1 Introduction .......................................................................................................................... 2
  1.1 Architecture Overview ................................................................................................. 4
    1.1.1 Adapter Services ...................................................................................................... 4
      1.1.1.1 Outbound Interaction/Request-response Service ............................................... 4
      1.1.2 Inbound Interaction/Event Service ....................................................................... 4
    1.1.2 BPEL PM integration with JCA Outbound Interaction .............................................. 5
      1.1.2.1 Flow .................................................................................................................. 7
    1.1.3 BPEL PM integration with JCA Inbound Interaction .................................................. 7
      1.1.3.1 Flow .................................................................................................................. 9
    1.1.4 Adapter Design-time ............................................................................................... 5
  2 Steps to integrate the JCA 1.5 Resource Adapter with BPEL PM for Outbound Interaction via WSIF ................................................................. 11
    2.1 Implement XML Record Converter Interface ............................................................. 11
    2.2 Understanding the WSDL structure .......................................................................... 11
    2.3 WSDL for Outbound Interaction – JCA Binding .......................................................... 12
      2.3.1 WSDL namespace .............................................................................................. 12
      2.3.2 Abstract WSDL Section ..................................................................................... 12
      2.3.2.1 Port Type .......................................................................................................... 12
      2.3.3 Binding Section ..................................................................................................... 13
      2.3.3.1 jca:operation .................................................................................................... 14
      2.3.3.2 jca:header ......................................................................................................... 14
      2.3.3.3 jca:binding ....................................................................................................... 16
      2.3.4 Service Element ................................................................................................... 17
      2.3.4.1 jca:address ...................................................................................................... 17
    2.4 Deployment of Oracle JCA 1.5 Resource Adapter .......................................................... 18
  3 Steps to integrate the BPEL PM with the JCA 1.5 Resource Adapter Inbound Interaction ............................................................................................................ 19
    3.1 XML Record Converter .............................................................................................. 19
    3.2 WSDL for Inbound Interaction- Oracle JCA1.5 extension ........................................... 19
      3.2.1 WSDL namespace .............................................................................................. 19
      3.2.2 WSDL Binding Section ..................................................................................... 19
      3.2.2.1 jca:operation .................................................................................................... 20
      3.2.2.2 jca:header ......................................................................................................... 20
      3.2.2.3 jca:binding ....................................................................................................... 21
      3.2.3 Service WSDL Extension ..................................................................................... 21
      3.2.3.1 jca:address ...................................................................................................... 21
    3.3 Deploy Resource Adapter ........................................................................................... 21
  4 Appendix .......................................................................................................................... 22
    oracle.tip.adapter.api.record Interface XMLRecord ................................................... 23
    getHeaderRecordElement ............................................................................................ 24
    setHeaderRecordElement ............................................................................................ 24
    setPayloadRecordElement ............................................................................................ 25
    Definition of RecordElement ....................................................................................... 25
1 Introduction

Enterprises need to integrate diverse set of applications that often come from different vendors and run on different platforms. Enterprise integration requires connectivity to a broad range of systems, including technology, messaging middleware, legacy, mainframe, B2B and packaged application systems. An adapter is a software that an application server or an application client uses to connect to a specific Enterprise Information System (EIS). The adapters enable customers and partners to leverage investments in existing custom or otherwise unique applications, and to integrate them with other applications. The Oracle approach is to provide a suite of standards-based integration adapters that implement bi-directional connectivity between Oracle applications and other EIS applications and enable faster, flexible, efficient and cost-
effective integration. This document describes the technical aspects involved in developing an Adapter for integration with Oracle BPEL PM, the BPEL based process orchestration product.

The Oracle BPEL PM is based on a Service Oriented Architecture (SOA) and every process activity is defined as a Web Service. Each business activity is a self-contained, self-describing, modular application whose interface is defined by a Web Service and the business process is modeled as a service flow. The Web Services can be distributed, located on heterogenous platforms and the invoker of the Web Service is agnostic of the implementation. The above architecture offers the benefits of SOA – namely flexibility, interoperability, reusability, extensibility and rapid implementation. It reduces the overall cost of management, modification, extension, and redeployment of existing business processes.

The Oracle AS Adapters also support SOA through the adoption of Open Standards. The Adapters are JCA 1.5 compliant starting from 10.1.2.0.1 (10gR2 Phase2) release. The Adapter Framework component is used to wrap any standard JCA 1.5 adapter interactions as a Web Service and has a WSIF (Web Service Invocation Framework) JCA Provider for the above purpose. The Adapter FW and SDK are one and the same.

The integration steps are simple once you have developed a standard JCA 1.0/1.5 Resource Adapter:

(1) Design-time - Configure the Adapter with the custom developed design-time tool. Generate WSDL definitions for the above and make it available to JDeveloper/Eclipse BPEL Designer. The Adapter-BPEL integration document describes the WSDL extension elements in detail.

