Oracle® Fusion Middleware

Administering the JMS Resource Adapter for Oracle WebLogic Server

12*c* (12.2.1.1.0)

E78173-01

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

ORACLE

Oracle Fusion Middleware Administering the JMS Resource Adapter for Oracle WebLogic Server, 12c (12.2.1.1.0)

E78173-01

Contents

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 that have purchased support 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 documentation describes the contents and organization of this guide—*Administering the JMS Resource Adapter for Oracle WebLogic Server*.

It includes the following topics:

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 1.7 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.

1.2 Guide to This Document

- This chapter, Introduction and Roadmap describes the organization and scope of this guide, including new features and related documentation.
- Understanding the WebLogic JMS Resource Adapter provides an overview of WebLogic JMS resource adapter components, concepts, and functionality.
- Administering the JMS Resource Adapter on Oracle GlassFish Server explains the components, design options, and other prerequisite considerations needed to use the JMS resource adapter to interoperate using WebLogic JMS in a foreign application server.
- Administering the JMS Resource Adapter on Oracle GlassFish Server describes additional configuration information and considerations when using deploying the JMS resource adapter on the .
- Understanding Message Consumption describes how to configure the deployment descriptor file, `ra.xml`, for a JMS resource adapter to have message-driven beans

(MDBs) asynchronously consume WebLogic JMS messages in a foreign application server as inbound messages.

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

1.3 Related Documentation

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

- *Administering JMS Resources for Oracle WebLogic Server* explains how to configure and manage WebLogic JMS resources.
- *Developing JMS Applications for Oracle WebLogic Server* explains how to develop WebLogic JMS applications.
- *Tuning Performance of Oracle WebLogic Server* provides information about monitoring performance and tuning the components in a WebLogic Server.
- *Developing Applications for Oracle WebLogic Server* explains how to develop 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 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.

For details about the JMS Resource Adapter example, see JMS Resource Adapter 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.

See Sample Applications and Code Examples in *Understanding Oracle WebLogic Server* for information about installing and configuring the suite of sample applications.

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

1.4.3 JMS Examples in the WebLogic Server Distribution

WebLogic Server optionally installs API code examples in *WL_HOME*`\samples\server\examples\src\examples`, where *WL_HOME* is the top-level directory of your WebLogic Server installation. Start the examples server and obtain information about the samples and how to run them from the WebLogic Server Start menu.

1.4.4 Additional JMS Examples Available for Download

Additional API examples are available for download at the Oracle Technology Network at `http://www.oracle.com/technology/index.html`. These examples are distributed as.zip files that you unzip into an existing WebLogic Server samples directory structure. You build and run the downloadable examples in the same manner as you would an installed WebLogic Server example.

1.5 New and Changed Features in This Release

See *What's New in Oracle WebLogic Server 12.2.1* for a comprehensive listing of the new WebLogic Server features introduced in this release.

2

Understanding the WebLogic JMS Resource Adapter

The following topics provide an overview of WebLogic JMS resource adapter components, concepts, and functionality:

- JMS Resource Adapter Overview
- Supported Application Servers
- Supported WebLogic Server Destinations
- Location of the JMS Resource Adapter Within WebLogic Server

2.1 JMS Resource Adapter Overview

WebLogic Server provides a Java EE Connector Architecture version 1.7 compliant resource adapter called the JMS resource adapter to provide an integration of a WebLogic JMS client with supported foreign application servers.

The JMS resource adapter includes the following features:

- Implementation of the Java EE Connector Architecture outbound and inbound contract for JMS.
- JNDI mapping to reference JMS connection factories and destinations.
- Message-drive bean (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.

  For more information, see 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.

  For more information, see 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 messaging system 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.

  For more information, see Support for the weblogic.jms.extension API.

- Support for the WebLogic security model.

  For more information, see Securing JMS Resource Adapter Connections.

- Advanced poison message handling. The JMS resource adapter can be configured to automatically redirect messages that have been redelivered multiple times to a designated error destination.

  For more information, see Using an Exception Queue.

- Simplified configuration of multiple destination JNDI mappings through the use of a single destination context resource adapter 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.

  For more information, see Using Automatic Destination Wrapping.

2.2 Supported Application Servers

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

Note:

Deploying the JMS resource adapter on Oracle WebLogic Server is not supported.

2.3 Supported WebLogic Server Destinations

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

2.4 Location of the JMS Resource Adapter Within WebLogic Server

The JMS resource adapter is contained in the file `wljmsra.jar`, which is located in the *`WL_HOME`*`\server\lib` directory of your WebLogic Server installation.

3

Administering the JMS Resource Adapter on Oracle GlassFish Server

The following topics describe the JMS resource adapter implementation, including the main components, design options, and other prerequisite considerations that are needed to use the JMS resource adapter to interoperate using WebLogic JMS in a foreign application server:

- JMS Resource Adapter Components
- How to Integrate the JMS Resource Adapter in a JMS Environment
- General Limitations and Considerations
- Support for the weblogic.jms.extension API
- JMS Resource Adapter Example

3.1 JMS Resource Adapter Components

The JMS resource adapter includes the following components:

- `ra.xml` file—An `xml` file used to configure the JMS resource adapter to interoperate between foreign application servers and WebLogic JMS.
- `wlthint3client.jar` file—The WebLogic Thin T3 Java client is a lightweight client that provides integration between applications running on foreign application servers and WebLogic Server. One common use case is integration with WebLogic JMS destinations.

  For more information, see Understanding the WebLogic Thin T3 Client in *Developing Stand-alone Clients for Oracle WebLogic Server*.
- The following classes and JAR files to support Oracle Wallet:
  - `oraclepki.jar`
  - `osdt_cert.jar`
  - `osdt_core.jar`
  - `WalletUtilWrapper.class`

  For more information, see Securing Credentials with Oracle Wallet.
- `weblogic.jms.ra.jar` file—The WebLogic Server resource adapter implementation that complies with the Java EE Connector Architecture version 1.7.

