Progress FUSE

FUSE™ ESB

Migrating to 4.1

Version 4.1
April 2009

PROGRESS SOFTWARE

Migrating to 4.1

Version 4.1

Publication date 22 Jul 2009

Legal Notices

Table of Contents

List of Examples

Part I. Differences Between Version 3.X and Version 4.1

Version 4.1 has a number of functional differences from 3.x versions.

Chapter 1. Feature Disparities

While FUSE ESB 4.1 retains most of the functionality of version 3.x, there are a number of issues you will encounter when moving to version 4.1.

Clustering

There is no longer a central way to configure a cluster of FUSE ESB containers. You create a cluster using explicit JMS endpoints that shuttle messages between the members of a cluster. For more information see "JMS Flows and Clustering" on page 29.

Flows

FUSE ESB 4.1 no longer uses *flows* for message exchanges. Applications that relied on specific features of the JMS flow or the JCA flow will need to find other ways to access those features.

Hot deployment

The default name of the hot deployment folder has been changed to `deploy`. You can change the location of the hot deployment folder by editing the org.apache.servicemix.filemonitor.monitorDir in `etc/config.properties`.

The default scan interval is 500ms. You can change the interval by editing the org.apache.servicemix.filemonitor.scanInterval in `etc/config.properties`.

Transactions

Transactions now work the same way for both synchronous and asynchronous JBI exchanges. The transaction is conveyed with the exchange in all cases. The resulting behavior is the same as using synchronous sends in previous versions of FUSE ESB.

Chapter 2. Configuration

Most of the configuration has been completely altered for FUSE ESB 4.1. This makes the job of configuring the FUSE ESB runtime easier.

Container XML configuration

The container no longer uses Spring XML configuration. Configuration is done using per-bundle properties files.

JMX configuration

The JMX configuration has been moved from the `service.properties` file to the `org.apache.servicemix.management.cfg` file.

For more information see "Changing the JMX Management Properties" in *Managing the Container*.

Logging configuration

The logging configuration has been moved from the `log4j.xml` file to the `org.ops4j.pax.logging.cfg` file.

For more information see "Logging Configuration" in *Managing the Container*

Chapter 3. API Changes

Some classes in FUSE ESB version 3.3 have been moved in FUSE ESB version 4.1.

Overview

A number of classes been moved during the upgrade to version 4.1. In most cases the classes have simply been packaged into different jars. In some cases, the classes have been moved to new packages.

New jars

The `servicemix-core` jar and the `servicemix-jbi` jar have been removed from FUSE ESB 4.1. The classes that were packaged in those jars have been moved to the following jars:

- `servicemix-common`
- `servicemix-utils`
- `org.apache.servicemix.specs.jbi-api-1.0`

If your projects have explicit dependencies on either of these artifacts, you need to update it to depend on the new artifacts.

Message exchange listeners

If you application uses a custom implementation of the `MessageExchangeListener` interface you will need to update your code. The interface has been moved from the `org.apache.servicemix` package to the `org.apache.servucemix.jbi.listener` package. Example 3.1 on page 13 shows the import statement for the `MessageExchangeListener` interface.

Example 3.1. Importing MessageExchangeListener

```
import org.apache.servicemix.jbi.listener.MessageExchangeL
istener
```

Chapter 4. JBI Issues

FUSE ESB 4.1 maintains full JBI compliance. There are a few differences between 4.1 and the 3.x versions.

OSGi Components

All of the JBI components included in FUSE ESB 4.1 are packaged as OSGi bundles. This means that:

- Applications do not need to use the JBI packaging to use the JBI components.

Note

Service units packaged as OSGi bundles will be subject to JBI lifecycle semantics.

- Service units can be deployed using a simple XML file.
- Endpoints deployed in JBI service units can communicate with OSGi services.

Component Configuration

All of JBI component configuration that used to reside in `installDir/conf/components.properties` are now configured using per-component configuration files. For example, the component configuration for the JMS component is located in `installDir/etc/servicemix-jms.cfg`.

In addition, each component's thread pool can be configured independent of the other components. Each component's configuration file includes a threadPoolCorePoolSize property and a threadPoolMaximumPoolSize property. The threadPoolCorePoolSize property specifies the minimum number of threads in a component's thread pool at start-up. The threadPoolMaximumPoolSize property specifies the maximum number of threads allowed in a component's thread pool.

