Oracle® Fusion Middleware

Administering JMS Resources for Oracle WebLogic Server

14c (14.1.2.0.0)
F47996-01
December 2024

ORACLE

Oracle Fusion Middleware Administering JMS Resources for Oracle WebLogic Server, 14c (14.1.2.0.0)

F47996-01

Contents

8 Monitoring JMS Statistics and Managing Messages

A JMS Resource Definition Elements Reference

B Configuring JMS Application Modules for Deployment (Deprecated)

Preface

This guide is a resource for system administrators who configure, manage, and monitor Oracle WebLogic JMS resources, including JMS servers, standalone destinations (queues and topics), distributed destinations, and connection factories.

Audience

The information in this document is relevant to production-phase administration, monitoring, and performance tuning. It does not address the pre-production development or testing phases of a software project.

It is assumed that the reader is familiar with WebLogic Server system administration. This document emphasizes the value-added features provided by WebLogic Server JMS and key information about how to use WebLogic Server features and components to maintain WebLogic JMS in a production environment.

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.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Documentation

Samples and Tutorials

Oracle provides a variety of code examples and tutorials that show WebLogic Server configuration and API use, and provide practical instructions on how to perform key development tasks. For more information, see Sample Applications and Code Examples in *Understanding Oracle WebLogic Server*.

New and Changed WebLogic Server Features

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

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

WebLogic Server Value-Added JMS Features

WebLogic JMS provides numerous WebLogic JMS Extension APIs that go beyond the standard JMS APIs specified by the JMS 1.1 and 2.0 Specification. These are available at http://www.oracle.com/technetwork/java/jms/index.html. Moreover, WebLogic JMS is tightly integrated into the WebLogic Server platform, allowing you to build secure Jakarta EE applications that can be easily monitored and administered through the WebLogic Remote Console. In addition to fully supporting XA transactions, WebLogic JMS also features high availability through its clustering and service migration features, while also providing interoperability with other releases of WebLogic Server and third-party messaging providers.

The following sections provide an overview of the unique features and powerful capabilities of WebLogic JMS.

Enterprise-Grade Reliability

WebLogic JMS includes the following reliability features:

- Out of the box transaction support:
  - Fully supported transactions, including distributed transactions, between JMS applications and other transaction-capable resources using the Jakarta Transaction (JTA), as described in Using Transactions with WebLogic JMS in *Developing JMS Applications for Oracle WebLogic Server*.
  - Fully integrated Transaction Manager, as described in Introducing Transactions in *Developing JTA Applications for Oracle WebLogic Server*.
- File or database-persistent message storage (both fully XA transaction capable). See Using the WebLogic Persistent Store in *Administering the WebLogic Persistent Store*.
- Message Store-and-Forward (SAF) that is clusterable and improves reliability by locally storing messages sent to unavailable remote destinations. See Understanding the Store-and-Forward Service in *Administering the Store-and-Forward Service for Oracle WebLogic Server*.
- If a server or network failure occurs then, JMS producer and consumer objects attempts to transparently fail-over to another server instance, if one is available. See Automatic JMS Client Failover in *Developing JMS Applications for Oracle WebLogic Server*.
- Supports connection clustering using connection factories targeted on multiple WebLogic Servers, as described in Configuring WebLogic JMS Clustering.
- System-assisted configuration of Uniform Distributed Queues, Replicated Distributed Topics, and Partitioned Distributed Topics that provide high availability, load balancing, and failover support in a cluster, as described in Using Distributed Destinations and Developing

Advanced Pug/Sub Applications in *Developing JMS Applications for Oracle WebLogic Server*.

- Automatic whole server migration provides improved cluster reliability and server migration WebLogic Server now supports automatic and manual migration of a clustered server instance and all the services it hosts from one machine to another, as described in Configuring WebLogic JMS Clustering.
- WebLogic JMS provides complete in-place restart support to provide the retry mechanism to auto-restart a failed file-based or JDBC-based store and its upper services without any server conflicts. See Restart In Placein *Administering the WebLogic Persistent Store*.
- Redirection of failed or expired messages to error destinations, as described in Managing Rolled Back, Recovered, Redelivered, or Expired Messages in *Developing JMS Applications for Oracle WebLogic Server*.
- Supports the JMS Delivery Count message property `JMSXDeliveryCount`, which specifies the number of message delivery attempts, where the first attempt is 1, the second is 2, and so on. WebLogic Server makes a best effort for a persistent delivery count, so that the delivery count does not reset back to one after a server restart. See Message in *Developing JMS Applications for Oracle WebLogic Server*.
- Provides three levels of load balancing: network-level, JMS connections, and distributed destinations.

Enterprise-Level Features

WebLogic JMS includes the following enterprise-level features:

- WebLogic Server fully supports the JMS 1.1 and JMS 2.0 specifications (available at `http://www.oracle.com/technetwork/java/jms/index.html`), in compliance with the Jakarta EE 8.0 platform specification, and provides numerous WebLogic JMS Extensions that go beyond the standard JMS APIs.
- Robust message and destination management capabilities:
  - Administrators can manipulate most messages in a running JMS Server, using either the WebLogic Remote Console or runtime APIs. See Managing JMS Messages.
  - Administrators can pause and resume message production, message insertion (in-flight messages), and message consumption operations on a given JMS destination, or on all the destinations hosted by a single JMS server, using either the WebLogic Remote Console or runtime APIs. See Controlling Message Operations on Destinations.
  - Message-Driven Beans (MDBs), EJBs also supply message pause and resume functionality, and can even automatically and temporarily pause during error conditions. See Programming and Configuring MDBs: Details in *Developing Message-Driven Beans for Oracle WebLogic Server*.
- Modular deployment of JMS resources, which are defined by an Extensible Markup Language (XML) file so that you can migrate your application and the required JMS configuration from environment to environment without opening an enterprise application file, and without extensive manual JMS reconfiguration. See Overview of JMS Modules.
- JMS message producers can group ordered messages into a single Unit-of-Order, which guarantees that all such messages are processed serially in the order in which they were created. See Using Message Unit-of-Order in *Developing JMS Applications for Oracle WebLogic Server*.
- To provide an even more restricted notion of a group than the Message Unit-of-Order feature, the Message Unit-of-Work (UOW) feature allows JMS producers to identify certain

messages as components of a UOW message group, and allows a JMS consumer to process them as such. For example, a JMS producer can designate a set of messages that must be delivered to a single client without interruption, so that the messages can be processed as a unit. See Using Unit-of-Work Message Groups in *Developing JMS Applications for Oracle WebLogic Server*.

- Message life cycle logging provides an administrator with better transparency about the existence of JMS messages from the JMS server viewpoint, in particular basic life cycle events, such as message production, consumption, and removal. See Message Life Cycle Logging.
- Timer services available for scheduled message delivery, as described in Setting Message Delivery Times in *Developing JMS Applications for Oracle WebLogic Server*.
- Flexible expired message policies to handle expired messages, as described in Handling Expired Messages in *Tuning Performance of Oracle WebLogic Server*.
- Support messages containing XML. See Defining XML Message Selectors Using the XML Selector Method in *Developing JMS Applications for Oracle WebLogic Server*.
- The WebLogic Thin T3 Client JAR (`wlthint3client.jar`) is a light-weight, performant alternative to the `wlfullclient.jar` and `wlclient.jar`, which are (IIOP) remote client jars. The Thin T3 client has a minimal footprint while providing access to a rich set of APIs that are appropriate for client usage. See Developing a WebLogic Thin T3 Client in *Developing Standalone Clients for Oracle WebLogic Server*.
- The JMS Store-and-Forward client enables standalone JMS clients to reliably send messages to server-side JMS destinations, even when the JMS client cannot temporarily reach a destination (for example, due to a network connection failure). While disconnected from the server, messages sent by the JMS SAF client are stored locally on the client and are forwarded to server-side JMS destinations when the client reconnects. See Reliably Sending Messages Using the JMS SAF Client in *Developing Stand alone Clients for Oracle WebLogic Server*.
- Automatic pooling of JMS client resources in server-side applications via JMS resource-reference pooling. Server-side applications use standard JMS APIs, but get automatic resource pooling. See Enhanced Jakarta EE Support for Using WebLogic JMS With EJBs and Servlets in *Developing JMS Applications for Oracle WebLogic Server*.

Performance

WebLogic JMS includes enterprise-class performance features, such as automatic message paging, message compression, and Document Object Model (DOM) support for XML messages:

- WebLogic Server uses highly optimized disk access algorithms and other internal enhancements to provide a unified messaging kernel that improves both JMS-based and Web Services messaging performance. See Using the WebLogic Persistent Store in *Administering Server Environments for Oracle WebLogic Server*.
- You may greatly improve the performance of typical non-persistent messaging with One-Way Message Sends. When configured on a connection factory, associated producers can send messages without internally waiting for a response from the target destination's host JMS server. You can choose to allow queue senders and topic publishers to do one-way sends, or to limit this capability to topic publishers only. You can also specify a "One-Way Window Size" to determine when a two-way message is required to regulate the producer before it can continue making additional one-way sends.

- Message paging automatically begins during peak load periods to free up virtual memory. See Paging Out Messages To Free Up Memory in *Tuning Performance of Oracle WebLogic Server*.
- Administrators can enable the compression of messages that exceed a specified threshold size to improve the performance of sending messages travelling across Java Virtual Machine (JVM) boundaries using either the WebLogic Remote Console or runtime APIs. See Compressing Messages in *Tuning Performance of Oracle WebLogic Server*.
- Synchronous consumers can also use the same efficient behavior as asynchronous consumers by enabling the Prefetch Mode for Synchronous Consumers option on the consumer's JMS connection factory, using either the WebLogic Remote Console or runtime APIs. See Using the Prefetch Mode to Create a Synchronous Message Pipeline in *Developing JMS Applications for Oracle WebLogic Server*.
- There are a wide variety of performance tuning options for JMS messages. See Tuning WebLogic JMS in *Tuning Performance of Oracle WebLogic Server*.
- There is MDB transaction batching supported by processing multiple messages in a single transaction. See Using Batching with Message-Driven Beans in *Developing Message-Driven Beans for Oracle WebLogic Server*.
- JMS SAF provides better performance than the WebLogic Messaging Bridge across clusters. See Tuning WebLogic JMS Store-and-Forward in *Tuning Performance of Oracle WebLogic Server*.
- DOM (Document Object Model) support for sending XML messages greatly improves performance for implementations that already use a DOM, because those applications do not have to flatten the DOM before sending XML messages. See Sending XML Messages in *Developing JMS Applications for Oracle WebLogic Server*.
- Message flow control during peak load periods, including blocking overactive senders, as described in Controlling the Flow of Messages on JMS Servers and Destinations and Defining Quota in *Tuning Performance of Oracle WebLogic Server*.
- The automatic pooling of connections and other objects by the JMS wrappers using JMS resource-reference pooling. See Enhanced Jakarta EE Support for Using WebLogic JMS With EJBs and Servlets in *Developing JMS Applications for Oracle WebLogic Server*.
- Multicasting of messages for simultaneous delivery to many clients using IP multicast, as described in Using Multicasting with WebLogic JMS in *Developing JMS Applications for Oracle WebLogic Server*.

Tight Integration with WebLogic Server

WebLogic JMS includes the following features to enable tight integration with WebLogic Server:

- JMS can be accessed locally by server-side applications without a network call because the destinations can exist on the same server as the application.
- Uses same ports, protocols, and user identities as WebLogic Server (T3, IIOP, and HTTP tunnelling protocols, optionally with Secure Socket Layer (SSL)).
- Web Services, Jakarta Enterprise Beans (including MDBs), and servlets supplied by WebLogic Server can work in close concert with JMS.
- Can be configured and monitored by using the same WebLogic Remote Console, or by using the JMS API.

- Supports WebLogic Scripting Tool (WLST) to initiate, manage, and make persistent configuration changes interactively or by using an executable script. See Using WLST to Manage JMS Servers and JMS System Module Resources.
- Provides complete Java Management Extensions (JMX) administrative and monitoring APIs, as described in *Developing Custom Management Utilities Using JMX for Oracle WebLogic Server*.
- Fully-integrated Transaction Manager, as described in Introducing Transactions in *Developing JTA Applications for Oracle WebLogic Server*.
- Leverages sophisticated security model built into WebLogic Server (policy engine), as described in Understanding WebLogic Security and Resource Types You Can Secure with Policies in *Securing Resources Using Roles and Policies for Oracle WebLogic Server*.

