Internet. You can change this behavior so that Maven will prefer searching repositories that are on a local network.

NOTE

Maven can run in an offline mode. In offline mode Maven only looks for artifacts in its local repository.

A.3. ADDING THE RED HAT MAVEN REPOSITORIES

In order to access artifacts from the Red Hat Maven repositories, you need to add them to Maven's **settings.xml** file. Maven looks for your **settings.xml** file in the **.m2** directory of the user's home directory. If there is not a user specified **settings.xml** file, Maven will use the system-level **settings.xml** file at **M2_HOME/conf/settings.xml**.

To add the Red Hat repositories to Maven's list of repositories, you can either create a new **.m2/settings.xml** file or modify the system-level settings. In the **settings.xml** file, add **repository** elements for the Red Hat repositories as shown in Adding the Red Hat Fuse Repositories to Maven .

Adding the Red Hat Fuse Repositories to Maven

```
<?xml version="1.0"?>
<settings>

  <profiles>
    <profile>
      <id>extra-repos</id>
      <activation>
        <activeByDefault>true</activeByDefault>
      </activation>
      <repositories>
       <repository>
          <id>redhat-ga-repository</id>
          <url>https://maven.repository.redhat.com/ga</url>
          <releases>
             <enabled>true</enabled>
          </releases>
```

```
        <snapshots>
          <enabled>false</enabled>
        </snapshots>
      </repository>
      <repository>
        <id>redhat-ea-repository</id>
        <url>https://maven.repository.redhat.com/earlyaccess/all</url>
        <releases>
          <enabled>true</enabled>
        </releases>
        <snapshots>
          <enabled>false</enabled>
        </snapshots>
      </repository>
      <repository>
        <id>jboss-public</id>
        <name>JBoss Public Repository Group</name>
        <url>https://repository.jboss.org/nexus/content/groups/public/</url>
      </repository>
    </repositories>
    <pluginRepositories>
      <pluginRepository>
        <id>redhat-ga-repository</id>
        <url>https://maven.repository.redhat.com/ga</url>
        <releases>
          <enabled>true</enabled>
        </releases>
        <snapshots>
          <enabled>false</enabled>
        </snapshots>
      </pluginRepository>
      <pluginRepository>
        <id>redhat-ea-repository</id>
        <url>https://maven.repository.redhat.com/earlyaccess/all</url>
        <releases>
          <enabled>true</enabled>
        </releases>
        <snapshots>
          <enabled>false</enabled>
        </snapshots>
      </pluginRepository>
      <pluginRepository>
        <id>jboss-public</id>
        <name>JBoss Public Repository Group</name>
        <url>https://repository.jboss.org/nexus/content/groups/public</url>
      </pluginRepository>
    </pluginRepositories>
  </profile>
 </profiles>

 <activeProfiles>
  <activeProfile>extra-repos</activeProfile>
 </activeProfiles>

</settings>
```

A.4. ARTIFACTS

The basic building block in the Maven build system is an *artifact*. The output of an artifact, after performing a Maven build, is typically an archive, such as a JAR or a WAR.

A.5. MAVEN COORDINATES

A key aspect of Maven functionality is the ability to locate artifacts and manage the dependencies between them. Maven defines the location of an artifact using the system of *Maven coordinates*, which uniquely define the location of a particular artifact. A basic coordinate tuple has the form, **{*groupId*, *artifactId*, *version*}**. Sometimes Maven augments the basic set of coordinates with the additional coordinates, *packaging* and *classifier*. A tuple can be written with the basic coordinates, or with the additional *packaging* coordinate, or with the addition of both the *packaging* and *classifier* coordinates, as follows:

```
groupId:artifactId:version
groupId:artifactId:packaging:version
groupId:artifactId:packaging:classifier:version
```

Each coordinate can be explained as follows:

groupdId

Defines a scope for the name of the artifact. You would typically use all or part of a package name as a group ID – for example, **org.fusesource.example**.

artifactId

Defines the artifact name (relative to the group ID).

version

Specifies the artifact's version. A version number can have up to four parts: **n.n.n.n**, where the last part of the version number can contain non-numeric characters (for example, the last part of **1.0-SNAPSHOT** is the alphanumeric substring, **0-SNAPSHOT**).

packaging

Defines the packaged entity that is produced when you build the project. For OSGi projects, the packaging is **bundle**. The default value is **jar**.

classifier

Enables you to distinguish between artifacts that were built from the same POM, but have different content.

The group ID, artifact ID, packaging, and version are defined by the corresponding elements in an artifact's POM file. For example:

```
<project ... >
  ...
  <groupId>org.fusesource.example</groupId>
  <artifactId>bundle-demo</artifactId>
  <packaging>bundle</packaging>
  <version>1.0-SNAPSHOT</version>
  ...
</project>
```

For example, to define a dependency on the preceding artifact, you could add the following **dependency** element to a POM:

-

```
<project ... >
  ...
  <dependencies>
    <dependency>
      <groupId>org.fusesource.example</groupId>
      <artifactId>bundle-demo</artifactId>
      <version>1.0-SNAPSHOT</version>
    </dependency>
  </dependencies>
  ...
</project>
```

NOTE

It is **not** necessary to specify the **bundle** package type in the preceding dependency, because a bundle is just a particular kind of JAR file and **jar** is the default Maven package type. If you do need to specify the packaging type explicitly in a dependency, however, you can use the **type** element.