Deprecated components

The following JBI components are not included in FUSE ESB 4.1:

- lightweight container

- jsr181

Packaging

Older versions of the JBI tooling generated JAR files for service assemblies and only converted them to ZIP files when they were deployed. Version 4.1 of the tooling creates ZIP files for service assemblies and places them in the assembly project's `target` folder.

Shared libraries

Previous versions of FUSE ESB allowed you to reference shared libraries from within service units. This functionality is not included in 4.1. Service units will have to explicitly include all of the required classes and support files they require.

Chapter 5. FUSE Services Framework Issues

FUSE ESB 4.1 uses FUSE Services Framework 2.2. This introduces a few migration issues.

Migrating from FUSE ESB 3.3 or Earlier

Overview

FUSE ESB 3.3, and earlier versions use FUSE Services Framework version 2.0. FUSE ESB 4.1 has jumped two releases of FUSE Services Framework. This section outlines the delta between FUSE Services Framework 2.0 and FUSE Services Framework 2.1. The following section outlines the delta between version 2.1 and version 2.2.

JAXB 2.1 annotations

The code generated by the 4.1 code generators adds some JAXB 2.1 specific annotations. These annotations are not compatible with previous versions of FUSE Services Framework.

WS-Addressing

JAX-WS 2.1 supports WS-Addressing directly in the APIs. WSDLs that use the EnpointReferenceType will now generate the JAX-WS 2.1 `EndpointReference` instead of the FUSE Services Framework proprietary type that was generated in FUSE Services Framework 2.0.

You can use the **wsdl2java** command's `-noAddressBinding` option to force the generation of the old-style endpoint reference code.

Tooling

The tool and the Maven code generator plug-in's goal have been replaced with more generic tools.

The tool has been replaced with the **java2ws** tool.

The Maven code generator plug-in's **java2wsdl** goal has been replaced with the new java2ws plug-in. Example 5.1 on page 18 shows an example of using the new plug-in.

Example 5.1. Using the Maven java2ws Plug-In

```
<project>
  ...
  <plugin>
    <groupId>org.apache.cxf</groupId>
    <artifactId>cxf-java2ws-plugin</artifactId>
    <version>2.1</version>
    <dependencies>
      <dependency>
        <groupId>org.apache.cxf</groupId>
        <artifactId>cxf-rt-frontend-jaxws</artifactId>
        <version>2.1</version>
```

```
      </dependency>
      <dependency>
        <groupId>org.apache.cxf</groupId>
        <artifactId>cxf-rt-frontend-simple</artifactId>
        <version>2.1</version>
      </dependency>
    </dependencies>
    <executions>
      <execution>
        <id>generate-test-sources</id>
        <phase>generate-test-sources</phase>
        <configuration>
         <className>org.apache.hello_world.Greeter</className>

          <genWsdl>true</genWsdl>
          <verbose>true</verbose>
        </configuration>
        <goals>
          <goal>java2ws</goal>
        </goals>
      </execution>
    </executions>
  </plugin>
  ...
</project>
```

ASM

The JAX-WS frontend now requires ASM 2.x or 3.x to be able to process some of the JAXB annotations on the SEI. There can be conflicts with other applications, such as Hibernate, that use asm.

If you don't use those annotations on the SEI or if you have generated wrapper classes, you can remove `asm jar` from your installation.

If you use the annotations, the workaround for Hibernate is to remove Hibernate's ASM 1.x jar used by it and replace Hibernate's `cglib jar` with the `cglib-nodeps jar` that includes a special internal version of ASM that would not conflict with the 2.x/3.x version used by FUSE Services Framework.

Migrating from FUSE ESB 3.4

JAX-RS 1.0

FUSE Services Framework supports the 1.0 release of the JAX-RS specification. Applications written using the JAX-RS APIs available previous releases will need to be updated to work with this release.

Policy engine

In 4.1 the policy engine is turned on by default. This means that WS-Policy assertions contained in WSDL documents are evaluated. If the runtime supports a given policy assertion, it will act accordingly.

To disable the policy engine you can add the configuration snipit in Example 5.2 on page 20 to your application's configuration.