(2) Run-time - Implement the XMLRecordConverter interface to translate from your JCA Record implementation to Oracle XMLRecord implementation. The javadoc for the interface as well as a sample has been provided in this location.

(3) End to end testing.

You could also wrap up your custom adapter as a Web Service and expose this to BPEL Process Manager. This is a loose coupling strategy and does not need an Adapter SDK. Both these approaches (JCA/web Service) are suitable for outbound (from J2EE/BPEL PM to EIS) invoke operations. Only the JCA 1.5 integration allows the BPEL PM to receive inbound events (from EIS to J2EE/BPEL PM). The BPEL PM acts as a pseudo JCA 1.5 container and implements the JCA 1.5 specific System Contracts. This is agnostic to the JCA 1.5 Resource Adapter which gets deployed in the Oracle AS OC4J container.

You can use any custom design-tool for the configuration of the Adapter but a WSDL must be generated at the end of the design-time phase for consumption by the BPEL PM design-time (Jdeveloper/Eclipse based BPEL Designer). The WSDL for the
JCA interactions have a JCA extension. The JCA extension elements are described in detail in the later sections. The Adapter is a JCA 1.0/1.5 Resource Adapter deployed in the same OC4J container as that of the BPEL PM product. Please note that the JCA 1.5 Resource Adapter and the BPEL PM instance must be deployed in the same OC4J container.

The Adapter Framework (AF) is the glue layer that integrates the standard JCA 1.5 Resource Adapter seamlessly with the BPEL PM product at run-time. The Adapter Framework has a WSIF JCA Provider for wrapping the JCA interactions as Web Services and performs the translation between Web Service messages to JCA interaction messages based on the WSDLs generated at design-time.

### 1.1 Architecture Overview

The Oracle Adapters exploits the WSIF and JCA standards to expose itself as a Web Service for both outbound and inbound operations.

#### 1.1.1 Adapter Services

There are 2 kinds of run-time Adapter Services.

##### 1.1.1.1 Outbound Interaction/Request-response Service

The synchronous request-response communication paradigm, is also known as an Outbound Interaction in JCA terminology. In the above paradigm, the Adapter receives requests from adapter client applications, translates these requests into the data format native to the EIS, and invokes the appropriate method on the EIS application. The Adapter also returns the EIS response to the adapter client applications, after performing reverse translation.

##### 1.1.1.2 Inbound Interaction/Event Service

The event-notification is an asynchronous communication paradigm and is also known as an Inbound Interaction in JCA terminology.
Event notification can be achieved in 2 ways:

- When listening for events, the adapter is registered as a listener for an EIS application that is configured to push events out to the adapter.
- Alternatively, the adapter polls the EIS, usually an interface or a staging database table, for events of interest to the client application.

These events might be generated within the EIS application when a transaction completes successfully, or they may be non-transactional events initiated by the EIS application.

The Adapter also has a third-type of function, a design-time function for configuration and creation of Adapter run-time services. The Adapter Service configuration consists of the EIS connection details and the schema for the EIS business object/service/interfaces.

The Adapter is a standard JCA 1.5 Resource Adapter and is deployed within OC4J, the J2EE container of the Oracle Application Service (Oracle AS). The OC4J container in Oracle AS 10.1.2 supports JCA 1.0 standards only and the JCA 1.5 Resource Adapter is deployed as a JCA 1.0 Resource Adapter in this release. The Resource Adapter is deployed in managed mode for the JCA Outbound Interaction and in a non-managed mode for the JCA Inbound Interaction. The OC4J container will support JCA 1.5 standards in Oracle AS 10.1.3 release and the JCA 1.5 Resource Adapter will be deployed in managed mode for both JCA Outbound and Inbound Interactions.

### 1.1.2 Adapter Design-time

Both the JCA Outbound and Inbound Interactions are exposed as WSDLs. The WSDL extension section contains JCA elements. The WSDL JCA Binding elements contains the metadata for the AF to invoke any JCA Outbound Interaction and activate any inbound JCA 1.5 Endpoint in order to receive inbound events.

The Adapter Vendor must provide a Tool to create and expose the Adapter configurations as WSDL for consumption by Oracle BPEL PM. The Oracle BPEL PM is configured using Oracle Jdeveloper/IBM Eclipse IDE. The Oracle BPEL PM requires “Partner Links’ to be created to invoke or receive messages from a Web
Service. The “Partner Link” is a BPEL standard for defining an external service. The Partner Link wizard allows file browsing of WSDLs.

1.1.3 BPEL PM integration with JCA Outbound Interaction

The BPEL PM has a light-weight, stateless Adapter Framework (AF) component to integrate with the JCA 1.5 Resource Adapter. The BPEL PM uses the Web Service Invocation Framework (WSIF) technology to invoke the Outbound Interaction of the Resource Adapter. WSIF is used for Java invocation of Web Services and separates the abstract part of the Web Service definition from the physical binding (transport and protocol) producing a true Service Oriented Architecture (SOA). The binding definition can be postponed to run-time providing true flexibility.

