Oracle® Fusion Middleware

Administering the JMS Resource Adapter for Oracle WebLogic Server

12*c* (12.1.2)

E28959-02

August 2013

This document describes how to configure and manage a JMS Resource Adapter hosted by a third-party application server to interoperate with WebLogic Server using WebLogic JMS.

ORACLE

Oracle Fusion Middleware Administering the JMS Resource Adapter for Oracle WebLogic Server, 12*c* (12.1.2)

E28959-02

Contents

5 Understanding Message Consumption

6 Sending Outbound JMS Messages

7 Configuring Destinations and Naming Contexts

12 JMS RA Deployment Descriptor Elements and Properties

Preface

This preface describes the document accessibility features and conventions used in this guide—*Administering the JMS Resource Adapter for Oracle WebLogic Server*.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1 Introduction and Roadmap

This chapter describes the contents and organization of this guide—*Administering the JMS Resource Adapter for Oracle WebLogic Server*.

1.1 Document Scope and Audience

This document is a resource for system administrators who use WebLogic JMS in a foreign application server and want to interoperate using a Java EE Connector Architecture resource adapter that integrates a WebLogic JMS client.

It is assumed that the reader is familiar with programming in Java Platform, Enterprise Edition (Java EE), Java EE Connector Architecture version 1.6 resource adapters, and JMS concepts. This document emphasizes the value-added features provided by WebLogic Server and key information about how to configure and use the WebLogic JMS Resource adapter (JMS RA).

1.2 Guide to This Document

- This chapter, Chapter 1, "Introduction and Roadmap," describes the organization and scope of this guide, including new features and related documentation.
- Chapter 2, "Understanding the WebLogic JMS Resource Adapter," provides an overview of WebLogic JMS Resource Adapter (JMS RA) components, concepts, and functionality.
- Chapter 3, "Administering the JMS RA," explains the components, design options, and other prerequisite considerations needed to use the JMS RA to interoperate using WebLogic JMS in a foreign application server
- Chapter 4, "Administering the JMS RA on Oracle GlassFish Server," describes additional configuration information and considerations when using deploying the JMS RA on the Oracle GlassFish Server.
- Chapter 5, "Understanding Message Consumption," describes how to configure the JMS RAs `ra.xml` file to configure MDBs to asynchronously consume WebLogic JMS messages in a foreign application server as inbound messages.

- Chapter 6, "Sending Outbound JMS Messages," describes how to send JMS messages using the JMS RA.
- Chapter 7, "Configuring Destinations and Naming Contexts," provides information on configuring `adminobject` elements to define destinations and naming contexts for inbound and outbound communication.
- Chapter 8, "Understanding Resource Providers," describes how to use and configure Resource Providers. A resource provider defines the JNDI properties that allow the JMS RA to connect to the WebLogic JMS provider.
- Chapter 9, "Understanding Transaction Processing," describes transaction processing and recovery when using the JMS RA to interoperate between a foreign application server and WebLogic Server.
- Chapter 10, "Understanding Failure Management," describes how the JMS RA responds to WebLogic Server and foreign application server failures.
- Chapter 11, "Securing JMS RA Connections," describes security considerations for the JMS RA.
- Chapter 12, "JMS RA Deployment Descriptor Elements and Properties," provides information about the WebLogic JMS RA deployment descriptor file, `ra.xml`.

1.3 Related Documentation

For information on topics related to configure and use the JMS RA, see the following documents:

- *Administering JMS Resources for Oracle WebLogic Server* is a guide to configuring and managing WebLogic JMS resources.
- *Developing JMS Applications for Oracle WebLogic Server* is a guide to developing WebLogic JMS applications.
- *Tuning Performance of Oracle WebLogic Server* provides information on how to monitor performance and tune the components in a WebLogic Server.
- *Developing Applications for Oracle WebLogic Server* is a guide to developing WebLogic Server applications.
- *Deploying Applications to Oracle WebLogic Server* is the primary source of information about deploying WebLogic Server applications.
- *Developing Resource Adapters for Oracle WebLogic Server* contains information on WebLogic resource adapters and the WebLogic Server implementation of the Java EE Connector Architecture.

1.4 Samples and Tutorials for the JMS Developer

In addition to this document, Oracle provides a variety of code samples and tutorials for JMS developers. The examples and tutorials illustrate WebLogic Server JMS in action, and provide practical instructions on how to perform key JMS development tasks.

Oracle recommends that you run some or all of the JMS examples before developing your own JMS applications.

1.4.1 JMS Resource Adapter Example

This example demonstrates how to utilize the WebLogic JMS Resource Adapter (JMS RA) deployed on a foreign application server to interoperate with the WebLogic JMS service. This example is included when you install the examples in your WebLogic distribution. See Section 3.5, "JMS RA Example"

1.4.2 Avitek Medical Records Application (MedRec) and Tutorials

MedRec is an end-to-end sample Java EE application shipped with WebLogic Server that simulates an independent, centralized medical record management system. The MedRec application provides a framework for patients, doctors, and administrators to manage patient data using a variety of different clients.

MedRec demonstrates WebLogic Server and Java EE features, and highlights Oracle-recommended best practices. MedRec is optionally installed with the WebLogic Server installation. You can start MedRec from the *ORACLE_HOME*`\user_projects\domains\medrec` directory, where *ORACLE_HOME* is the directory you specified as the Oracle Home when you installed Oracle WebLogic Server. For more information, see "Sample Applications and Code Examples" in *Understanding Oracle WebLogic Server*.

MedRec includes a service tier comprised primarily of Enterprise Java Beans (EJBs) that work to process requests from web applications, web services, and workflow applications, and future client applications. The application includes message-driven, stateless session, stateful session, and entity EJBs.

1.5 New and Changed Features in This Release

For a comprehensive listing of the new WebLogic Server features introduced in this release, see *What's New in Oracle WebLogic Server*.

2

Understanding the WebLogic JMS Resource Adapter

This chapter describes an overview of WebLogic JMS Resource Adapter components, concepts, and functionality.

- Section 2.1, "JMS RA Overview"
- Section 2.2, "Supported Application Servers"
- Section 2.3, "Supported WebLogic Server Destinations"
- Section 2.4, "Location of the JMS RA in the WebLogic Server Distribution"

2.1 JMS RA Overview

WebLogic Server provides a Java EE Connector Architecture version 1.6 compliant resource adapter called the JMS Resource Adapter (JMS RA) to provide an integration of a WebLogic JMS client with supported foreign application servers.

The JMS RA includes the following features:

- Implements the JCA outbound and inbound contract for JMS.
- JNDI mapping to reference JMS connection factories and destinations.
- MDB integration (including dynamic adjustment to changing message load)
- JMS connection pooling
- Lazy resolution of JMS operations (including start order independence, tolerance of dynamic management such as starts and stops of JMS providers, and connection retries in case of provider failure). See Section 9.2, "Lazy Enlistment of Connections in a Transaction."
- Cluster-capable XA support for WebLogic JMS that transparently integrates with non-WebLogic Transaction Managers and correctly recovers from all typical failure conditions. See Section 10.3, "Transaction Recovery."
- Support for asynchronous message processing that ensures all active members of a distributed destination are always serviced – no trapped messages. In addition, asynchronous messaging provides advanced publish/subscribe options, such as a single logical durable subscription partitioned across a distributed topic.
- Support for WebLogic JMS extensions that allow you to cast adapter wrapped objects to WebLogic JMS interfaces. See Section 3.4, "Support for the weblogic.jms.extension API."

- Support for the WebLogic security model. See Section 11, "Securing JMS RA Connections."
- Advanced poison message handling. The JMS RA can be configured to automatically redirect messages that have been redelivered multiple times to a designated error destination. See Section 5.1.4, "Using an Exception Queue."
- Simplified configuration of multiple destination JNDI mappings through the use of a single destination context RA administrative object. This allows applications to directly reference any number of destinations and avoid the need to configure multiple administrative objects and `resource-env` references. See Section 7.2, "Using Automatic Destination Wrapping."

2.2 Supported Application Servers

This release of the JMS RA is supported for deployment on Oracle GlassFish version 3.1 and higher.

Note: Deploying the JMS RA is not supported on any Oracle WebLogic Server release. The JMS RA fails gracefully if deployed on WebLogic Server.

2.3 Supported WebLogic Server Destinations

This release of the JMS RA supports foreign application server interoperability with destinations in Oracle WebLogic Server releases 12.1.2 and higher.

2.4 Location of the JMS RA in the WebLogic Server Distribution

The `wljmsra.jar` file is located in the `WL_HOME\server\lib` directory of your WebLogic Server installation.

3

Administering the JMS RA

This chapter describes the JMS RA implementation, including the main components, design options, and other prerequisite considerations needed to use the JMS RA to interoperate using WebLogic JMS in a foreign application server.

- Section 3.1, "JMS RA Components"
- Section 3.2, "How to Integrate the JMS RA in Your JMS Environment"
- Section 3.3, "General Limitations and Considerations"
- Section 3.4, "Support for the weblogic.jms.extension API"
- Section 3.5, "JMS RA Example"

3.1 JMS RA Components

The JMS RA has the following basic components:

- `ra.xml` file: An xml file used to configure the JMS RA to interoperate between foreign application servers and WebLogic JMS.
- `wlthint3client.jar` file: The WebLogic Thin T3 java client is a light-weight client that provides integration between applications running on foreign application servers and WebLogic Server. One common use case is integration with WebLogic JMS destinations. See "Understanding the WebLogic Thin T3 Client" in *Developing Stand-alone Clients for Oracle WebLogic Server*.
- Classes and JAR files for Oracle Wallet Support:
  - `oraclepki.jar`
  - `osdt_cert.jar`
  - `osdt_core.jar`
  - `WalletUtilWrapper.class`

  See Section 11.4, "Securing Credentials with Oracle Wallet."
- `weblogic.jms.ra.jar` file: The WebLogic Server Java EE Connector Architecture version 1.6 compliant resource adapter implementation.

3.2 How to Integrate the JMS RA in Your JMS Environment

The following section provides a basic overview of the steps required to integrate the JMS RA with a foreign application server, allowing applications to interoperate with WebLogic JMS.:

- The application code needs to be JMS compliant and may utilize supported `weblogic.jms.extensions`. For information on JMS RA features, support, and limitations, see Section 2, "Understanding the WebLogic JMS Resource Adapter" and Section 3.3, "General Limitations and Considerations." For basic information on programing WebLogic JMS, see "Understanding WebLogic JMS" in *Developing JMS Applications for Oracle WebLogic Server*.
- Configure the `ra.xml` file for your environment:
  - Section 5, "Understanding Message Consumption"
  - Section 6, "Sending Outbound JMS Messages"
  - Section 7, "Configuring Destinations and Naming Contexts"
  - Section 8, "Understanding Resource Providers"
- Deploy and manage the JMS RA. Use the foreign application server's native support for management tasks such as deployment, configuration, and monitoring. See:
  - Section 4, "Administering the JMS RA on Oracle GlassFish Server"

  The Oracle Fusion Middleware Error Messages Reference provides information about the error messages you may encounter when using the JMS RA and other Oracle Fusion Middleware components.

3.3 General Limitations and Considerations

This section provides information on general limitations and considerations when using the JMS RA:

- If two or more JMS RAs are deployed on the same application server, all the deployments must use the same WebLogic release version of the `wljmsra.rar` file. Different release versions may contain different classes due to changes in features and bug fixes and it is not possible to predict which version of the JMS RA's classes are loaded by the application server at any given time.
- The JMS RA does not automatically unsubscribe (remove) a durable subscriber if the listening MDB becomes unavailable. Although WebLogic Server MDBs support the ability to unsubscribe, they also create and name the subscription. Foreign application servers typically allow users to specify subscription names in MDB configurations, making it difficult to determine if it is appropriate to unsubscribe a given durable subscriber. Even it can be determined that it is appropriate to remove a given subscription, the JCA contract does not provide a standard implementation to provide the callback interface to remove a subscriber.
- Automatic JMS Client Failover is not supported. See Section 6.3, "JMS RA Outbound Connection Limitations."
- Applications utilizing the WebLogic JMS RA utilize WebLogic JMS connections that are associated with a foreign application server's connection manager. These managed JMS connections are created, used, and retuned to the foreign application server's connection pool for reuse by other applications as outlined in various Java EE specifications. This has the following limitations:
  - Using `setClientID` on a connection is not allowed. Once a `ClientID` is set on a connection, it can not be removed and every application that reuses the connection unknowingly has a `ClientID` value. An application using such a connection is also prevented form calling `setClientID` as this operation has already been performed on the connection.

  - As specified by the JEE 6 specification, other methods that may interfere with connection management by the container, such as `setExceptionListener`, are not supported.
  - Set connection properties, such as `setSubscriptionSharingPolicy`, as part of the connection factory settings. Otherwise, you will need to ensure that the connection/session is created in the managed connection pool before attempting to set the property directly on the connection.
- For a message-driven bean's message listener, only `REQUIRED` and `NOT_SUPPORTED` may be used for the value of the `<trans-attribute>` element. See http://docs.oracle.com/javaee/6/api/javax/ejb/TransactionAttributeType.html.