Example 5.2. Turning Off the Policy Engine

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:cxf="http://cxf.apache.org/core"
       xmlns:p="http://cxf.apache.org/policy"
       xsi:schemaLocation="
http://cxf.apache.org/core http://cxf.apache.org/schemas/core.xsd
http://cxf.apache.org/policy http://cxf.apache.org/schemas/policy.xsd
http://www.springframework.org/schema/beans http://www.springframe
work.org/schema/beans/spring-beans-2.0.xsd">
  ...
  <cxf:bus>
    <cxf:features>
      <p:policies enabled="false" />
    </cxf:features>
  </cxf:bus>
  ...
</beans>
```

JMS

Starting in version 4.1, the JMS transport defaults to using the JMS 1.1 API. If you are using a JMS 1.0 provider you can change this behavior by setting the `useJms11` configuration attribute to `false`.

Maven wsdl2java plug-in

The wsdl2java plug-in now defaults to placing generated code into `${project.build.directory}/generated-sources/cxf`. This follows the best practices for using Maven.

You can change the default behavior using the `sourceRoot` variable.

Part II. Differences Between Version 4.0 and Version 4.1

Version 4.1 has a number of functional differences from version 4.0.

Chapter 6. FUSE Services Framework Issues

FUSE ESB 4.1 uses FUSE Services Framework 2.2. This introduces a few migration issues.

JAX-RS 1.0

FUSE Services Framework supports the 1.0 release of the JAX-RS specification. Applications written using the JAX-RS APIs available previous releases will need to be updated to work with this release.

Policy engine

In version 4.1 the policy engine is turned on by default. This means that WS-Policy assertions contained in WSDL documents are evaluated. If the runtime supports a given policy assertion, it will act accordingly.

To disable the policy engine you can add the configuration snipit in Example 6.1 on page 23 to your application's configuration.

Example 6.1. Turning Off the Policy Engine

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:cxf="http://cxf.apache.org/core"
       xmlns:p="http://cxf.apache.org/policy"
       xsi:schemaLocation="
http://cxf.apache.org/core http://cxf.apache.org/schemas/core.xsd
http://cxf.apache.org/policy http://cxf.apache.org/schemas/policy.xsd
http://www.springframework.org/schema/beans http://www.springframe
work.org/schema/beans/spring-beans-2.0.xsd">
  ...
  <cxf:bus>
    <cxf:features>
      <p:policies enabled="false" />
    </cxf:features>
  </cxf:bus>
```

```
  ...
</beans>
```

JMS

Starting in version 4.1, the JMS transport defaults to using the JMS 1.1 API. If you are using a JMS 1.0 provider you can change this behavior by setting the `useJms11` configuration attribute to `false`.

Maven wsdl2java plug-in

The wsdl2java plug-in now defaults to placing generated code into `${project.build.directory}/generated-sources/cxf`. This follows the best practices for using Maven.

You can change the default behavior using the `sourceRoot` variable.

Chapter 7. FUSE Mediation Router Issues

FUSE ESB 4.1 uses FUSE Mediation Router 1.6. This introduces a few migration issues.

JBI component

Stream caching is turned on by default in 4.1. It is no longer necessary to include the `streamCaching()` processor to routes that use JBI endpoints if the message body is read multiple times.

Jetty component

The `camel-jetty` producer, or `to` endpoint, is no longer provided. It should be replaced by a `camel-http` producer.

Tip

This API change also means that the dependency on the `jetty-client` module is removed.

JAXB data marshaling

When using the JAXB data format in conjunction with XJC to create Java classed from XML Schema, the run time will default to using the instance value provided in the XML Schema for `JAXBElements`. The actual value of the associated element in the unmarshaled message is ignored.

If you want to get the value of the element mapped to a `JAXBElement` object form the unmarshaled message body, you need to set the `JaxbDataFormat` object's ignoreJAXBElement property to `false`.

Example 7.1 on page 25 shows a route that uses the unmarshaled message values.

Example 7.1. Using Unmashaled Message Data

```
<camelContext id="camel" xmlns="ht
tp://camel.apache.org/schema/spring">
  <route>
    <from uri="direct:start"/>
    <marshal>
      <jaxb  ignoreJAXBElement="false" />
    </marshal>
    <to uri="mock:result"/>
```

```
  </route>
</camelContext>
```

Part III. Migrating Applications to FUSE ESB 4.1

Applications that rely on clustering need to be migrated to using the new clustering mechanism. Applications using FUSE Services Framework or FUSE Mediation Router should be updated to use the OSGi packaging and deployment features.