The WSIF Provider converts the Web Service invocation (from BPEL PM invoke activity) to a JCA Outbound Interaction call and performs the reverse translation in the other direction. In other words, Web Service invocation launched by the BPEL “invoke” activity is converted to a JCA CCI Outbound Interaction and the JCA response is converted back to a Web Service response. This end to end invocation is synchronous. The WSIF Provider also supports one-way asynchronous JCA Outbound Interaction invocation. The WSIF Provider element hides the JCA implementation details from the BPEL PM process and the Web Service aspect from the JCA Resource Adapter. The Adapter Service needs to be captured as a Web Service with JCA binding to enable the WSIF Provider to perform the above translation. The WSIF Provider uses the JCA artifacts in the Service and Binding section of the Adapter WSDL to convert the Web Service into a JCA CCI call.

Oracle has its own implementation of CCI Record interface called the Oracle XML Record interface. The Oracle JCA WSIF Provider understands only the above Record interface and an XMLConverter class needs to be specified to convert to and fro.
1.1.3.1 Flow

- At design-time, the Adapter Service is exposed as a Web Service and is consumed during configuration of the “Partner Link” activity of the BPEL process.
- The WSDL extensions specify the JNDI address of the Resource Adapter and the InteractionSpec class name and the InteractionSpec parameters.
- At run-time, the “invoke” activity of the BPEL PM is used to invoke the Partner Link service – in this case a JCA Resource Adapter Outbound Interaction.
- The WSIF JCA Provider component of the AF takes control from this point.
- The WSIF JCA Provider receives the Web Service request from BPEL PM.
- The WSDL associated with this Web Service request contains the JNDI name of the JCA Connection Factory class in the WSDL extension element.
- The WSIF Provider performs the JNDI lookup and obtains a Connection instance.
- The WSIF Provider creates an InteractionSpec Bean object from the WSDL extension JCA artifacts.
- The WSIF Provider then calls the XMLRecord Converter class if specified in the WSDL extension section.
- The WSIF Provider invokes the JCA OutboundInteraction by passing the InteractionSpec, the InputRecord and an empty OutputRecord.
- The JCA 1.5 Resource Adapter calls the EIS application and returns the EIS response to the WSIF Provider.
- The WSIF Provider converts the output Record to XMLRecord if the XMLRecordConverter class is specified in the WSDL extension section.
- This XML Record is translated to a Web Service Response for consumption by the BPEL PM instance.
- The whole end to end scenario is synchronous and any JCA exception is forwarded as a Web Service Fault message.

1.1.4 BPEL PM integration with JCA Inbound Interaction

The BPEL PM receives events from the JCA 1.5 Resource Adapter via the AF. The AF plays the part of a JCA 1.5 container and implements the message-inflow contracts for receiving events from the adapter. In addition, it also implements the life-cycle management contract for start/stop of the JCA 1.5 Resource Adapter as well as Work Management for allocating threads. This enables Adapter Vendors to develop towards JCA 1.5, even though Oracle AS 10.1.2 will only ship with an OC4J version, which supports JCA 1.0. The JCA 1.5 Resource Adapter will be agnostic of the fact that the JCA 1.5 system contracts are implemented by AF and not by the OC4J container. The migration to a fully-managed mode in Oracle AS 10.1.3 release is expected to be seamless.
The JCA Inbound Interaction is captured in a WSDL at design-time in a fashion similar to the WSDL generation of the Outbound Interaction. The JCA Inbound WSDL Binding section contains the JCA 1.5 ActivationSpec. The ActivationSpec captures the inbound connectivity and inbound interaction details (as per JCA 1.5 specification). The JCA Inbound WSDL Service section contains the JCA 1.5 Resource Adapter class name.

The AF performs all the JCA 1.5 specific system contracts and the JCA 1.5 Resource Adapter is completely agnostic of the fact that it is being called by the AF component instead of the OC4J container. The ResourceAdapter class name in the WSDL Service section is used by the AF (psuedo JCA 1.5 container) to control the Life-cycle management of the Resource Adapter. JCA 1.5 specification mandates that the ResourceAdapter class implement the start (BootStrapContext ) and Stop() methods.

The AF implements the **BootStrapContext** interface and passes it as an input parameter when invoking the start() method of the ResourceAdapter. The BootStrapContext contains the **WorkManager** and **XATerminator** objects for work management and transaction co-ordination. There are however some limitations. The WorkManager implements the scheduleWork() method only. The AF does not implement doWork() and startWork() methods and instead points to the scheduleWork() method. The Resource Adapter can use the **WorkManager** object passed by the AF to submit **Work** instances for execution. The AF also does not support transaction-inflow.

