Progress FUSE

FUSE Services Framework

J2EE Integrations

Version 2.2.x
April 2009

PROGRESS SOFTWARE

J2EE Integrations

Version 2.2.x

Publication date 17 Jul 2009

Legal Notices

Table of Contents

List of Figures

List of Tables

List of Examples

Chapter 1. Introduction

Using the FUSE Services Framework JCA Connector, developers can easily connect their J2EE applications to Web services and expose their J2EE applications as Web services from within their chosen J2EE application server.

J2EE Connector Architecture Overview

Overview

The J2EE Connector Architecture (JCA) outlines a standard architecture for enabling J2EE applications to access resources in diverse Enterprise Information Systems (EISs). The goal is to standardize access to non-relational resources in the same way the JDBC API standardizes access to relational data.

The J2EE Connector Architecture is implemented in a J2EE application server and an EIS-specific resource adapter. The resource adapter plugs into the J2EE application server and provides a system library specific to, and connectivity to, that EIS.

The FUSE Services Framework JCA Connector is a JCA 1.5 resource adapter.

More information

For more information on the J2EE Connector Architecture, see the JCA 1.5 Specification[1].

[1] http://java.sun.com/j2ee/connector/download.html

FUSE Services Framework JCA Connector Overview

Overview

The FUSE Services Framework JCA Connector is a J2EE Connector Architecture 1.5 resource adapter. It enables you to expose Web services to your J2EE applications and allows you to expose your J2EE applications as Web services.

The term Web services is used here to include SOAP over HTTP based services and any service that has been exposed as a Web service by FUSE Services Framework. The FUSE Services Framework JCA Connector transparently connects your J2EE applications over multiple transports to any FUSE Services Framework-enabled back-end service.

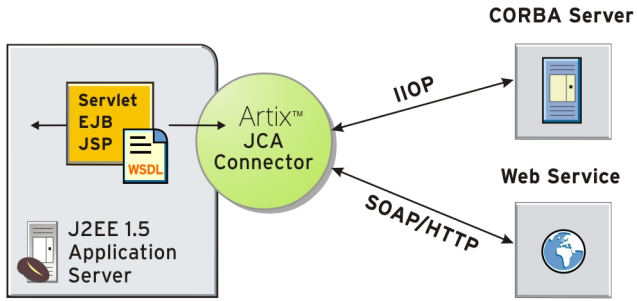
Note

To use the FUSE Services Framework JCA Connector your application server must support JCA 1.5 and EJB 2.1 or higher.

Graphical representation

Figure 1.1 on page 13 illustrates, at a high-level, how the FUSE Services Framework JCA Connector exposes a Web service to a J2EE application. It acts as a bridge between J2EE and SOAP over HTTP Web services. This is the simplest example. It also illustrates that the FUSE Services Framework JCA Connector can be used as a bridge between J2EE and a CORBA server that has been exposed as a Web service by FUSE Services Framework.

Figure 1.1. Connecting J2EE Applications to Web services



The FUSE Services Framework JCA Connector also enables inbound connections, allowing you to expose your J2EE application as a Web service.

FUSE Services Framework JCA Connector RAR file

The FUSE Services Framework JCA Connector resource adapter is packaged as a standard J2EE Connector Architecture resource adapter archive (RAR) file and is called `cxf.rar`. The `cxf.rar` file contains all of the classes that the FUSE Services Framework JCA Connector needs to manage both inbound and outbound connections.

FUSE Services Framework JCA Connector deployment descriptor

The FUSE Services Framework JCA Connector deployment descriptor file, `ra.xml`, contains information about FUSE Services Framework JCA Connector's resource implementation, configuration properties, transaction and security support. It describes the capabilities of the resource adapter and provides a deployer with enough information to properly configure the resource adapter in an application server environment.

An application server relies on the information in the deployment descriptor to know how to interact properly with the resource adapter. The deployment descriptor is packaged in the FUSE Services Framework JCA Connector RAR file.

Connection management

For information on how to use the FUSE Services Framework JCA Connector to manage inbound connections, see "Exposing a J2EE application as a Web Service" on page 15.

For information on how to use the FUSE Services Framework JCA Connector to manage outbound connections, see "Exposing a Web Service to a J2EE Application" on page 39.

Chapter 2. Exposing a J2EE application as a Web Service

This chapter describes how to use the FUSE Services Framework JCA Connector for inbound connections.

Introduction

Overview

The FUSE Services Framework JCA Connector's inbound support makes use of the JCA 1.5 specification's message inflow contract and EJB 2.1 or higher message-driven beans (MDBs). The JCA 1.5 specification defines a framework that allows the FUSE Services Framework JCA Connector to be notified when a MDB starts. The FUSE Services Framework JCA Connector then activates the FUSE Services Framework service endpoint facade, which receives client requests and invokes on the MDB's listener interface.

The instructions in this chapter assume that you are familiar with writing EJBs, including Message Driven Beans and Stateless Session Beans.

Note

To use the FUSE Services Framework JCA Connector your application server must support JCA 1.5 and EJB 2.1 or higher; for example, WebSphere 6.1.

More information

For more information about the JCA 1.5 message inflow contract, see *Chapter 12, Message Inflow* of the JCA 1.5 Specification[1].

In addition, if you are interested in knowing more about what goes on behind the scenes when a resource adapter, such as the FUSE Services Framework JCA Connector, invokes an application asynchronously through a MDB, see JCA 1.5, Part 3:Message Inflow[2].

Usage scenarios

You can use the FUSE Services Framework JCA Connector to expose your J2EE application as a Web service using any of the following scenarios:

- Java first, where you implement your service as one of the following:

  a. Message Driven Bean (MDB). In this case, incoming requests do not need to be dispatched to another EJB; the MDB includes the service implementation.

  See "Service Implemented as a Message Driven Bean" on page 18 for more details.

[1] http://java.sun.com/j2ee/connector/download.html
[2] http://www.ibm.com/developerworks/java/library/j-jca3/

b. Stateless Session Bean (SLSB). In this case, you use an FUSE Services Framework-provided generic MDB to dispatch incoming requests to your SLSB.

   See "Service Implemented as a Stateless Session Bean" on page 24 for more details.

- WSDL first, where your starting point is the service WSDL file. You use FUSE Services Framework to generate JAX-WS compliant Java code from the WSDL file and implement your service as a SLSB. Here, again, you use the FUSE Services Framework-provided generic MDB to dispatch incoming requests to your SLSB.

  See "WSDL First—Service Implemented as a SLSB" on page 29 for more details.

The rest of this chapter describes these scenarios in more detail.