3.4 Support for the weblogic.jms.extension API

The following sections describe how this release of the JMS RA supports the `weblogic.jms.extensions` API.

- Section 3.4.1, "Supported weblogic.jms.extension Interfaces"
- Section 3.4.2, "Supported weblogic.jms.extension Classes"

3.4.1 Supported weblogic.jms.extension Interfaces

The JMS RA uses custom wrapper classes to automatically implement the supported `weblogic.jms.extensions` interfaces and provide support for WebLogic features such as Unit-of-Order (UOO) and message scheduling. The JMS RA `ra.xml` file is pre-populated with entries corresponding to the correct WebLogic JMS wrapper classes so that you do not need any additional configuration to access supported WebLogic JMS features.

Note: Use WebLogic Connection Factories when using `weblogic.jms.extensions`. See Section 6.2.1, "JMS RA Connection Factories."

This release of the JMS RA supports the following `weblogic.jms.extensions` interfaces

- weblogic.jms.extensions.WLConnection
- weblogic.jms.extensions.WLSession
- weblogic.jms.extensions.WLQueueSession
- weblogic.jms.extensions.WLTopicSession
- weblogic.jms.extensions.WLDestination
- weblogic.jms.extensions.WLMessageProducer
- weblogic.jms.extensions.WLMessage
- weblogic.jms.extensions.XMLMessage

3.4.2 Supported weblogic.jms.extension Classes

All `weblogic.jms.extensions` are supported.

3.5 JMS RA Example

Oracle provides a JMS RA example for you to review when you install the examples in your WebLogic distribution. This example demonstrates how to utilize the JMS RA deployed on a foreign application server to interoperate with the WebLogic JMS service using a simple employee clock-in application.

The example illustrates how to:

- Configure and use a WebLogic Server cluster
- Configure and use WebLogic connection factories.
- Configure and use distributed topics and queues.
- Deploy the JMS RA to supported foreign application server. See Section 2.2, "Supported Application Servers."
- Configure JMS RA `config-properties` which include:
  - A WebLogic JNDI context for the WebLogic Server providing the JMS service.
  - An `adminobject` element which map the local JNDI name `sample/destination/queue` to the destination bound to the WebLogic JNDI as `DistributedQueue`
  - An `adminobject` element which maps the local JNDI name `sample/destination/topic` to the destination bound in the WebLogic JNDI as `DistributedTopic`
  - A connection factory bound in the local JNDI name as sample/factory that maps to the connection factory bound to WebLogic's JNDI as weblogic.jms.ConnectionFactory.
- Configure `resource-ref` and `resource-env-ref` elements in a servlet's `web.xml` that map to the local JNDI names defined for the JMS RA's connection factory and destinations.
- Configure `activation-config` elements required to allow message-driven beans (MDBs) to consume inbound messages.

The JMS RA example is located in *`EXAMPLES_HOME`*`\wl_server\examples\src\examples\jms\resourceAdapter`, where *`EXAMPLES_HOME`* represents the directory in which the WebLogic Server code examples are configured. For more information, see "Sample Applications and Code Examples" in *Understanding Oracle WebLogic Server*.

4

Administering the JMS RA on Oracle GlassFish Server

This chapter describes additional configuration information and considerations when using deploying the JMS RA on the Oracle GlassFish Server.

This chapter includes the following sections:

- Section 4.1, "Securing Credentials using the JCA Container"
- Section 4.2, "Oracle GlassFish Server Support for Lazy Connection Enlistment."
- Section 4.3, "Limitations and Considerations when Administering Oracle GlassFish Server"
- Section 4.4, "Oracle GlassFish Server References"

4.1 Securing Credentials using the JCA Container

When possible, Oracle recommends using the Oracle GlassFish Server JCA container to provide methods to set credentials using secure methods.

For more information, see Section 11, "Securing JMS RA Connections."

4.2 Oracle GlassFish Server Support for Lazy Connection Enlistment

Oracle GlassFish Server supports Section 9.2, "Lazy Enlistment of Connections in a Transaction."

4.3 Limitations and Considerations when Administering Oracle GlassFish Server

The following section describes the limitations and considerations when deploying the JMS RA on Oracle GlassFish Server:

- The Glassfish `XAResource` timeout value is 30 seconds by default and is independent of the transaction timeout. If a user requires a longer `XAResource` timeout, they must configure the Glassfish system property `server-config.transaction-service.property.xaresource-txn-timeout` to a value equal to their transaction timeout value.
- There is a known GlassFish 3.1.x issue where outbound MDB connections remain associated with a rolled back transaction.

  You may receive a `javax.transaction.xa.XAException: The resource already has an active association with a transaction` message if

your application uses `javax.ejb.MessageDrivenContext.setRollbackOnly()` within an XA enabled MDB's `onMessage` method which in turns uses an XA backed outbound connection factory. This exception is thrown after a rollback and the associated connection is taken from the connection pool to be reused in another transaction. If you have a licensed version of GlassFish, contact your Oracle Glassfish Support representative. If you have an open source copy, a patch is available at http://java.net/jira/browse/GLASSFISH-19094.

- You may receive a `ClassNotFoundException` when deploying the JMS RA to a Glassfish 3.1 cluster environment. If so, copy the `wlthint3client.jar` file packaged in the `wljmsra.rar` file to the <*Glassfish_install_dir*>`/lib` directory. See http://java.net/jira/browse/GLASSFISH-19111.
- You may receive the following exception `Failed to generate class for weblogic.messaging.dispatcher.FastDispatcherImpl_12120_WLStub` if you undeploy and deploy a WAR file multiple times. The workaround is to restart the GlassFish server.
- For additional information on known issues and any available workarounds for Oracle GlassFish Server 3.1 software, see "Oracle GlassFish Server 3.1.2 and 3.1.2.2 Release Notes" at http://docs.oracle.com/cd/E26576_01/doc.312/e24939/release-notes.htm.

4.4 Oracle GlassFish Server References

This section provides links to Oracle GlassFish Server reference documentation:

- GlassFish Download: http://glassfish.java.net/public/downloadsindex.html#top
- http://docs.oracle.com/cd/E18930_01/html/821-2432/index.html

5 Understanding Message Consumption

This chapter describes how to configure the JMS RAs `ra.xml` file to configure MDBs to asynchronously consume WebLogic JMS messages in a foreign application server as inbound messages.

- Section 5.1, "Configuring MDBs to Consume Inbound Messages"
- Section 5.2, "Configuring Advanced WebLogic JMS Resources"
- Section 5.3, "Best Practices for Inbound Communication"

5.1 Configuring MDBs to Consume Inbound Messages

Applications which require MDBs to asynchronously consumes messages from a WebLogic destination use the JMS RAs inbound implementation. The behavior of the MDBs is determined by configuring property values in the following files before deploying your application:

- Section 5.1.1, "Configuring inbound-resourceadapter Properties in the ra.xml File"
- Section 5.1.2, "Configuring activation-config Properties in the ejb-jar.xml File"
- Section 5.1.3, "Thread Management"

5.1.1 Configuring inbound-resourceadapter Properties in the ra.xml File

The following sections provide information on how to define `inbound-resourceadapter` properties for the inbound configuration in the `ra.xml` file:

- Section 5.1.1.1, "Required activation-config Properties"
- Section 5.1.1.2, "Optional activation-config Properties"
- Section 5.1.1.3, "Example inbound-resourceadapter Configuration"

5.1.1.1 Required activation-config Properties

The required `inbound-resourceadapter` properties are specified in the WebLogic JMS Resource-Adapter's `ra.xml` file located in your WebLogic Server distribution, see Section 2.4, "Location of the JMS RA in the WebLogic Server Distribution."

Required JMS properties include:

- `ConnectionFactory`
- `destination`
- `destinationType`

The values for `ConnectionFactory`, `destination`, and `destinationType` are defined as `activation-config` properties in the `ejb-jar.xml` file. The JNDI names configured in the `ejb-jar.xml` must also map to the JNDI names assigned to the JMS RA values of the `connection-definition` and `adminobject` elements. See:

- Section 5.1.2, "Configuring activation-config Properties in the ejb-jar.xml File"
- Section 6.2.2, "Configuring JMS RA Connection Factory Properties."
- Section 7, "Configuring Destinations and Naming Contexts."

5.1.1.2 Optional activation-config Properties

The JMS RA supports a number of additional `activation-config` properties. For more information, see Section 5.1.2, "Configuring activation-config Properties in the ejb-jar.xml File" and Section 12.4, "JMS RA Inbound Properties."

5.1.1.3 Example inbound-resourceadapter Configuration

The following code example shows an `inbound-resourceadapter` configuration.

Example 5–1 Example inbound-resourceadapter Configuration

```
. . .
<inbound-resourceadapter>
      <messageadapter>
        <messagelistener>
          <messagelistener-type>
            javax.jms.MessageListener
          </messagelistener-type>
          <activationspec>
            <activationspec-class>
              weblogic.jms.ra.ActivationSpecImpl
            </activationspec-class>
            <required-config-property>
              <config-property-name>ConnectionFactory</config-property-name>
            </required-config-property>
            <required-config-property>
              <config-property-name>Destination</config-property-name>
            </required-config-property>
            <required-config-property>
              <config-property-name>DestinationType</config-property-name>
            </required-config-property>
          </activationspec>
        </messagelistener>
      </messageadapter>
</inbound-resourceadapter>
. . .
```

5.1.2 Configuring activation-config Properties in the ejb-jar.xml File

The following sections provide information on how to configure `activation-config` properties in the `ejb-jar.xml` file:

- Section 5.1.2.1, "The ejb-jar.xml file and Annotations"
- Section 5.1.2.3, "Configuring Optional activation-config Properties in the ejb-jar.xml File"
- Section 5.1.2.4, "Example ejb-jar.xml File with JMS RA activation-config Properties"

5.1.2.1 The ejb-jar.xml file and Annotations

For elements in `ejb-jar.xml`, see the schema at http://java.sun.com/xml/ns/javaee/ejb-jar_3_0.xsd.

The `ejb-jar.xml` is optional starting from EJB 3.0. Annotations can be used to declare metadata in place of descriptor elements. See "Using EJB 3.1 Compliant MDBs" in *Developing Message-Driven Beans for Oracle WebLogic Server*.

5.1.2.2 Configuring Required activation-config Properties in the ejb-jar.xml File

You must configure an `activation-config-property` property for every `required-config-property` property in the `inbound-resourceadapter` of the `ra.xml`.

Typical required JMS properties include:

- `ConnectionFactory`: The JNDI location of a JMS Connector connection factory.
- `destination`: The JNDI location of a JMS Connector destination.
- `destinationType` where the type is one of the following:
  - `javax.jms.Topic`
  - `javax.jms.Queue`
  - `javax.jms.Destination`

In addition, If your MDB application utilizes optional `activation-config` properties, the foreign application server may require these optional `activation-config` properties be defined in `ra.xml` as `required-config-property` elements. Please refer to documentation provided by the vendor of your application server.

5.1.2.2.1 Example Queue Configuration

The following example shows `activation-config` properties for a queue configuration. In this example, the MDB that dequeues messages from a queue with the JNDI name `wljmsra/queue` using a connection from the connection factory with the JNDI name `wljmsra/xacf`.

```
. . .
<activation-config-property>
  <activation-config-property-name>
    ConnectionFactory
  </activation-config-property-name>
  <activation-config-property-value>
    wljmsra/xacf
  </activation-config-property-value>
</activation-config-property>
<activation-config-property>
  <activation-config-property-name>
    Destination
  </activation-config-property-name>
  <activation-config-property-value>
    wljmsra/queue
  </activation-config-property-value>
</activation-config-property>
<activation-config-property>
  <activation-config-property-name>
    DestinationType
  </activation-config-property-name>
  <activation-config-property-value>
```

```
    javax.jms.Queue
  </activation-config-property-value>
</activation-config-property>
. . .
```

5.1.2.2.2 Example Topic Configuration

The following example shows `activation-config` properties for a topic configuration. In this example, the MDB that dequeues messages from a topic with the JNDI name wljmsra/pdtopic1 using a connection from the connection factory with the JNDI name wljmsra/txacf1.

```
. . .
<activation-config-property>
  <activation-config-property-name>
    ConnectionFactory
  </activation-config-property-name>
  <activation-config-property-value>
    wljmsra/txacf1
  </activation-config-property-value>
</activation-config-property>
<activation-config-property>
  <activation-config-property-name>
    Destination
  </activation-config-property-name>
  <activation-config-property-value>
    wljmsra/pdtopic1
  </activation-config-property-value>
</activation-config-property>
<activation-config-property>
  <activation-config-property-name>
    DestinationType
  </activation-config-property-name>
  <activation-config-property-value>
    javax.jms.Topic
  </activation-config-property-value>
</activation-config-property>
. . .
```

5.1.2.3 Configuring Optional activation-config Properties in the ejb-jar.xml File

Configure any additional properties needed to achieve the appropriate MDB behavior for your environment. See Section 12.4, "JMS RA Inbound Properties."

5.1.2.4 Example ejb-jar.xml File with JMS RA activation-config Properties

The following example shows an `ejb-jar.xml` file with JMS RA `activation-config` properties.