3.2 How to Integrate the JMS Resource Adapter in a JMS Environment

Complete the following tasks to integrate the JMS resource adapter with a foreign application server, allowing applications to interoperate with WebLogic JMS:

- Make sure that the application is JMS-compliant and can use the `weblogic.jms.extensions` API.

  See Understanding the WebLogic JMS Resource Adapter and General Limitations and Considerations for information about JMS resource adapter features, support, and limitations.

  See Understanding WebLogic JMS in *Developing JMS Applications for Oracle WebLogic Server* for basic information about programing WebLogic JMS.

- Configure the `ra.xml` file for your environment, as described in the following chapters:

  – Understanding Message Consumption

  – Sending Outbound JMS Messages

  – Configuring Destinations and Naming Contexts

  – Understanding Resource Providers

- Deploy and manage the JMS resource adapter. Use the foreign application server's native support for management tasks such as deployment, configuration, and monitoring. For more information, see Administering the JMS Resource Adapter on Oracle GlassFish Server.

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

3.3 General Limitations and Considerations

Note the following considerations and limitations when you use the JMS resource adapter:

- If two or more JMS resource adapters are deployed on the same application server, those deployments must use the same version of Oracle WebLogic Server in the `wljmsra.rar` file. Different release versions of the JMS resource adapter may contain differences that can produce unpredictable results.

- The JMS resource adapter does not automatically unsubscribe (that is, remove) a durable subscriber if the listening message-driven bean (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 Java Connector Architecture contract does not provide a standard implementation to provide the callback interface to remove a subscriber.

- Automatic JMS client failover is not supported.

For more information, see JMS Resource Adapter Outbound Connection Limitations.

- Applications that use the WebLogic JMS resource adapter automatically use WebLogic JMS connections that are associated with a foreign application server's connection manager. These managed JMS connections are created, used, and returned to the foreign application server's connection pool for reuse by other applications, as described in various Java EE specifications. This has the following limitations:
  - Using the `setClientID` method on a connection is not supported. If a `ClientID` property is set on a connection, then the connection can't be removed, and every application that reuses the connection gets a `ClientID` value. An application using such a connection is also prevented from invoking the `setClientID` method on that connection because this operation will have already been performed.
  - As specified by the Java EE 7 specification, other methods that may interfere with connection management by the container, such as `setExceptionListener`, are not supported.
  - You must set connection properties, for example by using the `setSubscriptionSharingPolicy` method, as part of the connection factory settings. Otherwise, you need to make sure that the connection and session are created in the managed connection pool before you attempt to set the property directly on the connection.
- When configuring the message listener for an MDB, the only values you may use with the `<trans-attribute>` element are `REQUIRED` and `NOT_SUPPORTED`.

  For more information, see the description of the `TransactionAttributeType` enumerated type at the following URL:

  `http://docs.oracle.com/javaee/7/api/javax/ejb/TransactionAttributeType.html`

3.4 Support for the weblogic.jms.extension API

The following topics explain how the JMS resource adapter supports the `weblogic.jms.extensions` API:

- Supported weblogic.jms.extension Interfaces
- Supported weblogic.jms.extension Classes

3.4.1 Supported weblogic.jms.extension Interfaces

The JMS resource adapter uses custom wrapper classes to automatically implement the `weblogic.jms.extensions` interfaces and provide support for WebLogic features such as Unit-of-Order (UOO) and message scheduling. The JMS resource adapter `ra.xml` file is prepopulated 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 the `weblogic.jms.extensions` API, as explained in JMS Resource Adapter Connection Factories.

The JMS resource adapter 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

The JMS resource adapter supports the following `weblogic.jms.extensions` classes:

- `weblogic.jms.extensions.ClientSAFEncrypt`
- `weblogic.jms.extensions.ClientSAFFactory`
- `weblogic.jms.extensions.ClientSAFGenerate`
- `weblogic.jms.extensions.ClientSAFParser`
- `weblogic.jms.extensions.ConsumerInfo`
- `weblogic.jms.extensions.DestinationInfo`
- `weblogic.jms.extensions.JMSDestinationAvailabilityHelper`
- `weblogic.jms.extensions.JMSHelper`
- `weblogic.jms.extensions.JMSMessageFactoryImpl`
- `weblogic.jms.extensions.JMSMessageInfo`
- `weblogic.jms.extensions.JMSModuleHelper`
- `weblogic.jms.extensions.JMSRuntimeHelper`
- `weblogic.jms.extensions.Schedule`

3.5 JMS Resource Adapter Example

Oracle provides a JMS resource adapter example that is available when you install and configure the examples component of WebLogic Server. This example demonstrates how to use a JMS resource adapter that is deployed on a foreign application server.

This example interoperates with the WebLogic JMS service using a simple employee clock-in application.

This example shows 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 resource adapter on one of the supported foreign application servers listed in Supported Application Servers.
- Configure JMS resource adapter `config-properties` file, which specifies:
  - A WebLogic JNDI context for the WebLogic Server instance that provides the JMS service
  - An `adminobject` element, which maps 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 a sample or factory that maps to the connection factory bound to WebLogic JNDI as `weblogic.jms.ConnectionFactory`
- Configure the `resource-ref` and `resource-env-ref` elements in a servlet's `web.xml` deployment descriptor file that map to the local JNDI names defined for the JMS resource adapter's connection factory and destinations
- Configure `activation-config` elements required to allow MDBs to consume inbound messages

When installed, the JMS resource adapter example is located in the *`ORACLE_HOME`*`\wl_server\samples\server\examples\src\samples\jms\resourceAdapter` directory. For more information, see Sample Applications and Code Examples in *Understanding Oracle WebLogic Server*.