Chapter 8. JMS Flows and Clustering

In FUSE ESB 3.x clustering was implemented using the JMS flow. In FUSE ESB 4.1 clustering is implemented using a clustering engine that works in conjunction with the NMR. The clustering engine uses FUSE Message Broker and specifically configured JBI endpoints to build clusters.

Overview

The JMS flow in FUSE ESB 3.x allowed users to gain a level of persistence and failure tolerance out of their applications. The JMS flow used the internal FUSE Message Broker to manage temporary JMS destinations used for message exchanges. The JMS flow also made it possible to create a cluster of FUSE ESB containers over which you could distribute the endpoints used in an application.

While the JMS flow is not present in FUSE ESB 4.1, you can get some of the same benefits by using the included clustering engine. The clustering engine uses FUSE Message Broker, or any other JMS broker, to make it possible specify which endpoints in a JBI application participate in a cluster.

Setting up the clustering engine

FUSE ESB 4.1 has the clustering engine pre-installed and configured to use the included FUSE Message Broker. The default configuration, shown in Example 8.1 on page 29, should meet most basic requirements.

Example 8.1. Default Cluster Engine Configuration

```
<bean id="clusterEngine" class="org.apache.servicemix.jbi.cluster.engine.ClusterEngine">
  <property name="pool">
    <bean class="org.apache.servicemix.jbi.cluster.requestor.ActiveMQJmsRequestorPool">
      <property name="connectionFactory" ref="connectionFactory" />
      <property name="destinationName" value="${destinationName}" />
    </bean>
  </property>
  <property name="name" value="${clusterName}" />
</bean>

<osgi:list id="clusterRegistrations"
           interface="org.apache.servicemix.jbi.cluster.engine.ClusterRegistration"
           cardinality="0..N">
  <osgi:listener ref="clusterEngine" bind-method="register" unbind-method="unregister" />
</osgi:list>

<osgi:reference id="connectionFactory" interface="javax.jms.ConnectionFactory" />

<osgi:service ref="clusterEngine">
  <osgi:interfaces>
```

```
    <value>org.apache.servicemix.nmr.api.Endpoint</value>
    <value>org.apache.servicemix.nmr.api.event.Listener</value>
    <value>org.apache.servicemix.nmr.api.event.EndpointListener</value>
    <value>org.apache.servicemix.nmr.api.event.ExchangeListener</value>
  </osgi:interfaces>
  <osgi:service-properties>
    <entry key="NAME" value="${clusterName}" />
  </osgi:service-properties>
</osgi:service>

<osgix:cm-properties id="clusterProps" persistent-id="org.apache.servicemix.jbi.cluster.con
fig">
  <prop key="clusterName">${servicemix.name}</prop>
  <prop key="destinationName">org.apache.servicemix.jbi.cluster</prop>
</osgix:cm-properties>

<ctx:property-placeholder properties-ref="clusterProps" />
```

FUSE ESB has a preconfigured FUSE Message Broker instance that automatically starts when the container is started, so you should not need to start a broker instance for the clustering engine to work.

Note

You can configure the clustering engine to use a different JMS broker by changing the configuration to use `org.apache.servicemix.jbi.cluster.requestor.GenericJmsRequestorPool` instead of `org.apache.servicemix.jbi.cluster.requestor.ActiveMQJmsRequestorPool`.

Using in an application

When using an OSGi packaged JBI service assembly, you can include the clustered endpoints definitions directly in the Spring configuration. In addition to the endpoint definition, you need to add a bean that registers the endpoint with the clustering engine.

Example 8.2 on page 30 shows an HTTP consumer endpoint that is part of a cluster.

Example 8.2. OSGi Packaged JBI Endpoint

```
<http:consumer id="myHttpConsumer" service="test:myService" endpoint="myEndpoint" />

<bean class="org.apache.servicemix.jbi.cluster.engine.OsgiSimpleClusterRegistration">
  <property name="endpoint" ref="myHttpConsumer" />
</bean>
```

When using a JBI packaged service assembly, you need to create a Spring application to register the endpoint as a clustered endpoint. This configuration is a little more involved and requires that you provide more information about the endpoint.