JCA 1.5 mandates that the ResourceAdapter implement the endpointActivation (MessageEndpointFactory, ActivationSpec) and endpointDeactivation( ActivationSpec,
MessageEndpointFactory) methods to activate and deactivate a message endpoint. The AF calls the endPointActivation method on the Resource Adapter for every BPEL process endpoint that is interested in receiving an inbound event from the Resource Adapter. The ActivationSpec bean is created from the WSDL (binding section) artifacts specified at design-time and is passed along with the MessageEndPointFact to the endPointActivation method of the Resource Adapter. The ResourceAdapter in turn creates MessageEndpoints from the MessageEndpointFactory and calls the onMessage() method of the MessageEndpoint on receipt of an EIS event. The MessageEndpoints created by the MessageEndpointFactory are managed by the AF and the events received via the onMessage() method of these endpoints are in turn posted to the respective BPEL processes.

The AF invokes the endpointDeactivation() method to deactivate all registered inbound endpoints before calling the stop() method of the ResourceAdapter.

Note: A JCA 1.0 Resource Adapter can also be used for BPEL PM integration. The BPEL PM will not be able to receive inbound events via it’s receive activity in this case.

1.1.4.1.1 Flow

- The “ResourceAdapter” class name and the “ActivationSpec” are captured in the WSDL extension section of the JCA Inbound Interaction WSDL at design-time and made available to BPEL PM and hence AF at run-time.
- It creates an instance of the JCA 1.5 “ResourceAdapter” if it is does not exist and calls the “start” method of the JCA Resource Adapter class.
• Each Inbound Interaction operation referenced by the BPEL PM processes will result in invoking the “EndPointActivation” method of the JCA 1.5 Resource Adapter instance. The AF creates the “ActivationSpec” class (Java Bean) based on the ActivationSpec details present in the WSDL extension section of the JCA Inbound Interaction and passes it as input to the “endPointActivation” method along with the MessageEndPointFactory class.

• The JCA 1.5 Resource Adapter will create a “MessageEndPoint” instance from the MessageEndPointFactory class. The messageEndPoint has a messageListener component and the JCA 1.5 Resource Adapter will invoke the “onMessage” method on receipt of an EIS event.

• The AF will receive the event via the MessageListener class and forwards it to the “receive” activity of the BPEL PM instance that registered interest in the event.

• All the endpoints will be deactivated when the AF is shutdown and the “EndPointDeactivation” method is called. The “ActivationSpec” and the “MessageEndPointFactory” are both passed as inputs to the above method.

The AF also implements the JCA 1.5 Work Management contracts and provides “Worker threads” to the JCA 1.5 Resource Adapter.
2 Steps to integrate the JCA 1.5 Resource Adapter with BPEL PM for Outbound Interaction via WSIF

- Implement the XML Record Converter interface
- Design-time: Generate WSDLs
- Deploy the Adapter
- Perform run-time tests

2.1 Implement XML Record Converter Interface

Oracle has defined an “XMLRecord” class that consists of 2 Record Elements - one mandatory Payload RecordElement and one optional Header RecordElement. The Record elements can be UTF-8 encoded XML string, or a binary opaque byte stream. The AF understands only Oracle “XMLRecord”. Adapter Vendors need to write an XML Record Converter class to map between their JCA Record implementation and the Oracle XMLRecord implementation. The translator class should be specified in the WSDL JCA extension section as one of the properties. The Java API as well as an example translator class can be found in the appendix section.

2.2 Understanding the WSDL structure
The WSDL has an abstract section defined by a “Port Type” WSDL element, an implementation section defined by the “Binding” WSDL element and the address section defined by the “Service” WSDL element. The “Port Type” element defines the methods offered by the Service. The “Operation” WSDL element as the name indicates provides definition of a method in the Service. The input and output of operations are expressed by the “Message” element of the WSDL. The messages can in turn be made of parts and these parts can be of different data types. The data type definition is defined in the “Type” WSDL element. The “Binding” WSDL element maps the abstract definition of a “Port Type” WSDL element to an implementation. The “Service” WSDL element is a collection of “Port” elements. The “Port” WSDL element defines how to locate a “Binding” and hence a “Port Type”.

2.3 WSDL for Outbound Interaction – JCA Binding

The “Binding” section maps the abstract Adapter WSDL to a JCA CCI Outbound Interaction implementation and refers to the JCA CCI Interaction Spec. The “Service” section ties the abstract “Port Type” WSDL element and the “Binding” WSDL element to a JNDI address.