4

Administering the JMS Resource Adapter on Oracle GlassFish Server

This chapter includes the following topics:

4.1 Configuration Information and Other Considerations

Note following configuration information and considerations to bear in mind when deploying the JMS resource adapter on :

- Oracle recommends using the Java Connector Architecture container to provide methods to set credentials using secure methods.

  For more information, see Securing JMS Resource Adapter Connections.

- supports lazy connection enlistment. For more information, see Lazy Enlistment of Connections in a Transaction.

- The Glassfish `XAResource` timeout value is 30 seconds by default and is independent of the transaction timeout. If you require a longer `XAResource` timeout, you must set the Glassfish system property `server-config.transaction-service.property.xaresource-txn-timeout` to a value equal to your transaction timeout value.

- Oracle GlassFish 3.1.*x* has a known issue regarding how outbound message-driven bean (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` exception if your application uses the `javax.ejb.MessageDrivenContext.setRollbackOnly` method 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 Oracle GlassFish, contact your Oracle Glassfish Support representative. If you have an open source version, a patch is available at `http://java.net/jira/browse/GLASSFISH-19094`.

- You may receive a `ClassNotFoundException` exception when deploying the JMS resource adapter on a Glassfish 3.1 cluster environment. If this happens, copy the `wlthint3client.jar` file that is packaged in the `wljmsra.rar` file to the <*Glassfish_install_dir*>`/lib` directory. For more information, see `http://java.net/jira/browse/GLASSFISH-19111`.

- You may receive the following exception if you undeploy and redeploy a WAR file multiple times.

  ```
  Failed to generate class for
  weblogic.messaging.dispatcher.FastDispatcherImpl_12120_WLStub
  ```

  The workaround for this exception is to restart the GlassFish server.

- See Oracle GlassFish Server 3.1.2 and 3.1.2.2 Release Notes in the *Oracle GlassFish Server Release Notes* for a summary of new product features in the 3.1.2 and 3.1.2.2 releases, and descriptions and workarounds for known issues and limitations.

4.2 Additional GlassFish Server Resources

See the following references for additional GlassFish Server resources:

- The Oracle GlassFish Server download page is available at the following URL:

  `https://glassfish.java.net/download.html`

- *Oracle GlassFish Server 3.1 Quick Start Guide*, which is available at the following URL:

  `http://docs.oracle.com/cd/E18930_01/html/821-2432/index.html`

5

Understanding Message Consumption

The following topics describe how to configure the JMS resource adapters `ra.xml` file to configure message-driven beans (MDBs) to asynchronously consume WebLogic JMS messages in a foreign application server as inbound messages:

- Configuring MDBs to Consume Inbound Messages
- Configuring Advanced WebLogic JMS Resources
- Best Practices for Inbound Communication

5.1 Configuring MDBs to Consume Inbound Messages

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

- Configuring inbound-resourceadapter Properties in the ra.xml File
- Configuring activation-config Properties in the ejb-jar.xml File
- Thread Management

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

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

- Required activation-config Properties
- Optional activation-config Properties
- Example inbound-resourceadapter Configuration

5.1.1.1 Required activation-config Properties

You specify the required `inbound-resourceadapter` properties in the WebLogic JMS Resource-Adapter's `ra.xml` file, which is located in the *`WL_HOME`*`\server\lib` directory of your WebLogic Server installation.

The JMS properties you must specify include:

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

You define the values for the `ConnectionFactory`, `destination`, and `destinationType` properties 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 that are assigned to the JMS resource adapter values of the `connection-definition` and `adminobject` elements.

See also the following topics:

- Configuring activation-config Properties in the ejb-jar.xml File
- Configuring JMS Resource Adapter Connection Factory Properties
- Configuring Destinations and Naming Contexts

5.1.1.2 Optional activation-config Properties

The JMS resource adapter supports a number of additional `activation-config` properties.

See Configuring activation-config Properties in the ejb-jar.xml File and JMS Resource Adapter Inbound Properties.

5.1.1.3 Example inbound-resourceadapter Configuration

The following code example shows an `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 topics provide information on how to configure `activation-config` properties in the `ejb-jar.xml` file:

- The ejb-jar.xml File and Annotations
- Configuring Optional activation-config Properties in the ejb-jar.xml File

- Example ejb-jar.xml File with JMS Resource Adapter 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 as of EJB 3.0. You can use annotations to declare metadata in place of descriptor elements.

For information about how to program and implement EJB 3.2 compliant MDBs, see Using EJB 3.2 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` file.

The 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`—The type is one of the following:
  - `javax.jms.Topic`
  - `javax.jms.Queue`
  - `javax.jms.Destination`

In addition, if your MDB application uses optional `activation-config` properties, the foreign application server may require that these optional `activation-config` properties are defined in the `ra.xml` file as `required-config-property` elements.

For more information, see the documentation provided by the vendor of your application server.

5.1.2.2.1 Example Queue Configuration

The following example shows the `activation-config` properties for a queue configuration. In this example, the MDB that dequeues messages from a queue with the JNDI name `wljmsra/queue` uses 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 the `activation-config` properties for a topic configuration. In this example, the MDB that dequeues messages from a topic with the JNDI name `wljmsra/pdtopic1` uses 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 obtain the appropriate MDB behavior for your environment. For more information, see JMS Resource Adapter Inbound Properties.

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

The following example shows an `ejb-jar.xml` file with JMS resource adapter `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, see the JMS resource adapter sample application example in your WebLogic Server distribution as described in JMS Resource Adapter Example.

5.1.3 Thread Management

The JMS resource adapter provides the following properties to manage the threads in destinations used by MDBs that consume inbound messages.