Example 5–2 ejb-jar.xml file with JMS RA activation-config Properties

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar xmlns="http://java.sun.com/xml/ns/javaee">
    <display-name>WebLogic RA Demo</display-name>
    <enterprise-beans>
        <message-driven>
            <display-name>My queue MDB</display-name>
            <ejb-name>queueMDB</ejb-name>
            <ejb-class>jms.ra.DisQueueMDB</ejb-class>
```

```
                <messaging-type>javax.jms.MessageListener</messaging-type>
                <transaction-type>Container</transaction-type>
                <activation-config>
                    <activation-config-property>
                        <activation-config-property-name>
                                ConnectionFactory
                        </activation-config-property-name>
                        <activation-config-property-value>
                                java:sample/factory
                        </activation-config-property-value>
                    </activation-config-property>
                    <activation-config-property>
                        <activation-config-property-name>
                                Destination
                        </activation-config-property-name>
                        <activation-config-property-value>
                                java:sample/destination/queue
                        </activation-config-property-value>
                    </activation-config-property>
                    <activation-config-property>
                        <activation-config-property-name>
                                DestinationType
                        </activation-config-property-name>
                        <activation-config-property-value>
                                javax.jms.Queue
                        </activation-config-property-value>
                    </activation-config-property>
                    <activation-config-property>

<activation-config-property-name>UserName</activation-config-property-name>

<activation-config-property-value></activation-config-property-value>
                </activation-config-property>
                <activation-config-property>

<activation-config-property-name>Password</activation-config-property-name>

<activation-config-property-value></activation-config-property-value>
                </activation-config-property>
                <activation-config-property>

<activation-config-property-name>ClientId</activation-config-property-name>

<activation-config-property-value>queueMDB</activation-config-property-value>
                </activation-config-property>
                <activation-config-property>

<activation-config-property-name>MessageSelector</activation-config-property-name>

<activation-config-property-value></activation-config-property-value>
                </activation-config-property>
            </activation-config>
        </message-driven>
       . . .
      </message-driven>
    </enterprise-beans>
   . . .
</ejb-jar>
```

For additional `ejb-xml.jar` file examples, review the Resource Adapter example in your WebLogic Server distribution.

5.1.3 Thread Management

The JMS RA provides three properties to manage the threads in destinations used by MDBs consuming inbound messages.

- `minListenerThreads`: The minimum number of listener threads created for an individual physical destination. See `minListenerThreads`.
- `maxTotalListenerThreads`: The maximum number of listener threads available for a destination. See `maxTotalListenerThreads`.
- `maxListenerThreads`: The maximum number of listener threads created for an individual physical destination within a destination. See `maxListenerThreads`.

5.1.3.1 Setting the Maximum Threads for a Physical Destination

This section provides information on how to configure the maximum number of threads (`maxListenerThreads`) for a destination:

- Queues: Using more than one thread may be useful in increasing the rate at which messages are consumed.
- Topics: This value should always be 1. Each listener thread gets its own session and `TopicSubscriber`.
  - Durable subscribers: It is an error to have more than one subscriber with the same subscription name.
  - Nondurable subscribers: This value should always be 1. Creating more threads creates more subscribers which translates into more copies of each message to process.

5.1.3.2 Setting the Maximum Threads for a Distributed Destination

The `maxTotalListenerThreads` property allows you to limit or to provide addition processing threads for a distributed destination.

- If the value of `maxTotalListenerThreads` is less than the number of physical destinations in the associated distributed destination, the JMS RA uses a value equal to the number of physical destinations in the distributed destination and logs a warning message.
- If the Foreign application server cannot provide threads equal to or more than the number of physical destinations in the distributed destination, the JMS RA logs a warning message.
- The JMS RA implements a fairness policy for allocating threads whereby any individual physical destination which currently being serviced by less than `maxListenerThreads` and needs more threads can reallocate threads from any other physical destination in the same distributed destination which has at least 2 more threads that it does. This fairness policy is independent of the foreign application server's `WorkManager` which may not grant the JMS RA's requests for additional `maxTotalListenerThreads` new threads.

  The process of reallocating threads from one destination to another takes some time, as both the determination of the thread deficiency and the transfer of the existing thread only take place between the processing of messages (between calls to `onMessage`). Since two threads are involved, the transfer process take up to two `onMessage` processing times.

5.1.4 Using an Exception Queue

The JMS RA allows users to configure an exception queue for inbound communications to handle poison messages from the inbound MDB queue. When `UseExceptionQueue=true`, messages that would otherwise be discarded are sent to the exception queue. Messages are normally sent to an exception queue when the `maxDeliveryCount` value is exceeded. See `maxDeliveryCount`.

Messages are processed to the exception queue using the following rules:

- Messages are not set directly to the exception queue.
- A new message of the same type is created.
  - The properties and body from the original message are copied to the new message.

    To prevent the original headers from being overwritten by the resource provider, each is copied to a `GJRA_CopyOfJMS{`*`Header`*`}` property. As `javax.jms.Destination` is not a valid property type, each destination header is translated into a descriptive message. Note: `JMSX*` properties are not converted. For example, `JMSXDeliveryCount`.

    Note: If any part of the copy process fails, processing continue with the next rule. For `Bytes/Map/Stream` message types, this may result in part of the message body is copied and part of the message body is not.

  - If the copy process is 100% successful, the boolean property `GJRA_CopySuccessful` is added with the value `true`.
  - A string property called `GJRA_DeliveryFailureReason` is added which contains the description of why the message was not delivered.
  - If the MDB `onMessage` method generated an exception immediately prior to the delivery failure, a string property called `GJRA_onMessageExceptions` is added which contains the exception information.
- The copy of the original message is sent to the exception queue.

  Note: Only one attempt is made to send the copy of the original message to the exception queue. If this attempt fails, the message is discarded without being placed in the exception queue. See `includeBodiesInExceptionQueue` for more information.

The connection factory used for the primary destination is also used for the exception queue. If the primary destination (specified by the `Destination` property) is a topic, then the connection factory must support both queues and topics. For example, the `<connectionfactory-interface>` element must be either `javax.jms.ConnectionFactory` or `javax.jms.XAConnectionFactory`.

5.1.5 Setting userName/password

The `userName/password` properties allow you to pass authentication parameters to the resource provider. When neither of these properties are set, connections used for the MDB's inbound message handling are created using the no-argument version of

the createConnection method. When one or both are set, the userName/password properties are passed to the createConnection method as the user/password arguments. If only one of the properties is not set, null is used to replace that property value in the createConnection argument list.)

5.2 Configuring Advanced WebLogic JMS Resources

The following sections provide information on how to configure advanced message processing for inbound messages:

- Section 5.2.1, "The JMS RA and Sharable Subscriptions"
- Section 5.2.2, "Using Ordered Message Processing"
- Section 5.2.3, "Design Strategies when Consuming from DistributedTopics"
- Section 5.2.4, "Consuming from Stand-alone (Non-distributed) Topics"

5.2.1 The JMS RA and Sharable Subscriptions

The JMS RA uses a SHARABLE Subscription Sharing Policy and UNRESTRICTED Client ID Policy when processing inbound messages, overriding any configured connection factory settings. A subscriptionName must be provided for durable subscriptions as the JMS RA's MDBs do not generate subscription names. Subscriptions are shared when:

- Non-durable: Consumers on the same distributed topic can share a non-durable subscriptions only if they have the same clientID and messageSelector.
- Durable: Consumers on the same distributed topic can share a durable subscriptions only they have the same clientID, message selector, and subscriptionName.

Always configure a clientId to ensure the MDB consumes incoming messages. For example:

```
. . .
<activation-config>
  <activation-config-property>

<activation-config-property-name>clientId</activation-config-property-name>
    <activation-config-property-value>myMDB</activation-config-property-value>
  </activation-config-property>
</activation-config>
. . .
```

See "Configure Shared Subscriptions" in *Administering JMS Resources for Oracle WebLogic Server*.

5.2.2 Using Ordered Message Processing

If your application requires single-threaded processing of subscription messaging, configure your application to use WebLogic JMS Unit-of-Ordering (UOO) processing. See "Using Message Unit-of-Order" in *Developing JMS Applications for Oracle WebLogic Server*.

5.2.3 Design Strategies when Consuming from DistributedTopics

The following sections provide information on design strategies that can be used to develop high availability applications using distributed topics:

- Section 5.2.3.1, "Replicated vs Partitioned Distributed Topics"
- Section 5.2.3.3, "One-Copy-Per-Server Design Strategy for Distributed Topics"
- Section 5.2.3.2, "One-Copy-Per-Application Design Strategy for Distributed Topics"

5.2.3.1 Replicated vs Partitioned Distributed Topics

This section provides an overview of replicated and partitioned distributed topics, both are supported for inbound message consumption.

- Replicated Distributed Topics: All physical topic members receive each message sent. When a message arrives at one of the physical topic members, a copy of the message is automatically internally forwarded to the other members of the topic.
- Partitioned Distributed Topics: The distributed topic member receiving the message is the only member that is aware of the message. The message is not forwarded to other members, and subscribers on other members do not get a copy of the message. Incoming messages can be load balanced among the distributed topic members using the `JMS Affinity` and `Load Balance` attributes. See "Load Balancing Partitioned Distributed Topics" in *Administering JMS Resources for Oracle WebLogic Server*.

5.2.3.2 One-Copy-Per-Application Design Strategy for Distributed Topics

`One-Copy-Per-Application` is the default design pattern available and has the following characteristics:

- Each application as a whole (that is all instances of the application together) receives one copy of each message that is published to the DT. That is each instance only receives a subset of the messages that are sent to the DT.
- An `UNRESTRICTED` Client ID Policy
- An `SHARABLE` Subscription Sharing Policy
- Uses the same subscription name if the subscribers are durable
- All consumers subscribe to the same topic instance (or member of a DT)

5.2.3.2.1 Implementing One-Copy-Per-Application To implement the `One-Copy-Per-Application` design strategy, you must specify the `ProviderProperties` property in your EJB with a value of `TopicMessageDistributionMode=One-Copy-Per-Application` in the `activation-config` element as shown in Section 5.2.3.2.2, "Example One-Copy-Per-Application EJB Configuration."

Note: If you do not specify `TopicMessageDistributionMode=One-Copy-Per-Server`, the JMS RA defaults to `TopicMessageDistributionMode=One-Copy-Per-Application` to avoid message duplication.

5.2.3.2.2 Example One-Copy-Per-Application EJB Configuration The following code snippet from an `ejb-jar.xml` file implements `One-Copy-Per-Application` message processing:

```
. . .
<mdb-resource-adapter>
```

```
  <resource-adapter-mid>wljmsra</resource-adapter-mid>
    <activation-config>
      <activation-config-property>
        <activation-config-property-name>ClientId</activation-config-property-name
>
        <activation-config-property-value>RDT2MDB</activation-config-property-valu
e>
      </activation-config-property>
      <activation-config-property>
        <activation-config-property-name>ProviderProperties</activation-config-pro
perty-name>
        <activation-config-property-value>TopicMessageDistributionMode=One-Copy-Pe
r-Application</activation-config-property-value>
      </activation-config-property>
    </activation-config>
</mdb-resource-adapter>

<!-- Mapping a Queue admin-object to the Resource-Adapter name -->
  <resource-env-ref>
    <resource-env-ref-name>jms/ResultTopic</resource-env-ref-name>
    <jndi-name>wljmsra/rtopic1</jndi-name>
  </resource-env-ref>

<!-- Mapping a Connection Factory to the Resource-Adapter name -->
  <resource-ref>
    <res-ref-name>jms/ResultXACFFactory</res-ref-name>
    <jndi-name>wljmsra/xacf</jndi-name>
  </resource-ref>
. . .
```

5.2.3.3 One-Copy-Per-Server Design Strategy for Distributed Topics

`One-Copy-Per-Server` is a design pattern where each instance of an application gets one copy of each message that is published to the Topic.

5.2.3.3.1 Implementing One-Copy-Per-Server To implement the `One-Copy-Per-Server` design strategy, you must:

- Specify the `weblogic.jms.ra.providers.wl.ServerID` property when starting the foreign server instance. For example:
  `-Dweblogic.jms.ra.providers.wl.ServerID=`*`aUniqueIdForTheServer`*

  where *`aUniqueIdForTheServer`* is a unique identifyier for your foreign server.

  For Oracle Glassfish, you can configure this property using the `asadmin` command:

  `asadmin> create-jvm-options --user` *`myUsername`* `--password` *`myPassword`* `--host localhost --port 4848 -Dweblogic.jms.ra.providers.wl.ServerID="`*`aUniqueIdForTheServer`*`"`

- Specify the `ProviderProperties` property in your EJB with a value of `TopicMessageDistributionMode=One-Copy-Per-Server` in the `activation-config` element as shown in Section 5.2.3.2.2, "Example One-Copy-Per-Application EJB Configuration."

5.2.3.3.2 Example One-Copy-Per-Server The following code snippet from an `ejb-jar.xml` file implements `One-Copy-Per-Server` message processing:

```
. . .
<mdb-resource-adapter>
  <resource-adapter-mid>wljmsra</resource-adapter-mid>
    <activation-config>
      <activation-config-property>
        <activation-config-property-name>ClientId</activation-config-property-name
>
        <activation-config-property-value>RDTMDB</activation-config-property-value
>
      </activation-config-property>
      <activation-config-property>
        <activation-config-property-name>ProviderProperties</activation-config-pro
perty-name>
        <activation-config-property-value>TopicMessageDistributionMode=One-Copy-Pe
r-Server</activation-config-property-value>
      </activation-config-property>
    </activation-config>
</mdb-resource-adapter>
<!-- Mapping a Queue admin-object to the Resource-Adapter name -->
  <resource-env-ref>
    <resource-env-ref-name>jms/ResultTopic</resource-env-ref-name>
    <jndi-name>wljmsra/rtopic1</jndi-name>
  </resource-env-ref>
<!-- Mapping a Connection Factory to the Resource-Adapter name -->
  <resource-ref>
    <res-ref-name>jms/ResultXACFFactory</res-ref-name>
    <jndi-name>wljmsra/xacf</jndi-name>
  </resource-ref>
. . .
```

5.2.4 Consuming from Stand-alone (Non-distributed) Topics

On each foreign application server instance that hosts the MDB application, an MDB pool is created for the topic, whether the topic is running in the same cluster or in a different cluster. For an MDB cluster of `N` nodes, `N` MDB pools are created. Each MDB pool creates an individual subscription on the topic, and subscribers from different MDB pools do not share the same subscription. See Section 5.2.3.3.1, "Implementing One-Copy-Per-Server."

5.3 Best Practices for Inbound Communication

This section provides information on tuning and best practices for JMS RA for inbound communication:

- Ensure that the JMS RA allocates more threads than distributed destination members.
- Always configure a `clientID` when using topics. The JMS RA uses a `SHARABLE` Subscription Sharing Policy when processing inbound messages which requires a configured `clientId` for either durable or non-durable subscribers.

  If no `clientID` is set, the MDB will not receive the message and the error condition logged in the foreign application server log. See Section 5.2.1, "The JMS RA and Sharable Subscriptions."
- For queues, increasing the value of `maxListenerThreads` to more than one thread may increase the rate at which messages are consumed.

- The JMS RA does not support sharable connections. If you use the `<resource-ref>` element in your application's JEE descriptors (`web.xml`, `ejb-jar.xml`) to identify the JMS Connection Factories that are looked up and used in that application, you must set the `<res-sharing-scope>` child-element to `unsharable`. The default value for this element is `sharable`. For example:

```
. . .
<resource-ref>
   <res-ref-name>jms/ReplyFactory</res-ref-name>
   <res-type>javax.jms.ConnectionFactory</res-type>
   <res-auth>Application</res-auth>
   <res-sharing-scope>unsharable</res-sharing-scope>
</resource-ref>
```

  If your application uses `@Resource` injection, see http://docs.oracle.com/javaee/6/api/javax/annotation/Resource.html#shareable%28%29

6 Sending Outbound JMS Messages

This chapter describes how to send JMS messages using the JMS RA.

6.1 JMS RA Outbound Communication Basics

The JMS RA is a thin wrapper of WebLogic JMS client. Users can choose to configure their connection client-id to be either RESTRICTED or UNRESTRICTED and their subscriptions to be either Sharable or Exclusive.

6.2 Defining Outbound Connections

The JMS RA provides a number of predefined JMS connection factories. Each JMS connection factory interface has its own `<connection-definition>` element. Each `<connection-definition>` defines the classes and interfaces as required by the JCA specification and additional configuration properties.

6.2.1 JMS RA Connection Factories

The JMS RA supports the following connection factories:

- WebLogic JMS non-XA ConnectionFactory
- WebLogic JMS non-XA QueueConnectionFactory
- WebLogic JMS non-XA TopicConnectionFactory
- WebLogic JMS XA ConnectionFactory
- WebLogic JMS XA QueueConnectionFactory
- WebLogic JMS XA TopicConnectionFactory

See the `<outbound-resourceadapter>` element of the ra.xml file in Section 12.2, "Example of the JMS RA ra.xml File."

6.2.2 Configuring JMS RA Connection Factory Properties

The JMS RA supports additional configuration properties that can be set in the `ra.xml file` or in the JEE container of the foreign application server where the JMS RA is deployed. See Section 12.5, "JMS RA Outbound Configuration Properties." The following is an example WebLogic XA Connection Factory with additional configuration properties.

Example 6–1 Example WebLogic XA Connection Factory Connection Definition

```
. . .
<connection-definition>
          <managedconnectionfactory-class>
              weblogic.jms.ra.WLManagedXAConnectionFactory
          </managedconnectionfactory-class>
          <config-property>
            <config-property-name>LoggerName</config-property-name>
            <config-property-type>java.lang.String</config-property-type>
            <config-property-value>weblogic.jms.ra</config-property-value>
          </config-property>
          <config-property>
            <config-property-name>LogLevel</config-property-name>
            <config-property-type>java.lang.String</config-property-type>
            <config-property-value>FINEST</config-property-value>
          </config-property>
          <config-property>
            <config-property-name>group</config-property-name>
            <config-property-type>java.lang.String</config-property-type>
            <config-property-value></config-property-value>
          </config-property>
          <config-property>
            <config-property-name>rpResourceLocation</config-property-name>
            <config-property-type>java.lang.String</config-property-type>
            <config-property-value></config-property-value>
          </config-property>
          <config-property>
            <config-property-name>autoCloseSession</config-property-name>
            <config-property-type>java.lang.Boolean</config-property-type>
            <config-property-value>false</config-property-value>
          </config-property>
          <connectionfactory-interface>
              weblogic.jms.ra.WLXAConnectionFactoryInterface
          </connectionfactory-interface>
          <connectionfactory-impl-class>
              weblogic.jms.ra.WLXAConnectionFactory
          </connectionfactory-impl-class>
          <connection-interface>
              weblogic.jms.ra.WLXAConnectionInterface
          </connection-interface>
          <connection-impl-class>
              weblogic.jms.ra.WLXAConnection
          </connection-impl-class>
        </connection-definition>
. . .
```

6.3 JMS RA Outbound Connection Limitations

The following are connection management limitations:

- The WebLogic Automatic JMS Client Failover feature enables JMS client to automatically reconnect to another live server instance in a cluster if a server or network failure occurs. This feature interferes with a JMS RA's ability to associate the best ITMs for managed connections. The JMS RA automatically disables this feature and throws an exception if application code tries to enable automatic reconnect programmatically.
- `UP/DOWN` events are not currently supported at the WebLogic instance level. You also can not create a connection to a specific WebLogic instance. This may result in repeated use of a bad connection as JMS RA connection purging may not immediately identify bad connections before they are resused.
- Because `UP` are not currently supported at the WebLogic instance level, the JMS RA does not support connection rebalancing.
- The JMS RA outbound implementation is not aware of the `UP/DOWN` events of server instances of a WebLogic cluster that an outbound application is communicating with. Because of this limitation:
  - You may see the continued use of a bad connection as JMSRA connection purging may not immediately identify a bad connection before it is reused.
  - JMS RA does not automatically re-balance the connections.

6.4 Understanding How Outbound Messages are Load Balanced

A WebLogic cluster consists of multiple WebLogic server instances running simultaneously and working together to provide increased scalability and reliability. A JMS distributed destination usually has a set of members distributed across multiple server instances, with each member hosted by a separate JMS server instance. See "Understanding WebLogic Server Clustering" in *Administering Clusters for Oracle WebLogic Server*.

The following sections provide information on how the JMS RA load balances outbound WebLogic JMS communication.

- Section 6.4.1, "RMI Load Balancing using the Connection Factory"
- Section 6.4.2, "Load Balancing Messages to Distributed Destinations"

6.4.1 RMI Load Balancing using the Connection Factory

The JMS RA supports RMI object load balancing using WebLogic JMS connection factories to creates connections and sessions that point to different WebLogic server instances. See "Load Balancing in a Cluster" in *Administering Clusters for Oracle WebLogic Server*.

6.4.2 Load Balancing Messages to Distributed Destinations

Load balancing to WebLogic Distributed Destinations (DDs) is automatically supported without additional configuration. DDs appear to the JMS RA as a single, logical destination. Both a distributed destination and its members are advertised in WebLogic JNDI. When messages are load balanced to a member of distributed destination on a WebLogic server instance that is different from the instance that the JMS RA's managed connection points to, the Interposed Transaction Manager (ITM) of the managed connection is able to enlist the messages sent to a different WebLogic instance, transparently relaying XA calls to the corresponding transaction coordinator on that instance if necessary.

Note: Individual distributed topic members must be referenced when creating, using, and unsubscribing a durable subscription.

6.5 Configuring Transaction Support for Outbound Communication

JMS RA provides outbound XA capability for applications using WebLogic JMS according to the JCA standard. See Section 9.1.1, "Transaction Support for Outbound Communication."

6.6 Configuring Authentication for Outbound Communication

Configure the `authentication-mechanism` element in the `ra.xml` file. See Section 11.1, "JCA Security."

7

Configuring Destinations and Naming Contexts

This chapter provides information on configuring `adminobject` elements to define destinations and naming contexts for inbound and outbound communication.

This chapter includes the following sections:

- Section 7.1, "Context adminobject Objects"
- Section 7.2, "Using Automatic Destination Wrapping"
- Section 7.3, "Destination adminobjects"
- Section 7.4, "Example adminobject Stanza"

7.1 Context adminobject Objects

Context `adminobjects` are used to look up queues and topics dynamically using Automatic Destination Wrapping. See Section 7.2, "Using Automatic Destination Wrapping."

Auto-Destination Wrapping allows applications to bind an `<admin-object>` in the application server's JNDI that represents a WebLogic JNDI context. Applications can then lookup this WebLogic JNDI context to lookup JMS destinations by their name in the WebLogic JNDI. This can simplify configuration for applications with a large number of destinations because only one `<admin-object>` has to be configured with the application-server.

Use the following elements to define a context `adminobject`:

- `<adminobject-interface>`: Use `weblogic.jms.ra.WLDestinationContextInterface`
- `<adminobject-class>`: Use `weblogic.jms.ra.WLDestinationContext`.

Use the following configuration properties in the `<config-property>` element:

- `group`: The name of the group to associate with this `adminobject` definition.
- `rpContextLocation`: Provides information to the resource adapter describing how to look up JMS destinations using the context `admin-object`. This will be in the form `connector:<connectorName>/`.

The macros defined in a `groupDefinition` may be used for the value of connector in this property value. For instance, `connector:{connectorName}/` can be used in place of `connector:wlDevelopment/` when the value of the macro `{connectorName}` is *wlDevelopment* within a group. See Section 8.2, "Advanced Resource Provider Configuration using groupDefinitions."

7.2 Using Automatic Destination Wrapping

Automatic Destination Wrapping allows an administered object to automatically wrap provider destinations, alleviating the need to create an administered object for each provider destination and explicitly binding the administered object to the destination's JNDI. A single `adminobject` is defined for each resource provider allowing you to dynamically construct JNDI url's to access each destination within this context.

Applications look up JMS RA `admin-objects` bound in JNDI. These `admin-objects` may represent JMS destinations or a destination context. When used for JMS destinations, you must defined one `admin-object` for each JMS destination. If an applications uses 20 JMS destinations, then you must configure 20 individual `admin-objects`, one for each destination. Applications must also define and map these 20 JMS destinations in their JEE descriptors as `resource-env-references`.

The destination context `admin-object` allows you to integrate a single destination context that in turn, may be used to lookup any number of JMS destinations at the WebLogic Server serving as the JMS provider. Applications only need to define one `resource-env-ref` mapping for the destination context. Applications lookup the destination context and append the remote WebLogic JNDI name for the destination.

Automatic Destination Wrapping simplifies configuration when:

- You cannot determine the WebLogic Server JMS destinations at the deployment time. For example, the JMS RA only gets the source and target destinations from end users at runtime.
- If your application needs to access a large number of destinations. This avoids the need to configure a matching set of JMS RA destinations for all destinations of a resource provider. Otherwise, each resource provider destination has to be mapped to a configured `adminobject` before it can be used by the application.

For example, an `adminobject` representing a WebLogic JMS server is bound to JNDI name `myContext` in foreign application server. If a WebLogic JMS destination with JNDI name `jms/bar` configured in this WebLogic server JNDI tree, Automatic Destination Wrapping uses `myContext/jms/bar` in the foreign server JNDI lookup.

The following is an example for automatic wrapping of queues.