2.3.1 WSDL namespace

The name space for the Outbound interaction are based on the WSDL J2C specification and are given below:

```
xmlns:jca=http://xmlns.oracle.com/pcbpel/wsdl/jca/
```

You can also specify the adapter specific binding elements using a similar convention. Example of binding elements for Oracle AS Technology Adapters are given below:

- `http://xmlns.oracle.com/ias/pcbpel/wsdl/adapter/file` (File adapter)
- `http://xmlns.oracle.com/ias/pcbpel/wsdl/adapter/database` (Database adapter)

2.3.2 Abstract WSDL Section

2.3.2.1 Port Type

The “Port Type” can represent a “2-way” synchronous JCA Outbound Interaction invocation or a “1-way” asynchronous Outbound Interaction. These are represented as follows:
2.3.3 Binding Section

There are 3 extension elements under the Binding section and they are as follows:

- `<jca:binding>`
- `<jca:operation>`
- `<jca:header>`

```xml
<portType name="oneWay">
  <operation name="sendRequest">
    <input message="request"/>
  </operation>
</portType>

<portType name="requestResponse">
  <operation name="getResponse">
    <input message="request"/>
    <output message="response"/>
  </operation>
</portType>

<binding name="" type=PORT_TYPE >
  <jca_binding />
  <operation name="">
    <jca:operation
      InteractionSpec=""
      InteractionSpec Parameter List
    />
    <input>
      <jca:header />
    <input>
    <output />
  </operation>
</binding>
```

1-way asynchronous
CCI J2CA Outbound
Interaction invocation

2-way synchronous
CCI J2CA Outbound
Interaction Invocation

XML Converter class
To convert to Oracle XML
Record

JCA InteractionSpec and
InteractionSpec Parameter
List

WSDL message part
that contains Operation
headers
2.3.3.1 jca:operation

The **jca:operation** element defines one mandatory attribute - the Java class name of the **javax.resource.cci.InteractionSpec** implementation class and zero or more optional any-name/any-value attributes for the InteractionSpec parameters. This Extension Element must be an immediate child of a WSDL Operation element (**wsdl:operation**). The WSIF Provider creates the InteractionSpec bean class and initializes it with the name/value pairs specified in the **jca:operation** element using Java reflection. Format for the **jca:operation** is shown below.

```xml
<binding name="" type="">
  <jca:binding />
  <operation name="">
    <jca:operation InteractionSpec = "InteractionSpec class name"
                    InteractionSpec parameters – name/value pair list
    />
    <input>
      <jca:header />
    </input>
  </operation>
</binding>
```

Example below shows the **jca:operation** element for a FileAdapter operation. The InteractionSpec class name is “oracle.tip.adapter.file.outbound.FileInteractionSpec”. The parameters for the InteractionSpec namely “OutputDirectory”, “NumberOfMessages”, “ElapsedTime”, “FileSize”, “StaticName” are used to create the InteractionSpec Bean instance by the AF to invoke the FileAdapter Outbound Interaction.

```xml
<binding name="FileOutbound_JCABinding" type="tns:FileOutbound_PortType">
  <jca:binding />
  <operation name="Send">
    <jca:operation InteractionSpec = "oracle.tip.adapter.file.outbound.FileInteractionSpec"
                    OutputDirectory="\.\sample\FileAdapter\ComplexStructure\outputDir"
                    NumberOfMessages=""
                    ElapsedTime=""
                    FileSize=""
                    StaticName="MyFile"
                    IsSequenceNumberInName="true"
                    TimeStampPatternInName="yyyy.MM.dd.SSS" />
    <input>
      <jca:header />
    </input>
  </operation>
</binding>
```

2.3.3.2 jca:header
Oracle has defined a JCA CCI Record class, “**XMLRecord**” class that consists of 2 Record Elements - one mandatory “**Payload RecordElement**” and one optional “**Header RecordElement**”. The `jca:header` element defines the header element for the operation and represents the Header RecordElement of the “**XML Record**”. This Extension Element must be an immediate child of the WSDL Input element - `wsdl:input` under `jca:operation` element. The `jca:header` element defines two mandatory attributes namely message and part. This informs the WSIF Provider about which WSDL Message and Part elements define the Header Record Element for an operation.

```xml
<operation name="... ">
  <jca:operation ...>
    <input>
      <jca:header message="tns:<HeaderMessageName>" part="<HeaderPartName/>">
      </jca:header>
    </input>
  </jca:operation>
</operation>
```

The Headers and Payload Record Elements *must* be defined as separate WSDL Message elements. This is illustrated in the following example:

```xml
<definition xmlns="http://schemas.xmlsoap.org/wsdl/" ...>
  <types>
    <schema ...>
      <element name="FileHeader">
        <complexType>
          <sequence>
            <element minOccurs="0" name="FileName" type="string"/>
            ...
          </sequence>
        </complexType>
      </element>
      <element name="myPayloadType">
        ...
      </element>
    </schema>
  </types>

  <!-- Header message -->
  <message name="FileHeader">
    <part element="tns:FileHeader" name="Header"/>
  </message>

  <!-- Payload message -->
  <message name="myPayloadMessage">
    <part name="myPayload" type="tns:myPayloadType"/>
  </message>

  <portType name="new_customer_Write_portType">
    <operation name="new_customer_Write">
      <input message="tns:myPayloadMessage"/>
    </operation>
  </portType>
</definition>
```
2.3.3.3 jca:binding

This is used to specify the XML Record Converter to convert from Adapter Vendor specific CCI Record structure to Oracle XML Record structure. It is the child of the WSDL Binding element - wsdl:binding.