Interoperability with Other Messaging Services

WebLogic JMS includes the following features for interoperability with other messaging services:

- Fully supports direct interoperability with prior WebLogic Server releases as described in WebLogic Server Compatibility in *Information Roadmap for Oracle WebLogic Server* .
- Forwards messages transactionally by the WebLogic Messaging Bridge to other JMS providers — as well as to other instances and releases of WebLogic JMS, as described in *Administering the WebLogic Messaging Bridge for Oracle WebLogic Server*.
- Supports mapping of other JMS providers so their objects appear in the WebLogic JNDI tree as local JMS objects. Also references remote instances of WebLogic Server in another cluster or domain in the local Java Naming and Directory Interface (JNDI) tree. See Foreign Server Configuration.
- Uses MDBs to transactionally receive messages from multiple JMS providers. See Programming and Configuring MDBs: Details in *Developing Message-Driven Beans for Oracle WebLogic Server*.
- Provides automatic transaction enlistment of non-WebLogic JMS client resources in server-side applications using JMS resource-reference pooling. See Enhanced Jakarta EE Support for Using WebLogic JMS With EJBs and Servlets in *Developing JMS Applications for Oracle WebLogic Server*.
- Provides integration with Oracle Tuxedo messaging provided by WebLogic Tuxedo Connector. See How to Configure the Oracle Tuxedo Queuing Bridge in the *Administering WebLogic Tuxedo Connector for Oracle WebLogic Server*.
- The WebLogic JMS C API enables programs written in 'C' to participate in JMS applications. This implementation of the JMS C API uses JNI in order to access a Java Virtual Machine (JVM). See WebLogic JMS C API in *Developing JMS Applications for Oracle WebLogic Server*.
- Uses Oracle Streams Advanced Queuing (AQ) to provide database-integrated message queuing functionality that leverages the functions of Oracle Database to manage messages. WebLogic Server interoperates with Oracle AQ using a Foreign JMS and JDBC data source configuration in a WebLogic Server domain. Both local and remote JMS clients can use Oracle AQ destinations from WebLogic JNDI. See Interoperating with Oracle AQ JMS.

2

Understanding JMS Resource Configuration

Learn about basic WebLogic JMS concepts and features, and how they work with other application components and Oracle WebLogic Server.

This chapter includes the following sections:

- Overview of JMS and Oracle WebLogic Server
  The WebLogic Server implementation of JMS is an enterprise-class messaging system that is tightly integrated into the WebLogic Server platform.
- What are JMS Configuration Resources?
  JMS configuration resources, such as destinations and connections factories, are stored outside of the WebLogic domain as module descriptor files. These files are defined by XML documents that conform to the `weblogic-jms.xsd` schema.
- Overview of JMS Servers
  JMS servers are environment-related configuration entities that act as management containers for destination resources within JMS modules that are targeted to specific JMS servers.
- Overview of JMS Modules
  JMS modules are application-related definitions that are independent of the domain environment. You create and manage JMS resources either as *system modules* or as *application modules*.
- Other Environment-Related System Resources for WebLogic JMS
  In addition to JMS Servers and System Modules, an administrator may also need to configure one of the following artifacts.

Overview of JMS and Oracle WebLogic Server

The WebLogic Server implementation of JMS is an enterprise-class messaging system that is tightly integrated into the WebLogic Server platform.

It fully supports the JMS 2.0 Specification, available at `http://www.oracle.com/technetwork/java/jms/index.html`, and also provides numerous WebLogic JMS Extensions that go beyond the standard JMS APIs.

- What Is the Jakarta Messaging?
- WebLogic JMS Architecture and Environment
- Domain Configuration

What Is the Jakarta Messaging?

An enterprise messaging system enables applications to asynchronously communicate with one another through the exchange of messages. A message is a request, report, and/or event that contains information needed to coordinate communication between different applications. A message provides a level of abstraction, allowing you to separate the details about the destination system from the application code.

JMS is a standard API for accessing enterprise messaging systems that is implemented by industry messaging providers. Specifically, JMS:

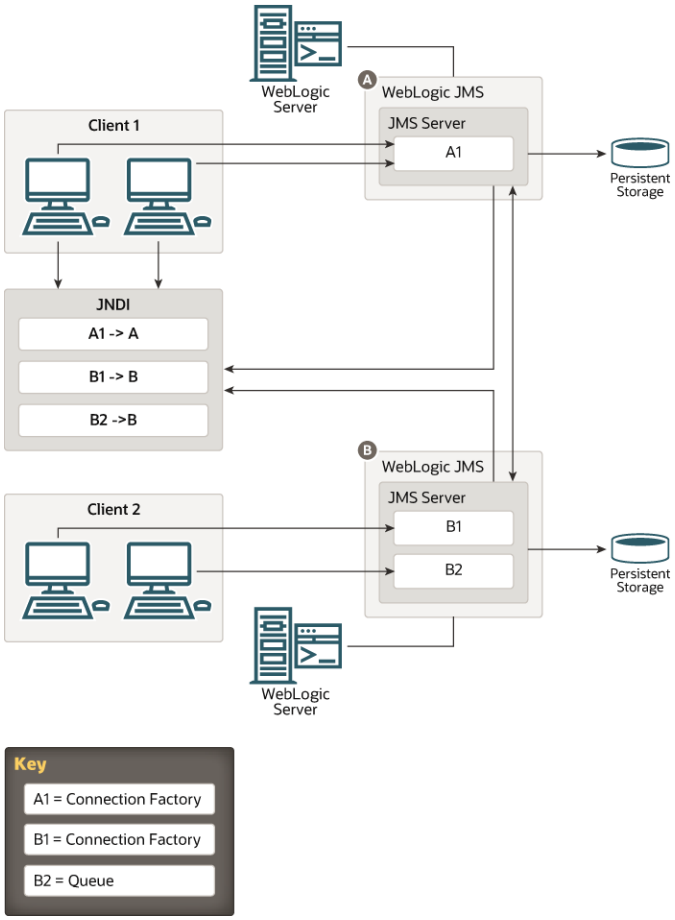
- Enables Jakarta applications that share a messaging system to exchange messages
- Simplifies application development by providing a standard interface for creating, sending, and receiving messages

WebLogic JMS accepts messages from *producer* applications and delivers them to *consumer* applications. For more information about JMS API programming with WebLogic Server, see Understanding the Simplified API Programming Model in *Developing JMS Applications for Oracle WebLogic Server*.

WebLogic JMS Architecture and Environment

Figure 2-1 illustrates the WebLogic JMS architecture.

Figure 2-1 WebLogic JMS Architecture



The major components of the WebLogic JMS architecture include:

- A JMS server is an environment-related configuration entity that acts as a management container for JMS queue and topic resources defined within JMS modules that are targeted to specific JMS servers. A JMS server's primary responsibility for its targeted destinations is to maintain information on what persistent store is used for any persistent messages that arrive on the destinations, and to maintain the states of durable subscribers created on the

destinations. You can configure one or more JMS servers per domain, and a JMS server can manage one or more JMS modules. See Overview of JMS Servers.

- JMS modules contain configuration resources, such as standalone queue and topic destinations, distributed destinations, and connection factories, and are defined by XML documents that conform to the `weblogic-jms.xsd` schema. See What Are JMS Configuration Resources?
- Client JMS applications either produce messages to destinations or consume messages from destinations.
- JNDI (Java Naming and Directory Interface) provides a server *lookup* facility.
- WebLogic persistent storage (a server instance's default store, a user-defined file store, or a user-defined JDBC-accessible store) for storing persistent message data.

Domain Configuration

In general, the WebLogic Server domain configuration file (`config.xml`) contains the configuration information required for a domain. This configuration information can be further classified into *environment-related* information and *application-related* information. Examples of environment-related information are the identification and definition of JMS servers, JDBC data sources, WebLogic persistent stores, and server network addresses. These system resources are usually unique from domain to domain.

The configuration and management of these system resources are the responsibility of a WebLogic administrator, who usually receives this information from an organization's system administrator or MIS department. To accomplish these tasks, an administrator can use the WebLogic Remote Console, various command-line tools, such as WebLogic Scripting Tool (WLST), or JMX APIs for programmatic administration.

Examples of application-related definitions that are independent of the domain environment are the various Jakarta EE application components configurations, such as EAR, WAR, JAR, RAR files, and JMS and JDBC modules. The application components are originally developed and packaged by an application development team, and may contain optional programs (compiled Java code) and respective configuration information (also called descriptors, which are mostly stored as XML files). In the case of JMS and JDBC modules, however, there are no compiled Java programs involved. These pre-packaged applications are given to WebLogic Server administrators for deployment in a WebLogic domain.

The process of deploying an application links the application components to the environment-specific resource definitions, such as which server instances should host a given application component (targeting), and the WebLogic persistent store to use for persisting JMS messages.

After the initial deployment is completed, an administrator has only limited control over deployed applications. For example, administrators are only allowed to ensure the proper life cycle of these applications (deploy, undeploy, redeploy, remove, so on.) and to tune the parameters, such as increasing or decreasing the number of instances of any given application to satisfy the client needs. Other than life cycle and tuning, any other modification to these applications must be completed by the application development team.

What are JMS Configuration Resources?

JMS configuration resources, such as destinations and connections factories, are stored outside of the WebLogic domain as module descriptor files. These files are defined by XML documents that conform to the `weblogic-jms.xsd` schema.

JMS modules do not include JMS server definitions, which are stored in the WebLogic domain configuration file, as described in Overview of JMS Servers.

You create and manage JMS resources either as *system modules*, similar to the way they were managed prior to this release, or as *application modules*. JMS application modules are a WebLogic-specific extension of Jakarta EE modules and can be deployed either with a Jakarta EE application (as a packaged resource) or as standalone modules that can be made globally available. See Overview of JMS Modules.

Overview of JMS Servers

JMS servers are environment-related configuration entities that act as management containers for destination resources within JMS modules that are targeted to specific JMS servers.

A JMS server's primary responsibility for its targeted destinations is to maintain information on what persistent store is used for any persistent messages that arrive on the destinations, and to maintain the states of durable subscribers created on the destinations. As a container for targeted destinations, any configuration or runtime changes to a JMS server can affect all of its destinations.

JMS servers persist in the domain's `config.xml` file and multiple JMS servers can be configured on the various WebLogic Server instances in a cluster, as long as they are uniquely named. Client applications use either the JNDI tree or the `java:/comp/env` naming context to look up a connection factory and create a connection to establish communication with a JMS server. Each JMS server handles requests for all targeted module destinations. Requests for destinations not handled by a JMS server are forwarded to the appropriate server instance.

Overview of JMS Modules

JMS modules are application-related definitions that are independent of the domain environment. You create and manage JMS resources either as *system modules* or as *application modules*.

JMS system modules are typically configured using the WebLogic Remote Console or WebLogic Scripting Tool (WLST), which adds a reference to the module in the domain's `config.xml` file. JMS application modules are a WebLogic-specific extension of Jakarta EE modules and can be deployed either with a Jakarta EE application (as a packaged resource) or as stand-alone modules that can be made globally available.

The main difference between system modules and application modules is in the ownership. System modules are owned and modified by the WebLogic administrator and are available to all applications. Application modules are owned and modified by the WebLogic developers, who package the JMS resource modules with the application's EAR file.

With modular deployment of JMS resources, you can migrate your application and the required JMS configuration from environment to environment, such as from a testing environment to a production environment, without opening an enterprise application file (such as an EAR file) or a standalone JMS module, and without extensive manual JMS reconfiguration.

These sections describe the different types of JMS modules and the resources that they can contain:

- JMS System Modules
- JMS Application Modules (Deprecated)
- Comparing JMS System Modules and Application Modules
- Configurable JMS Resources in Modules

- JMS Schema
- JMS Interop Modules (Deprecated)

JMS System Modules

WebLogic Administrators typically use the WebLogic Remote Console or the WebLogic Scripting Tool (WLST) to create and deploy (target) JMS modules, and to configure the module's configuration resources, such as queues, and topic connection factories.

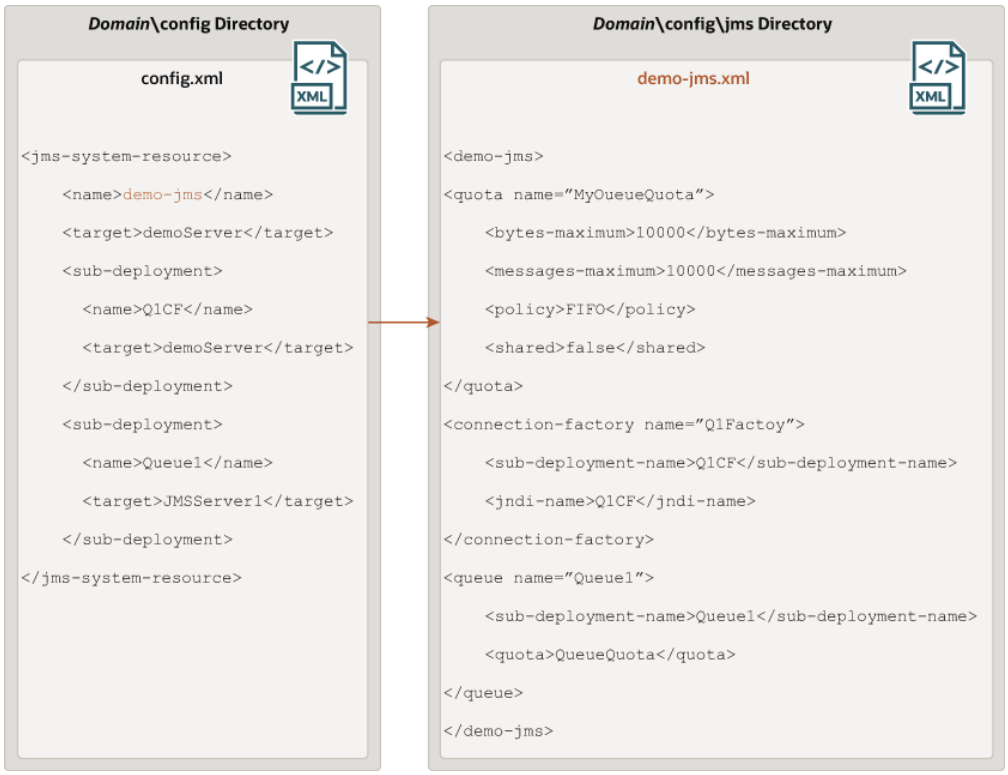
JMS modules that you configure this way are considered *system modules*. JMS system modules are owned by the administrator, who can at any time add, modify, or delete resources. System modules are globally available for targeting to servers and clusters configured in the domain, and therefore are available to all applications deployed on the same targets and to client applications.