```
. . .
<adminobject-config location="myContext">
   <adminobject-class>weblogic.jms.ra.WLDestinationContext</adminobject-class>
   <config-property name="group" value="wls"/>
   <config-property name="rpContextLocation" value="connector:{rp_name}/"/>
</adminobject-config>
. . .
```

where *`rp_name`* is the name of the resource provider defined in `ra.xml` file. See Section 12.6, "JMS RA adminobject Configuration Properties."

In the application, look up `myContext` as a `javax.naming.Context`:

```
. . .
@Resource(mappedName="myContext")
private javax.naming.Context wljmsraContext;
. . .
```

Now you can look up the queue destinations using this context:

```
. . .
Queue theMdbQueue = (Queue) wlraContext.lookup("com.oracle.jms.qa.myQ1");
. . .
```

7.3 Destination adminobjects

There are two types of `adminobject` objects which can be created for JMS queues and topics:

- For Queues, the `<adminobject-class>`: `oracle.j2ee.ra.jms.generic.WLQueueAdmin`
- For Topics, the `<adminobject-class>`: `oracle.j2ee.ra.jms.generic.WLTopicAdmin`

7.4 Example adminobject Stanza

```
. . .
<!-- Context admin object -->
<adminobject>
   <adminobject-interface>weblogic.jms.ra.WLDestinationContextInterface</adminobje
ct-interface>
   <adminobject-class>weblogic.jms.ra.WLDestinationContext</adminobject-class>
   <config-property>
      <config-property-name>group</config-property-name>
      <config-property-type>java.lang.String</config-property-type>
      <config-property-value></config-property-value>
   </config-property>
   <config-property>
      <config-property-name>rpContextLocation</config-property-name>
      <config-property-type>java.lang.String</config-property-type>
      <config-property-value></config-property-value>
   </config-property>
<adminobject>

<!-- Queue admin object -->
<adminobject>
   <adminobject-interface>weblogic.jms.ra.WLQueueAdminInterface</adminobject-inter
face>
   <adminobject-class>weblogic.jms.ra.WLQueueAdmin</adminobject-class>
   <config-property>
      <config-property-name>group</config-property-name>
      <config-property-type>java.lang.String</config-property-type>
      <config-property-value></config-property-value>
   </config-property>
   <<config-property>
      <config-property-name>rpResourceLocation</config-property-name>
      <config-property-type>java.lang.String</config-property-type>
      <config-property-value></config-property-value>
   </config-property>
</adminobject>

<!-- Topic admin object -->
<adminobject>
   <adminobject-interface>weblogic.jms.ra.WLTopicAdminInterface</adminobject-inter
face>
   <adminobject-class>weblogic.jms.ra.WLTopicAdmin</adminobject-class>
   <config-property>
      <config-property-name>group</config-property-name>
      <config-property-type>java.lang.String</config-property-type>
```

```
      <config-property-value></config-property-value>
   </config-property>
   <config-property>
      <config-property-name>rpResourceLocation</config-property-name>
      <config-property-type>java.lang.String</config-property-type>
      <config-property-value></config-property-value>
   </config-property>
</adminobject>
. . .
```

8

Understanding Resource Providers

This chapter describes how to use and configure Resource Providers. A resource provider defines the JNDI properties that allow the JMS RA to connect to the WebLogic JMS provider.

- Section 8.1, "Basic Resource Provider Configuration"
- Section 8.2, "Advanced Resource Provider Configuration using groupDefinitions"
- Section 8.3, "Example Resource Provider Configuration."

8.1 Basic Resource Provider Configuration

The JMS RA utilizes a `resource-adapter` `config-property` named `resourceProviderDefintions` to define the JNDI properties. This property is used by the JMS RA to access WebLogic JMS. You can configure multiple resource providers.

Note: When describing `resourceProviderDefinitions`, you may need to use the escape character "%" because your WebLogic Server JNDI url may have commas. A comma is also used as a delimiter to separate properties.

For example: If the url is `t3://host:port,host2:port2`, the JMS RA would fail to parse the url because of the comma after "port".

To workaround this issue, escape the comma using "%". For example: `t3://host:port%,host2,port`

Use the following steps to configure a resource provider:

1. Specify `resourceProviderDefinitions` as the `config-property-name`.

   ```
   <config-property-name>resourceProviderDefinitions</config-property-name>
   ```

2. Specify `java.lang.String` as the `config-property-type`.

   ```
   <config-property-type>java.lang.String</config-property-type>
   ```

3. Use the following name/value pair pattern to define specific JNDI properties for a resource provider:

   ```
   (RP_NAME: jndiEnv=(property1=value1,property2=value2,...))
   ```

   where:

RP_NAME is a unique name used to define the JNDI properties of a resource provider and is used with the `rpResourceLocation config-property` for `<connection-definition>` and `<adminobject>` elements. See "Sending Outbound JMS Messages" on page 6-1.

property1=value1,property2=value2,... is a coma separated list of name/value pairs that define JNDI properties for a resource provider.

Note: Each defined resource provider (*RP_NAME* value) must be unique within the `ra.xml` file.

For an example configuration, see Section 8.3, "Example Resource Provider Configuration."

8.2 Advanced Resource Provider Configuration using groupDefinitions

The JMS RA uses the `groupDefinitions` property to provide an advanced method for configuring resource providers. It enables you to create a compatible set of messaging objects while providing flexible address resolution of connection factories and `adminobjects`. `groupDefinitions` depends on the following components:

- Group: A set of compatible messaging objects, such as connection factories and associated destination `adminobjects`.
- Macro: A Group component that is substituted for a `rpResourceLocation` configuration property in connection factories and `adminobjects`.

Use the following steps to configure `groupDefinitions`:

1. Specify `groupDefinitions` as the `config-property-name`.

   ```
   <config-property-name>groupDefinitions</config-property-name>
   ```

2. Specify `java.lang.String` as the `config-property-type`.

   ```
   <config-property-type>java.lang.String</config-property-type>
   ```

3. Use the following pattern to define a Group:

   ```
   <config-property-value>
      GROUP_NAME: macro
   </config-property-value
   ```

   where:

   GROUP_NAME is a unique name used to represent a Group configuration.

   macro is a coma separated list of name/value pairs that define a set of compatible messaging objects. The JMS RA substitutes these pairs for `rpResourceLocation` configuration properties in connection factories and `adminobjects`.

For an example configuration, see Section 8.3, "Example Resource Provider Configuration."

8.3 Example Resource Provider Configuration

The following section provides an example resource provider configuration. Two resource providers are configured: rp1and rp2. Two Groups are configured: GroupA and GroupB

Example 8–1 Example Resource Provider Configuration

```
. . .
     <config-property>
        <config-property-name>resourceProviderDefinitions</config-property-name>
        <config-property-type>java.lang.String</config-property-type>
// Example Configuration for two resource providers: rp1 and rp2
        <config-property-value>
           (rp1: jndiEnv=(java.naming.factory.initial=
                    weblogic.jms.WrappedInitialContextFactory,
                    java.naming.provider.url=t3://@@@HOST@@@:7002,
                    java.naming.security.principal=wxyzUser1,
                    java.naming.security.credentials=wxyzPass1))
            (rp2: jndiEnv=(java.naming.factory.initial=
                    weblogic.jms.WrappedInitialContextFactory,
                    java.naming.provider.url=t3://anotherhost:8002,
                    java.naming.security.principal=wxyzUser1,
                    java.naming.security.credentials=wxyzPass1))
        </config-property-value>
. . .
     </config-property>
        <config-property>
        <config-property-name>groupDefinitions</config-property-name>
        <config-property-type>java.lang.String</config-property-type>
        <config-property-value>
                (GroupA:  connectorName=rp1,
                          cf=myCF,
                          xacf=myXACF,
                          topic1=myT1,
                          topic2=myT2,
                          queue1=myQ1,
                          queue2=myQ12)
                (GroupB:  connectorName=rp2,
                          cf=example/cf,
                          xacf=example/xacf,
                          queue1=queue1,
                          queue2=queue2)
        </config-property-value>
     </config-property>
. . .
```

9 Understanding Transaction Processing

This chapter describes transaction processing and recovery when using the JMS RA to interoperate between a foreign application server and WebLogic Server.

- Section 9.1, "JMS RA Transaction Support"
- Section 9.2, "Lazy Enlistment of Connections in a Transaction"
- Section 9.3, "WebLogic Cluster-wide Recovery."

9.1 JMS RA Transaction Support

There are two major use cases for JMS in a third party application server. Case one is outbound communication where users interact with the JMS provider synchronously. The second case is inbound communications where the users interact with the JMS provider asynchronously, for example through message-driven bean (MDB).

9.1.1 Transaction Support for Outbound Communication

JMS RA provides outbound XA capability for applications using WebLogic JMS according to the JCA standard. See Section 12.3.32, "transaction-support."

The JMS RA's managed connection holds a WebLogic JMS client connection and session pointing to a WebLogic cluster member. The managed connection also references the interposed transaction manager (ITM) stub pointing to the same WebLogic cluster member. Although a WebLogic ITM is able to handle JMS operations occurring at different WebLogic instances, it is more efficient to have JMS operations and the ITM located on the same cluster member e same instance (ITM affinity). When a JMS RA's managed connection is involved in a global transaction:

- The GlassFish server instance calls the `getXAResource()` method on the managed connection to enlist its operations into the global transaction (per JCA 1.6 transaction contract between resource adapter and application server).
- If `Cluster-wide Recovery` is enabled, the interposed transaction manager transparently handles any XA calls and forwards them to the correct cluster member instance to maintain ITM affinity. See "Cluster-wide Recovery" in *Developing JTA Applications for Oracle WebLogic Server*.
- The managed connection returns a `XAResource` wrapper pointing to the corresponding ITM.
- The wrapper relays the XA calls to the ITM's `XAResource`.

9.1.2 Transaction Support for Inbound Transactions

In the inbound use case, messages are received by implementing either the resource adapter inbound communication contract or native MDB code.

- In the resource adapter contract is used, the resource adapter receives messages from the MDB destination (a WebLogic Server destination) and distributes them to MDBs. In most cases, the resource adapter spawns multiple worker threads to receive messages concurrently. Each thread uses a separate endpoint and a separate WebLogic JMS session. For a foreign application server to enlist message receiving operations into a global transaction, the JMS resource adapter creates the endpoint using the `MessageEndpointFactory createEndpoint(XAResource)` method passing a `XAResource` wrapper which includes the ITM stub pointing to the same WebLogic server instance as the WebLogic JMS session receiving messages. If the WebLogic destination is a distributed destination, the worker threads are distributed evenly across the destination members.
- If native MBDs are use, the application server is responsible for the message receiving. The use scenario of the JMS resource adapter is the same as the outbound usage. The transaction management is also the same.

9.2 Lazy Enlistment of Connections in a Transaction

Typically, when an application component requests a connection handle in the context of a transaction, the connection is automatically, or "eagerly", enlisted in the transaction. This is true even if the connection is ultimately unused, which results in unnecessary overhead.

With "lazy" enlistment, a new connection is enlisted in the transaction only if it is actually used in the transaction— only if data is transmitted through the connection.

Both a JMS RA and an foreign application server must support lazy enlistment for this feature to be used in your application environment.

9.3 WebLogic Cluster-wide Recovery

You can configure cluster-wide transaction recovery of distributed transactions across all the interposed transaction managers of a cluster by selecting the **Enable Cluster-Wide Recovery** attribute on the "Cluster: Configuration: JTA" page in the *Oracle WebLogic Server Administration Console Online Help*.

When enabled, each interposed transaction manager in a WebLogic cluster is aware of the transaction distribution across the entire cluster. This allows the interposed transaction manager on each cluster member to determine if it should handle a given XA call or forward it to the appropriate interposed transaction manager on another cluster member.

For additional information on WebLogic transaction processing, see:Section 10, "Understanding Failure Management" and "Participating in Transactions Managed by a Third-Party Transaction Manager" in *Developing JTA Applications for Oracle WebLogic Server*.

10 Understanding Failure Management

This chapter describes how the JMS RA responds to WebLogic Server and foreign application server failures.

This chapter includes the following sections:

- Section 10.1, "WebLogic Server Failure"
- Section 10.2, "WebLogic Distributed Destination Member Failure"
- Section 10.3, "Transaction Recovery"

10.1 WebLogic Server Failure

The JMS RA detects the connection failure through the exception listener and JMS operations. When a WebLogic instance failure is detected, the corresponding connections/sessions are closed an new connections and sessions are created. The failed connections are purged from the connection pool to prevent them from being resused.

> **Note:** When a failed WebLogic server instance is restarted or a new server instance joins the a cluster, the resource adapter does not load balance the existing connections in the pool to the new server or restarted instances.

10.2 WebLogic Distributed Destination Member Failure

When a WebLogic distributed destination member fails, the JMS RA the inbound MDB container closes any existing connections/sessions and opens new connections/sessions to an available member.

- If the failed member is restored or a new member is added, the JMS RA rebalances the workload so that the messages are evenly distributed to all distributed destination members. The JMS resource adapter listens to the destination member UP/DOWN events to achieve the rebalance. Requires `Server Affinity=fasle` and `LoadBalancing=true`.
- For failures that cannot be recovered from quickly, the adapter logs the failure and retries periodically.

10.3 Transaction Recovery

The following sections provide on the transaction recovery process when a foreign application server or WebLogic Server become unavailable before a transaction is complete:

- Section 10.3.1, "Transaction Recovery When WebLogic Server is Unavailable"
- Section 10.3.2, "Transaction Recovery When the Foreign Server is Unavailable"

10.3.1 Transaction Recovery When WebLogic Server is Unavailable

The following sections provide information on how a transaction directed by a foreign transaction manager is processed when the WebLogic Server instance is unavailable.

10.3.1.1 Failure Before Prepare

If the WebLogic Server instance participating in the transaction directed by a third-party transaction manager becomes unavailable before the transaction is prepared, the transaction is lost.

10.3.1.2 Failure After Prepare

If the WebLogic Server instance participating in the transaction directed by a foreign transaction manager becomes unavailable after the transaction is prepared, the result is an in-doubt transaction.

The foreign transaction manager continues to retry the transaction periodically until one of the following occurs:

- The transaction times out and the resources are recovered.
- The original WebLogic Server instance becomes available and the transaction is resolved.
- If the WebLogic environment supports Cluster-wide Recovery, the transaction is resolved when it is forwarded to the appropriate interposed transaction manager (ITM) on another cluster member. See "Cluster-wide Recovery" in *Developing JTA Applications for Oracle WebLogic Server* and "InterposedTransactionManager" in *Java API Reference for Oracle WebLogic Server*.

  If the foreign transaction manager uses the same `XAResource` to complete the transaction, the **JMS RA** `XAResource` wrapper lookups the ITM stub each time the retry happens. If the foreign transaction manager uses a different `XAResource` to complete the transaction, the ITM on the live instance forwards the XA calls to the target instance transparently. In either case, the transaction is only completed when the failed ITM is restored by either a restart or by migration.

10.3.2 Transaction Recovery When the Foreign Server is Unavailable

The recovery process begins when the foreign server is restarted. If `Cluster-wide Recovery` is enabled, the ITM of any affected clustered WebLogic Server returns all in-doubt transactions that occurred when the `recover` method is called. Each ITM handles the `XAResource` commit/rollback calls for all in-doubt transactions either by itself or by forwarding to the responsible ITM. See "Cluster-wide Recovery" in *Developing JTA Applications for Oracle WebLogic Server*

Note: For the recovery process to succeed, every single ITM in the cluster must be available as every ITM in the cluster must be polled to compile the complete list of in-doubt transactions.

The foreign transaction manager interacts with the `XAResource` wrapper provided by the JMS RA. The `XAResource` wrapper relays the XA calls to ITM stubs. The JMS RA uses the standard JMS API and the proprietary WebLogic JMS extension API as the client interface. An application (such as a EJB or Servlet) running inside the foreign server interacts with the JMS RA and WebLogic JMS through JMS RA wrappers. In addition to relaying the JMS operations to WebLogic client objects, the JMS RA wrappers also perform lazy enlistment and end transaction branches by intercepting the appropriate JMS calls.

10.4 Understanding WebLogic Service Migration

The following section provides information on JMS RA behavior during WebLogic service migration.

If a WebLogic JMS server is configured to use a migratable target, a JMS RA instance reconnects to the migrated WebLogic JMS service after a service migration completes.

Note: WebLogic JMS service migration provides high availability for JMS services, it does not provide load balancing. See "Migratable Services" in *Administering Clusters for Oracle WebLogic Server*. For information on load balancing, see

The following sections provide information how the JMS RA handles inbound and outbound communication during JMS service migration:

- Section 10.4.1, "Inbound Communication"
- Section 10.4.2, "Outbound Communication"

10.4.1 Inbound Communication

During JMS service migration, the JMS RA closes existing WebLogic JMS connections and sessions. New connections and sessions are created that point to WebLogic Server instance where the JMS server and its distributed destination members have migrated.

10.4.2 Outbound Communication

During JMS service migration, the distributed destination members of a failed WebLogic Server instance are migrated to an available WebLogic Server instance.The WebLogic JMS client associated with the JMS RA detects the migration and redirects JMS operations to the new migrated destination members.

11 Securing JMS RA Connections

This chapter describes security considerations for the JMS RA.

- Section 11.1, "JCA Security"
- Section 11.2, "WebLogic JMS Security"
- Section 11.3, "Specifying a Username/Password"
- Section 11.4, "Securing Credentials with Oracle Wallet"
- Section 11.5, "Secure Wire Communication"

11.1 JCA Security

The JMS RA is fully compliant with the JCA security contract as described in *Java™ EE Connector Architecture Specification, version 1.6* at http://download.oracle.com/otn-pub/jcp/connector_architecture-1.6-fr-oth-JSpec/connector-1_6-final-spec.pdf.

For outbound communication, in the `ra.xml` file you can specify authentication-mechanism-type, credential-interface, and reauthentication-support. For example:

```
. . .
<outbound-resourceadapter>
. . .
    <authentication-mechanism>
            <authentication-mechanism-type>
               BasicPassword
            </authentication-mechanism-type>

<credential-interface>javax.resource.spi.security.PasswordCredential</credential-i
nterface>
        </authentication-mechanism>
        <reauthentication-support>false</reauthentication-support>
. . .
</outbound-resourceadapter>
. . .
```

11.2 WebLogic JMS Security

The following sections provide information on WebLogic JMS security:

- Section 11.2.1, "Overview of JMS Security Models"
- Section 11.2.2, "Protecting JMS Resources"

11.2.1 Overview of JMS Security Models

WebLogic JMS uses a thread-based security model. The subject of the thread is established in the JNDI look up as the JNDI username and credential. WebLogic JMS assumes all related operations are done within the same thread under the same subject that is used for later authorizations at the server. The username and password used to create JMS connections are ignored in the authorization phase.

11.2.2 Protecting JMS Resources

You can secure JMS resources that are deployed either as a service or an application. To secure JMS destinations, you create security policies and security roles for all destinations (JMS queues and JMS topics) as a group, or an individual destination (JMS queue or JMS topic) on a JMS server. See "Java Messaging Service (JMS) Resources" in *Securing Resources Using Roles and Policies for Oracle WebLogic Server*.

11.3 Specifying a Username/Password

There are four methods for specifying Username and Password:

- Section 11.3.1, "Specifying a Username/Password for In-bound Connections using the JCA Container"
- Section 11.3.2, "Specifying a Username/Password for In-bound Connections using JNDI"
- Section 11.3.3, "Specifying a Username/Password for In-bound Connections using a Connection Factory"
- Section 11.3.4, "Specifying a Username/Password for Out-bound Connections"

When possible, Oracle recommends using the host application server's JCA container. Most vendors provide JCA containers that provide methods to dynamically set credentials using secure methods. Other methods typically store credentials in clear text. If you chose a method that does not encrypt credentials, use Oracle Wallet to secure them. See Section 11.4, "Securing Credentials with Oracle Wallet.".

11.3.1 Specifying a Username/Password for In-bound Connections using the JCA Container

Your application can provide the username/password in the `activation-spec` of an inbound resource adapter. The `activation spec` is then passed into the JMS RA by the foreign application server's JCA container.

For detailed information on how to specify a username/password using the JCA container of your foreign application server, see Section 4, "Administering the JMS RA on Oracle GlassFish Server."

11.3.2 Specifying a Username/Password for In-bound Connections using JNDI

You can configure the `jndiEnv` property in the `resourceProviderDefinitions` to include the usename/password as shown below:

```
<config-property-name>resourceProviderDefinitions</config-property-name>
      <config-property-type>java.lang.String</config-property-type>
      <config-property-value>
        (weblogicAdmin:
jndiEnv=(java.naming.factory.initial=weblogic.jms.WrappedInitialContextFactory,
                                  java.naming.provider.url='t3://host:port',
```

```
                                    java.naming.security.principal=xxxx,
                                    java.naming.security.credentials=yyyy))
    </config-property-value>
    </config-property>
```

See Section 11.4.1, "Example JNDI Configurations for Setting Credentials."

11.3.3 Specifying a Username/Password for In-bound Connections using a Connection Factory

The JMS RA simplifies security interoperability with foreign application servers by providing connection based security model using a new JNDI initial context factory: `weblogic.jms.WrappedInitialContextFactory`. The resulting subject is determined using the following rules:

- A subject is associated with each connection created using the connection username/password.
- If the connection is created without username password, then the JNDI username/password is used.
- All subsequent JMS operations will use the resultant subject from the connection creation call regardless what is on the thread.

11.3.4 Specifying a Username/Password for Out-bound Connections

For outbound connections, you can use `createConnection(java.lang.String, java.lang.String)`.

The JMS RA does not provide configuration attributes to implement the username/password for outbound connections. See Section 11.5, "Secure Wire Communication" for information on securing out-bound communication.

Some foreign application servers may provide secure credentials between domains. If so, consult the your vendor documentation for more information.

11.4 Securing Credentials with Oracle Wallet

Oracle Wallet provides an simple and easy method to secure credentials between multiple domains. It allows you to update credentials by updating the Wallet instead of having to change individual credentials.

To secure your credentials, you must:

- Create a wallet file and add the necessary credentials using the JMS RA encryption utility. This step creates a `cwallet.sso` file at the specified location which maps an alias to the secured credentials. See Section 11.4.2, "Using the wljmsra Encryption utility."
- Provide the alias in the JMS RA deployment descriptor or connection pool configuration.
- Provide the location of the `cwallet.sso` file in the JMS RA deployment descriptor or connection pool configuration.

11.4.1 Example JNDI Configurations for Setting Credentials

The following table provide examples of how you can set JMS RA JNDI environment properties.

Table 11–1 JNDI Properties for Setting Credentials

JNDI Settings	Behavior
java.naming.security.principal=*principal* java.naming.security.credentials=*credentials*	The JMS RA run time uses the values of *principal* and *credentials* to access WebLogic Server destinations.
java.naming.security.principal=*principal* java.naming.security.credentials=->*alias* weblogic.jms.walletDir=*directory*	The JMS RA run time uses the value of *principal* as the username and the value of *alias* to retrieve and use the password stored in the `cwallet.sso` file located in the directory specified by the value of *directory*.
java.naming.security.principal=->*alias1* java.naming.security.credentials=->*alias2* weblogic.jms.walletDir=*directory*	The JMS RA run time uses the value of *alias1* to retrieve and use the username and *alias2* to retrieve and use the password stored in the `cwallet.sso` file located in the directory specified by the value of *directory*.
java.naming.security.principal=->*alias* java.naming.security.credentials=->* weblogic.jms.walletDir=*directory*	The JMS RA run time uses the value of *alias* to retrieve and use the username and password stored in the `cwallet.sso` file located in the directory specified by the value of *directory*.
java.naming.security.principal=*principal* java.naming.security.credentials=->* weblogic.jms.walletDir=*directory*	The JMS RA run time uses the value of *principal* as the username and the value of *principal* to retrieve and use the password stored in the `cwallet.sso` file located in the directory specified by the value of *directory*.

11.4.2 Using the wljmsra Encryption utility

The JMS RA provides a command-line utility to add application credentials into a an Oracle Wallet file. To run the utility, change directories to the `wlserver/server/lib` directory of your installation and enter the following command to display the valid commands:

```
java -jar wljmsra.rar help
Usage:
create <dir>: Create wallet under given directory.
add <alias> <value> [dir]: Add value using the alias.
replace <alias> <value> [dir]: Replace value of the alias.
remove <alias> [dir]: Remove an alias.
dump [dir]: List all aliases in the wallet.
help: This help.
```

11.4.2.1 Create a Wallet

The following example uses the encryption utility to create a wallet file in the directory `mywallet`:

```
java -jar wljmsra.rar create mywallet
```

```
Info: Created wallet under directory 'mywallet'.
```

11.4.2.2 Create an Alias

The following example uses the encryption utility to create an alias:

```
java -jar wljmsra.rar add user6 pwd6
Info: Added alias 'user6'.
```

11.4.2.3 Replace an Alias

The following example uses the encryption utility to replace an alias:

```
java -jar wljmsra.rar replace user6 newpwd6
Info: Replaced alias 'user6'.
```

11.4.2.4 Remove an Alias

The following example uses the encryption utility to remove an alias:

```
java -jar wljmsra.rar remove user6
Info: Removed alias 'user6'.
```

11.4.2.5 List the Aliases in a Wallet

The following example uses the encryption utility to list the aliases in a wallet:

```
java -jar wljmsra.rar dump mywallet
Info: Aliases found in wallet under 'mywallet'.
user4
Info: 1 aliases found.
```

11.5 Secure Wire Communication

Oracle recommends using SSL/t3s to secure information sent over the wire. See "Configuring SSL" in *Administering Security for Oracle WebLogic Server*.

12

JMS RA Deployment Descriptor Elements and Properties

This chapter provides information about the WebLogic JMS RA deployment descriptor file, `ra.xml`.

This chapter includes the following sections:

12.1 Overview of the JMS RA ra.xml

The JCA 1.6 compliant deployment descriptor for the JMS RA is called `ra.xml`. This file is used to integrate a WebLogic JMS client with supported foreign application servers.

12.2 Example of the JMS RA ra.xml File

The following example shows the `ra.xml` deployment descriptor:

Example 12–1 ra.xml file

```
<?xml version="1.0" encoding="UTF-8"?>

<connector 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/connector_1_5.xsd"
           version="1.5">

   <display-name>Oracle JMS Connector</display-name>
   <vendor-name>Oracle Corporation</vendor-name>
   <eis-type>JMS</eis-type>
   <resourceadapter-version>2.0</resourceadapter-version>

   <resourceadapter>
      <resourceadapter-class>weblogic.jms.ra.ResourceAdapterImpl</resourceadapter-
```

```
class>
      <config-property>
         <config-property-name>resourceProviderDefinitions</config-property-name>
         <config-property-type>java.lang.String</config-property-type>
         <config-property-value></config-property-value>
      </config-property>
      <config-property>
         <config-property-name>groupDefinitions</config-property-name>
         <config-property-type>java.lang.String</config-property-type>
         <config-property-value>
            (:)
         </config-property-value>
      </config-property>