Example JCA binding showcasing the “AcmeXMLRecordConverter” is shown below.
2.3.4 Service Element

2.3.4.1 jca:address

This Extension Element must be an immediate child of the base WSDL Port element (wsdl:port). The jca:address element has one and only one attribute – adapterInstanceJndi to define the JNDI name of a ManagedConnectionFactory of a deployed JCA adapter.

```
<wsdl:definitions ...
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/
...

<wsdl:service name="<service_name>">
  <wsdl:port name="<port_name>" binding="tns:<binding_name>">
    <jca:address adapterInstanceJndi=""/>
  </wsdl:port>
</wsdl:service>
```

For example, consider an AQ JCA Adapter instance – “myAqAdapter1“ deployed in OC4J and with JNDI name as “eis/myAQAdapter1” . The deployment descriptor oc4j-ra.xml will look similar to the section below:

```
$J2EE_HOME/application-deployments/default/myAqAdapter1/oc4j-ra.xml:
<oc4j-connector-factories>
  <connector-factory location="eis/myAqAdapter1" connector-name="AQ Adapter">
    ...<config-property name="ServerName" value="foo"/>
  </connector-factory>
</oc4j-connector-factories>
```

The above ManagedConnectionFactory will be referred by the WSDL in the “Service” section in the following manner:

```
<wSDL:definitions ...
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/
...

<wSDL:service name="<service_name>">
  <wSDL:port name="<port_name>" binding="tns:<binding_name>"/>
```

The JCA WSIF Provider will perform a JNDI lookup based on the above `<jca:address>` element in the OC4J's JNDI Name Space (Initial Context) to create a JCA CCI Connection object.

```java
Context ic = new InitialContext();
String jcaJndiLocation = jcaAddress.getAdapterInstanceJndi();
ConnectionFactory cf = (ConnectionFactory) ic.lookup(jcaJndiLocation);
connection = cf.getConnection();
```

## 2.4 Deployment of Oracle JCA 1.5 Resource Adapter

- The JCA 1.5 Resource Adapter must be packaged as a .rar file and must have a deployment descriptor ra.xml. The Adapter Vendor must provide a Resource Adapter Archive (RAR) with an embedded ra.xml deployment descriptor for deploying the Adapter within the OC4J container. Here are the contents of a sample RAR file:
  
  META-INF/ra.xml  
  META-INF/oc4j-ra.xml  
  howto.html  
  images/icon.jpg  
  ra.jar  
  cci.jar

- The JCA 1.5 (1.0) Resource Adapter and the BPEL PM process must be deployed in the same OC4J container.

- The ResourceAdapter can be deployed using the following command:
  
  `<ORACLE AS HOME>/ dcm/bin/dcmctl deployapplication –a <name> -f <location of rar file>`

- OC4J creates a `oc4j-ra.xml` for each deployed Resource Adapter if it is not included in the .rar file. This file is used to specify the connection factory and it’s JNDI location. OC4J creates a `<connector-factory>` element for the above. You can use the oc4j-ra.xml to override the configuration properties in the ra.xml. The `<connector-factory>` element contains the configuration properties.

- You can create multiple `<connector-factory>` elements in the oc4j-ra.xml and each such factory needs to be attached to a unique JNDI name.
3 Steps to integrate the BPEL PM with the JCA 1.5 Resource Adapter Inbound Interaction

- Implement the XML Record Converter interface
- Design-Time: Generate WSDLs
- Deploy Adapter
- Perform end to end tests

3.1 XML Record Converter

Please see section 2.1.

3.2 WSDL for Inbound Interaction- Inbound JCA1.5 extension

The Metadata for Inbound Interaction is also captured as a WSDL. The Oracle JCA 1.5 extension also known as “Inbound Extension” is used to specify the ResourceAdapter class name and the ActivationSpec parameters. This is understood only by the Adapter Framework component. The “Binding” section maps the abstract Adapter WSDL to a JCA CCI Inbound Interaction implementation and refers to the JCA CCI ActivationSpec. The “Service” section ties the abstract “Port Type” WSDL element and the “Binding” WSDL element to a ResourceAdapter class name.