Service Implemented as a Message Driven Bean

Overview

In this scenario you implement your service as a MDB. When it starts, the MDB notifies the FUSE Services Framework JCA Connector. The FUSE Services Framework JCA Connector activates the FUSE Services Framework service endpoint facade, which receives client requests and invokes directly on the MDB. Incoming invocations do not have to be dispatched to another EJB.

In addition, there is no need for a service WSDL file. FUSE Services Framework uses the service endpoint interface to build a service model as it is defined in the activation specification `serviceInterfaceClass` property in your application's deployment descriptor file, `ejb-jar.xml`.

Advantages

The advantages of using this approach is that it preforms faster than either of the SLSB scenarios because the MDB does not need to dispatch incoming requests to another EJB.

In addition, you do not need to implement EJB Home, Local or Remote interfaces.

Disadvantages

The disadvantage of this approach is that the service endpoint interface has to be exposed as the `messagelistener-type` element in the FUSE Services Framework JCA Connector's deployment descriptor. This means that you must edit the FUSE Services Framework JCA Connector's deployment descriptor file.

Sample application

FUSE Services Framework includes a working example of this scenario. It is located in the `samples/integration/jca/inbound-mdb` directory of your FUSE Services Framework installation.

If you want to build and run this sample, please follow the instructions outlined in the `README.txt` file located in this directory. The example code shown in this section is taken from this sample application.

High-level Implementation Steps

Complete the following steps to expose your J2EE application, implemented as a MDB, as a Web service using the FUSE Services Framework JCA Connector:

1. Write a MDB that implements the service that you want to expose. See, for instance, `GreeterBean.java` located in

InstallDir/samples/integration/jca/inbound-mdb/src/demo/ejb and shown in Example 2.1 on page 19.

Example 2.1. Message Driven Bean—GreeterBean.java

```
package demo.ejb;

import javax.ejb.MessageDrivenBean;
import javax.ejb.MessageDrivenContext;

import org.apache.hello_world_soap_http.Greeter;

public class GreeterBean implements MessageDrivenBean, Greeter {

      public String sayHi() {
           System.out.println("sayHi called ");
           return "Hi there!";
      }

      public String greetMe(String user) {
           System.out.println("greetMe called user = " + user);
           return "Hello " + user;
      }

      //---------------- EJB Methods
      public void ejbCreate() {
       }

      public void ejbRemove() {
      }

      public void setMessageDrivenContext(MessageDrivenContext mdc) {
      }
}
```

2. Write a deployment descriptor for your MDB. See, for instance, the `ejb-jar.xml` file located in *InstallDir*/samples/integration/jca/inbound-mdb/etc and shown in Example 2.2 on page 19.

Example 2.2. Message Driven Bean Deployment Descriptor—ejb-jar.xml

```
<?xml version="1.0"?>
...
<ejb-jar xmlns="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
```

```
http://java.sun.com/xml/ns/j2ee/ejb-jar_2_1.xsd"
version="2.1">

<enterprise-beans>
   <message-driven>
      <ejb-name>Greeter MDB</ejb-name>
      <ejb-class>demo.ejb.GreeterBean</ejb-class>
      <messaging-type>
         org.apache.hello_world_soap_http.Greeter
      </messaging-type>
      <transaction-type>Bean</transaction-type>

      <activation-config>
         <!-- displayName -->
         <activation-config-property>
            <activation-config-property-name>
               displayName
            </activation-config-property-name>
            <activation-config-property-value>
               MyCxfEndpoint
            </activation-config-property-value>
          </activation-config-property>

          <!-- service endpoint interface -->
          <activation-config-property>
             <activation-config-property-name>
                serviceInterfaceClass
             </activation-config-property-name>
             <activation-config-property-value>
                org.apache.hello_world_soap_http.Greeter
             </activation-config-property-value>
          </activation-config-property>

          <!-- address -->
          <activation-config-property>
             <activation-config-property-name>
                address
             </activation-config-property-name>
             <activation-config-property-value>
                http://localhost:9999/GreeterBean
             </activation-config-property-value>
           </activation-config-property>

         </activation-config>
      </message-driven>
   </enterprise-beans>

        <assembly-descriptor>
        <method-permission>
```

```
            <unchecked/>
            <method>
               <ejb-name>GreeterBean</ejb-name>
               <method-name>*</method-name>
            </method>
         </method-permission>
          <container-transaction>
              <description/>
              <method>
                 <description/>
                 <ejb-name>GreeterBean</ejb-name>
                 <method-name>*</method-name>
              </method>
              <trans-attribute>Supports</trans-attribute>
           </container-transaction>
       </assembly-descriptor>
</ejb-jar>
```

For more information about the supported activation configuration properties, see "Inbound Activation Configuration" on page 55.

If you are using EJB 3.0, the only change you need to make to the deployment descriptor is in the opening `ejb-jar` element. For EJB 3.0 it should read as shown in Example 2.3 on page 21.

Example 2.3. EJB 3.0 Deployment Descriptor

```
<ejb-jar xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee

     http://java.sun.com/xml/ns/javaee/ejb-jar_3_0.xsd"
     version="3.0">
```

3. Package you application in an EJB JAR file.
4. Make a copy of the FUSE Services Framework JCA Connector's deployment descriptor file, `ra.xml`, which is located in the `samples/integration/jca/websphere/inbound-mdb/etc` directory of your installation.
5. Edit the `ra.xml` file so that the `messagelistener-type` element defines the same interface as the `messaging-type` element defined in your MDB deployment descriptor. This ensures that the FUSE Services Framework JCA Connector is notified when the MDB starts.

6. Build the FUSE Services Framework JCA Connector RAR file. It must have the following structure and contents:

Table 2.1. RAR File Structure and Contents: Service Implemented as MDB

Directory	Contents
`META-INF`	The `ra.xml` file that you modified.
`Root`	• All of the JARs in the `modules` directory of the FUSE Services Framework installation, except the `*manifest*.jar` files. • The `cxf-integration-jca-*.jar`s from the `modules/integration` directory of the FUSE Services Framework installation. • All of the JARs in the `lib` directory of the FUSE Services Framework installation, except the `cxf*.jar` files.

The sample application's `build.xml` file includes a **generate.rar** target that builds the RAR file (see Example 2.4 on page 22).