   <outbound-resourceadapter>

   <!-- *************  WebLogic JMS Connection Factories  ************* -->

      <!--  WebLogic JMS non-XA javax.jms.ConnectionFactory  -->
      <connection-definition>
         <managedconnectionfactory-class>
            weblogic.jms.ra.WLManagedConnectionFactory
         </managedconnectionfactory-class>
         <connectionfactory-interface>
            weblogic.jms.ra.WLConnectionFactoryInterface
         </connectionfactory-interface>
         <connectionfactory-impl-class>
            weblogic.jms.ra.WLConnectionFactory
         </connectionfactory-impl-class>
         <connection-interface>
            weblogic.jms.ra.WLConnectionInterface
         </connection-interface>
         <connection-impl-class>
            weblogic.jms.ra.WLConnection
         </connection-impl-class>
      </connection-definition>

      <!--  WebLogic JMS non-XA javax.jms.QueueConnectionFactory  -->

      <connection-definition>
         <managedconnectionfactory-class>
            weblogic.jms.ra.WLManagedQueueConnectionFactory
         </managedconnectionfactory-class>
         <connectionfactory-interface>
            weblogic.jms.ra.WLQueueConnectionFactoryInterface
         </connectionfactory-interface>
         <connectionfactory-impl-class>
            weblogic.jms.ra.WLQueueConnectionFactory
         </connectionfactory-impl-class>
         <connection-interface>
            weblogic.jms.ra.WLQueueConnectionInterface
         </connection-interface>
         <connection-impl-class>
            weblogic.jms.ra.WLQueueConnection
         </connection-impl-class>
      </connection-definition>

      <!--  WebLogic JMS non-XA javax.jms.TopicConnectionFactory  -->

      <connection-definition>
```

```
   <managedconnectionfactory-class>
      weblogic.jms.ra.WLManagedTopicConnectionFactory
   </managedconnectionfactory-class>
   <connectionfactory-interface>
      weblogic.jms.ra.WLTopicConnectionFactoryInterface
   </connectionfactory-interface>
   <connectionfactory-impl-class>
      weblogic.jms.ra.WLTopicConnectionFactory
   </connectionfactory-impl-class>
   <connection-interface>
      weblogic.jms.ra.WLTopicConnectionInterface
   </connection-interface>
   <connection-impl-class>
      weblogic.jms.ra.WLTopicConnection
   </connection-impl-class>
</connection-definition>

<!--  WebLogic JMS XA javax.jms.ConnectionFactory  -->

<connection-definition>
   <managedconnectionfactory-class>
      weblogic.jms.ra.WLManagedXAConnectionFactory
   </managedconnectionfactory-class>
   <connectionfactory-interface>
      weblogic.jms.ra.WLXAConnectionFactoryInterface
   </connectionfactory-interface>
   <connectionfactory-impl-class>
      weblogic.jms.ra.WLXAConnectionFactory
   </connectionfactory-impl-class>
   <connection-interface>
      weblogic.jms.ra.WLXAConnectionInterface
   </connection-interface>
   <connection-impl-class>
      weblogic.jms.ra.WLXAConnection
   </connection-impl-class>
</connection-definition>

<!--  WebLogic JMS XA javax.jms.QueueConnectionFactory  -->

 <connection-definition>
   <managedconnectionfactory-class>
      weblogic.jms.ra.WLManagedXAQueueConnectionFactory
   </managedconnectionfactory-class>
   <connectionfactory-interface>
      weblogic.jms.ra.WLXAQueueConnectionFactoryInterface
   </connectionfactory-interface>
   <connectionfactory-impl-class>
      weblogic.jms.ra.WLXAQueueConnectionFactory
   </connectionfactory-impl-class>
   <connection-interface>
      weblogic.jms.ra.WLXAQueueConnectionInterface
   </connection-interface>
   <connection-impl-class>
      weblogic.jms.ra.WLXAQueueConnection
   </connection-impl-class>
</connection-definition>

<!--  WebLogic JMS XA javax.jms.TopicConnectionFactory  -->

<connection-definition>
```

```
      <managedconnectionfactory-class>
         weblogic.jms.ra.WLManagedXATopicConnectionFactory
      </managedconnectionfactory-class>
      <connectionfactory-interface>
         weblogic.jms.ra.WLXATopicConnectionFactoryInterface
      </connectionfactory-interface>
      <connectionfactory-impl-class>
         weblogic.jms.ra.WLXATopicConnectionFactory
      </connectionfactory-impl-class>
      <connection-interface>
         weblogic.jms.ra.WLXATopicConnectionInterface
      </connection-interface>
      <connection-impl-class>
         weblogic.jms.ra.WLXATopicConnection
      </connection-impl-class>
   </connection-definition>

   <!-- ******************  miscellaneous outbound   ****************** -->

   <transaction-support>XATransaction</transaction-support>

   <authentication-mechanism>
      <authentication-mechanism-type>
         BasicPassword
      </authentication-mechanism-type>
      <credential-interface>
         javax.resource.spi.security.PasswordCredential
      </credential-interface>
   </authentication-mechanism>
   <reauthentication-support>false</reauthentication-support>

   </outbound-resourceadapter>

   <inbound-resourceadapter>
      <messageadapter>
         <messagelistener>
            <messagelistener-type>
               javax.jms.MessageListener
            </messagelistener-type>
            <activationspec>
               <activationspec-class>
                  weblogic.jms.ra.ActivationSpecImpl
               </activationspec-class>
               <required-config-property>
                  <config-property-name>ConnectionFactory</config-property-name
>
               </required-config-property>
               <required-config-property>
                  <config-property-name>Destination</config-property-name>
               </required-config-property>
               <required-config-property>
                  <config-property-name>DestinationType</config-property-name>
               </required-config-property>
            </activationspec>
         </messagelistener>
      </messageadapter>

   </inbound-resourceadapter>

   <!-- ******************  WebLogic JMS Destinations  ****************** -->
```

```
      <!-- WebLogic JMS javax.jms.Queue, weblogic.jms.extensions.WLDestination -->
      <adminobject>
         <adminobject-interface>
            weblogic.jms.ra.WLQueueAdminInterface
         </adminobject-interface>
         <adminobject-class>
            weblogic.jms.ra.WLQueueAdmin
         </adminobject-class>
      </adminobject>

      <!-- WebLogic JMS javax.jms.Topic, weblogic.jms.extensions.WLDestination -->
      <adminobject>
         <adminobject-interface>
            weblogic.jms.ra.WLTopicAdminInterface
         </adminobject-interface>
         <adminobject-class>
            weblogic.jms.ra.WLTopicAdmin
         </adminobject-class>
      </adminobject>

      <!-- javax.naming.Context for looking up
weblogic.jms.extensions.WLDestination -->

      <adminobject>
         <adminobject-interface>
            weblogic.jms.ra.WLDestinationContextInterface
         </adminobject-interface>
        <adminobject-class>
            weblogic.jms.ra.WLDestinationContext
         </adminobject-class>
      </adminobject>

   </resourceadapter>

</connector>
```

12.3 JMS RA Element Descriptions

The element hierarchy of the ra.xml deployment descriptor file is provided below. The number of occurrences allowed is listed in braces following element name. Each element is described in detail in the following sections.

```
<connector> {1}
   <display-name> {0 or 1}
   <vendor-name {0 or 1}
   <eis-type> {1}
   <resourceadapter-version> {1}
   <resourceadapter> {1}
      <resourceadapter-class> {1}
      <config-property> {0 or more}
         <config-property-name> {1}
         <config-property-type> {1}
         <config-property-value {0 or 1}
      <outbound-resourceadapter> {1}
         <connection-definition> {1 or more}
            <managedconnectionfactory-class{1}
            <config-property> {0 or more}
               <config-property-name> {1}
```