Example 8.3 on page 31 shows an HTTP consumer endpoint that is part of a cluster.

Example 8.3. OSGi Packaged JBI Endpoint

```
<http:consumer id="myHttpConsumer" service="test:myService" endpoint="myEndpoint" />

<bean class="org.apache.servicemix.jbi.cluster.engine.OsgiSimpleClusterRegistration">
  <property name="serviceName" value="test:myService" />
  <property name="endpointName" value="myEndpoint" />
</bean>
```

Chapter 9. Migrating JAX-WS Services to Native FUSE ESB Applications

FUSE ESB 4.1 makes it possible to deploy JAX-WS services as native applications without wrapping them as JBI endpoints. Services developed this way have direct access to the containers transport layer.

Overview

Migrating a JBI based JAX-WS service to a native application is straight-forward. The code and WSDL from the JBI implementation do not need to change. You will, however, need to modify the following:

- the project's file structure
- the project's POM
- the service's Spring configuration

The project structure

JBI projects for a JAX-WS service contain at least three subprojects:

- the service engine service unit
- the binding component service unit
- the service assembly

JAX-WS services using the native FUSE Services Frameworkintegration are built as a single project. Your project folder only needs a top-level POM and a `src` folder.

The source code from the JBI project's service engine project should be moved under the `src` folder. In addition, the WSDL from the JBI project should be moved to `src/main/resources/`.

The POM

You cannot migrate your POM file from version 3.x to version 4.1. You will need to do the following:

1. Create a new top-level POM that includes all of the dependencies for a FUSE ESB project.
2. Set the `packaging` element to `bundle`.

3. If your JBI project's service engine used the FUSE Services Framework code generation plug-ins, you will need to move them to your new top-level POM.

   The entry for the FUSE Services Framework code generation plug-ins will be similar to the entry shown in Example 9.1 on page 34.

Example 9.1. FUSE Services Framework Code Generation Plug-ins

```
<plugin>
  <groupId>org.apache.cxf</groupId>
    <artifactId>cxf-codegen-plugin</artifactId>
    <version>${cxf.version}</version>
    <executions>
      <execution>
        <phase>generate-sources</phase>
        <configuration>
          ...
        </configuration>
        <goals>
          ...
        </goals>
      </execution>
    </executions>
</plugin>
```

4. Move the compilation and JAX-WS dependencies to your new top-level POM.

5. Add the OSGi bundle plug-in to you new top-level POM.

   Example 9.2 on page 34 shows the entry for the OSGi bundle plug-in.

Example 9.2. OSGi Bundle Plug-in Configuration

```
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <configuration>
    <instructions>
      <Bundle-SymbolicName>bundleId</Bundle-SymbolicName>
      <Import-Package>importList</Import-Package>
      <Private-Package>currentPackage</Private-Package>
      <Require-Bundle>org.apache.cxf.cxf-bundle</Require-Bundle>
    </instructions>
  </configuration>
</plugin>
```

- *bundleId* is the identifier used by the container for the bundle.
- *importList* is a comma-seperated list of packages the application needs to import from other bundles.

6. Add the required bundle dependencies to the OSGi bundle plug-in.

   Example 9.3 on page 35 shows the minimum list of packages that a native FUSE Services Framework application needs.

Example 9.3. OSGi Bundle Plug-in Configuration

```
javax.jws,
javax.wsdl,
META-INF.cxf,
META-INF.cxf.osgi,
org.apache.cxf.bus,
org.apache.cxf.bus.spring,
org.apache.cxf.bus.resource,
org.apache.cxf.configuration.spring,
org.apache.cxf.resource,
org.springframework.beans.factory.config
```

The configuration

FUSE Services Framework JBI projects have their configuration spread over two service units' `xbean.xml` files. The service implementation's configuration is specified in the service engine's service unit using an `cxfse:endpoint` element. The transport's configuration is specified in the binding component's service unit using an `cxfbc:consumer` element.

Native FUSE Services Framework projects have their configuration located in a single Spring configuration file. This file is located in the `src/main/resources/META_INF/spring`. The FUSE Services Framework service is configured using the standard FUSE Services Framework configuration elements.

Example 9.4 on page 35 shows a sample configuration for a FUSE Services Framework service.