When you create a JMS system module WebLogic Server creates a JMS module file in the `config\jms` subdirectory of the domain directory, and adds a reference to the module in the domain's `config.xml` file as a `JMSSystemResource` element. This reference includes the path to the JMS system module file and a list of target servers and clusters on which the module is deployed.

The JMS module conforms to the `weblogic-jms.xsd` schema, as described in JMS Schema. System modules are also accessible through WebLogic Management Extension (JMX) utilities, as a JMSSystemResourceMBean. The naming convention for JMS system modules is *`MyJMSModule`*`-jms.xml`.

Figure 2-2 shows an example of a JMS system module listing in the domain's `config.xml` file and the module that it maps to in the `config\jms` directory.

Figure 2-2 Reference from config.xml to a JMS System Module



For more information about configuring JMS system modules, see Configuring Basic JMS System Resources.

JMS Application Modules (Deprecated)

JMS configuration resources can also be managed as deployable application modules, similar to standard Jakarta EE descriptor-based modules. JMS Application modules can be deployed either with a Jakarta EE application as a *packaged module*, where the resources in the module are optionally made available to only the enclosing application (i.e., application-scoped), or as a *standalone module* that provides global access to the resources defined in that module.

Application developers typically create application modules in an enterprise-level IDE or another development tool that supports editing XML descriptor files, then package the JMS modules with an application and pass the application to a WebLogic Administrator to deploy, manage, and tune.

As discussed in Domain Configuration, JMS application modules do not contain compiled Java programs as part of the package, enabling administrators or application developers to create and manage JMS resources on demand.

For more information about configuring JMS application modules, see Configuring JMS Application Modules for Deployment.

> **Note:**
>
> WebLogic JMS Application Modules for Deployment are deprecated, including packaged and standalone modules. Support for JMS Application Modules will be removed in a future release. Oracle recommends creating required JMS configuration using system modules.

Comparing JMS System Modules and Application Modules

A key to understanding WebLogic JMS configuration and management is that *who* creates a JMS resource and *how* a JMS resource is created determine how a resource is deployed and modified. Both WebLogic administrators and programmers can configure JMS modules.

In contrast to system modules, deployed application modules are owned by the developer who created and packaged the module, rather than the administrator who deploys the module, which means that the administrator has more limited control over deployed resources. When deploying an application module, an administrator can change resource properties that were specified in the module, but the administrator cannot add or delete resources. As with other Jakarta EE modules, deployment configuration changes for a application module are stored in a deployment plan for the module, leaving the original module untouched.

Table 2-1 lists the JMS module types and how they can be configured and modified.

Table 2-1 JMS Module Types and Configuration and Management Options

Module Type	Created with	Dynamically Add/Remove Modules	Modify with JMX Remotely	Modify with Deployment Tuning Plan (non-remote)	Modify with Remote Console	Scoping	Default Sub module Targeting
System	Remote Console or WLST	Yes	Yes	No	Yes using JMX	Global and local	No
Application	IDE or XML editor	No must be redeployed	No	Yes using deployment plan	Yes using deployment plan	Global, local, and application	Yes

For more information about preparing JMS application modules for deployment, see Configuring JMS Application Modules for Deployment and Deploying Applications and Modules with weblogic.deployer in *Deploying Applications to Oracle WebLogic Server*.

Configurable JMS Resources in Modules

The following configuration resources are defined as part of a system module or an application module:

- Queue and topic destinations, as described in Queue and Topic Destination Configuration.
- Connection factories, as described in Connection Factory Configuration.
- Templates, as described in JMS Template Configuration.
- Destination keys, as described in Destination Key Configuration.
- Quota, as described in Quota Configuration.
- Distributed destinations, as described in Configuring Distributed Destination Resources.
- Foreign servers, as described in Configuring Foreign Server Resources to Access Third-Party JMS Providers.
- JMS store-and-forward (SAF) configuration items, as described in JMS Store-and-Forward (SAF).

All other JMS environment-related resources must be configured by the administrator as domain configuration resources. This includes:

- JMS servers (required), as described in Overview of JMS Servers.
- Store-and-Forward agents (optional), as described in JMS Store-and-Forward (SAF).
- Path service (optional), as described in Path Service.
- Messaging bridges (optional), as described in Messaging Bridges.
- Persistent stores (optional), as described in Persistent Stores.

For more information about configuring JMS system modules, see Configuring Basic JMS System Resources.

JMS Schema

In support of the modular configuration model for JMS resources, Oracle provides a schema for WebLogic JMS objects: `weblogic-jms.xsd`. When you create JMS resource modules

(descriptors), the modules must conform to the schema. IDEs and other tools can validate JMS resource modules based on this schema.

The `weblogic-jms.xsd` schema is available online at `http://xmlns.oracle.com/weblogic/weblogic-jms/1.4/weblogic-jms.xsd`.

JMS Interop Modules (Deprecated)

> **Note:**
>
> JMS Interop Modules are deprecated in WebLogic Server 12.1.1. If you have a module named `interop-jms.xml` in your `config.xml`, convert it to a regular system module. See JMS System Module Configuration.

A JMS interop module is a special type of JMS system resource module. It is created and managed as a result of a JMS configuration upgrade for this release, and/or through the use of WebLogic JMX MBean APIs from prior releases.

JMS interop modules differ in many ways from JMS system resource modules, as follows.

- The JMS module descriptor is always named `interop-jms.xml` and the file exists in the domain's `config\jms` directory.
- Interop modules are *owned* by the system, as opposed to other JMS system resource modules, which are owned mainly by an administrator.
- Interop modules are targeted everywhere in the domain.
- The JMS resources that exist in a JMS interop module can be accessed and managed using deprecated JMX (MBean) APIs.
- The MBean of a JMS interop module is `JMSInteropModuleMBean`, which is a child MBean of `DomainMBean`, and can be looked up from `DomainMBean` like any other child MBean in a domain.

A JMS interop module can also implement many of the WebLogic Server 9.x or later features, such as Message Unit-of-Order and destination quota. However, it *cannot* implement the following WebLogic Server 9.x or later features:

- Uniform distributed destination resources
- JMS Store-and Forward resources

> **Note:**
>
> Use of any new features in the current release in a JMS interop module may possibly break compatibility with JMX clients prior to WebLogic Server 9.x.

Other Environment-Related System Resources for WebLogic JMS

In addition to JMS Servers and System Modules, an administrator may also need to configure one of the following artifacts.

- Persistent Stores
- JMS Store-and-Forward
- Path Service
- Messaging Bridges

Persistent Stores

The WebLogic persistent store provides a built-in, high-performance storage solution for all subsystems and services that require persistence. For example, it can store persistent JMS messages or temporarily store messages sent using the Store-and-Forward feature. Each WebLogic Server instance in a domain has a default persistent store that requires no configuration and can be simultaneously used by subsystems that prefer to use the system's default storage. However, you can also configure a dedicated file-based store or JDBC database-accessible store to suit your JMS implementation. For more information about configuring a persistent store for JMS, see Using the WebLogic Persistent Store in *Administering Server Environments for Oracle WebLogic Server*.

JMS Store-and-Forward

The SAF service enables WebLogic Server to deliver messages reliably between applications that are distributed across WebLogic Server instances. For example, with the SAF service, an application that runs on or connects to a local WebLogic Server instance can reliably send messages to a destination that resides on a remote server. If the destination is not available at the moment that the messages are sent, either because of network problems or system failures, then the messages are saved on a local server instance, and are forwarded to the remote destination as soon as it becomes available.

JMS modules use the SAF service to enable local JMS message producers to reliably send messages to remote JMS queues or topics. See Configuring SAF for JMS Messages in *Administering the Store-and-Forward Service for Oracle WebLogic Server*.

Path Service

The WebLogic Server path service is a persistent map that can be used to store the mapping of a group of messages to a messaging resource by pinning messages to a distributed queue member or store-and-forward path. For more information about configuring a path service, see Using the WebLogic Path Service.

Messaging Bridges

The Messaging Bridge lets you to configure a forwarding mechanism between any two messaging products, providing interoperability between separate implementations of WebLogic JMS, or between WebLogic JMS and another messaging product. The messaging bridge instances and bridge source and target destination instances persist in the domain's

`config.xml` file. See Understanding the Messaging Bridge in *Administering the WebLogic Messaging Bridge for Oracle WebLogic Server*.

3

Configuring Basic JMS System Resources

Learn how to configure and manage basic JMS system resources for Oracle WebLogic Server, such as JMS servers and JMS system modules.

This chapter includes the following sections:

- Methods for Configuring JMS System Resources
  WebLogic Administrators can use multiple methods to configure and deploy (target) JMS resources. Methods include the WebLogic Remote Console, the WebLogic Scripting Tool (WLST), and so on.
- Main Steps for Configuring Basic JMS System Resources
  Use the WebLogic Remote Console to configure a persistent store, a JMS server, and a basic JMS system module.
- JMS Configuration Naming Requirements
  Within a domain, each server, machine, cluster, virtual host, and any other resource type must have a unique name and cannot use the same name as the domain. This unique naming rule also applies to all configuration objects, including configurable JMS objects that can be configured as JMS servers, JMS system modules, and JMS application modules.
- JMS Server Configuration
  JMS destinations are configured in JMS modules and are targeted to JMS servers which are configured separately.
- JMS System Module Configuration
  JMS system modules are owned by the administrator, who can delete, modify, or add JMS system resources at any time. The configuration resources described as part of a JMS system module are queue and topic destinations, connection factories, templates, destination keys, and quotas.
- Specifying the Unmapped Resource Reference Mode for Connection Factories
  When you declare a JMS connection factory in the EJB or Servlet using the `@Resource` annotation or the `resource-ref element` in the deployment descriptors, and if this resource reference is not mapped to a JNDI name directly by a lookup attribute, a `mappedName` attribute, or a `jndi-name`, then WebLogic Server allows you to specify the behavior of such unmapped resource references to JMS connection factories.
- Connection Factory Configuration
  Connection factories are resources that enable JMS clients to create JMS connections.
- Queue and Topic Destination Configuration
  A JMS destination identifies a queue (point-to-point) or topic (publish/subscribe) resource within a JMS module. Each queue and topic resource is targeted to a specific JMS server.
- JMS Template Configuration
  A JMS template provides a way to centrally specify settings that are shared across multiple destinations.
- Destination Key Configuration
  As messages arrive on a specific destination, by default they are sorted in FIFO (first-in, first-out) order, which sorts ascending based on each message's unique JMSMessageID.

However, you can use a destination key to configure a different sorting scheme for a destination, such as LIFO (last-in, first-out).

- Quota Configuration
  A quota resource defines a maximum number of messages and bytes, and is responsible for enforcing the defined maximums. It is then associated with one or more destinations.
- Message Limit in a Subscription
  WebLogic JMS provides an option to set a limit on the messages in a topic subscription. If a subscription reaches its configured limit, then by default, the oldest messages in the subscription are ejected to make room for newer messages.
- Foreign Server Configuration
  A foreign server resource lets you to reference third-party JMS providers within a local WebLogic Server JNDI tree.
- Distributed Destination Configuration
  A distributed destination resource is a single set of destinations (queues or topics) that is accessible as a single, logical destination to a client. For example, a distributed topic has its own JNDI name.
- JMS Store-and-Forward (SAF) Configuration
  JMS SAF resources build on the WebLogic Store-and-Forward (SAF) service to provide highly available JMS message production.

Methods for Configuring JMS System Resources

WebLogic Administrators can use multiple methods to configure and deploy (target) JMS resources. Methods include the WebLogic Remote Console, the WebLogic Scripting Tool (WLST), and so on.