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

5.1.3.1 Setting the Maximum Threads for a Physical Destination

Note the following when configuring 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—You may note 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 results in excessive 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 additional 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 resource adapter 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 resource adapter logs a warning message.
- The JMS resource adapter implements a fairness policy for allocating threads whereby any individual physical destination that is currently being serviced by less than maxListenerThreads, and that needs more threads, can reallocate threads from any other physical destination in the same distributed destination that has at least 2 more threads than it does. This fairness policy is independent of the foreign

application server's `WorkManager` instance, which may not grant the JMS resource adapter's requests for additional `maxTotalListenerThreads` new threads.

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

5.1.4 Using an Exception Queue

The JMS resource adapter allows you to configure an exception queue for inbound communications to handle poison messages from the inbound MDB queue. When the `UseExceptionQueue` property is enabled, 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. For more information, 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. Because `javax.jms.Destination` is not a valid property type, each destination header is translated into a descriptive message. Note that `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 only part of the message body being copied.

  - If the copy process is successful, the boolean property `GJRA_CopySuccessful` is added with the value `true`.
  - The string property `GJRA_DeliveryFailureReason` is added that contains the description explaining why the message was not delivered.
  - If the MDB `onMessage` method generates an exception immediately prior to the delivery failure, the string property `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. For more information, see includeBodiesInExceptionQueue.

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 User Name and Password Properties

The user name and password properties allow you to pass authentication parameters to the resource provider. When neither of these properties is set, then 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`and `password` properties are passed to the `createConnection` method as the user name and 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 topics provide information on how to configure advanced message processing for inbound messages:

- The JMS Resource Adapter and Sharable Subscriptions
- Using Ordered Message Processing
- Design Strategies When Consuming from DistributedTopics
- Consuming from Standalone (Nondistributed) Topics

5.2.1 The JMS Resource Adapter and Sharable Subscriptions

The JMS resource adapter uses a `SHARABLE` subscription sharing policy and `UNRESTRICTED` client ID policy when it processes inbound messages by overriding any configured connection factory settings. A `subscriptionName` must be provided for durable subscriptions because the JMS resource adapter's MDBs do not generate subscription names. Subscriptions are shared when they are durable or nondurable, as follows:

- Durable—Consumers on the same distributed topic can share a durable subscriptions only if they have the same `clientID`, message selector, and `subscriptionName`.
- Nondurable—Consumers on the same distributed topic can share a nondurable subscriptions only if they have the same `clientID` and `messageSelector`.

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>
. . .
```

For more information, 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, then you must configure your application to use WebLogic JMS unit-of-ordering (UOO) processing.

For more information, 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:

- Replicated Versus Partitioned Distributed Topics
- One-Copy-Per-Server Design Strategy for Distributed Topics
- One-Copy-Per-Application Design Strategy for Distributed Topics

5.2.3.1 Replicated Versus Partitioned Distributed Topics

Replicated and partitioned distributed topics are supported for inbound message consumption as follows:

- 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 and 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 distributed topic. That is each instance only receives a subset of the messages that are sent to the distributed topic.
- 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 distributed topic)

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.

For more information, see Example One-Copy-Per-Application EJB Configuration.

Note:

If you do not specify `TopicMessageDistributionMode=One-Copy-Per-Server`, the JMS resource adapter 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-value>
      </activation-config-property>
      <activation-config-property>
        <activation-config-property-name>ProviderProperties</activation-config-
property-name>
        <activation-config-property-value>TopicMessageDistributionMode=One-Copy-Per-
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`*

  In the preceding property specification, *`aUniqueIdForTheServer`* represents a unique identifier for your foreign server.

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

  ```
  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. For more information, see Example One-Copy-Per-Application EJB Configuration.

5.2.3.3.1.1 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-
property-name>
        <activation-config-property-value>TopicMessageDistributionMode=One-Copy-Per-
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 Standalone (Nondistributed) 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.

For more information, see Implementing One-Copy-Per-Server .

5.3 Best Practices for Inbound Communication

Note the following tuning recommendations and best practices for configuring a JMS resource adapter for inbound communication:

- Ensure that the JMS resource adapter allocates more threads than distributed destination members.
- Always configure a `clientID` when using topics. The JMS resource adapter uses a `SHARABLE` subscription sharing policy when processing inbound messages, which requires a configured `clientId` for either durable or nondurable subscribers.

  If no `clientID` is set, then the MDB does not receive the message and the error condition logged in the foreign application server log.

  For more information, see The JMS Resource Adapter 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 resource adapter does not support sharable connections. If you use the `<resource-ref>` element in your application's deployment descriptor file (`web.xml`, and `ejb-jar.xml`) to identify the JMS connection factories that are looked up and used in that application, then 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/7/api/javax/annotation/Resource.html#shareable--`.

6

Sending Outbound JMS Messages

The following topics describe how to send JMS messages using the JMS resource adapter:

- JMS Resource Adapter Outbound Communication Basics
- Defining Outbound Connections
- JMS Resource Adapter Outbound Connection Limitations
- Understanding How Outbound Messages Are Load Balanced
- Configuring Transaction Support for Outbound Communication
- Configuring Authentication for Outbound Communication

6.1 JMS Resource Adapter Outbound Communication Basics

The JMS resource adapter is a thin wrapper of the 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 resource adapter 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 Java Connector Architecture specification and additional configuration properties.

6.2.1 JMS Resource Adapter Connection Factories

The JMS resource adapter 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 Example JMS Resource Adapter ra.xml File.

6.2.2 Configuring JMS Resource Adapter Connection Factory Properties

The JMS resource adapter supports additional configuration properties that can be set in the `ra.xml` file or in the Java EE container of the foreign application server where the JMS resource adapter is deployed, as explained in JMS Resource Adapter Outbound Configuration Properties.