3.2.1 WSDL namespace

Please refer to section 2.3.1

3.2.2 WSDL Binding Section

The WSDL Binding extension elements are as follows:

- `<jca:binding>`
- `<jca:operation>`
- `<jca:header>`

```xml
<binding name="" type=PORT_TYPE >
  <jca_binding />
  <operation name="" />
    <jca:operation>
      ActivationSpec="" 
      ActivationSpec Parameter List
    
    <input>
      <jca:header />
    
    <input>
    <output />
  </operation>
</binding>
```
3.2.2.1 jca:operation

The jca:operation element defines one mandatory attribute - the Java class name of the javax.resource.cci.ActivationSpec implementation class and zero or more optional any-name/any-value attributes for the ActivationSpec parameters. This Extension Element must be an immediate child of a WSDL Operation element (wsdl:operation). The AF creates the ActivationSpec bean class and initializes it with the name/value pairs specified in the jca:operation element using Java reflection. Format for the jca:operation is shown below.

```xml
<binding name="" type="">
  <jca:binding />
  <operation name="">
    <jca:operation>
      ActivationSpec = "ActivationSpec class name"
      ActivationSpec parameters – name/value pair list
    
    </jca:operation>
  </operation>
</binding>
```

Example of a FileAdapter WSDL Binding section for an InboundInteraction is given below:

```xml
<binding name="FileInbound_JCABinding" type="tns:FileInbound_PortType">
  <jca:binding />
  <operation name="Receive">
    <jca:operation>
      ActivationSpec = "oracle.tip.adapter.file.inbound.FileActivationSpec"
      PollingInterval = "2000"
      IsDebatching = "true"
      PublishSize = "2"
      ArchiveDirectory= ".".
      InputDirectory = ".\samples\demos\FileAdapter\ComplexStructure\inputDir"
      MaxMessages = "1"
      RegexInclusion = ".+\.txt"
      RegexExclusion = "doc"/
    
    </jca:operation>
  </operation>
</binding>
```

3.2.2.2 jca:header

Please refer to section 2.2.2.2
### 3.2.2.3 jca:binding

Please refer to section 2.2.2.3

### 3.2.3 Service WSDL Extension

#### 3.2.3.1 jca:address

This Extension Element must be an immediate child of the base WSDL Port element (wsdl:port). The jca:address element has one and only one attribute – AdapterClassName to specify the JCA 1.5 Resource Adapter class name.

```xml
<wsdl:service name="<service_name>">
  <wsdl:port name="<port_name>" binding="tns:<binding_name>">
    <jca:address AdapterClassName="" />
  </wsdl:port>
</wsdl:service>
```

For example, consider a FileAdapter with ResourceAdapter class name as “oracle.tip.adapter.file.FileResourceAdapter”. The WSDL Service extension element for the InboundInteraction will be defined in the following manner for the above Resource Adapter.

```xml
<wsdl:service name="<service_name>">
  <wsdl:port name="<port_name>" binding="tns:<binding_name>">
    <jca:address AdapterClassName="oracle.tip.adapter.file.FileResourceAdapter" />
  </wsdl:port>
</wsdl:service>
```

### 3.3 Deploy Resource Adapter

The JCA 1.5 Resource Adapter is deployed as a JCA 1.0 Resource Adapter with OC4J container. Please see section 2.3 for further details.
4 Appendix

4.1 XML Record Converter API


/* Copyright (c) 2004, Oracle. All rights reserved. */

/*
  DESCRIPTION
  <short description of component this file declares/defines>

PRIVATE CLASSES
  <list of private classes defined - with one-line descriptions>

NOTES
  <other useful comments, qualifications, etc.>

MODIFIED    (MM/DD/YY)
  bstern      08/29/04 - bstern_af_i18n
  bstern      08/24/04 - Creation
*/

/**
   * @version $Header: /home/cvsroot/collaxa/everest/src/classes/oracle/tip/adapter/api/record/XMLRecordConverter.java,v 1.2 2004/09/10 23:18:14 gmi Exp $ *
   * @author bstern
   * @since release specific (what release of product did this appear in)
   */

package oracle.tip.adapter.api.record;

/**
   * Callout interface for XMLRecord conversion thereby enabling third party
   * cci.Record definitions to plugin to and be used by the Adapter Framework.
   */
public interface XMLRecordConverter {

/**
   * Converts a 3rd party Record type to an OraBPEL XML Record
public oracle.tip.adapter.api.record.XMLRecord
    convertToXMLRecord(javax.resource.cci.Record record)
    throws javax.resource.ResourceException;

/**
 * Converts an OraBPEL XML Record to a 3rd party Record type
 */
public javax.resource.cci.Record
    convertFromXMLRecord(oracle.tip.adapter.api.record.XMLRecord xmlRecord)
    throws javax.resource.ResourceException;

/**
 * Instantiates a CCI Record of a 3rd party Record type. This is needed to
 * support Interaction.execute() with 3rd party output Record types
 */
public javax.resource.cci.Record
    createThirdPartyRecord()
    throws javax.resource.ResourceException;

} 

oracle.tip.adapter.api.record

Interface XMLRecord

All Superinterfaces:
    Cloneable, javax.resource.cci.Record, Serializable

public interface XMLRecord extends javax.resource.cci.Record

All adapter interactions will be Invoked with an XMLRecord (Interaction's input Record) and the Interaction will return (or populate) an XMLRecord (Interaction's output Record.) Adapters will raise XMLRecords to initiate an event on behalf of an EIS instance. XMLRecord is the superclass of both.
Method Summary