Example 2.4. generate.rar Target

```
<target name="generate.rar" depends="init">
    <copy file="${basedir}/etc/ra.xml" todir="${build.classes.dir}/cxf-rar/META-INF"/>
    <copy todir="${build.classes.dir}/cxf-rar">
        <fileset dir="${cxf.home}/lib">
            <include name="*.jar"/>
            <exclude name="cxf*.jar"/>
         </fileset>
         <fileset dir="${cxf.home}/modules">
           <include name="*.jar"/>
           <exclude name="*manifest*.jar"/>
         </fileset>
         <fileset dir="${cxf.home}/modules/integration">
             <include name="*.jar"/>
             <exclude name="*-jbi-*.jar" />
         </fileset>
      </copy>
      <jar destfile="${build.classes.dir}/lib/cxf.rar"
           basedir="${build.classes.dir}/cxf-rar"/>
</target>
```

The `cxf.home` variable must be set to the *`InstallDir`* directory. This is done for you when you set your environment (see "Setting Up Your Environment" in *Configuring and Deploying Endpoints*.

7. Deploy the FUSE Services Framework JCA Connector RAR file and your EJB JAR file to your J2EE application server. For details, see "Deploying FUSE Services Framework JCA Connector" on page 49.

Service Implemented as a Stateless Session Bean

Overview

In this scenario you implement your service as a Stateless Session Bean (SLSB). FUSE Services Framework provides a generic MDB implementation that notifies the FUSE Services Framework JCA Connector when it starts. The FUSE Services Framework JCA Connector then activates the FUSE Services Framework service endpoint facade, which dispatches client requests to the generic MDB. The MDB dispatches incoming requests to your SLSB, using the SLSB's EJB local reference (as implemented in its Local Home interface).

Advantages

The advantage of this approach is that you do not have to edit the FUSE Services Framework JCA Connector deployment descriptor.

In addition, there is no need for a service WSDL file. FUSE Services Framework uses the service endpoint interface to build a service model as it is defined in the activation specification `serviceInterfaceClass` property in your application's deployment descriptor file, `ejb-jar.xml`.

Disadvantages

The disadvantage of this approach is that it may not preform as fast as the approach described in "Service Implemented as a Message Driven Bean" on page 18.

Sample application

FUSE Services Framework includes a working example of this scenario. You can find it in the `samples/integration/jca/inbound-mdb-dispatch` directory of your FUSE Services Framework installation.

If you want to build and run this sample, please follow the instructions outlined in the `README.txt` file located in this directory. The example code shown in this section is taken from this sample application.

High-level Implementation Steps

Complete the following steps to expose your J2EE application, implemented as a SLSB, as a Web service using the FUSE Services Framework JCA Connector:

1. Write a SLSB that implements the service that you want to expose. See, for instance, `GreeterBean.java` located in

   InstallDir`/samples/integration/jca/inbound-mdb-dispatch/src/demo/ejb`

   and shown in Example 2.5 on page 25.

Example 2.5. Stateless Session Bean—GreeterBean.java

```
package demo.ejb;

import javax.ejb.CreateException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

public class GreeterBean implements SessionBean {

      //------------- Business Methods
      public String sayHi() {
            System.out.println("sayHi invoked");
            return "Hi from an EJB";
      }

      public String greetMe(String user) {
            System.out.println("greetMe invoked user:" + user);
            return "Hi " + user + " from an EJB";
      }

      //------------- EJB Methods
      public void ejbActivate() {
      }

      public void ejbRemove() {
      }

      public void ejbPassivate() {
      }

      public void ejbCreate() throws CreateException {
      }

      public void setSessionContext(SessionContext con) {
      }
}
```

2. Write an EJB Local Home interface for your SLSB. See, for instance, `GreeterLocalHome.java` located in

   InstallDir`/samples/integration/jca/inbound-mdb-dispatch/src/demo/ejb`

   and shown in Example 2.6 on page 25.

   Example 2.6. GreeterLocalHome.java

   ```
   package demo.ejb;
   ```

```
import javax.ejb.CreateException;
import javax.ejb.EJBLocalHome;

public interface GreeterLocalHome extends EJBLocalHome {
     GreeterLocal create() throws CreateException;
}
```

3. Write a deployment descriptor for your SLSB and ensure that it includes:

   a. A `message-driven` element under the `enterprise-beans` element that references to the generic MDB as follows:

      - The value of `ejb-class` is

        `org.apache.cxf.jca.inbound.DispatchMDBMessageListenerImpl`

      - The value of `messaging-type` is

        `org.apache.cxf.jca.inbound.DispatchMDBMessageListener`

   b. An `ejb-local-ref` element, which is required by the MDB so it can look up the local EJB object reference for your SLSB.

   See, for instance, the `ejb-jar.xml` located in *`InstallDir`*`/samples/integration/jca/inbound-mdb-dispatch/etc` and shown in Example 2.7 on page 26.

Example 2.7. Stateless Session Bean Deployment Descriptor—ejb-jar.xml

```
<?xml version="1.0"?>
                          ...
<ejb-jar xmlns="http://java.sun.com/xml/ns/j2ee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
       http://java.sun.com/xml/ns/j2ee/ejb-jar_2_1.xsd"
       version="2.1">

   <enterprise-beans>
      <session>
         <ejb-name>DispatchedGreeterBean</ejb-name>
         <home>demo.ejb.GreeterHome</home>
         <remote>demo.ejb.GreeterRemote</remote>
         <local-home>demo.ejb.GreeterLocalHome</local-home>
         <local>demo.ejb.GreeterLocal</local>
         <ejb-class>demo.ejb.GreeterBean</ejb-class>
         <session-type>Stateless</session-type>
         <transaction-type>Container</transaction-type>
```

```
      </session>

      <message-driven>
        <ejb-name>GreeterEndpointActivator</ejb-name>
        <ejb-class>org.apache.cxf.jca.inbound.DispatchMDBMessageListenerImpl</ejb-class>
        <messaging-type>org.apache.cxf.jca.inbound.DispatchMDBMessageListener
        </messaging-type>
        <transaction-type>Bean</transaction-type>

        <activation-config>
             <!-- display name-->
             <activation-config-property>
                 <activation-config-property-name>
                    DisplayName
                 </activation-config-property-name>
                 <activation-config-property-value>
                    DispatchedGreeterEndpoint
                 </activation-config-property-value>
             </activation-config-property>
             <!-- service endpoint interface -->
             <activation-config-property>
                 <activation-config-property-name>
                    serviceInterfaceClass
                 </activation-config-property-name>
                 <activation-config-property-value>
                    org.apache.hello_world_soap_http.Greeter
                 </activation-config-property-value>
             </activation-config-property>
             <!-- address -->
             <activation-config-property>
                 <activation-config-property-name>
                    address
                 </activation-config-property-name>
                 <activation-config-property-value>
                    http://localhost:9999/GreeterBean
                 </activation-config-property-value>
             </activation-config-property>
             <!-- targetBeanJndiName -->
             <activation-config-property>
                 <activation-config-property-name>
                     targetBeanJndiName
                </activation-config-property-name>
                <activation-config-property-value>
                    java:comp/env/DispatchedGreeterLocalHome
                </activation-config-property-value>
             </activation-config-property>
         </activation-config>

         <ejb-local-ref>
```

```
            <ejb-ref-name>DispatchedGreeterLocalHome</ejb-ref-name>
            <ejb-ref-type>Session</ejb-ref-type>
            <local-home>demo.ejb.GreeterLocalHome</local-home>
            <local>demo.ejb.GreeterLocal</local>
            <ejb-link>DispatchedGreeterBean</ejb-link>
        </ejb-local-ref>
      </message-driven>

    </enterprise-beans>
</ejb-jar>
```

For more information about the supported activation configuration properties, see "Inbound Activation Configuration" on page 55.