The following is an example WebLogic XA connection factory with additional configuration properties:

```
. . .
<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 Resource Adapter Outbound Connection Limitations

Note the following connection management limitations:

- The WebLogic automatic JMS client failover feature enables a 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 resource adapter's ability to associate the best Interposed Transaction Managers (ITMs) for managed connections. The JMS resource adapter automatically disables this feature and throws an exception if application code tries to enable automatic reconnect programmatically.
- The `UP` and `DOWN` events are not currently supported in WebLogic Server. You also cannot create a connection to a specific WebLogic Server instance. This may result in repeated use of a bad connection because JMS resource adapter connection purging may not immediately identify bad connections before they are resused.
- Because `UP` events are not currently supported in WebLogic Server, the JMS resource adapter does not support connection rebalancing.
- The JMS resource adapter outbound implementation is not aware of the `UP` and `DOWN` events of Managed Server instances in a cluster with which an outbound application is communicating. Because of this limitation, note the following:
  - You may see the continued use of a bad connection because JMS resource adapter connection purging may not immediately identify a bad connection before it is reused.
  - The JMS resource adapter does not automatically rebalance connections.

6.4 Understanding How Outbound Messages Are Load Balanced

A WebLogic Server cluster consists of multiple Managed 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 Managed Server instances, with each member hosted by a separate JMS server instance. For more information, see Understanding WebLogic Server Clustering in *Administering Clusters for Oracle WebLogic Server*.

The following topics provide information about how the JMS resource adapter load balances outbound WebLogic JMS communication:

- RMI Load Balancing Using the Connection Factory
- Load Balancing Messages to Distributed Destinations

6.4.1 RMI Load Balancing Using the Connection Factory

The JMS resource adapter supports RMI object load balancing using WebLogic JMS connection factories to create connections and sessions that point to different WebLogic Server instances.

For more information, 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 resource adapter 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 a distributed destination on a WebLogic Server instance that is different from the

instance that the JMS resource adapter'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 Server 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

The JMS resource adapter provides outbound XA capability for applications using WebLogic JMS according to the Java Connector Architecture standard. For more information, see Transaction Support for Outbound Communication.

6.6 Configuring Authentication for Outbound Communication

Configure the `authentication-mechanism` element in the `ra.xml` file, as explained in Java Connector Architecture Security.

7

Configuring Destinations and Naming Contexts

The following topics provide information on configuring `adminobject` elements to define destinations and naming contexts for inbound and outbound communication:

- About Context Objects for Administered Objects
- Using Automatic Destination Wrapping
- Administered Objects for Queues and Topics
- Example Administered Object Stanza

7.1 About Context Objects for Administered Objects

A context object for an administered object is used to look up queues and topics dynamically using automatic destination wrapping, which is described in Using Automatic Destination Wrapping.

Automatic destination wrapping allows applications to bind an administered object in the application server's JNDI tree that represents a WebLogic JNDI context. Applications can then look up this WebLogic JNDI context to look up JMS destinations by their name in the WebLogic JNDI tree. This can simplify the configuration for applications that have a large number of destinations because only one administered object must be configured with the application server.

Use the following elements to define a context object for an administered object:

- `<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 object for the administered object. You specify this using the following syntax:

  `connector:<connectorName>/`

The macros defined in a `groupDefinition` may be used for the value of connector in this property value. For example, you can use `connector:{connectorName}/` in place of `connector:wlDevelopment/` when the value of the macro `{connectorName}` within a group is *wlDevelopment*. For more information, see Advanced Method for Configuring Resource Providers.

7.2 Using Automatic Destination Wrapping

Automatic destination wrapping allows an administered object to automatically wrap provider destinations, eliminating the need to create an administered object for each provider destination and explicitly binding the administered object to the destination's JNDI. A single administered object 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 resource adapter administered objects bound in JNDI. These administered objects may represent JMS destinations or a destination context. When used for JMS destinations, you must defined one administered object for each JMS destination. If an applications uses 20 JMS destinations, then you must configure 20 individual administered objects, one for each destination. Applications must also define and map these 20 JMS destinations in their Java EE descriptors as `resource-env-references`.

The destination context administered object allows you to integrate a single destination context, which in turn may be used to lookup any number of JMS destinations at the WebLogic Server instance serving as the JMS provider. Applications need to define only 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 deployment time. For example, the JMS resource adapter gets the source and target destinations from end users only at runtime.
- If your application needs to access a large number of destinations. This avoids the need to configure a matching set of JMS resource adapter destinations for all destinations of a resource provider. Otherwise, each resource provider destination has to be mapped to a configured administered object before it can be used by the application.

For example, an administered object 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>
. . .
```

In the preceding example, *`rp_name`* represents the name of the resource provider defined in the `ra.xml` file. For more information, see Administered Object Configuration Properties.

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

```
. . .
@Resource(mappedName="myContext")
```

```
private javax.naming.Context wljmsraContext;
. . .
```

Then you can look up the queue destinations using the following context:

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

7.3 Administered Objects for Queues and Topics

You can create the following type of administered objects for JMS queues and topics:

- For queues, you create an instance of a `<adminobject-class>` object: `oracle.j2ee.ra.jms.generic.WLQueueAdmin`
- For topics, you create an instance of a `<adminobject-class>` object: `oracle.j2ee.ra.jms.generic.WLTopicAdmin`