You can use the following tools to configure JMS system resources:

- The WebLogic Remote Console lets you to configure, modify, and target JMS-related resources:
  - JMS servers, as described in JMS Server Configuration.
  - JMS system modules, as described in JMS System Module Configuration.
  - Store-and-Forward services for JMS, as described in Configuring SAF for JMS Messages in *Administering the Store-and-Forward Service for Oracle WebLogic Server*.
  - Persistent stores, as described in Using the WebLogic Persistent Store in *Administering Server Environments for Oracle WebLogic Server*.
- WebLogic Scripting Tool (WLST) is a command-line scripting interface that allows system administrators and operators to initiate, manage, and make persistent WebLogic Server configuration changes interactively or by using an executable script. See Using WLST to Manage JMS Servers and JMS System Module Resources.
- Java Management Extensions (JMX) is the solution for monitoring and managing resources on a network. See Overview of WebLogic Server Subsystem MBeans in *Developing Custom Management Utilities Using JMX for Oracle WebLogic Server*.
- The `JMSModuleHelper` extension class contains methods to create and manage JMS module configuration resources in a given module. See Using JMS Module Helper to Manage Applications in *Developing JMS Applications for Oracle WebLogic Server* or the `JMSModuleHelper Class` Javadoc.

> **Note:**
>
> For information about configuring and deploying JMS application modules in an enterprise application, see Configuring JMS Application Modules for Deployment.

- WebLogic Server REST APIs let you to create and deploy new JMS system resource and link it up to the cluster. See Configuring System Resources in *Administering Oracle WebLogic Server with RESTful Management Services*.

Main Steps for Configuring Basic JMS System Resources

Use the WebLogic Remote Console to configure a persistent store, a JMS server, and a basic JMS system module.

For a high level overview, see Messaging in the *Oracle WebLogic Remote Console Online Help*. For instructions about using the WebLogic Remote Console to manage a WebLogic Server domain, see Domain Configuration.

WebLogic JMS provides default values for some configuration options; you must provide values for all others. After WebLogic JMS is configured, applications can send and receive messages using the JMS API. For information on tuning the default configuration parameters, see Tuning WebLogic JMS in *Tuning Performance of Oracle WebLogic Server* or JMSBean in the *MBean Reference for Oracle WebLogic Server*.

1. If you require persistent messaging, then use one of the following storage options:
   - To store persistent messages in a file-based store, you can simply use the server's default persistent store, which requires no configuration on your part. However, you can also create a dedicated file store for JMS. See Creating a Custom (User-Defined) File Store in the *Administering the WebLogic Persistent Store*.
   - To store persistent messages in a JDBC-accessible database, you must create a JDBC store. See Using a JDBC Store in *Administering the WebLogic Persistent Store*.
2. Configure a JMS server to manage the messages that arrive in the queue and topic destinations in a JMS system module. See Overview of JMS Servers.
3. Configure a JMS system module to contain your destinations, as well as other resources, such as quotas, templates, destination keys, distributed destinations, and connection factories. See JMS System Modules.
4. Before creating any queues or topics in your system module, you can optionally create other JMS resources in the module that can be referenced from within a queue or topic, such as JMS templates, quota settings, and destination sort keys:
   - Define quota resources for your destinations. Destinations can be assigned their own quotas; multiple destinations can share a quota; or destinations can share the JMS server's quota. See Quota Configuration.
   - Create JMS templates, which let you define multiple destinations with similar option settings. See JMS Template Configuration.
   - Configure destination keys to create custom sort orders of messages as they arrive in a destination. See Destination Key Configuration.

   After these resources are configured, you can select them when you configure your queue or topic resources.
5. Configure a queue or topic destination or both in your system module:

- Configure a standalone topic for the delivery of messages to multiple recipients (publish/subscribe). See Queue and Topic Destination Configuration.
- Configure a standalone queue for the delivery of messages to exactly one recipient (point-to-point). See Queue and Topic Destination Configuration.

6. If the default connection factories provided by WebLogic Server are not suitable for your application, then create a connection factory to enable your JMS clients to create JMS connections.

   For more information about using the default connection factories, see Using the Default Connection Factories Defined by WebLogic Server. For more information about configuring a Connection Factory, see Connection Factory Configuration Parameters.

WebLogic JMS provides default values for some configuration options; you must provide values for all others. After WebLogic JMS is configured, applications can send and receive messages using the JMS API.

- Advanced Resources in JMS System Modules

Advanced Resources in JMS System Modules

Beyond basic JMS resource configuration, you can add these advanced resources to a JMS system module:

- Create a Uniform Distributed Destination resource to configure a set of queues or topics that distributed across the cluster, with each member belonging to a separate JMS server in the cluster. See Configuring Distributed Destination Resources.
- Create a JMS Store-and-Forward resource to reliably forward messages to remote destinations, even when a destination is unavailable at the time that message is sent, as described in Configuring SAF for JMS Messages in *Administering the Store-and-Forward Service for Oracle WebLogic Server*.
- Create a Foreign Server resource to reference third-party JMS providers within a local WebLogic Server JNDI tree. See Configuring Foreign Server Resources to Access Third-Party JMS Providers.

JMS Configuration Naming Requirements

Within a domain, each server, machine, cluster, virtual host, and any other resource type must have a unique name and cannot use the same name as the domain. This unique naming rule also applies to all configuration objects, including configurable JMS objects that can be configured as JMS servers, JMS system modules, and JMS application modules.

The resource names inside JMS modules must be unique per resource type (for example, queues, topics, and connection factories). However, two different JMS modules can have a resource of the same type that can share the same name.

The JNDI name of any bindable JMS resource (excluding quotas, destination keys, and JMS templates) across JMS modules must use the following naming requirements:

- Global names must be unique across the cluster.
- Local names must be unique across the server.
- If there is a naming conflict, then the JMS resource is not bound into the JNDI tree. If there is any doubt, then make the JNDI name globally unique.

> **Note:**
>
> WebLogic Domain, WebLogic Server, and WebLogic JMS Server names have additional unique naming requirements when two different WebLogic domains interoperate with each other, or when a client communicates with more than one WebLogic domain. Minimally, the two associated domains must have different domain names, and, if global transactions span the domains, then stores should not share the same name even if they are in different domains.

JMS Server Configuration

JMS destinations are configured in JMS modules and are targeted to JMS servers which are configured separately.

A JMS server's primary responsibility for its targeted destinations is to maintain information about what persistent store is used for any persistent messages that arrive on the destinations, and to maintain the states of durable subscribers created on the destinations. As a container for targeted destinations, any configuration or run-time changes to a JMS server can affect all of its destinations.

> **Note:**
>
> A sample `examplesJMSServer` configuration is provided with the product in the Examples Server. For more information about developing basic WebLogic JMS applications, refer to Developing a Basic JMS Application in *Developing JMS Applications for Oracle WebLogic Server*.

JMS server configuration covers the following information:

- JMS Server Configuration Parameters
- JMS Server Targeting
- JMS Server Monitoring Parameters
- Session Pools and Connection Consumers

JMS Server Configuration Parameters

The WebLogic Remote Console lets you configure, modify, target, and delete JMS server resources in a system module. See Create a JMS Server in the *Oracle WebLogic Remote Console Online Help*.

You can configure the following parameters for JMS servers:

- General configuration parameters, including persistent storage, message paging defaults, a template to use when your applications create temporary destinations, and expired message scanning.
- Threshold and quota parameters for destinations in JMS system modules targeted to a particular JMS server.

  For more information about configuring messages and bytes quota for JMS servers and destinations, see Defining Quota in *Tuning Performance of Oracle WebLogic Server*.

- Message logging parameters for a JMS server's log file, which contains the basic events that a JMS message traverses, such as message production, consumption, and removal.

  For more information about configuring message life cycle logging on JMS servers, see Message Life Cycle Logging.

- Destination pause and resume controls that lets you pause message production, message insertion (in-flight messages), and message consumption operations on all the destinations hosted by a single JMS Server.

  For more information about pausing message operations on destinations, see Controlling Message Operations on Destinations.

Some JMS server options can be dynamically configured. When options are modified at runtime, only incoming messages are affected; stored messages are not affected. For more information about the default values for all JMS server options, see JMSServerBean and JMSServerRuntimeMBean in *MBean Reference for Oracle WebLogic Server*.

JMS Server Targeting

You can target a JMS server to either an independent WebLogic Server instance or to a cluster (dynamic or mixed), or to a migratable target server where it will be deployed.

- Weblogic Server instance — The server target where you want to deploy the JMS server. When a target WebLogic Server starts, the JMS server starts as well. If no target WebLogic Server is specified, then the JMS server does not start.
- Cluster-Targeted JMS — You can target JMS service artifacts such as JMS Server, SAF Agents, Persistent Stores, and Path Service to a cluster.

> **Note:**
>
> Oracle recommends using Cluster targeting with new Store configuration that supports Automatic Service Migration, instead of using Migratable Targets. See Simplified JMS Cluster and High Availability Enhancements.

- Migratable Target — Migratable targets define a set of WebLogic Server instances in a cluster that can potentially host an *exactly-once* service, such as a JMS server. When a migratable target server starts, the JMS server boots as well on the specified *user-preferred* server in the cluster. However, a JMS server and all of its destinations can be migrated to another server within the cluster in response to a server failure or due to a scheduled migration for system maintenance. For more information about configuring a migratable target for JMS services, see Migration of JMS-related Services.

For JMS Server targeting best practices, see Targeting Best Practices.

JMS Server Monitoring Parameters

You can monitor runtime statistics for active JMS servers, destinations, and server session pools. See Monitor a JMS Server in the *Oracle WebLogic Remote Console Online Help*.

- Monitor all active JMS servers — A table shows all instances of the JMS server deployed across the WebLogic Server domain.
- Monitor all active JMS destinations — A table displays showing all active JMS destinations for the current domain.
- Monitor all active JMS session pool runtimes — A table displays showing all active JMS session pools for the current domain.

For more information about monitoring JMS objects, see Monitoring JMS Statistics and Managing Messages.

Session Pools and Connection Consumers

> **Note:**
>
> Session pool and connection consumer configuration objects were deprecated in WebLogic Server 9.x. They are not a required part of the Jakarta EE specification, do not support JTA user transactions, and are largely superseded by Message-Driven Beans (MDBs), which are a required part of Jakarta EE. For more information on designing MDBs, see *Developing Message-Driven Beans for Oracle WebLogic Server*.

Server session pools enable an application to process messages concurrently. After you define a JMS server, you can configure one or more session pools for each JMS server. Some session pool options can be dynamically configured, but the new values do not take effect until the JMS server is restarted. See Defining Server Session Pools in *Developing JMS Applications for Oracle WebLogic Server*.

Connection consumers are queues (point-to-point) or topics (publish/subscribe) that retrieve server sessions and process messages. After you define a session pool, configure one or more connection consumers for each session pool. See Defining Server Session Pools in *Developing JMS Applications for Oracle WebLogic Server*.

JMS System Module Configuration

JMS system modules are owned by the administrator, who can delete, modify, or add JMS system resources at any time. The configuration resources described as part of a JMS system module are queue and topic destinations, connection factories, templates, destination keys, and quotas.

With the exception of standalone queue and topic resources that must be targeted to a single JMS server, the connection factory, distributed destination, foreign server, and JMS SAF destination resources in system modules can be made globally available by targeting them to server instances and clusters configured in the WebLogic domain. These resources are therefore available to all applications deployed on the same targets and to client applications. The naming convention for JMS system modules is `MyJMSModule-jms.xml`.

The WebLogic Remote Console lets you configure, modify, target, monitor, and delete JMS system modules in your environment. For information about JMS system module configuration , see Configure Resources for JMS System Modules in the *Oracle WebLogic Remote Console Online Help*.

You define the following "basic" configuration resources as part of a JMS system module:

- Queue and topic destinations, as described in Queue and Topic Destination Configuration.
- Connection factories, as described in Connection Factory Configuration.
- Templates, as described in JMS Template Configuration.
- Destination keys, as described in Destination Key Configuration.
- Quota, as described in Quota Configuration.

You can also define the following "advanced" clustering configuration resources as part of a JMS system module:

- Foreign servers, as described in Configuring Foreign Server Resources to Access Third-Party JMS Providers.
- Distributed destinations, as described in Configuring Distributed Destination Resources.
- JMS Store-and-Forward configurations, as described in Configuring SAF for JMS Messages in *Administering the Store-and-Forward Service for Oracle WebLogic Server*.

A sample `examples-jms` module is provided with the WLS Code Examples. For more information about starting the Examples server, see Starting Instances of WebLogic Server in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

For information about alternative methods for configuring JMS system modules, such as using the WebLogic Scripting Tool (WLST), see Methods for Configuring JMS System Resources.

JMS System Module and Resource Subdeployment Targeting

JMS system modules must be targeted to one or more WebLogic Server instances or to a cluster. JMS resources that can be targeted and are defined in a system module must also be targeted to JMS server or WebLogic Server instances within the scope of a parent module's targets. Additionally, JMS resources that can be targeted and are inside a system module can be further grouped into *subdeployments* during the configuration or targeting process to provide further loose coupling of JMS resources in a WebLogic domain.