If you are using EJB 3.0, the only change you need to make to the deployment descriptor is in the opening `ejb-jar` element. For EJB 3.0 it should read as shown in Example 2.3 on page 21.

4. Package your application in an EJB JAR file.

5. Build the FUSE Services Framework JCA Connector RAR file. It must have the following structure and contents:

   1. `META-INF` directory — Must contain the `ra.xml`, located in *`InstallDir`*`/samples/integration/jca/inbound-mdb-dispatch/etc`.

   2. `Root` directory — Must contain the JAR files listed under `Root` in Table 2.1 on page 22.

   The sample application's `build.xml` file includes a **generate.rar** target that you can use to build the RAR file (see Example 2.4 on page 22).

   Note that the `ra.xml` file `activation spec` is set to `org.apache.cxf.jca.inbound.DispatchMDBActivationSpec`, which includes a `targetBeanJndiName` configuration property that enables you to specify your SLSB's JNDI name.

6. Deploy the FUSE Services Framework JCA Connector RAR file and your EJB JAR file to your J2EE application server. For details, see "Deploying FUSE Services Framework JCA Connector" on page 49.

WSDL First—Service Implemented as a SLSB

Overview

In this scenario your service is defined in a WSDL file. You use the **wsdl2java** tool to generate starting point JAX-WS compliant Java code from which you implement your service as a Stateless Session Bean (SLSB).

It is similar to the scenario described in "Service Implemented as a Stateless Session Bean" on page 24. Again you make use of the generic MDB implementation provided by FUSE Services Framework. It notifies the FUSE Services Framework JCA Connector when it starts and the FUSE Services Framework JCA Connector then activates the FUSE Services Framework service endpoint facade. The service endpoint facade dispatches client requests to the generic MDB. The MDB performs a JNDI lookup to obtain a reference to your SLSB and dispatches incoming requests to it.

Differences between the WSDL-first SSLB and Java-first SLSB

The primary differences between this approach and the approach described in "Service Implemented as a Stateless Session Bean" on page 24 is that:

- You can configure the FUSE Services Framework runtime directly by including a `cxf.xml` configuration file in your EJB JAR file.
- FUSE Services Framework creates a service bean based on the service WSDL file and you must include the WSDL file in the EJB JAR file.
- Your EJB deployment descriptor must contain additional activation configuration properties, including:
  - busConfigLocation — Specifies the location of the configuration file.
  - wsdlLocation — Specifies the location of the service's WSDL file.
  - endpointName — Specifies the service's `portType` element's QName.
  - serviceName — Specifies the service's `service` element's QName.

For more information on activation configuration properties, see "Inbound Activation Configuration" on page 55.

Advantages

One advantage of using this approach is the ability to directly configure the FUSE Services Framework runtime.

Sample application

FUSE Services Framework includes a working example of this scenario. You can find it in the *InstallDir*/samples/integration/jca/inbound-mdb-dispatch-wsdl directory of your FUSE Services Framework installation.

If you want to build and run this sample, please follow the instructions outlined in the README.txt file located in this directory. The example code shown in this section is taken from this sample application.

Implementation steps

Complete the following steps if you want to use the FUSE Services Framework JCA Connector to expose your J2EE application, defined in a WSDL file and implemented as a SLSB, as a Web service:

1. Set your environment using the **fuse_env** script, which is located in the *InstallDir*/bin directory.

   For more information on the **fuse_env** script, see "Setting Up Your Environment" in *Configuring and Deploying Endpoints*.

2. Obtain a copy of, or details of the location of, the WSDL file that defines the Web service that your application implements.

   This step assumes that the WSDL file already exists. If you need to develop a WSDL file, see *Writing WSDL Contracts*.

3. Map the WSDL file to Java to obtain starting point JAX-WS compliant Java code. FUSE Services Framework provides a **wsdl2java** tool that does this for you.

   For more information on the **wsdl2java** tool, see wsdl2java in *Tool Reference*.

4. Write a stateless session bean (SLSB) that implements the service that you want to expose. See, for instance, GreeterBean.java located in

InstallDir/samples/integration/jca/inbound-mdb-dispatch-wsdl/src/demo/ejb and shown in Example 2.8 on page 31.

Example 2.8. WSDL First SLSB—GreeterBean.java

```
package demo.ejb;

import java.util.logging.Logger;
import javax.ejb.CreateException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

import org.apache.hello_world_soap_http.Greeter;
import org.apache.hello_world_soap_http.PingMeFault;
import org.apache.hello_world_soap_http.types.FaultDetail;

public class GreeterBean implements SessionBean, Greeter {

      private static final Logger LOG =
            Logger.getLogger(GreeterBean.class.getPackage().getName());

       //------------- Business Methods
       // (copied from wsdl_first sample)

       public String greetMe(String me) {
             LOG.info("Executing operation greetMe");
             System.out.println("Executing operation greetMe");
             System.out.println("Message received: " + me + "\n");
             return "Hello " + me;
       }

       public void greetMeOneWay(String me) {
             LOG.info("Executing operation greetMeOneWay");
             System.out.println("Executing operation greetMeOneWay\n");
             System.out.println("Hello there " + me);
       }

       public String sayHi() {
             LOG.info("Executing operation sayHi");
             System.out.println("Executing operation sayHi\n");
             return "Bonjour";
       }

       public void pingMe() throws PingMeFault {
             FaultDetail faultDetail = new FaultDetail();
             faultDetail.setMajor((short)2);
             faultDetail.setMinor((short)1);
             LOG.info("Executing operation pingMe, throwing PingMeFault exception");
             System.out.println("Executing operation pingMe, throwing PingMeFault
```

```
            exception\n");
            throw new PingMeFault("PingMeFault raised by server", faultDetail);
        }

        //------------- EJB Methods
        public void ejbActivate() {
        }

        public void ejbRemove() {
        }

        public void ejbPassivate() {
        }

        public void ejbCreate() throws CreateException {
        }

        public void setSessionContext(SessionContext con) {
        }

}
```

5. Write an EJB Local Home interface for your SLSB. See, for instance, `GreeterLocalHome.java` located in

   InstallDir/samples/integration/jca/inbound-mdb-dispatch-wsdl/src/demo/ejb

   and shown in Example 2.9 on page 32.