7.4 Example Administered Object Stanza

You can create the following administered object stanza:

```
. . .
<!-- Context admin object -->
<adminobject>
   <adminobject-interface>weblogic.jms.ra.WLDestinationContextInterface</adminobject-
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-
interface>
   <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-
interface>
   <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

The following topics describe how to configure and use resource providers that define the JNDI properties that allow the JMS resource adapter to connect to the WebLogic JMS provider:

- Basic Resource Provider Configuration
- Advanced Method for Configuring Resource Providers
- Example Resource Provider Configuration

8.1 Basic Resource Provider Configuration

The JMS resource adapter utilizes a configuration property named `resourceProviderDefinitions` in the `ra.xml` file to define resource providers, such as JNDI properties. The `resourceProviderDefinitions` property is used by the JMS resource adapter to access WebLogic JMS. You can configure multiple resource providers in an `ra.xml` file.

Note:

When specifying the `resourceProviderDefinitions` property, you may need to use the per cent character (%) as an escape character if your WebLogic Server JNDI URL includes one or more commas to delimit multiple properties.

For example, if the URL is represented as `t3://host:port,host2:port2`, then the JMS resource adapter will fail to parse the URL because a comma is inserted after `host:port`.

To resolve this issue, insert a per cent character (%) immediately prior to the comma. For example: `t3://host:port%,host2,port`

To configure a resource provider, complete the following steps in the `ra.xml` file:

1. **a.** Specify the `resourceProviderDefinitions` property as the value of the `<config-property-name>` element. For example:

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

   b. Specify `java.lang.String` as the value of the `<config-property-type>` element. For example:

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

   c. Define the specific JNDI properties for a resource provider by using the following name-value pair pattern:

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

In this name-value pattern:

- *RP_NAME* is a unique name for the JNDI properties of a resource provider and is used with the `rpResourceLocation` configuration property of the `<connection-definition>` and `<adminobject>` elements. For more information, see Sending Outbound JMS Messages.

Note:

Each defined resource provider name (*RP_NAME*) must be unique in the `ra.xml` file.

- *property1=value1,property2=value2,...* is a comma-separated list of name-value pairs that define the JNDI properties for a resource provider.

See Example Resource Provider Configuration for an example configuration.

8.2 Advanced Method for Configuring Resource Providers

The JMS resource adapter supports the `groupDefinitions` property to provide an advanced method for configuring resource providers. Using this property enables you to create a compatible set of messaging objects while providing flexible address resolution of connection factories and administered objects. Using the `groupDefinitions` property has a dependency on the following components:

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

To configure a `groupDefinitions` property in the `ra.xml` file, complete the following steps:

1. Specify `groupDefinitions` as the value of the `<config-property-name>` element. For example:

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

2. Specify `java.lang.String` as value of the `<config-property-type>` element. For example:

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

3. Define a `Group` configuration by using the following pattern:

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

   In the preceding pattern:

   - *GROUP_NAME* is a unique name that represents a `Group` configuration.
   - *macro* is a comma-separated list of name-value pairs that define a set of compatible messaging objects. The JMS resource adapter substitutes these pairs for `rpResourceLocation` configuration properties in connection factories and administered objects.

See Example Resource Provider Configuration for an example configuration.

8.3 Example Resource Provider Configuration

The following shows an example resource provider configuration. In this example:

- Two resource providers are configured: `rp1`and `rp2`.
- Two Group objects are configured: `GroupA` and `GroupB`.

```
. . .
     <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

The following topics describe transaction processing and recovery when using the JMS resource adapter to interoperate between a foreign application server and WebLogic Server:

- JMS Resource Adapter Transaction Support
- Lazy Enlistment of Connections in a Transaction
- WebLogic Cluster-Wide Recovery

9.1 JMS Resource Adapter Transaction Support

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

9.1.1 Transaction Support for Outbound Communication

The JMS resource adapter provides outbound XA capability for applications using WebLogic JMS in accordance with the Java Connector Architecture specification. The JMS resource adapter's managed connection holds a WebLogic JMS client connection and session pointing to a WebLogic Server cluster member. The managed connection also references the interposed transaction manager (ITM) stub pointing to the same WebLogic Server cluster member. Although a WebLogic ITM is able to handle JMS operations occurring in different WebLogic Server instances, it is more efficient to have JMS operations and the ITM located on the same cluster member and same instance (ITM affinity).

When a the managed connection for a JMS resource adapter is involved in a global transaction, note the following:

- The Oracle GlassFish server instance invokes the `getXAResource` method on the managed connection to enlist its operations into the global transaction (in accordance with the Java Connector Architecture transaction contract between a resource adapter and an application server).
- If cluster-wide transaction recovery is enabled, the ITYM transparently handles any XA calls and forwards them to the correct cluster member instance to maintain ITM affinity.

  For information about how to configure cluster-wide transaction recovery of distributed transactions across all the ITMs of a cluster, 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` wrapper.

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. Note the following:

- If the resource adapter contract is used, then 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 that includes the ITM stub pointing to the same WebLogic Server instance as the WebLogic JMS session that is receiving messages. If the WebLogic destination is a distributed destination, the worker threads are distributed evenly across the destination members.
- If native MBDs are in use, the application server is responsible for receiving messages. The use scenario of the JMS resource adapter is the same as for 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 enlisted in the transaction. This occurs even if the connection is never used, 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; that is, only if data is transmitted through the connection.

Both a JMS resource adapter and a 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 ITMs of a cluster by selecting **Enable Cluster-Wide Recovery** in the Cluster: Configuration: JTA page in the WebLogic Server Administration Console.

When cluster-wide recovery is enabled, each ITM in a WebLogic Server cluster is aware of the transaction distribution across the entire cluster. This allows the ITM on each cluster member to determine whether it should handle a given XA call or forward it to the appropriate ITM on another cluster member.

For more information about WebLogic Server transaction processing, see 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

The following topics describe how the JMS resource adapter responds to WebLogic Server and foreign application server failures:

- WebLogic Server Failure
- WebLogic Distributed Destination Member Failure
- Transaction Recovery
- Understanding WebLogic Service Migration

10.1 WebLogic Server Failure

The JMS resource adapter detects the connection failure through the exception listener and JMS operations. When a WebLogic Server instance failure is detected, the corresponding connections and sessions are closed and new ones are created. The failed connections are purged from the connection pool to prevent them from being reused.

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 to restarted instances.

10.2 WebLogic Distributed Destination Member Failure

When a WebLogic Server distributed destination member fails, the JMS resource adapter for the inbound MDB container closes any existing connections and sessions and opens new ones to an available cluster member.