Default Targeting

When using the WebLogic Remote Console to configure resources in a JMS system module, you can choose whether to simply accept the parent module's default targets or to proceed to an advanced targeting page where you can use the subdeployment mechanism for targeting the resource. However, standalone queue and topic resource types cannot use default targets and must be targeted to a subdeployment that is targeted to a single JMS server and it is a best practice to use subdeployment targeting instead of default targeting for all destination types regardless.

When you select the default targeting mechanism, in the WebLogic Remote Console, its target status is reflected by the **Default Targeting Enabled** option on the resource type's **General** page.

For more information about configuring JMS system resources, see Configure Resources for JMS System Modules in the *Oracle WebLogic Remote Console Online Help*.

> **Note:**
>
> Default targeting is not recommended for any type of destination. Instead, use subdeployment targeting. See Targeting Best Practices.

Advanced (Subdeployment) Targeting

When targeting standalone queue and topic resources, or when bypassing the default targeting mechanism for other resource types, you must use advanced targeting (also known as subdeployment targeting). A subdeployment is a mechanism by which system module resources that can be targeted (such as standalone destinations, distributed destinations, and connection factories) are grouped and targeted to specific server resources within a system module's targeting scope.

Although a JMS system module can be targeted to a wide array of WebLogic Server instances in a domain, a module's standalone queues or topics can be targeted only to a single JMS server. Connection factories, uniform distributed destinations (UDDs), and foreign servers can be targeted to one or more JMS servers, one or more WebLogic Server instances, or to a cluster.

Therefore, standalone queues or topics cannot be associated with a subdeployment if other members of the subdeployment are targeted to multiple JMS servers, which would be the case, for example, if a connection factory is targeted to a cluster that is hosting JMS servers in a domain. UDDs, however, can be associated with such subdeployments because the purpose of UDDs is to distribute its members to multiple JMS servers in a domain.

Table 3-1 shows the valid targeting options for JMS system resource subdeployments:

Table 3-1 JMS System Resource Subdeployment Targeting

JMS Resource	Valid Targets
Queue	JMS server
Topic	JMS server
Connection factory	JMS server(s) | server instance(s) | cluster
Uniform distributed queue	JMS server(s) | server instance(s) | cluster
Uniform distributed topic	JMS server(s) | server instance(s) | cluster
Foreign server	JMS server(s) | server instance(s) | cluster
SAF imported destinations	SAF Agent(s) | server instance(s) | cluster

Note:

Connection factory, uniform distributed destination, foreign server, and SAF imported destination resources can also be configured to default to their parent module's targets, as explained in Default Targeting.

Default targeting, server instance targeting, and cluster targeting is not recommended for any type of destination (including non-distributed destinations, distributed destinations, or SAF imported destinations). Instead, use a subdeployment target that contains JMS servers, or, for SAF imported destinations, that contains contains SAF agent(s). See Targeting Best Practices.

An example of a simple subdeployment for standalone queues or topics would be to group them with a connection factory so that these resources are collocated on a specific JMS server, which can help reduce network traffic. Also, if the targeted JMS server should be

migrated to another WebLogic Server instance, the connection factory and all its connections also migrate along with the JMS server's destinations.

For example, if a system module named *jmssysmod-jms.xml*, is targeted to a WebLogic Server instance that has two configured JMS servers: *jmsserver1* and *jmsserver2*, and you want to collocate two queues and a connection factory on only *jmsserver1*, then you can group the queues and connection factory in the same subdeployment, named *jmsserver1group*, to ensure that these resources are always linked to *jmsserver1*, provided the connection factory is not already targeted to multiple JMS servers.

```
<weblogic-jms xmlns="http://xmlns.oracle.com/weblogic/weblogic-jms">
  <connection-factory name="connfactory1">
    <sub-deployment-name>jmsserver1group</sub-deployment-name>
    <jndi-name>cf1</jndi-name>
  </connection-factory>
 <queue name="queue1">
    <sub-deployment-name>jmsserver1group</sub-deployment-name>
    <jndi-name>q1</jndi-name>
  </queue>
 <queue name="queue2">
    <sub-deployment-name>jmsserver1group</sub-deployment-name>
    <jndi-name>q2</jndi-name>
  </queue>
</weblogic-jms>
```

And the following is how the *jmsserver1group* subdeployment targeting would look in the domain's configuration file:

```
  <jms-system-resource>
   <name>jmssysmod-jms</name>
   <target>wlsserver1</target>
   <sub-deployment>
     <name>jmsserver1group</name>
     <target>jmsserver1</target>
   </sub-deployment>
   <descriptor-file-name>jms/jmssysmod-jms.xml</descriptor-file-name>
  </jms-system-resource>
```

For information about deploying standalone JMS modules, see Deploying JDBC, JMS, and WLDF Application Modules in *Deploying Applications to Oracle WebLogic Server*.

Specifying the Unmapped Resource Reference Mode for Connection Factories

When you declare a JMS connection factory in the EJB or Servlet using the `@Resource` annotation or the `resource-ref element` in the deployment descriptors, and if this resource reference is not mapped to a JNDI name directly by a lookup attribute, a `mappedName` attribute, or a `jndi-name`, then WebLogic Server allows you to specify the behavior of such unmapped resource references to JMS connection factories.

If a JNDI name is mapped to the resource reference, then the unmapped resource reference mode does not take effect and the resource reference either resolves to the specified object in the JNDI or generates an exception, `javax.naming.NameNotFoundException`.

For more information about resource references, see Enhanced Support for Using WebLogic JMS with EJBs and Servlets in *Developing JMS Applications for Oracle WebLogic Server*

The value that you specify for the Connection Factory Unmapped Resource Reference Mode parameter for a server determines the behavior of resource references to JMS connection factories. Possible values are:

- **ReturnDefault:** If the resource reference does not match the local JNDI name of a configured foreign JMS provider or if it does not match with an object bound to the JNDI tree, then it returns **java:comp/DefaultJMSConnectionFactory**, which is the default JMS connection factory defined by the Jakarta EE 8 specification. See Using the Default JMS Connection Factory Defined by Jakarta EE 8.
- **FailSafe**: The resource reference resolves to an object bound to the JNDI tree with the same name as the resource reference name, if one can be found in JNDI. Otherwise, it throws an exception, `javax.naming.NameNotFoundException`.

> **Note:**
>
> Oracle recommends configuring the reference mode to FailSafe. See Configure JMS Resources.

For more information about the values for the unmapped resource reference mode, see the definition of the `JMSConnectionFactoryUnmappedResRefMode` attribute in MBean Reference for Oracle WebLogic Server.

Connection Factory Configuration

Connection factories are resources that enable JMS clients to create JMS connections.

A connection factory supports concurrent use, enabling multiple threads to access the object simultaneously. WebLogic JMS provides pre-configured default connection factories that can be enabled or disabled on a per-server basis, as described in Using the Default Connection Factories Defined by WebLogic Server.

Otherwise, you can configure one or more connection factories to create connections with predefined options that better suit your application. Within each JMS module, connection factory resource names must be unique. And, all connection factory JNDI names in any JMS module must be unique across an entire WebLogic domain, as defined in JMS Configuration Naming Requirements. WebLogic Server adds them to the JNDI space during startup, and the application then retrieves a connection factory using the WebLogic JNDI APIs.

You can establish cluster-wide, transparent access to JMS destinations from any server in the cluster, either by using the default connection factories for each server instance, or by configuring one or more connection factories and targeting them to one or more server instances in the cluster. This way, each connection factory can be deployed on multiple WebLogic Server instances. For more information on configuring JMS clustering, see Configuring WebLogic JMS Clustering.

- Using the Default JMS Connection Factory Defined by Jakarta EE 8
- Using Default Connection Factories Defined by WebLogic Server
- Connection Factory Configuration Parameters
- Connection Factory Targeting

Using the Default JMS Connection Factory Defined by Jakarta EE 8

WebLogic Server supports the default JMS connection factory defined by the Jakarta EE 8 platform specification. This default connection factory can be looked up using the `java:comp/DefaultJMSConnectionFactory` JNDI name or it can be accessed using the `@Resource` annotation. The `java:comp/DefaultJMSConnectionFactory` JNDI name resolves to the equivalent of the default `weblogic.jms.XAConnectionFactory`.

> **Note:**
>
> The lookup of `DefaultJMSConnectionFactory` using `@Resource` and `Context.lookup()` always returns pooled and wrapped connection factory, to ensure that the resource provides adequate performance and also to ensure that the Jakarta EE restriction on a server-side JMS API usage is properly enforced.

The Jakarta EE 8 default connection factory can be accessed only from within a Jakarta EE 8 application unlike the default connection factories defined by WebLogic Server. The Jakarta EE 8 connection factory cannot be accessed from standalone clients except by using the application client container facility.

> **Note:**
>
> - Oracle recommends using custom connection factories instead of using default connection factories because the default connection factories cannot be tuned. Custom connection factories that can be tuned often prove useful for tuning applications even after the application is in production
> - A WebLogic Server administrator cannot disable the Default JMS Connection Factory per WebLogic Server.
>
> See Look Up a Connection Factory in JNDI in *Developing JMS Applications for Oracle WebLogic Server*.

Using Default Connection Factories Defined by WebLogic Server

WebLogic Server defines two default connection factories, which can be looked up using the following JNDI names:

- `weblogic.jms.ConnectionFactory`
- `weblogic.jms.XAConnectionFactory`

You only need to configure a new connection factory if the pre-configured settings of the default factories are not suitable for your application. For more information on using the default connection factories, see Understanding WebLogic JMS in *Developing JMS Applications for Oracle WebLogic Server*.

The main difference between the pre-configured settings for the default connection factories and a user-defined connection factory is the default value for the "XA Connection Factory Enabled" option to enable JTA transactions. For more information about the XA Connection

Factory Enabled option, and to see the default values for the other connection factory options, see JMSConnectionFactoryBean in the *MBean Reference for Oracle WebLogic Server*.

Also, using default connection factories means that you have no control over targeting the WebLogic Server instances where the connection factory may be deployed. However, you can enable and or disable the default connection factories on a per-WebLogic Server basis.

> **Note:**
>
> Oracle recommends using custom connection factories instead of default connection factories because default connection factories are not tunable. Custom connection factory tunables often prove useful for tuning applications even after the application is in production.

Connection Factory Configuration Parameters

The WebLogic Remote Console lets you to configure, modify, target, and delete connection factory resources in a system module.

1. In the **Edit Tree**, go to **Services**, then **JMS System Resources**, then *myJMSSystemResource*.
2. From the **Navigation Tree**, as children of *myJMSSystemResource*, select the **Connection Factories** you want to configure.

You can modify the following parameters for connection factories:

- General configuration parameters, including modifying the default client parameters, default message delivery parameters, load balancing parameters, unit-of-order parameters, and security parameters.
- Transaction parameters, which enable you to define a value for the transaction time-out option and to indicate whether an XA queue or XA topic connection factory is returned, and whether the connection factory creates sessions that are JTA aware.

  > **Note:**
  >
  > When selecting the **XA Connection Factory Enabled** option to enable JTA transactions with JDBC stores, you must verify that the configured JDBC data source uses a non-XA JDBC driver. This limitation does not remove the XA capabilities of layered subsystems that use JDBC stores. For example, WebLogic JMS is fully XA-capable regardless of whether it uses a file store or any JDBC store.

- Flow control parameters, which enable you to tell a JMS server or destination to slow down message producers when it determines that it is becoming overloaded.

Some connection factory options can be dynamically configured. When options are modified at runtime, only incoming messages are affected; stored messages are not affected. For more information about the default values for all connection factory options, see JMSConnectionFactoryBean in *MBean Reference for Oracle WebLogic Server*.

Connection Factory Targeting

You can target connection factories to one or more JMS server, to one or more WebLogic Server instances, or to a cluster.

- JMS server(s) — You can target connection factories to one or more JMS servers along with destinations. You can also group a connection factory with standalone queues or topics in a subdeployment targeted to a specific JMS server, which guarantees that all these resources are collocated to avoid extra network traffic. Another advantage of such a configuration would be if the targeted JMS server needs to be migrated to another WebLogic Server instance, then the connection factory and all its connections also migrate along with the JMS server destinations. However, when standalone queues or topics are members of a subdeployment, a connection factory can be targeted only to the same JMS server.
- WebLogic Server instance(s) — To establish transparent access to JMS destinations from any server in a domain, you can target a connection factory to multiple WebLogic Server instances simultaneously.
- Cluster — To establish cluster wide, transparent access to JMS destinations from any server in a cluster, you can target a connection factory to all server instances in the cluster, or even to specific servers within the cluster.

For more information about JMS system module subdeployment targeting, see JMS System Module and Resource Subdeployment Targeting. For information on connection factory targeting best practices, see Targeting Best Practices.

Queue and Topic Destination Configuration

A JMS destination identifies a queue (point-to-point) or topic (publish/subscribe) resource within a JMS module. Each queue and topic resource is targeted to a specific JMS server.