   Example 2.9. WSDL First—GreeterLocalHome.java

   ```
   package demo.ejb;

   import javax.ejb.CreateException;
   import javax.ejb.EJBLocalHome;

   public interface GreeterLocalHome extends EJBLocalHome {
   GreeterLocal create() throws CreateException;
   }
   ```

6. Write a FUSE Services Framework configuration file if you want to configure the runtime directly. See, for instance, the `cxf.xml` configuration file located in

   InstallDir/samples/integration/jca/inbound-mdb-dispatch-wsdl/etc

   and shown in Example 2.10 on page 33. It shows how you configure logging.

For more information on how to configure FUSE Services Framework, see *Configuring and Deploying Endpoints*.

For information on how to configure FUSE Services Framework security, see *Security Guide*.

Example 2.10. cxf.xml—Configuring Logging

```
<beans xmlns="http://www.springframework.org/schema/beans"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:cxf="http://cxf.apache.org/core"
               xsi:schemaLocation="
               http://cxf.apache.org/core http://cxf.apache.org/schemas/core.xsd
               http://www.springframework.org/schema/beans
               http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
...
      <cxf:bus>
            <cxf:features>
                  <cxf:logging/>
            </cxf:features>
      </cxf:bus>
</beans>
```

7. Write a deployment descriptor for your SLSB and ensure that it includes:

   a. A `message-driven` element under the `enterprise-beans` element that references to the generic MDB as follows:

      - The value of the `ejb-class` element is `org.apache.cxf.jca.inbound.DispatchMDBMessageListenerImpl`
      - The value of the `messaging-type` element is `org.apache.cxf.jca.inbound.DispatchMDBMessageListener`

   b. An `ejb-local-ref` element, which is required by the MDB so it can look up the local EJB object reference for your SLSB.

   See, for instance, the `ejb-jar.xml` file in *`InstallDir`*`/samples/integration/jca/inbound-mdb-dispatch-wsdl/etc` and shown in Example 2.11 on page 33.

Example 2.11. WSDL First SLSB Deployment Descriptor—ejb-jar.xml

```
<?xml version="1.0"?>
<ejb-jar xmlns="http://java.sun.com/xml/ns/j2ee"
```

```
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
              http://java.sun.com/xml/ns/j2ee/ejb-jar_2_1.xsd"
              version="2.1">

    <enterprise-beans>
       <session>
          <ejb-name>GreeterWithWsdlBean</ejb-name>
          <local-home>demo.ejb.GreeterLocalHome</local-home>
          <local>demo.ejb.GreeterLocal</local>
          <ejb-class>demo.ejb.GreeterBean</ejb-class>
          <session-type>Stateless</session-type>
          <transaction-type>Container</transaction-type>
       </session>

       <message-driven>
          <ejb-name>GreeterEndpointActivator</ejb-name>
          <ejb-class>org.apache.cxf.jca.inbound.DispatchMDBMessageListenerImpl
          </ejb-class>
          <messaging-type>org.apache.cxf.jca.inbound.DispatchMDBMessageListener
          </messaging-type>
          <transaction-type>Bean</transaction-type>

          <activation-config>
             <!-- bus configuration location -->
             <activation-config-property>
                <activation-config-property-name>
                   busConfigLocation
               </activation-config-property-name>
               <activation-config-property-value>
                  etc/cxf.xml
               </activation-config-property-value>
            </activation-config-property>
            <!-- wsdl location -->
            <activation-config-property>
               <activation-config-property-name>
                  wsdlLocation
               </activation-config-property-name>
               <activation-config-property-value>
                  wsdl/hello_world.wsdl
               </activation-config-property-value>
            </activation-config-property>
            <!-- service name -->
            <activation-config-property>
               <activation-config-property-name>
                  serviceName
               </activation-config-property-name>
               <activation-config-property-value>
                  {http://apache.org/hello_world_soap_http}SOAPService
```

```
            </activation-config-property-value>
         </activation-config-property>
         <!-- endpoint name -->
         <activation-config-property>
            <activation-config-property-name>
               endpointName
            </activation-config-property-name>
            <activation-config-property-value>
               {http://apache.org/hello_world_soap_http}SoapPort
            </activation-config-property-value>
         </activation-config-property>
         <!-- service interface class -->
         <activation-config-property>
            <activation-config-property-name>
               serviceInterfaceClass
            </activation-config-property-name>
            <activation-config-property-value>
               org.apache.hello_world_soap_http.Greeter
            </activation-config-property-value>
         </activation-config-property>
         <!-- address -->
         <activation-config-property>
            <activation-config-property-name>
               address
            </activation-config-property-name>
            <activation-config-property-value>
               http://localhost:9000/SoapContext/SoapPort
            </activation-config-property-value>
         </activation-config-property>
         <!-- display name-->
         <activation-config-property>
            <activation-config-property-name>
               displayName
            </activation-config-property-name>
            <activation-config-property-value>
               GreeterWithWsdlEndpoint
            </activation-config-property-value>
         </activation-config-property>
         <!-- targetBeanJndiName -->
         <activation-config-property>
            <activation-config-property-name>
               targetBeanJndiName
            </activation-config-property-name>
            <activation-config-property-value>
               java:comp/env/GreeterWithWsdlLocalHome
            </activation-config-property-value>
         </activation-config-property>
      </activation-config>
```

```
          <ejb-local-ref>
             <ejb-ref-name>GreeterWithWsdlLocalHome</ejb-ref-name>
             <ejb-ref-type>Session</ejb-ref-type>
             <local-home>demo.ejb.GreeterLocalHome</local-home>
             <local>demo.ejb.GreeterLocal</local>
             <ejb-link>GreeterWithWsdlBean</ejb-link>
          </ejb-local-ref>
       </message-driven>

    </enterprise-beans>
</ejb-jar>
```

The `ejb-jar.xml` file in this scenario includes additional activation configuration properties. These properties are used during endpoint activation and point to:

- The configuration file: busConfigLocation
- The service WSDL file: wsdlLocation
- The service name QName as defined in the WSDL file: serviceName
- The `portType` element's QName as defined in the WSDL file: endpointName

If you are using EJB 3.0, the only change you need to make to the deployment descriptor is in the opening `ejb-jar` element. For EJB 3.0 it should read as shown in Example 2.3 on page 21.