- If the failed cluster member is restored, or a new member is added, the JMS resource adapter 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` and `DOWN` events to achieve the rebalance. To obtain this behavior, you must disable server affinity and enable load balancing. For information, see Load Balancing for JMS in *Administering Clusters for Oracle WebLogic Server*.
- For failures that cannot be recovered from quickly, the adapter logs the failure and retries periodically.

10.3 Transaction Recovery

The following topics provide information about the transaction recovery process when a foreign application server or WebLogic Server becomes unavailable before a transaction is complete:

- Transaction Recovery When WebLogic Server is Unavailable
- Transaction Recovery When the Foreign Server is Unavailable

10.3.1 Transaction Recovery When WebLogic Server is Unavailable

This section provides information about how a transaction directed by a foreign transaction manager is processed when the WebLogic Server instance is unavailable.

- 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.

- 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 Server 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 resource adapter `XAResource` wrapper looks up 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 completed only after 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 and 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* for more information.

Note:

For the recovery process to succeed, every ITM in the cluster must be available because all ITMs in the cluster are polled to compile the complete list of in-doubt transactions.

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

10.4 Understanding WebLogic Service Migration

The following topic provides information about JMS resource adapter behavior during WebLogic Server service migration.

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

Note:

WebLogic JMS service migration provides high availability for JMS services, but it does not provide load balancing.

See Migratable Services in *Administering Clusters for Oracle WebLogic Server* for more information.

Note the following details regarding how the JMS resource adapter handles inbound and outbound communication during JMS service migration:

- Inbound communication

  During JMS service migration, the JMS resource adapter 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.

- 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 resource adapter detects the migration and redirects JMS operations to the new migrated destination members.

11

Securing JMS Resource Adapter Connections

The following topics describe security considerations for the JMS resource adapter:

- Java Connector Architecture Security
- WebLogic JMS Security
- Specifying User Name and Password Credentials
- Securing Credentials with Oracle Wallet
- Secure Communication

11.1 Java Connector Architecture Security

The JMS resource adapter is fully compliant with the Java Connector Architecture security contract, as described in *Java™ EE Connector Architecture Specification, version 1.7* at `http://download.oracle.com/otndocs/jcp/connector_architecture-1_7-mrel-spec/`.

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

For example:

```
. . .
<outbound-resourceadapter>
. . .
    <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>
. . .
```

11.2 WebLogic JMS Security

The following topics provide information about WebLogic JMS security:

- Overview of JMS Security Models
- 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 lookup as the JNDI user name and password credentials. WebLogic JMS assumes all related operations are done within the same thread under the same subject that is used for later authorizations in the server. The user name 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* for more information.

11.3 Specifying User Name and Password Credentials

You can apply any of the following methods to specify the user name and password credentials:

- Specifying a User Name and Password for Inbound Connections Using the Java Connector Architecture Container
- Specifying a User Name and Password for Inbound Connections Using JNDI
- Specifying a User Name and Password for Inbound Connections Using a Connection Factory
- Specifying a User Name and Password for Outbound Connections

Oracle recommends using the host application server's Java Connector Architecture container. Most vendors provide Java Connector Architecture 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 Securing Credentials with Oracle Wallet for more information.

11.3.1 Specifying a User Name and Password for Inbound Connections Using the Java Connector Architecture Container

Your application can provide the user name and password credentials in the `activation-spec` of an inbound resource adapter. The `activation spec` is then passed into the JMS resource adapter by the foreign application server's Java Connector Architecture container.

See Administering the JMS Resource Adapter on Oracle GlassFish Server for detailed information about how to specify a user name and password using the Java Connector Architecture container of your foreign application server.

11.3.2 Specifying a User Name and Password for Inbound Connections Using JNDI

You can configure the `jndiEnv` property in the `resourceProviderDefinitions` to include the user name and password credentials as follows:

```
<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 Example JNDI Configurations for Setting Credentials for more information.

11.3.3 Specifying a User Name and Password for Inbound Connections Using a Connection Factory

The JMS resource adapter 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 user name and password.
- If the connection is created without user name and password, then the JNDI user name and password are used.
- All subsequent JMS operations use the resultant subject from the connection creation call regardless of what is on the thread.

11.3.4 Specifying a User Name and Password for Outbound Connections

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

The JMS resource adapter does not provide configuration attributes to implement the user name and password for outbound connections.

See Secure Communication for information about securing outbound communication.

Some foreign application servers may provide secure credentials between domains. If so, consult 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 resource adapter encryption utility. This step creates a `cwallet.sso` file at the specified location that maps an alias to the secured credentials.

  See Using the wljmsra Encryption Utility for more information.

- Provide the alias in the JMS resource adapter deployment descriptor or connection pool configuration.

- Provide the location of the `cwallet.sso` file in the JMS resource adapter deployment descriptor or connection pool configuration.

11.4.1 Example JNDI Configurations for Setting Credentials

The following table provide examples showing how you can set JMS resource adapter JNDI environment properties.

JNDI Settings	Behavior
`java.naming.security.principal=`*`principal`* `java.naming.security.principal=`*`principal`* `java.naming.security.credentials=`*`credentials`*	The JMS resource adapter runtime uses the values of *`principal`* and *`credentials`* to access Oracle WebLogic Server destinations.
`java.naming.security.principal=`*`principal`* `java.naming.security.credentials=->`*`alias`* `weblogic.jms.walletDir=`*`dirctory`*	The JMS resource adapter runtime uses the value of *`principal`* as the user name 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=`*`dirctory`*	The JMS resource adapter runtime uses the value of *`alias1`* to retrieve and use the user name 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=`*`dirctory`*	The JMS resource adapter runtime uses the value of *`alias`* to retrieve and use the user name 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=`*`dirctory`*	The JMS resource adapter runtime uses the value of *`principal`* as the user name 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 resource adapter provides a command-line utility to add application credentials into an Oracle Wallet file. To run the utility, change to the *`WL_HOME`*`server/lib` directory 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.
```