A JMS server's primary responsibility for its targeted destinations is to maintain information on what persistent store is used for any persistent messages that arrive on the destinations, and to maintain the states of durable subscribers created on the destinations.

You can optionally create other JMS resources in a module that can be referenced from within a queue or topic, such as JMS templates, quota settings, and destination sort keys:

- Quota : Assign quotas to destinations; multiple destinations can share a quota; or destinations can share the JMS server's quota. See Defining Quota in *Tuning Performance of Oracle WebLogic Server*.
- JMS Template : Define multiple destinations with similar option settings. You also need a JMS template to create temporary queues. See JMS Template Configuration.
- Destination Key : Create custom sort orders of messages as they arrive on a destination. See Destination Key Configuration.

See the following sections:

- Queue and Topic Configuration Parameters
- Queue and Topic Targeting
- Destination Monitoring and Management Parameters

Queue and Topic Configuration Parameters

A JMS queue defines a *point-to-point* destination type for a JMS server. A message delivered to a queue is distributed to a single consumer. A JMS topic identifies a *publish/subscribe* destination type for a JMS server. Topics are used for asynchronous peer communications. A message delivered to a topic is distributed to all consumers that are subscribed to that topic.

The WebLogic Remote Console lets you configure, modify, target, and delete queue and topic resources in a system module. Within each JMS module, queue and topic resource names must be unique. And, all queue and topic JNDI names in any JMS module must be unique across an entire WebLogic domain, as defined in JMS Configuration Naming Requirements.

1. In the **Edit Tree**, go to **Services**, then **JMS System Resources**, then *myJMSSystemResource*.
2. From the **Navigation Tree**, as children of *myJMSSystemResource*, select the **Queues** and **Topics** you want to configure.

You can configure the following parameters for a queue or a topic:

- General configuration parameters, including a JNDI name, a destination key for sorting messages as they arrive at the destination, or selecting a JMS template if you are using one to configure properties for multiple destinations.

  > **Note:**
  >
  > Although queue and topic JNDI names can be dynamically changed, there may be long-lived producers or consumers, such as MDBs, that continue trying to produce or consume messages to and from the original queue or topic JNDI name.

- Threshold and quota parameters, which define the upper and lower message and byte threshold and maximum quota options for the destination. See Quota Configuration.
- Message logging parameters, such as message type and user properties, and logging message life cycle information into a JMS log file.

  See Message Life Cycle Logging. Pause and resume controls for message production, message insertion (in-flight messages), and message consumption operations on a destination. See Controlling Message Operations on Destinations.
- Message Delivery override parameters, such as message priority and time-to-deliver values, which can override those values specified by a message producer or connection factory.
- Message Delivery failure parameters, such as defining a message redelivery limit, selecting a message Expiration Policy, and specifying an error destination for expired messages.
- For topics only, multicast parameters, including a multicast address, time-to-live (TTL), and port.

Some options can be dynamically configured. When options are modified at run time, only incoming messages are affected; stored messages are not affected. For more information about the default values for all options, see QueueBean and TopicBean in *MBean Reference for Oracle WebLogic Server*.

- Creating Error Destinations

- Creating Distributed Destinations

Creating Error Destinations

To help manage recovered or rolled back messages, you can also configure a target error destination for messages that have reached their redelivery limit. The error destination can be either a topic or a queue, but it must be a destination that is targeted to the same JMS server as the destinations it is associated with. See Configuring an Error Destination for Undelivered Messages in *Developing JMS Applications for Oracle WebLogic Server*.

Creating Distributed Destinations

A distributed destination resource is a group of destinations (queues or topics) that are accessible as a single, logical unit to a client (for example, a distributed topic has its own JNDI name). The members of the set are typically distributed across multiple servers within a cluster, with each member belonging to a separate JMS server. See Distributed Destination Configuration.

Queue and Topic Targeting

Stand alone queues and topics can be deployed only to a specific JMS server in a domain because they depend on the JMS servers they are targeted to for the management of persistent messages, durable subscribers, and message paging.

If you want to associate a group of queues or topics with a connection factory on a specific JMS server, then you can target the destinations and connection factory to the same subdeployment, which links these resources to the JMS server targeted by the subdeployment. However, when standalone destinations are members of a subdeployment, a connection factory can be targeted only to the same JMS server.

For more information on JMS system module subdeployment targeting, see JMS System Module and Resource Subdeployment Targeting. For Queue and Topic targeting best practices, see Targeting Best Practices.

Destination Monitoring and Management Parameters

You can monitor runtime statistics for queues and topics in system modules, as well as manage the messages on queues and durable subscribers on topics.

- For information about managing messages on queues, as described in Managing JMS Messages.
- For information about managing durable subscriber on topics, as described in Managing JMS Messages.

JMS Template Configuration

A JMS template provides a way to centrally specify settings that are shared across multiple destinations.

- You do not need to reenter every option setting each time you define a new destination; you can use the JMS template and override any setting to which you want to assign a new value.
- You can modify shared option settings dynamically simply by modifying the template.

- You can specify subdeployments for error destinations so that any number of destination subdeployments (groups of queue or topics) use only the error destinations specified in the corresponding template subdeployments.
- JMS Template Configuration Parameters

JMS Template Configuration Parameters

The WebLogic Remote Console lets you configure, modify, target, and delete JMS template resources in a system module.

1. In the **Edit Tree**, go to **Services**, then **JMS System Resources**, then *myJMSSystemResource*.
2. From the **Navigation Tree**, as children of *myJMSSystemResource*, select the **Templates** that you want to configure.

The options that can be configured for a JMS template are the same as those configured for a destination. See Queue and Topic Configuration Parameters. These configuration options are inherited by the destinations that use them, with the following exceptions:

- If the destination that is using a JMS template specifies an override value for an option, then the override value is used.
- If the destination that is using a JMS template specifies a message redelivery value for an option, then that redelivery value is used.
- The Name option is not inherited by the destination. This name is valid for the JMS template only. You must explicitly define a unique name for all destinations. See JMS Configuration Naming Requirements.
- The JNDI Name, Enable Store, and Template options are not defined for JMS templates.
- You can configure subdeployments for error destinations, so that any number of destination subdeployments (groups of queue or topics) use only the error destinations specified in the corresponding template subdeployments.

Any options that are not explicitly defined for a destination are assigned default values. If no default value exists, then specify a value within the JMS template or as a destination option override.

Some template options can be dynamically configured. When options are modified at runtime, only incoming messages are affected; stored messages are not affected. For more information about the default values for all topic options, see `TemplateBean` in *MBean Reference for Oracle WebLogic Server*.

Destination Key Configuration

As messages arrive on a specific destination, by default they are sorted in FIFO (first-in, first-out) order, which sorts ascending based on each message's unique JMSMessageID. However, you can use a destination key to configure a different sorting scheme for a destination, such as LIFO (last-in, first-out).

The WebLogic Remote Console lets you configure, modify, target, and delete destination key resources in a system module.

1. In the **Edit Tree**, go to **Services**, then **JMS System Resources**, then *myJMSSystemResource*.
2. From the **Navigation Tree**, as children of *myJMSSystemResource*, select the **Destination Keys** that you want to configure.

For more information about the default values for all destination key options, see DestinationKeyBean in the *MBean Reference for Oracle WebLogic Server*.

Quota Configuration

A quota resource defines a maximum number of messages and bytes, and is responsible for enforcing the defined maximums. It is then associated with one or more destinations.

See Defining Quota in *Tuning Performance of Oracle WebLogic Server*.

Message Limit in a Subscription

WebLogic JMS provides an option to set a limit on the messages in a topic subscription. If a subscription reaches its configured limit, then by default, the oldest messages in the subscription are ejected to make room for newer messages.

See Subscription Message Limits in *Tuning Performance of Oracle WebLogic Server*.

Foreign Server Configuration

A foreign server resource lets you to reference third-party JMS providers within a local WebLogic Server JNDI tree.

With a foreign server resource, you can quickly map a foreign JMS provider so that its associated connection factories and destinations appear in the WebLogic JNDI tree as local JMS objects. A foreign server resource can also be used to reference remote instances of WebLogic Server in another cluster or domain in the local WebLogic JNDI tree.

See Configuring Foreign Server Resources to Access Third-Party JMS Providers.

Distributed Destination Configuration

A distributed destination resource is a single set of destinations (queues or topics) that is accessible as a single, logical destination to a client. For example, a distributed topic has its own JNDI name.

The members of the set are typically distributed across multiple servers within a cluster, with each member belonging to a separate JMS server. Applications that use a distributed destination are more highly available than applications that use standalone destinations because WebLogic JMS provides load balancing and failover for the members of a distributed destination in a cluster.

See Configuring Distributed Destination Resources.

JMS Store-and-Forward (SAF) Configuration

JMS SAF resources build on the WebLogic Store-and-Forward (SAF) service to provide highly available JMS message production.

For example, a JMS message producer connected to a local server instance can reliably forward messages to a remote JMS destination, even though that remote destination may be temporarily unavailable when the message was sent. JMS Store-and-forward is transparent to JMS applications; therefore, JMS client code still uses the existing JMS APIs to access remote destinations.

See Configuring SAF for JMS Messages in *Administering the Store-and-Forward Service for Oracle WebLogic Server*.

4

Configuring Advanced JMS System Resources

You can learn how to configure advanced WebLogic JMS resources for Oracle WebLogic Server, such as a distributed destination in a clustered environment.
This chapter includes the following sections:

- Configuring WebLogic JMS Clustering
  A WebLogic Server *cluster* is a group of servers in a domain that work together to provide a more scalable, more reliable application platform than a single server. A cluster appears to its clients as a single server but it is a group of servers acting as one.
- Migration of JMS-Related Services
  JMS-related services are singleton services; therefore, are not active on all server instances in a cluster. Instead, the services are pinned to a single server in the cluster to preserve data consistency.
- Using the WebLogic Path Service
  The WebLogic Server path service is a persistent map used for storing the mapping between a group of messages in a JMS Message Unit-of-Order and a messaging resource in a cluster.
- Configuring Foreign Server Resources to Access Third-Party JMS Providers
  WebLogic JMS allows you to reference third-party JMS providers within a local WebLogic Server JNDI tree. With Foreign Server resources in JMS modules, you can quickly map a foreign JMS provider so that its associated connection factories and destinations appear in the WebLogic JNDI tree as local JMS objects.
- Configuring Distributed Destination Resources
  A distributed destination resource in a JMS module represents a single set of destinations (queues or topics) that are accessible as a single, logical destination to a client. For example, a distributed topic has its own JNDI name. The members of the set are typically distributed across multiple servers within a cluster, with each member belonging to a separate JMS server.
- Configure an Unrestricted ClientID
  The Client ID Policy specifies whether more than one JMS connection can use the same client ID in a cluster.
- Configure Shared Subscriptions
  The Subscription Sharing Policy specifies whether subscribers can share subscriptions with other subscribers on the same connection.

Configuring WebLogic JMS Clustering

A WebLogic Server *cluster* is a group of servers in a domain that work together to provide a more scalable, more reliable application platform than a single server. A cluster appears to its clients as a single server but it is a group of servers acting as one.

- Advantages of JMS Clustering
- How JMS Clustering Works

- Configuration Guidelines for JMS Clustering
- What About Failover?

Advantages of JMS Clustering

The advantages of clustering for JMS include the following:

- *Load balancing of destinations across multiple servers in a cluster*

  An administrator can establish load balancing of destinations across multiple servers in the cluster by:

  – Configuring a JMS server and targeting a WebLogic cluster. See Simplified JMS Cluster and High Availability Configuration.

  – Configuring multiple JMS servers and targeting them to the configured WebLogic Servers.

  – Configuring multiple JMS servers and targeting them to a set of migratable targets.

  Each JMS server is deployed on exactly one WebLogic Server instance and handles requests for a set of destinations.

- *High availability of destinations*

  – *Distributed destinations* : The queue and topic members of a distributed destination are usually distributed across multiple servers within a cluster, with each member belonging to a separate JMS server. Applications that use distributed destinations are more highly available than applications that use simple destinations because WebLogic JMS provides load balancing and failover for member destinations of a distributed destination within a cluster. For more information on distributed destinations, see Configuring Distributed Destination Resources.

  – *Store-and-Forward* : JMS modules use the SAF service to enable local JMS message producers to reliably send messages to remote queues or topics. If the destination is not available at the moment the messages are sent, either because of network problems or system failures, then the messages are saved on a local server instance, and are forwarded to the remote destination as soon as it becomes available. See Understanding the Store-and-Forward Service in *Administering the Store-and-Forward Service for Oracle WebLogic Server*.

  – For automatic failover, WebLogic Server supports migration at the server level—a complete server instance, and all of the services it hosts can be migrated to another machine, either automatically or manually. See Whole Server Migration in *Administering Clusters for Oracle WebLogic Server*.