<table>
<thead>
<tr>
<th>oracle.tip.adapter.api.record.RecordElement</th>
<th>getHeaderRecordElement()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gets the Header RecordElement.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>oracle.tip.adapter.api.record.RecordElement</th>
<th>getPayloadRecordElement()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gets the Payload RecordElement.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>setHeaderRecordElement(oracle.tip.adapter.api.record.RecordElement)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Set the header record element of the XMLRecord.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>setPayloadRecordElement(oracle.tip.adapter.api.record.RecordElement)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Set the header record element of the XMLRecord.</td>
</tr>
</tbody>
</table>

Methods inherited from interface javax.resource.cci.Record

clone, equals, getRecordName, getRecordShortDescription, hashCode, setRecordName, setRecordShortDescription

Method Detail

getHeaderRecordElement

public oracle.tip.adapter.api.record.RecordElement getHeaderRecordElement() |
| Gets the Header RecordElement. |
| Returns:                      |
| RecordElement containing the header data. |

getPayloadRecordElement

public oracle.tip.adapter.api.record.RecordElement getPayloadRecordElement() |
| Gets the Payload RecordElement. |
| Returns:                      |
| RecordElement containing the payload data. |

setHeaderRecordElement

public void setHeaderRecordElement(oracle.tip.adapter.api.record.RecordElement recordElement)
Set the header record element of the XMLRecord.

**Parameters:**

*recordElement* - RecordElement to add.

---

**setPayloadRecordElement**

```java
public void setPayloadRecordElement(oracle.tip.adapter.api.record.RecordElement recordElement)
```

Set the header record element of the XMLRecord.

**Parameters:**

*recordElement* - RecordElement to add.

---

**Definition of RecordElement**

_**oracle.tip.adapter.api.record**_

**Interface RecordElement**

public interface **RecordElement**

XMLRecord is comprised two RecordElements - Headers and Payload. RecordElements contain data.

For example, an AQ adapter's Purchase Order (PO) XMLRecord will have a header RecordElement containing the AQ queuing specific headers, and a payload RecordElement containing the PO payload as e.g. a DOMSource.

**See Also:**

XMLRecord

---

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Result</strong></td>
<td></td>
</tr>
<tr>
<td><code>getData()</code></td>
<td>Get the data for this RecordElement.</td>
</tr>
<tr>
<td>void <code>getDataAs(Result result)</code></td>
<td>Get the data for this RecordElement.</td>
</tr>
<tr>
<td><strong>boolean</strong></td>
<td></td>
</tr>
<tr>
<td><code>isDOMSource()</code></td>
<td>Predicate saying whether RecordElement data is of type DOMSource</td>
</tr>
<tr>
<td><code>isSAXSource()</code></td>
<td>Predicate saying whether RecordElement data is of type SAXSource</td>
</tr>
<tr>
<td><code>isStreamSource()</code></td>
<td>Predicate saying whether RecordElement data is of type StreamSource</td>
</tr>
<tr>
<td><strong>void</strong></td>
<td></td>
</tr>
<tr>
<td><code>setData(Source data)</code></td>
<td>Set the data for this RecordElement (for an inbound Interaction, or for an output Record for an Interaction.)</td>
</tr>
</tbody>
</table>
Method Detail

setData
public void setData(Source data)
Set the data for this RecordElement (for an inbound Interaction, or for an output Record for an Interaction.)
Parameters:
data - The data. Can be instances of DOMSource, SAXSource or StreamSource.
See Also:
XMLRecord.setHeaderRecordElement(oracle.tip.adapter.api.record.RecordElement),
XMLRecord.setPayloadRecordElement(oracle.tip.adapter.api.record.RecordElement), DOMSource, SAXSource, StreamSource

getData
public Result getData()
Get the data for this RecordElement.
Returns:
The data.
See Also:
XMLRecord.getHeaderRecordElement(),
XMLRecord.getPayloadRecordElement()

getDataAs
public void getDataAs(Result result)
Get the data for this RecordElement.
Parameters:
result - Handle to desired Result type (instance of DOMResult, SAXResult, StreamResult).
See Also:
XMLRecord.getHeaderRecordElement(),
XMLRecord.getPayloadRecordElement()

isDOMSource
public boolean isDOMSource()
Predicate saying whether RecordElement data is of type DOMSource
Returns:
boolean Is RecordElement data of instance DOMSource
See Also:
setData(javax.xml.transform.Source), DOMSource
isSAXSource
public boolean isSAXSource()
    Predicate saying whether RecordElement data is of type SAXSource
    Returns:
    boolean Is RecordElement data of instance SAXSource
    See Also:
    [setData(javax.xml.transform.Source), SAXSource]

isStreamSource
public boolean isStreamSource()
    Predicate saying whether RecordElement data is of type StreamSource
    Returns:
    boolean Is RecordElement data of instance StreamSource
    See Also:
    [setData(javax.xml.transform.Source), StreamSource]