```
            <config-property-type> {1}
            <config-property-value {0 or 1}
         <connectionfactory-interface> {1}
         <connectionfactory-impl-class> {1}
         <connection-interface> {1}
         <connection-impl-class> {1}
<transaction-support> {1}
<authentication-mechanism> {1}
   <authentication-mechanism-type> {1}
   <credential-interface> {1}
<reauthentication-support> {1}
<inbound-resourceadapter> {1}
   <messageadapter> {1}
      <messagelistener> {1}
         <messagelistener-type> {1}
         <activationspec> {1}
            <activationspec-class> {1}
            <required-config-property> {0 or more}
               <config-property-name> {1}
<adminobject> {1 or more}
   <adminobject-interface> {1}
   <adminobject-class> {1}
   <config-property> {0 or more}
      <config-property-name> {1}
      <config-property-type> {1}
      <config-property-value {0 or 1}
```

12.3.1 activationspec

The `<activationspec>` child element of the `<messagelistener>` element is used to specify an activation specification. The information includes fully qualified Java class name of an activation specification and a set of required configuration property names.

12.3.2 activationspec-class

The `<activationspec-class>` child element of the `<activationspec>` element is used to specify the fully qualified Java class name of the activation specification class. This class must implement the `javax.resource.spi.ActivationSpec` interface. The implementation of this class is required to be a JavaBean.

12.3.3 adminobject

The `<adminobject>` child element of the `<resourceadapter>` element is used to specify information about an administered object. Administered objects are specific to a messaging style or message provider. This contains information on the Java type of the interface implemented by an administered object, its Java class name and its configuration properties.

12.3.4 adminobject-class

The `<adminobject-class>` child element of the `<adminobject>` element is used to specify the fully qualified name of the Java type of the interface implemented by an administered object.

12.3.5 adminobject-interface

The <adminobject-interface> child element of the <adminobject> element is used to specify the fully qualified name of the Java type of the interface implemented by an administered object.

12.3.6 authentication-mechanism

The <authentication-mechanism> child element of the <outbound-resourceadapter> element specifies an authentication mechanism supported by the resource adapter.

The BasicPassword mechanism type should support the javax.resource.spi.security.PasswordCredential interface. The Kerbv5 mechanism type should support the org.ietf.jgss.GSSCredential interface or the deprecated javax.resource.spi.security.GenericCredential interface.

12.3.7 authentication-mechanism-type

The <authentication-mechanism-type> child element of the <authentication-mechanism> element specifies the authentication mechanism. Values are:

- BasicPassword
- Kerbv5

12.3.8 config-property-name

The <config-property-name> child element of the <config-property> or <required-config-property> element defines the name of a configuration property and is entered as a string. Valid names are specific to a resource adapter or ministered object implementation.

12.3.9 config-property-type

The <config-property-type> child element of the <config-property> element defines the data type of a configuration property value and is entered as a string. For example: java.lang.String

12.3.10 config-property-value

The <config-property-value> child element of the <config-property> element defines the value of a configuration property and is entered as a string.

12.3.11 connection-definition

The <connection-definition> child element of the <outbound-resourceadapter> element defines the classes and interfaces required by the JCA specification to define connection factories.

12.3.12 connectionfactory-impl-class

The <connectionfactory-imlp-class> child element of the <connection-definition> element defines the fully qualified name of the ConnectionFactory class that implements resource adapter specific ConnectionFactory interface. See http://java.sun.com/xml/ns/javaee/connector_1_6.xsd.

12.3.13 config-property

The <config-property> child element of the <resourceadapter> element, <adminobject> element, and <connection-definition> element that defines a conversation property for a resource adapter administered objects. A configuration property is defined in the same manner as it is defined in the standard connector deployment descriptor.

12.3.14 connectionfactory-interface

The <connectionfactory-interface> child element of the <connection-definition> element specifies the fully qualified name of the ConnectionFactory interface supported by the resource adapter. See http://java.sun.com/xml/ns/javaee/connector_1_6.xsd.

12.3.15 connection-impl-class

The <connection-imlp-class> child element of the <connection-definition> element specifies the fully qualified name of the Connection class that implements resource adapter specific Connection interface. See http://java.sun.com/xml/ns/javaee/connector_1_6.xsd.

12.3.16 connection-interface

The <connection-interface> child element of the <connection-definition> element specifies the fully qualified name of the Connection interface supported by the resource adapter.

12.3.17 connector

The <connector> element is the root element of the WebLogic JMS RA deployment descriptor file, ra.xml

12.3.18 credential-interface

The <authentication-mechanism> child element of the <outbound-resourceadapter> element specifies the interface that the resource adapter implementation supports for the representation of the credentials.Values are:

- javax.resource.spi.security.PasswordCredential
- org.ietf.jgss.GSSCredential
- javax.resource.spi.security.GenericCredential

12.3.19 display-name

The <display-name> element is an optional element that specifies the JMS RA display name, a short name that can be displayed by GUI tools.

12.3.20 eis-type

The <eis-type> element specifies the Enterprise Information System (EIS) resource as JMS for this deployment descriptor file.

12.3.21 inbound-resourceadapter

The <inbound-resourceadapter> child element of the <resourceadapter> element is used to specify information about an inbound resource adapter. This contains information specific to the implementation of the resource adapter library as specified through the <messageadapter> element.

12.3.22 managedconnectionfactory-class

The <managedconnectionfactory-class> child element of the <connection-definition> specifies the fully qualified name of the Java class that implements the javax.resource.spi.ManagedConnectionFactory interface. This Java class is provided as part of resource adapter's implementation of connector architecture specified contracts. The implementation of this class is required to be a JavaBean. See http://java.sun.com/xml/ns/javaee/connector_1_6.xsd.

12.3.23 messageadapter

The <messageadapter> child element of the <inbound-resourceadapter> element is used to specify messaging capabilities of the resource adapter. This contains information specific to the implementation of the resource adapter library as specified through the <messagelistener> element.

12.3.24 messagelistener

The <messagelistener> child element of the <messageadapter> element is used to specify the implementation of the message listener as specified through the <messagelistener-type> element.

12.3.25 messagelistener-type

The <messagelistener-type> child element of the <messageadapter> element is used to specify the specific message listener supported by the messaging resource adapter. It contains information on the Java type of the message listener interface and an activation specification.

12.3.26 reauthentication-support

The <reauthentication-support> child element of the <outbound-resourceadapter> element specifies whether the resource adapter implementation supports re-authentication of existing managed connection instances. Vaues are true or false.

12.3.27 required-config-property

The <required-config-property> child element of the <activationspec> element is used to specify required properties.

12.3.28 resourceadapter

The <resourceadapter> element encompasses the configuration of a single resource adapter that is deployed to a foreign JMS provider.

12.3.29 resourceadapter-class

The <resourceadapter-class> child element of the <resourceadapter> element specifies the JMS RA implementation class.

12.3.30 resourceadapter-version

The <resourceadapter-version> element specifies the release version number for this deployment descriptor file.

12.3.31 outbound-resourceadapter

The <outbound-resourceadapter> child element of the <resourceadapter> element defines the configuration that is used to connect to an Enterprise Information System (EIS) from a foreign application server. The configuration defines connection factories for the resource adapter.

12.3.32 transaction-support

The <transaction-support> child element of the <outbound-resourceadapter> element specifies the level of transaction support provided by the resource adapter. The value must be one of the following:

- NoTransaction
- LocalTransaction
- XATransaction

12.3.33 vendor-name

The optional vendor-name element specifies the JMS RA vendor, Oracle Corporation, that can be displayed by GUI tools.

12.4 JMS RA Inbound Properties

The following table provides information on additional properties used to configure the MDBs that consume inbound messages.

Table 12–1 JMS RA inbound Configuration Properties

Property	Value	Description
acknowledgeMode	String	Set to Auto-acknowledge or Dups-ok-acknowledge. This controls the quality-of-service (QoS) provided by listener threads which consume messages and call the MDB's onMessage method. **Note**: This release of the JMS RA ignores this property. A QoS at least as strong as "Auto-acknowledge is always used. Future versions may honor requests for Dups-ok-acknowledge. Default is Auto-acknowledge.

Table 12–1 (Cont.) JMS RA inbound Configuration Properties

Property	Value	Description
`clientId`	`String`	A string value used to identify a JMS client. If set, connection(s) used by the listener threads are set to use this value. A `clientId` is required for applications interoperating with WebLogic Server JMS topics.
`connectionFactory`	`String`	The JNDI name of the connection factory. The JMS RA uses this connection factory to create the JMS connection it uses to receive messages for an MDB's `onMessage`. If the exception queue is enabled (see useExceptionQueue), the JMS Connector also uses a connection created from this connection factory for the production of messages to the exception queue. If the `onMessage` method is configured to be XA transacted, then the connection factory specified by this property must XA-backed. You must supply a `connectionFactory`, it can not be a `null` value.
`destination`	`String`	The JNDI name of the destination. The JMS RA receives messages from this destination and passes the received messages to this MDB's `onMessage`. You must supply a `destination`, it can not be a `null` value.
`destinationType`	`JMStype`	One of the following: `javax.jms.Topic`, `javax.jms.Queue`, `javax.jms.Destination`. You must supply a `destinationType`, it can not be a `null` value.
`exceptionQueue`	`String`	The JNDI location of the j`javax.jms.Queue` used as the exception queue. See Using an Exception Queue. Required when `UseExceptionQueue` is `true`, and ignored when `UseExceptionQueue` is `false`.
`FailureRetryInterval`	`long`	The amount of time, in milliseconds, an application server waits to for a `WorkManager` to allocate a listener thread for an endpoint. Default is 60,000 milliseconds.

Table 12–1 (Cont.) JMS RA inbound Configuration Properties

Property	Value	Description
`includeBodiesInExceptionQueue`	`boolean`	When `true`, messages sent to the exception queue include the message body. If many messages are sent to the exception queue during normal operation and the message body has no use in the exception queue, you can set the value to `false` to improve performance. This property is ignored when Using an Exception Queue is `false`. There are two cases where this property does not apply: ■ If the original message did not have a message body, then the message sent to the exception queue will not have a message body. ■ If a copy of the original message can not be created for any reason, then the original may be sent to the exception queue instead. This may result in a message body being sent to the exception queue. Default value is `true`.
`listenerThreadMaxIdleDuration`	`long`	The amount of time, in milliseconds, a listener thread which is not receiving any messages is kept available before being returned to the pool. As long as an endpoint is active, idle threads are not dropped if the result it causes the number of available threads to be less than the value of `minListenerThreads`. Default value is 300,000.
`loggerName`	`String`	The name used to obtain a logger for this endpoint (see `java.util.logging.Logger.getLogger`).

Table 12–1 (Cont.) JMS RA inbound Configuration Properties

Property	Value	Description
logLevel	Level	Each log message has an associated severity level. The level gives a rough guide to the importance and urgency of a log message. Supported values are: ■ ConnectionPool ■ ConnectionOps ■ TransactionalOps ■ ListenerThreads ■ INFO ■ CONFIG ■ FINE ■ FINER ■ FINEST ■ SEVERE ■ WARNING ■ OFF
maxDeliveryCount	int	If a message has a JMSXDeliveryCount value greater than the value of MaxDeliveryCnt, the message is discarded. If an exception queue is enabled, a copy of the message is sent to the exception queue. If the value of MaxDeliveryCnt is 0, no messages are discarded. See useExceptionQueue. **Note**: When an MDB responds to a message by throwing an exception, the message is not considered delivered and it may be redelivered. If an MDB always respond to a given message by throwing an exception, and MaxDeliveryCnt is 0 to prevent messages from ever being discarded, the result may be an MDB stuck in an infinite loop that continuously fails to process the same message. Default value is 5.

Table 12–1 (Cont.) JMS RA inbound Configuration Properties

Property	Value	Description
`maxListenerThreads`	`int`	The maximum number of listener threads created for an endpoint. ■ For queues: Using more than one thread may be useful in increasing the rate at which messages are consumed. ■ For topics: This value should always be 1.Each listener thread gets its own session and `TopicSubscriber`. ■ For durable subscribers: This value can be greater than 1 if multiple topic subscribers use the same subscription name simultaneously and from multiple connections. ■ For nondurable subscribers: Use one thread. Creating more threads creates more subscribers which translates into more copies of each message to process. Default value is `1`.
`maxTotalListenerThreads`	`int`	The additional number of listener threads created for an endpoint. The JMS RA implements a fairness policy for allocating threads whereby any destination which is below `maxListenerThreads` and needs more threads can reallocate threads from any other endpoint which has at least 2 more threads that it does. This fairness policy continues functioning even if the application server's `WorkManager` does not grant the JMS RA's requests for additional `maxTotalListenerThreads` new threads. Default value is 1000.
`messageSelector`	`String`	A selector expression used to filter messages sent to the MDB's `onMessage` method. It is used filter messages for the JMS sessions created for the listener threads. Default value is null, no message filtering.

Table 12–1 (Cont.) JMS RA inbound Configuration Properties

Property	Value	Description
`minListenerThreads`	`int`	The minimum number of listener threads created for this endpoint. Valid values are range from 1 to `maxListenerThreads`. If the application server allows the creation of 1 or more listener threads, but does not create that number of threads specified by `MinListenerThreads`, the endpoint activation does not fail and uses the available number of listener threads. Default value is 1.
noLocal	boolean	If `true`, messages are not published locally. In some cases, a connection may both publish and subscribe to a topic. The subscriber `noLocal` attribute allows a subscriber to inhibit the delivery of messages published by its own connection. Default value is `false`.
`password`	`String`	The password of the resource provider. See Section 5.1.5, "Setting userName/password." Default value is `null`.
`providerProperties`	`String`	A list of additional vendor specific properties defined as name/value pairs separated by a comma (","). For example: `my_property_name=my_property_value,another_property_name=another_property_value` See "WLConnection" in *Java API Reference for Oracle WebLogic Server*.
`receiveTimeout`	`long`	The amount of time, in milliseconds, the JMS RA waits for a message to arrive before exiting the current transaction. The transaction manager limits the amount of time a transaction lasts. Set this value such that such that the transaction manager does timeout a transaction during `onMessage` unless there is a problem with the transaction. For example: If the transaction manager timeout is set to 30 seconds, and `onMessage` routine never takes more than 10 seconds unless there is a problem with the transaction, then set this value to 200,000 (20 seconds). Default value is 15,000.

Table 12–1 (Cont.) JMS RA inbound Configuration Properties

Property	Value	Description
subscriptionDurability	boolean	When Durable, specifies a subscription receives messages sent while a subscriber is not active. Valid values are Durable or NonDurable. Requires DestinationType of javax.jms.Topic or javax.jms.Destination) and a non-null subscriptionName property. Default value is NonDurable.
subscriptionName	String	The reference name mapped to a subscriber when creating a durable subscriber for the listener thread. This property is required when the value of subscriptionDurability is Durable and DestinationType is javax.jms.Topic or javax.jms.Destination. Ignored in all other cases. For a given JMS server, a subscription name is assigned to at most one MDB which must have at most one listener thread.
useExceptionQueue	boolean	When true, messages that would otherwise be discarded are sent to the exception queue. Messages are normally sent to an exception queue when the maxDeliveryCount value is exceeded. Requires a valid exceptionQueue name. See maxDeliveryCount.
userName	String	The user name of the resource provider. See Section 5.1.5, "Setting userName/password." Default value is null.

12.5 JMS RA Outbound Configuration Properties

Each JMS Connection Factory interface has its own <connection-definition> element. The <connection-definition> defines classes and interfaces as required by the JCA specification and it defines resource-adapter configuration properties. The configuration properties can be set here, in the ra.xml file or set using the configuration tools provided by the JEE container where this resource adapter is deployed.

The following outbound configuration properties are supported:

Table 12–2 JMS RA Outbound Configuration Properties

Property	Value	Description
`group`	String	The name of the group to associate with this connection definition. See Section 6.2.2, "Configuring JMS RA Connection Factory Properties."
`rpResourceLocation`	String	Provides information to the resource adapter describing how to look up the JMS connection factory associated with the `<connection-definition>` element. Typically the form is: `connector:`*`connectorName`*`/`*`remote-jndi-name`*. The macros defined in a `groupDefinition` may be used as values in this field. See Section 6.2.2, "Configuring JMS RA Connection Factory Properties."
`autoCloseSession`	boolean	When `true`, the JMS RA attempts to close sessions when closing connections. This is a workaround for some foreign JMS providers that do not close XA connections when a session is open.

Table 12–2 (Cont.) JMS RA Outbound Configuration Properties

Property	Value	Description
`providerCustomization`	String	Specifies the JMS-provider-specific customization code. The value is a fully qualified class name. Provider customization code may customize anything the RA customization API provides a hook for which may include inbound, outbound and recovery scenarios. The 'ProviderCustomization' property need not be set if the JMS provider's connection factories implement the "public String getOracleJMSRAProviderCustomization()" method. If both are specified, the 'ProviderCustomization' property (if set to a non-null/non-blank value) takes precedence over the getOracleJMSRAProviderCustomization() method. Required when there is a MDB using this connection factory for inbound messaging with a distributed destination. For this case, the property value must be set to `weblogic.jms.ra.providers.wl.WLProviderInfo`.)
`clientId`	String	Optional. All connections created from this `connection-definition` have the JMS `clientID` set to the `setClientId` value. This should only be used when no `clientId` has been pre-configured for the connection factory in the WebLogic Server instance.

12.6 JMS RA adminobject Configuration Properties

The following `adminobject` configuration properties are supported:

Table 12–3 JMS RA adminobject Configuration Properties

Property	Value	Description
`group`	String	The name of the group to associate with this connection definition. See Section 7, "Configuring Destinations and Naming Contexts."
`rpResourceLocation`	String	Provides information to the resource adapter describing how to look up the JMS connection factory associated with the `<connection-definition>` element. Typically the form is: `connector:`*`connectorName`*`/`*`remote-jndi-name`*. *`connectorName`* must be defined in the `resourceProviderDefinitions`. See Section 8.1, "Basic Resource Provider Configuration." The macros defined in a `groupDefinition` may be used as values in this field. See Section 7, "Configuring Destinations and Naming Contexts."
`rpContextLocation`	String	Provides information to the resource adapter describing how how to lookup the destination context that is used to locate the Queue or Topic in a WebLogic Server instance. Typically the form is: `connector:`*`connectorName`*`/. connectorName` must be defined in the `resourceProviderDefinitions`. See Section 8.1, "Basic Resource Provider Configuration."