8. Build your EJB JAR file and remember to include the service WSDL file in a `wsdl` directory and the FUSE Services Framework configuration file, if you have one, in an `etc` directory.

9. Build the FUSE Services Framework JCA Connector RAR file. It must have the following structure and contents:

   1. `META-INF` directory — Contains the `ra.xml`, located in *InstallDir*`/samples/integration/jca/inbound-mdb-dispatch-wsdl\etc`
   2. `Root` directory — Contains the JAR files listed under `Root` in Table 2.1 on page 22.

   The sample application `build.xml` file includes a **generate.rar** target that you can use to build the RAR file (see Example 2.4 on page 22).

10. Deploy the FUSE Services Framework JCA Connector RAR file and your EJB JAR file to your J2EE application server. For details, see "Deploying FUSE Services Framework JCA Connector" on page 49.

Chapter 3. Exposing a Web Service to a J2EE Application

You can use the FUSE Services Framework JCA Connector to connect your J2EE applications to Web services, otherwise known as outbound connections. This chapter walks you through the steps involved.

Introduction

Overview

The FUSE Services Framework JCA Connector includes a connection management API that allows you to get a connection from your J2EE application to a Web service. The FUSE Services Framework JCA Connector API usage pattern is consistent with general connection management in J2EE.

Sample applications

FUSE Services Framework includes working samples that demonstrate how outbound connections work. You can find them in the following directories of your FUSE Services Framework installation:

- `samples/integration/jca/outbound`
- `samples/integration/jca/outbound.was61`

If you want to build and run these samples, please follow the instructions outlined in the `README.txt` file located in each directory.

Implementation Steps

Steps

The following is a list of the steps that you need to complete to expose your J2EE application to a Web service using the FUSE Services Framework JCA Connector. It assumes that the Web service WSDL file already exists. If, however, you need to develop a WSDL file, please refer to *Writing WSDL Contracts*.

1. Set your FUSE Services Framework environment (see "Setting Up Your Environment" in *Configuring and Deploying Endpoints*).
2. Obtain a copy of, or details of the location of, the WSDL file that defines the Web service to which your application needs to connect.
3. Map the WSDL file to Java to obtain the Java interfaces that you will use when writing your application. FUSE Services Framework provides a **wsdl2java** tool that does this for you.

   For more information on the **wsdl2java** tool see wsdl2java in *Tool Reference*.
4. Write your application. For details, see "Writing Your Application" on page 43.
5. Package your application. For details, see "Packaging Your Application" on page 46.
6. Build the FUSE Services Framework JCA Connector RAR file. It must have the following structure and contents:

Table 3.1. Outbound Connections: RAR File Structure and Contents

Directory	Contents
`META-INF`	The `ra.xml` file that configures the resource adapter
`Root`	• All of the JARs in the `modules` directory of the FUSE Services Framework installation, except the `*manifest*.jar` files. • The `cxf-integration-jca-*.jar`s from the `modules/integration` directory of the FUSE Services Framework installation.

Directory	Contents
	• All of the JARs in the `lib` directory of the FUSE Services Framework installation, except the `cxf*.jar` files.

The sample application's `build.xml` file includes a **generate.rar** target that you can use to build the RAR file (see Example 2.4 on page 22).

7. Deploy the FUSE Services Framework JCA Connector RAR file and your application to your J2EE application server. For details, see "Deploying FUSE Services Framework JCA Connector" on page 49.

Writing Your Application

Connection Management API Definition

The FUSE Services Framework JCA Connector connection management API is packaged in `org.apache.cxf.jca.outbound` and consists of two interfaces:

- `CXFConnectionFactory`
- `CXFConnection`

The API is packaged in *`InstallDir`*`/modules/integration/cxf-integration-jca-`*`Version`*`-fuse.jar`.

The `CXFConnectionFactory` interface provides the methods to create a `CXFConnection` object that represents a Web service defined by the supplied parameters. It is the type returned from an environment naming context lookup of the FUSE Services Framework JCA Connector by a J2EE component and is the entry point to gaining access to a Web service.

The `CXFConnection` interface provides a handle to a connection managed by the J2EE application server. It is the super interface of the Web service proxy returned by the `CXFConnectionFactory` interface.

Usage pattern

To use the `CXFConnectionFactory` your application needs to:

1. Look up a `CXFConnectionFactory` object in the application server's JNDI registry.
2. Use the `CXFConnectionFactory.getConnection()` method to get a `CXFConnection` object.

   The `CXFConnectionFactory.getConnection()` method takes one parameter, a `CXFConnectionSpec` object, which the has following fields:

   - `serviceName` — the QName of the service. This is required.
   - `endpointName` — the QName of the endpoint; i.e. the port name. This is required.

- `wsdlURL` — the URL of the WSDL file. Note that the URL can point to a WSDL file located in the application WAR file or to a location outside the application WAR file, such as a file location on a file system. For more information, see "Finding WSDL at Runtime" in *Developing Applications Using JAX-WS*.
- `serviceClass` — the service interface class. This is required.
- `busConfigURL` — the URL of FUSE Services Framework configuration, if such configuration exists. It allows you to configure directly the FUSE Services Framework runtime. This is optional.

  For more information on how to configure the Artix bus, see *Configuring and Deploying Endpoints*.

  For information on how to configure Artix security, see *Security Guide*.

  The `busConfigURL` setting overrides any configuration that has been set using the FUSE Services Framework JCA Connector `busConfigLocation` activation configuration property (See "Inbound Activation Configuration" on page 55 for more detail).
- `address` — the transport address. This is optional.

3. Use the `CXFConnection.getService()` method to obtain a Web service client.
4. Close the `CXFConnection` object.
5. Invoke on the service.

   The Web service client can still be used after the `CXFConnection` object is closed.

Example of using the Connection Management API

The code shown in Example 3.1 on page 44 shows how to use the FUSE Services Framework JCA Connector connection management API.