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'.
```

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

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

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

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

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

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

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 Communication

Oracle recommends using SSL or t3s to secure information being sent.

See Configuring SSL in *Administering Security for Oracle WebLogic Server 12c (12.2.1)* for more information.

12

JMS Resource Adapter Deployment Descriptor Elements and Properties

The Java Connector Architecture compliant deployment descriptor for the JMS resource adapter is named `ra.xml`. This file is used to integrate a WebLogic JMS client with supported foreign application servers.

The following topics provide information about the WebLogic JMS resource adapter deployment descriptor file, `ra.xml`:

12.1 Example JMS Resource Adapter ra.xml File

The following is an example `ra.xml` deployment descriptor 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>
```

12.2 JMS Resource Adapter Element Descriptions

The element hierarchy of the `ra.xml` deployment descriptor file is provided below. The number of occurrences that is allowed is listed in braces and follows the element name.

```
<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}
```

12.2.1 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.

12.2.2 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.

12.2.3 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.

12.2.4 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.

12.2.5 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.

12.2.6 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.

12.2.7 authentication-mechanism-type

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

- BasicPassword
- Kerbv5

12.2.8 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 administered object.

12.2.9 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 java.lang.String.

12.2.10 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.

12.2.11 connection-definition

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

12.2.12 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 for more information.

12.2.13 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.

12.2.14 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 for more information.

12.2.15 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 for more information.

12.2.16 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.

12.2.17 connector

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

12.2.18 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

12.2.19 display-name

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

12.2.20 eis-type

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

12.2.21 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.

12.2.22 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 for more information.

12.2.23 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.

12.2.24 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.

12.2.25 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.

12.2.26 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`.

12.2.27 required-config-property

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

12.2.28 resourceadapter

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

12.2.29 resourceadapter-class

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

12.2.30 resourceadapter-version

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

12.2.31 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.

12.2.32 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

12.2.33 vendor-name

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

12.3 JMS Resource Adapter Inbound Properties

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

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 resource adapter 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`.
`clientId`	`String`	A string value used to identify a JMS client. If set, the 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 resource adapter 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, the JMS Connector also uses a connection created from this connection factory for the production of messages to the exception queue. See the description of the useExceptionQueue property, in this table, for more information. 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 cannot be a `null` value.
`destination`	`String`	The JNDI name of the destination. The JMS resource adapter receives messages from this destination and passes the received messages to this MDB's `onMessage`. You must supply a `destination`, it cannot be a `null` value.

Property	Value	Description
destinationType	JMSType	One of the following: javax.jms.Topic, javax.jms.Queue, javax.jms.Destination. You must supply a destinationType, it cannot be a null value.
exceptionQueue	String	The JNDI location of the jjavax.jms.Queue used as the exception queue. See Using an Exception Queue for more information. 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.
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. See Using an Exception Queue for more information. 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. See minListenerThreads for more information. Default value is 300,000.
loggerName	String	The name used to obtain a logger for this endpoint (see java.util.logging.Logger.getLogger).

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 the description of the useExceptionQueue property, in this table, for more information. **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.
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.

Property	Value	Description
`maxTotalListenerThreads`	`int`	The additional number of listener threads created for an endpoint. The JMS resource adapter implements a fairness policy for allocating threads whereby any destination which is below and needs more threads can reallocate threads from any other endpoint which has at least 2 more threads that it does. See maxListenerThreads for more information. This fairness policy continues functioning even if the application server's `WorkManager` does not grant the JMS resource adapter'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.
`minListenerThreads`	`int`	The minimum number of listener threads created for this endpoint. Valid values are range from 1 to maxListenerThreads. See maxListenerThreads for more information. 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 Setting User Name and Password Properties for more information 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*.

Property	Value	Description
receiveTimeout	long	The amount of time, in milliseconds, the JMS resource adapter 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 the 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.
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. See the description of the useExceptionQueue property, in this table, for more information. 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 the subscriptionDurability property 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 for more information.
userName	String	The user name of the resource provider. See Setting User Name and Password Properties for more information. Default value is null.

12.4 JMS Resource Adapter 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 Java Connector Architecture specification and it defines resource adapter configuration properties. The configuration properties can be set in the ra.xml file or set using the configuration tools provided by the Java EE container where this resource adapter is deployed.

The following table lists the outbound configuration properties that are supported:

Property	Value	Description
`group`	`String`	The name of the group to associate with this connection definition. See Configuring JMS Resource Adapter Connection Factory Properties for more information.
`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 Configuring JMS Resource Adapter Connection Factory Properties for more information.
`autoCloseSession`	`boolean`	When `true`, the JMS resource adapter 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.
`providerCustomization`	`String`	Specifies the JMS-provider-specific custom code. The value is a fully qualified class name. Provider custom code may customize anything for which the JMS resource adapter customization API provides a hook, 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 not set to a null or blank value) takes precedence over the `getOracleJMSRAProviderCustomization()` method.
`clientId`	`String`	Optional. All connections created from this `connection-definition` have the JMS `clientID` set to the `setClientId` value. This should be used only when no `clientId` has been preconfigured for the connection factory in the WebLogic Server instance.

12.5 Administered Object Configuration Properties

The following administered object configuration properties are supported:

Property	Value	Description
`group`	`String`	The name of the group to associate with this connection definition. See Configuring Destinations and Naming Contexts for more information.

Property	Value	Description
`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 Basic Resource Provider Configuration for more information. The macros defined in a `groupDefinition` may be used as values in this field. See Configuring Destinations and Naming Contexts for more information.
`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 Basic Resource Provider Configuration for more information.