    WebLogic Server also supports automatic migration at the service level for JMS, where a failed service instance can be restarted in place on its current WebLogic Server JVM or migrated to another running JVM in the same cluster. This is termed 'service migration' and there are two approaches for configuring it based on how JMS is configured and targeted. For more information, see Migratable Target and Simplified JMS Configuration and High Availability Enhancements. The latter is recommended for new configurations.

- *Cluster wide, transparent access to destinations from any server in a cluster*

  An administrator can establish cluster wide, transparent access to destinations from any server in the cluster by either using the default connection factories for each server instance in the cluster, or by configuring one or more connection factories and targeting them to one or more server instances in the cluster, or to the entire cluster. This way, each

connection factory can be deployed on multiple WebLogic Server instances. Connection factories are described in more detail in Connection Factory Configuration.

- *Scalability*
  - Load balancing of destinations across multiple servers in the cluster.
  - Distribution of the application load across multiple JMS servers through connection factories, thus reducing the load on any single JMS server and enabling session concentration by routing connections to specific servers.
- *Server affinity for JMS Clients*

  When configured for the cluster, load-balancing algorithms (round-robin-affinity, weight-based-affinity, or random-affinity), provide server affinity for JMS client connections. If a JMS application has a connection to a given server instance, JMS attempts to establish new JMS connections to the same server instance. For more information on server affinity, see Load Balancing in a Cluster in *Administering Clusters for Oracle WebLogic Server*.

For more information about the features and benefits of using WebLogic clusters, see Understanding WebLogic Server Clustering in *Administering Clusters for Oracle WebLogic Server*.

How JMS Clustering Works

An administrator can establish cluster wide, transparent access to JMS destinations from any server in a cluster, either by using the default connection factories for each server instance in a cluster, or by configuring one or more connection factories and targeting them to one or more server instances in a cluster, or to an entire cluster. This way, each connection factory can be deployed on multiple WebLogic Servers. For information about configuring and deploying connection factories, see Connection Factory Configuration Parameters.

A messaging application uses a Java Naming and Directory Interface (JNDI) context to look up a connection factory and then uses the connection factory to create a connection from the client into the cluster. If the client application is located outside of the connection factory's cluster, the connection will implicitly connect to one of servers in the cluster that are among the targets of the connection factory (this server may be different than the server the JNDI context itself is using). If the application is running on a WebLogic Server, and the same server is among the targets of the connection factory, then the client connection will simply connect to the local WebLogic Server. Each JMS server handles requests for a set of destinations. If requests for destinations are sent to a WebLogic Server connection host which is not hosting a JMS server or destinations, or are load balanced to a different WebLogic Server, the requests are forwarded by the connection host to the appropriate WebLogic Server instance in the same cluster that is hosting the desired JMS server and its destinations.

The administrator can also configure multiple JMS servers on the various servers in the cluster: as long as the JMS servers are uniquely named—and can then target JMS queue or topic resources to the various JMS servers. Alternatively, an administrator can target a JMS server in a cluster, and the cluster automatically creates an instance of a JMS server on each server. The application uses the Java Naming and Directory Interface (JNDI) to look up a connection factory and create a connection to establish communication with a JMS server. Each JMS server handles requests for a set of destinations. Requests for destinations not handled by a JMS server are forwarded to the appropriate WebLogic Server instance. For information about configuring and deploying JMS servers, see JMS Server Configuration.

- JMS Clustering Naming Requirements
- Distributed Destination Within a Cluster
- JMS Services As a Migratable Service Within a Cluster

JMS Clustering Naming Requirements

There are naming requirements when configuring JMS objects and resources, such as JMS servers, JMS modules, and JMS resources, to work in a clustered environment in a single WebLogic domain or in a multi domain environment. See JMS Configuration Naming Requirements.

Distributed Destination Within a Cluster

A distributed destination resource is a single set of destinations (queues or topics) that is accessible as a single, logical destination to a client (for example, a distributed topic has its own JNDI name). The members of the unit are usually distributed across multiple servers within a cluster, with each member belonging to a separate JMS server. Applications that use distributed destinations are more highly available than applications that use simple destinations because WebLogic Server provides load balancing and failover for member destinations of a distributed destination within a cluster. See Configuring Distributed Destination Resources.

JMS Services As a Migratable Service Within a Cluster

In addition to being part of a whole server migration, where all services hosted by a server can be migrated to another machine, JMS services are also part of the singleton service migration framework. This allows an administrator, for example, to migrate a JMS server and all of its destinations to another WebLogic Server within a cluster in response to a server failure or for scheduled maintenance. This includes both scheduled migrations as well as automatic migrations. For more information about JMS service migration, see Migration of JMS-related Services.

Configuration Guidelines for JMS Clustering

In order to use WebLogic JMS in a clustered environment, follow these guidelines:

1. Configure your clustered environment as described in Setting Up WebLogic Clusters in *Administering Clusters for Oracle WebLogic Server*.
2. Identify targets for any user-defined JMS connection factories using the WebLogic Remote Console. For connection factories, you can identify either a single-server, a cluster, or a migratable target.

   For more information about these connection factory configuration attributes, see Connection Factory Configuration.
3. Optionally, identify migratable server targets or clusters for JMS services (JMSServers and Persistent Stores) using the WebLogic Remote Console. For example, for JMS servers, you can identify:
   - A Configured server.
   - A Cluster. See Simplified JMS Cluster and High Availability Configuration.
   - A migratable target, which is a set of server instances in a cluster that can host an "exactly-once" service like JMS in case of a server failure in the cluster. For more information on migratable JMS server targets, see Migration of JMS-related Services.

   For more information about JMS server configuration attributes, see JMS Server Configuration.
4. Optionally, you can configure the physical JMS destinations in a cluster as part of a virtual distributed destination set, as discussed in Distributed Destination Within a Cluster. Note

that it is a best practice to always target destinations using a subdeployment target that in turn references one or more specific SAF Agents (for imported destinations) or JMS Servers (for all other types of destinations).

See Best Practices for JMS Beginners and Advanced Users for more information.

What About Failover?

> **Note:**
>
> The WebLogic JMS Automatic Reconnect feature is deprecated. The JMS Connection Factory configuration, `javax.jms.extension.WLConnection` API, and `javax.jms.extension.JMSContext` API for this feature will be removed or ignored in a future release. Oracle recommends that client applications handle connection exceptions as described in Client Resiliency Best Practices in *Administering JMS Resources for Oracle WebLogic Server*.

The resiliency of a JMS system is fundamentally addressed at two levels. First at the JVM and service level via migration as described later in this section and in the following section, and second at the API level by ensuring clients reconnect and retry after a failure. For more information about client reconnection and retry after a failure, see Client Resiliency Best Practices in *Administering JMS Resources for Oracle WebLogic Server*.

In addition, implementing the automatic service migration feature ensures that exactly-once services, like JMS, do not introduce a single point of failure for dependent applications in the cluster. For dyamic-cluster targeted JMS server failover, failback and restart-in-place features are available. See Migration of JMS-related Services. WebLogic Server also supports data migration at the server level—a complete server instance, and all of the services it hosts can be migrated to another machine, either automatically, or manually. See Whole Server Migration in *Administering Clusters for Oracle WebLogic Server*.

In a clustered environment, WebLogic Server also offers service continuity in the event of a single server failure by allowing you to configure distributed destinations, where the members of the unit are usually distributed across multiple servers within a cluster, with each member belonging to a separate JMS server. See Distributed Destination Within a Cluster.

Oracle also recommends implementing high-availability clustering software, which provides an integrated, out-of-the-box solution for WebLogic Server-based applications.

Migration of JMS-Related Services

JMS-related services are singleton services; therefore, are not active on all server instances in a cluster. Instead, the services are pinned to a single server in the cluster to preserve data consistency.

To ensure that singleton JMS services do not introduce a single point of failure for dependent applications in the cluster, you can configure JMS-related services for high availability by using cluster-targeted JMS or migratable-target JMS. See Migratable Target and Simplified JMS Configuration and High Availability Enhancements. The latter is recommended for new configurations. JMS services can also be manually migrated before performing scheduled server maintenance.

Migratable JMS-related services include:

- JMS Server : a management container for the queues and topics in JMS modules that are targeted to them. See JMS Server Configuration.
- Store-and-Forward (SAF) Service : store-and-forward messages between local sending and remote receiving endpoints, even when the remote endpoint is not available at the moment the messages are sent. Only the sending SAF agents configured for JMS SAF (sending capability only) are migratable. See Understanding the Store-and-Forward Service in *Administering the Store-and-Forward Service for Oracle WebLogic Server*.
- Path Service: a persistent map that can be used to store the mapping of a group of messages in a JMS Message Unit-of-Order to a messaging resource in a cluster. One path service is configured per cluster. See Using the WebLogic Path Service.
- Custom Persistent Store : a user-defined, disk-based file store or JDBC-accessible database for storing subsystem data, such as persistent JMS messages or store-and-forward messages. See Using the WebLogic Persistent Store in *Administering Server Environments for Oracle WebLogic Server*.

See Understanding the Service Migration Framework in *Administering Clusters for Oracle WebLogic Server*.

- Automatic Migration of JMS Services
- Manual Migration of JMS Services
- Persistent Store High Availability

Automatic Migration of JMS Services

An administrator can configure migratable targets so that hosted JMS services are automatically migrated from the current unhealthy hosting server to a healthy active server with the help of the Health Monitoring subsystem. For more information about configuring automatic migration of JMS-related services, see Roadmap for Configuring Automatic Migration of JMS-Related Services in *Administering Clusters for Oracle WebLogic Server*.

Manual Migration of JMS Services

An administrator can manually migrate JMS-related services to a healthy server if the host server fails or before performing server maintenance. For more information about configuring manual migration of JMS-related services, see Roadmap for Configuring Manual Migration of JMS-Related Services in *Administering Clusters for Oracle WebLogic Server*.

> **Note:**
>
> Manual migration requires migratable targets, and therefore is not an option when taking advantage of Simplified JMS Cluster and High Availability Configuration in a cluster targeted JMS configuration. This type of configuration has much less of a need for manual migration as Simplified JMS Cluster and High Availability Configuration supports automatic fail-back.

Persistent Store High Availability

As discussed in What About Failover?, a JMS service, including a custom persistent store, can be migrated as part of the "whole server" migration feature, or as part of a "service-level" migration for migratable JMS-related services.

File stores must use the same files throughout the lifetime regardless of where they run. This means that it is the responsibility of the administrator to make sure that a migrated file store can access the same files that it updated before it was migrated.

Migratable custom file stores can be configured on a shared disk that is available to the migratable target servers in the cluster or, if using a migratable target HA configuration, can be migrated to a backup server target by using pre/post-migration scripts. For more information about migrating persistent stores, see Custom Store Availability for JMS Services in *Administering Clusters for Oracle WebLogic Server*. See File Locations in *Administering the WebLogic Persistent Store*.

Similarly, default file stores must be located in a shared directory location when setting up whole server migration or JTA migration.

Finally, migrated JDBC Stores must still access the same database and schema as their original location.

Using the WebLogic Path Service

The WebLogic Server path service is a persistent map used for storing the mapping between a group of messages in a JMS Message Unit-of-Order and a messaging resource in a cluster.

The path service provides a way to enforce ordering by pinning messages to a member of a cluster that is hosting servlets, distributed queue members, or Store-and-Forward agents. One path service is configured per cluster. For more information about the Message Unit-of-Order feature, see Using Message Unit-of-Order in *Developing JMS Applications for Oracle WebLogic Server*.

In the Remote Console, to configure a path service in a cluster, in the **Edit Tree**, go to **Services**: **Path Services**: *myPathService*.

- Path Service High Availability
- Implementing Message UOO with a Path Service

Path Service High Availability

There are different ways to achieve high availability for the path service:

- You can use whole server migration to restart the WebLogic Server that runs the path service. See Oracle Fusion Middleware Administering Clusters . See Whole Server Migration in *Administering Clusters for Oracle WebLogic Server*.
- The path service can use a cluster targeted store with singleton distribution policy. See Simplified JMS Cluster and High Availability Configuration.
- Path Service and its store can be configured to use a migratable target. However, a migratable path service cannot use the default store, so a custom store must be configured and targeted to the same migratable target. As an additional best practice, the path service and its custom store should be the only users of that migratable target. See Understanding the Service Migration Framework in *Administering Clusters for Oracle WebLogic Server*.

Implementing Message UOO with a Path Service

Consider the following when implementing Message Unit-of-Order in conjunction with path service-based routing:

- Each path service mapping is stored in a persistent store. When configuring a path service, select a persistent store that takes advantage of a high availability solution. See Persistent Store High Availability.
- If one or more producers send messages using the same Unit-of-Order name then all messages they produce will share the same path entry and have the same member queue destination.
- If the required route for a Unit-of-Order name is unreachable, then the producer sending the message will throw a `JMSOrderException`. The exception is thrown because the JMS messaging system can not meet the quality-of-service required : only one distributed destination member consumes messages for a particular Unit-of-Order name.
- A path entry is automatically deleted when the last producer and last message reference are deleted.
- Depending on your system, using the path service may slow system throughput due to a remote disk operations to create, read, and delete path entries.
- A distributed queue and its individual members each represent a unique destination. For example:

  DXQ1 is a distributed queue with queue members Q1 and Q2. DXQ1 also has a Unit-of-Order name value of *Fred* mapped by the Path Service to the Q2 member.