Example 3.1. HelloWorldServlet—Outbound Connections

```
❶Context ctx = new InitialContext();
CXFConnectionFactory factory = (CXFConnectionFactory)ctx.lookup(EIS_JNDI_NAME);
```

```
❷CXFConnectionSpec spec = new CXFConnectionSpec();
    spec.setServiceClass(Greeter.class);
   spec.setServiceName(new QName("http://apache.org/hello_world_soap_http", "SOAPService"));

   spec.setEndpointName(new QName("http://apache.org/hello_world_soap_http", "SoapPort"));

    spec.setWsdlURL(getClass().getResource("/wsdl/hello_world.wsdl"));
    CXFConnection connection = null;
    try {
          connection = getConnection(spec);

          ❸Greeter greeter = connection.getService(Greeter.class);

          ❹connection.close();

          ❺greeter.sayHi();
...
         }
```

The code shown in Example 3.1 on page 44 does the following:

❶ Retrieves the connection factory from JNDI.

❷ Creates the connection and uses a `CXFConnectionSpec` object to specify:

- The service class.
- A QName that identifies which service in the WSDL file to use.
- A QName that identifies which port in the WSDL file to use
- The WSDL file URL.

❸ Obtains a Web service client.

❹ Closes the connection to the service and return to the application server connection pool.

Remember you can close the connection and continue using the client.

❺ Invokes methods on the service.

Accessing request/response contexts

The outbound samples show how you can use message contexts.

For more information on message contexts, see "Working with Contexts" in *Developing Applications Using JAX-WS*.

Packaging Your Application

Overview

When packaging and deploying your J2EE application you must declare the resource reference used in your code in your application deployment descriptor and map that resource reference to a resource. In addition, you need to package the Web service interface classes with your application.

Declaring the resource reference

You must declare the resource reference used in your code in your application deployment descriptor, `web.xml`, by adding a `resource-ref` tag. See Example 3.2 on page 46.

Example 3.2. Declaring the resource reference

```
<resource-ref>
     <res-ref-name>eis/CXFConnectionFactory</res-ref-name>
     <res-type>org.apache.cxf.jca.outbound.CXFConnectionFactory</res-type>
     <res-auth>Container</res-auth>
</resource-ref>
```

Mapping the resource reference

You must map the resource reference used in your code to the resource. How you do this is dependent on the application server that you are using. For example, if you are using WebSphere you can use the WebSphere Administrative Console to map the resource reference to the resource while deploying the FUSE Services Framework JCA Connector. See "Deploying FUSE Services Framework JCA Connector" on page 49 and the WebSphere documentation for details.

Packaging details

When packaging your application, include the Java classes that are generated by the **wsdl2java** tool and any other classes that are associated with your application. You can include the service's WSDL file, however, this is not necessary (see the description of `wsdlURL` in "Usage pattern" on page 43.

For example, the `outbound` sample application is packaged in a WAR file as follows:

- `WEB-INF/classes` — includes the application Java class files, the Java classes that are generated from the WSDL file.
- `WEB-INF/classes/wsdl` — WSDL file.

- `WEB-INF/lib` — includes a `common.jar` file that contains the `DemoServletBase.class` file, which the sample application extends.

Please refer to the J2EE specification and your J2EE vendor documentation for more information on application packaging.

Chapter 4. Deploying FUSE Services Framework JCA Connector

How you deploy the FUSE Services Framework JCA Connector is dependent on the J2EE application server that you are using. This chapter provides some basic deployment steps and uses WebSphere 6.1 as an example application server.

Introduction

Overview

How you deploy the FUSE Services Framework JCA Connector is dependent on the J2EE application server that you are using. This chapter describes how to set your environment and provides some basic deployment steps for WebSphere 6.1. It assumes that you have already built the FUSE Services Framework JCA Connector RAR file and your application JAR file. If not, please refer to either:

- on page 15
- on page 39

More detailed information

For more detailed information on how to deploy a JCA resource adapter, please refer to your J2EE application server documentation.

Setting your Environment

Overview

To use FUSE Services Framework JCA Connector with your application server, ensure that:

- The `cxf-manifest.jar`, which is located in *`InstallDir`*`/lib` directory, is on your `CLASSPATH`.
- That the JDK and the **ant** tool's `bin` directory are on your `PATH`.

Important

You do not need to, and should not, source the FUSE Services Framework environment before running your application server.

Deploying to WebSphere 6.1

Overview

This section provides basic information on deploying the FUSE Services Framework JCA Connector and your application to WebSphere 6.1. For more detailed information, please refer to your WebSphere documentation.

Prerequisites

The following prerequisites apply to WebSphere 6.1:

- Copy the `wsdl4j-*.jar` from your *`InstallDir`*`/lib` directory to your *`WAS_HOME`*`/AppServer/java/jre/lib/endorsed` directory. If the endorsed directory does not exist, create it.

  Restart WebSphere.

- Make sure your environment is set correctly. See "Setting your Environment" on page 51 for details.

Deploying the FUSE Services Framework JCA Connector

You must deploy the FUSE Services Framework JCA Connector to WebSphere before you deploy your application. In addition, please make sure that the FUSE Services Framework JCA Connector has not already been deployed to your application server.

To deploy the FUSE Services Framework JCA Connector in WebSphere 6.1 complete the following steps:

1. Logon to WebSphere Integrated Solution Console. The default address is:

   `http://`*`hostname`*`:9060/ibm/console/login.do`

2. Navigate to **Resources** → **Resource adapters** → **Resource adapters**.
3. On the Resource adapters page, click **Install RAR**.
4. On the Install RAR File page, select the **Local path** radio button if the browser that you are running is on the same machine as the WebSphere server. Otherwise, select the **Server path** radio button.
5. Specify or browse to where you have built the `cxf.rar` file and click **Next**.

6. On the next page, click **OK** to install the Resource Adapter.
7. On the next page, click the **CXF JCA Connector** link to edit the Resource Adapter.
8. On the Configuration page, click the **J2C activation specification** link.
9. On the next page, click **New** to create a new Activation Specification.
10. On the next page, enter **`MyActivationSpec`** in the **Name** textbox and click **OK**.

    The JNDI name is optional. If it is omitted, a JNDI name is created for you as `eis/<ActivationSpecName>`, where `<ActivationSpecName>` is `MyActivationSpec`.
11. Click **Save** to commit the configuration.

You can specify activation configuration values in the new activation specification you just created.

For inbound connections, the activation specification is associated with your MDB later. The MDB's deployment descriptor can define activation configuration values to override the values specified in the associated activation specification. See "Inbound Activation Configuration" on page 55 for more detail.

Deploying Your application

To deploy your application to WebSphere 6.1, complete the following steps:

1. Logon to WebSphere Integrated Solution Console. The default address is `http://<hostname>:9060/ibm/console/login.do`.
2. Navigate to **Applications** → **Install new Applications**.
3. On the **Preparing for the application installation** page, select the **Local path** radio button if the browser that you are running is on the same machine as the WebSphere server. Otherwise, select the **Server path** radio button.
4. Specify or browse to the path where you have your application JAR file stored and click **Next**.
5. On the **Step 1: Select installation options** page, click **Next**.
6. On the **Step 2: Map modules to servers** page, click **Next**.