Example 9.4. FUSE Services Framework Spring Configuration

```
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:jaxws="http://cxf.apache.org/jaxws"
```

```
 xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframe
work.org/schema/beans/spring-beans.xsd
http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd">

    <import resource="classpath:META-INF/cxf/cxf.xml" />
    <import resource="classpath:META-INF/cxf/cxf-extension-soap.xml" />
    <import resource="classpath:META-INF/cxf/cxf-extension-http.xml" />
    <import resource="classpath:META-INF/cxf/osgi/cxf-extension-osgi.xml" />

 <jaxws:endpoint id="PersonService"
                 implementor="org.apache.servicemix.samples.wsdl_first.PersonImpl"
                 address="/PersonService"
                 wsdlLocation="classpath:person.wsdl"/>

</beans>
```

More Information

For more information on using the native JAX-WS integration see *Developing and Deploying JAX-WS Services*.

Chapter 10. Migrating FUSE Mediation Router Applications to OSGi

In FUSE ESB 4.1 the recommended method for deploying FUSE Mediation Router applications is as OSGi bundles.

Overview

Migrating a JBI based FUSE Mediation Router application to OSGi is straight-forward. The code the JBI implementation does not need to change. You will, however, need to modify the following:

- the project's file structure
- the project's POM
- the Spring configuration

The project structure

JBI projects for a FUSE Mediation Router contain at least two subprojects:

- the service unit
- the service assembly

OSGi FUSE Mediation Router applications are built as a single project. Your project folder only needs a top-level POM and a `src` folder.

The source code from the JBI project's service engine unit should be moved under the `src` folder. Any Sping configuration and the `camel-context.xml` file should be moved to the `src/main/resources/META-INF/spring` folder.

The POM

You cannot migrate your POM file from version 3.x to version 4.1. You will need to do the following:

1. Create a new top-level POM that includes all of the dependencies for a FUSE ESB project.
2. Set the `packaging` element to `bundle`.
3. Add the OSGi bundle plug-in to you new top-level POM.

   Example 10.1 on page 38 shows the entry for the OSGi bundle plug-in.

Example 10.1. OSGi Bundle Plug-in Configuration

```
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <configuration>
    <instructions>
      <Bundle-SymbolicName>bundleId</Bundle-SymbolicName>
      <Import-Package>importList</Import-Package>
      <Private-Package>currentPackage</Private-Package>
    </instructions>
  </configuration>
</plugin>
```

- *bundleId* is the identifier used by the container for the bundle.
- *importList* is a comma-seperated list of packages the application needs to import from other bundles.

4. Add `org.apache.camel.osgi` to the `Import-Package` element of the bundle plug-in's configuration.

The configuration

FUSE Mediation Router JBI projects use a file called `camel-context.xml` to configure the routes. FUSE Services Framework OSGi projects configure routes in a configuration file located in the `src/main/resources/META_INF/spring`. This file, typically called `beans.xml`, is very similar to the `camel-context.xml` file used in the JBI project. The difference is that the `camelContext` element used in the OSGi project is in the http://activemq.apache.org/camel/schema/osgi namespace.

Example 10.2 on page 38 shows a sample configuration.

Example 10.2. FUSE Services Framework Spring Configuration

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:osgi="http://activemq.apache.org/camel/schema/osgi"
       xsi:schemaLocation="
       http://www.springframework.org/schema/beans
         http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
       http://activemq.apache.org/camel/schema/spring
         http://activemq.apache.org/camel/schema/spring/camel-spring.xsd
       http://activemq.apache.org/camel/schema/osgi
         http://activemq.apache.org/camel/schema/osgi/camel-osgi.xsd">
```

```
  <osgi:camelContext xmlns="http://activemq.apache.org/camel/schema/spring">
    <route>
      <from uri="timer://myTimer?fixedRate=true&period=2000"/>
      <bean ref="myTransform" method="transform"/>
      <to uri="log:ExampleRouter"/>
    </route>
  </osgi:camelContext>

  <bean id="myTransform" class="org.apache.servicemix.examples.camel.MyTransform">
    <property name="prefix" value="${prefix}"/>
  </bean>

</beans>
```

More Information

For more information on using FUSE Mediation Router see *Enterprise Integration Patterns*.