7. On the **Step 3: Bind listeners for message-driven beans** page, in the far right column, click the **Activation Specification** radio button.
8. Specify the Target Resource JNDI Name as below and click **Next**.

   ```
   eis/MyActivationSpec
   ```

9. On the **Step 4: Summary** page, click **Finish**.
10. Click the **Save** link to commit the configuration.
11. Navigate to **Applications** → **Enterprise Applications**.
12. Select the box next to your application JAR file and click **Start** to start the MDB.

For more detail, please consult your WebSphere documentation.

Chapter 5. Inbound Activation Configuration

This chapter describes the configuration for the FUSE Services Framework JCA Connector's inbound message listeners.

Introduction

Activation specifications are part of the configuration of inbound messaging support provided by a JCA 1.5 resource adapter, such as FUSE Services Framework JCA Connector. Resource adapters that support inbound messaging define one or more types of message listener in their deployment descriptors. This is defined in the `messagelistener` element in the `ra.xml` file. The message listener is the interface that the resource adapter uses to communicate inbound messages to the message endpoint. For each type of message listener that a resource adapter implements, the resource adapter defines an associated activation specification, which defines configuration properties for the receiving endpoint.

The FUSE Services Framework JCA Connector inbound support includes two types of message listener and two activation specification classes, one for each message listener type.

Table 5.1. Message Listeners and Activation Specifications

Message Listener Type	Activation Specification Class	Supported Properties
Target service interface, used when MDB also implements the target service. See "Service Implemented as a Message Driven Bean" on page 18 for an example use case.	`org.apache.cxf.jca.inbound.MDBActivationSpec`	See Table 5.2 on page 56.

`org.apache.cxf.jca.inbound.`
 	
`DispatchMDBMessageListener`
	`org.apache.cxf.jca.inbound.DispatchMDBActivationSpec`	See Table 5.2 on page 56

Message Listener Type	Activation Specification Class	Supported Properties
See "Service Implemented as a Stateless Session Bean" on page 24 for an example use case.		and Table 5.3 on page 57.

Supported Properties

Table 5.2 on page 56 shows the activation configuration properties that are supported when the target service interface is specified as the message listener type and `org.apache.cxf.jca.inbound.MDBActivationSpec` is specified as the activation specification class in the FUSE Services Framework JCA Connector `ra.xml` file.

Table 5.2. Service Implemented as MDB: Supported Activation Configuration Properties

Property Name	Required	Description
wsdlLocation	No	A string that specifies the location of the Web service WSDL file.
schemaLocations	No	String that specifies the schema locations, each one separated by a comma.
serviceInterfaceClass	Yes	String that specifies the service interface class name.
busConfigLocation	No	String that specifies the location of any FUSE Services Framework configuration files.
address	No (if specified in WSDL file)	String that specifies the transport address.
endpointName	Yes	String that specifies the QName of the service's `portType` element in the WSDL file.
serviceName	Yes	String that specifies the QName of the service's `service` element in the WSDL file.
displayName	Yes	String that specifies the name used for logging and as a key in a map of endpoints.

Table 5.2 on page 56 and Table 5.3 on page 57 show the activation configuration properties that are supported when `org.apache.cxf.jca.inbound.DispatchMDBMessageListener` is specified as the message listener type and `org.apache.cxf.jca.inbound.DispatchMDBActivationSpec` is specified as the activation specification class in the FUSE Services Framework JCA Connector `ra.xml` file.

Table 5.3. Service Implemented as a SLSB: Supported Activation Configuration Properties

Property Name	Required	Description
targetBeanJndiName	Yes	A string that specifies the JNDI name of the target session bean.

Setting activation configuration properties

Activation configuration properties can be set in any of the following:

- The application deployment descriptor.
- Activation specification, which can be set when deploying FUSE Services Framework JCA Connector.
- The FUSE Services Framework JCA Connector deployment descriptor, `ra.xml`.

Values specified in the `ejb-jar.xml` file override those set in the activation specification and the `ra.xml` file. Values specified in the activation specification override those set in the `ra.xml` file.

Examples of setting

For an example of how the activation configuration properties are set, see:

a. The `ra.xml` located in

InstallDir`/samples/integration/jca/inbound-mdb-dispatch/etc`,

the relevant sections of which are shown in Example 5.1 on page 57.

b. The `ejb-jar.xml` file located in

InstallDir`/samples/integration/jca/inbound-mdb-dispatch/etc`,

the relevant sections of which are shown in Example 5.2 on page 58.

Example 5.1. Activation Specification in ra.xml

```
<messagelistener>
      <messagelistener-type>
            org.apache.cxf.jca.inbound.DispatchMDBMessageListener
      </messagelistener-type>
      <activationspec>
            <activationspec-class>
                  org.apache.cxf.jca.inbound.DispatchMDBActivationSpec
            </activationspec-class>
            <required-config-property>
                <config-property-name>displayName
```

```
                </config-property-name>
            </required-config-property>
            <required-config-property>
                <config-property-name>targetBeanJndiName
                </config-property-name>
            </required-config-property>
        </activationspec>
</messagelistener>
```

Example 5.2. Activation Specification in ejb-jar.xml

```
<activation-config>
    <!-- display name-->
    <activation-config-property>
       <activation-config-property-name>
          DisplayName
       </activation-config-property-name>
       <activation-config-property-value>
          DispatchedGreeterEndpoint
       </activation-config-property-value>
       </activation-config-property>
    <!-- service endpoint interface -->
    <activation-config-property>
          <activation-config-property-name>
             serviceInterfaceClass
          </activation-config-property-name>
          <activation-config-property-value>
             org.apache.hello_world_soap_http.Greeter
          </activation-config-property-value>
    </activation-config-property>
    <!-- address -->
    <activation-config-property>
       <activation-config-property-name>
          address
       </activation-config-property-name>
       <activation-config-property-value>
          http://localhost:9999/GreeterBean
       </activation-config-property-value>
    </activation-config-property>
    <!-- targetBeanJndiName -->
    <activation-config-property>
       <activation-config-property-name>
          targetBeanJndiName
       </activation-config-property-name>
       <activation-config-property-value>
          java:comp/env/DispatchedGreeterLocalHome
       </activation-config-property-value>
    </activation-config-property>
</activation-config>
```

Index

T