  - If message M1 is sent to DXQ1, then it uses the Path Service to define a route to Q2.
  - If message M1 is sent directly to Q2, then, no routing by the Path Service is performed. This is because the application selected Q2 directly and the system was not asked to pick a member from a distributed destination.
  - If you want the system to use the path service, send messages to the distributed destination. If not, send directly to the member.
  - You can have more than one destination that has the same Unit-of-Order names in a distributed queue. For example:

    Queue Q3 also has a Unit-of-Order name value of *Fred*. If Q3 is added to DXQ1, then there are now two destinations that have the same Unit-of-Order name in a distributed queue. Even though, Q3 and DXQ1 share the same Unit-of-Order name value *Fred*, each has a unique route and destination that allows the server to continue to provide the correct message ordering for each destination.
- Empty the queues before removing them from a distributed queue or adding them to a distributed queue. Although the path service removes the path entry for the removed member, there is a short transition period where a message produced may throw an exception `JMSOrderException` when the queue has been removed but the path entry still exists.

Configuring Foreign Server Resources to Access Third-Party JMS Providers

WebLogic JMS allows you to reference third-party JMS providers within a local WebLogic Server JNDI tree. With Foreign Server resources in JMS modules, you can quickly map a foreign JMS provider so that its associated connection factories and destinations appear in the WebLogic JNDI tree as local JMS objects.

Foreign Server resources can also be used to reference remote instances of WebLogic Server in another cluster or domain in the local WebLogic JNDI tree. For more information about

integrating remote and foreign JMS providers, see Enhanced 2EE Support for Using WebLogic JMS With EJBs and Servlets in *Developing JMS Applications for Oracle WebLogic Server*.

These sections provide more information about how a foreign server works and a sample configuration for accessing a remote MQSeries JNDI provider.

- How WebLogic JMS Accesses Foreign JMS Providers
- Creating Foreign Server Resources
- Sample Configuration for MQSeries JNDI

How WebLogic JMS Accesses Foreign JMS Providers

When a foreign JMS server is deployed, it creates local connection factory and destination objects in the WebLogic Server JNDI. Then when a foreign connection factory or destination object is looked up on the local server, that object performs the actual lookup on the remote JNDI directory, and the foreign object is returned from that directory.

This method makes it easier to configure multiple WebLogic Messaging Bridge destinations, because the foreign server moves the JNDI initial context factory and connection URL configuration details outside of your Messaging Bridge destination configurations. You need to only provide the foreign Connection Factory and Destination JNDI name for each object.

For more information on configuring a messaging bridge, see Configuring and Managing a Messaging Bridge in *Administering the WebLogic Messaging Bridge for Oracle WebLogic Server*.

The ease-of-configuration concept also applies to configuring WebLogic Servlets, EJBs, and Message-Driven Beans (MDBs) with WebLogic JMS. For example, the `weblogic-ejb-jar.xml` file in the MDB can have a local JNDI name, and you can use the foreign JMS server to control where the MDB receives messages from. For example, you can deploy the MDB in one environment to talk to one JMS destination and server, and you can deploy the same `weblogic-ejb-jar.xml` file to a different server and have it talk to a different JMS destination without having to unpack and edit the `weblogic-ejb-jar.xml` file.

Creating Foreign Server Resources

A *Foreign Server* resource in a JMS module represents a JNDI provider that is outside the WebLogic JMS server. It contains information that allows a local WebLogic Server instance to reach a remote JNDI provider, thereby allowing a number of foreign connection factory and destination objects to be defined on one JNDI directory.

The WebLogic Remote Console lets you to configure, modify, target, and delete foreign server resources in a system module.

1. In the **Edit Tree**, go to **Services**, then **JMS System Resources**, then *myJMSSystemResource*.
2. From the **Navigation Tree**, as children of *myJMSSystemResource*, select the **Foreign Servers** you want to configure.

> **Note:**
>
> For information about configuring and deploying JMS application modules in an enterprise application, see Configuring JMS Application Modules for Deployment.

Some foreign server options are dynamically configured. When options are modified at run time, only incoming messages are affected; stored messages are not affected. For more information about the default values for all foreign server options, see ForeignServerBean in the *MBean Reference for Oracle WebLogic Server*.

After defining a foreign server, you can configure connection factory and destination objects. You can configure one or more connection factories and destinations (queues or topics) for each foreign server.

- Creating Foreign Connection Factory Resources
- Creating a Foreign Destination Resources

Creating Foreign Connection Factory Resources

A **Foreign Connection Factory** resource in a JMS module contains the JNDI name of the connection factory in the remote JNDI provider, the JNDI name that the connection factory is mapped to in the local WebLogic Server JNDI tree, and an optional user name and password.

The foreign connection factory creates non-replicated JNDI objects on each WebLogic Server instance that the parent foreign server is targeted to. (To create the JNDI object on every node in a cluster, target the foreign server to the cluster.)

Creating a Foreign Destination Resources

A *Foreign Destination* resource in a JMS module represents either a queue or a topic. It contains the destination JNDI name that is looked up on the foreign JNDI provider and the JNDI name that the destination is mapped to on the local WebLogic Server. When the foreign destination is looked up on the local server, a lookup is performed on the remote JNDI directory, and the destination object is returned from that directory.

Sample Configuration for MQSeries JNDI

Table 4-1 provides a possible a sample configuration when accessing a remote MQSeries JNDI provider.

Table 4-1 Sample MQSeries Configuration

Foreign JMS Object	Option Names	Sample Configuration Data

Foreign Server	Name
	JNDI Initial Context Factory
	JNDI Connection URL
	JNDI Properties
		`MQJNDI`
		`com.sun.jndi.fscontext.RefFSContextFactory`
		`file:/MQJNDI/`
		(If necessary, enter a comma-separated name=value list of properties.)

Foreign Connection Factory	Name
	Local JNDI Name
	Remote JNDI Name
	Username
	Password
		`MQ_QCF`
		`mqseries.QCF`
		`QCF`
		`weblogic_jms`
		`weblogic_jms`

Foreign Destination 1	Name
	Local JNDI Name
	Remote JNDI Name
		`MQ_QUEUE1`
		`mqseries.QUEUE1`
		`QUEUE_1`

Table 4-1 (Cont.) Sample MQSeries Configuration

Foreign JMS Object	Option Names	Sample Configuration Data
Foreign Destination 2	Name	`MQ_QUEUE2`
	Local JNDI Name	`mqseries.QUEUE2`
	Remote JNDI Name	`QUEUE_2`

Configuring Distributed Destination Resources

A distributed destination resource in a JMS module represents a single set of destinations (queues or topics) that are accessible as a single, logical destination to a client. For example, a distributed topic has its own JNDI name. The members of the set are typically distributed across multiple servers within a cluster, with each member belonging to a separate JMS server.

Applications that use a distributed destination are more highly available than applications that use standalone destinations because WebLogic JMS provides load balancing and failover for the members of a distributed destination in a cluster.

These sections provide information on how to create, monitor, and load balance distributed destinations:

- Uniform Distributed Destinations vs. Weighted Distributed Destinations
- Creating Uniform Distributed Destinations
- Creating Weighted Distributed Destinations
- Load Balancing Messages Across a Distributed Destination
- Distributed Destination Migration
- Distributed Destination Failover

Uniform Distributed Destinations vs. Weighted Distributed Destinations

> **Note:**
>
> Weighted Distributed Destinations were deprecated in WebLogic Server 10.3.4.0. Oracle recommends using Uniform Distributed Destinations.

WebLogic Server 9.x and later offers two types of distributed destination: uniform and weighted. In releases prior to WebLogic Server 9.x, WebLogic Administrators often needed to manually configure physical destinations to function as members of a distributed destination. This method provided the flexibility to create members that were intended to carry extra message load or have extra capacity; however, such differences often led to administrative and application problems because such a weighted distributed destination was not deployed consistently across a cluster. This type of distributed destination is officially referred to as a *weighted distributed destination* (or WDD).

A *uniform distributed destination* (UDD) greatly simplifies the management and development of distributed destination applications. Using uniform distributed destinations, you no longer need to create or designate destination members, but you can instead rely on WebLogic Server to

uniformly create the necessary members on the JMS servers to which a JMS module is targeted. This feature ensures the consistent configuration of all distributed destination parameters, particular with in regard to weighting, security, persistence, paging, and quotas.

The weighted distributed destination feature is still available for users who prefer to manually fine-tune distributed destination members. However, Oracle strongly recommends configuring uniform distributed destinations to avoid possible administrative and application problems due to a weighted distributed destination not being deployed consistently across a cluster.

For more information about using a distributed destination with your applications, see Using Distributed Destinations in *Developing JMS Applications for Oracle WebLogic Server*.

Creating Uniform Distributed Destinations

The WebLogic Remote Console enables you to configure, modify, target, and delete UDD resources in a JMS system module.

1. In the **Edit Tree**, go to **Services**, then **JMS System Resources**, then *myJMSSystemResource*.
2. From the **Navigation Tree**, as children of *myJMSSystemResource*, select the **Uniformed Distributed Queues** and **Uniformed Distributed Topics** you want to configure.

> **Note:**
>
> It is recommended that you create a single cluster targeted JMS Server and an associated persistent store to host the UDD resource, with optional HA configuration settings. This makes the UDD configuration simple, scalable, and highly available. See Simplified JMS Cluster and High Availability Configuration. A Replicated Distributed Topic is not supported by a cluster targeted JMS Server (use a Partitioned Distributed Topic instead).

Some uniform distributed destination options can be dynamically configured. When options are modified at run time, only incoming messages are affected; stored messages are not affected. For more information about the default values for all uniform distributed destination options, see the following entries in *MBean Reference for Oracle WebLogic Server*:

- UniformDistributedQueueBean
- UniformDistributedTopicBean

The following sections provide additional uniform distributed destination information:

- Targeting Uniform Distributed Queues and Topics
- Pausing and Resuming Message Operations on UDD Members
- Monitoring UDD Members
- Configuring Partitioned Distributed Topics

Targeting Uniform Distributed Queues and Topics

Unlike a standalone queue and topics resources in a module, which can only target a specific single-instance JMS server and only run on this one instance, a UDD can be targeted to multiple JMS server instances within the same server or cluster.

There are multiple ways to target a UDD but Oracle strongly recommends only one of them: configure UDDs to target a system module subdeployment that in turn directly references one or more JMS Servers. All other targeting options are strongly discouraged; for example, Oracle recommends against targeting a destination using 'default targeting' or targeting a subdeployment that in turn references a cluster or server name. Failure to follow this best practice can result in unintentional message loss.

For example, consider a system module named `jmssysmod-jms.xml` which is targeted to a cluster with three WebLogic Server instances: wlserver1, wlserver2, and wlserver3, where each server is in turn targeted by a configured JMS server, jmsserver1, jmsserver2, and jmsserver3. If you want to setup a uniform distributed queue in the same cluster, you can group the UDQ in a subdeployment named `jmsservergroup` to ensure that it is always linked to the exact desired JMS Server instances. You can optionally use the same subdeployment for a connection factory. Here is how the servergroup sub-deployment resources would look like in `jmssysmod-jms.xml`:

```
<weblogic-jms xmlns="http://xmlns.oracle.com/weblogic/weblogic-jms">
  <connection-factory name="MyCF">
    <sub-deployment-name>jmsservergroup</sub-deployment-name>
    <jndi-name>jms/MyCF</jndi-name>
  </connection-factory>
  <uniform-distributed-queue name="MyUDQ">
    <sub-deployment-name>jmsservergroup</sub-deployment-name>
    <jndi-name>jms/MyUDQ</jndi-name>
  </uniform-distributed-queue>
</weblogic-jms>
```

And here's how the corresponding subdeployment would be configured in the system module's corresponding stanza in the domain's `config.xml` file:

```
  <jms-system-resource>
    <name>jmssysmod-jms</name>
    <target>cluster1,</target>
    <sub-deployment>
      <name>jmsservergroup</name>
      <target>jmsserver1,jmsserver2,jmsserver3</target>
    </sub-deployment>
    <descriptor-file-name>jms/jmssysmod-jms.xml</descriptor-file-name>
  </jms-system-resource>
```

If you are using simplified JMS configuration that leverages a cluster targeted jms server named `MyClusteredJMSServer` instead of individually configured and targeted jms servers 'jmserver1', 'jmserver2', and 'jmserver3', then the above subdeployment's target simplifies to:

```
<target>MyClusteredJMSServer</target>
```

Instead of:

```
<target>jmsserver1,jmsserver2,jmsserver3</target>
```

> **Note:**
>
> - Remember, Oracle strongly recommends that a destination should always be configured to target subdeployments that in turn reference the exact desired JMS Server(s) for the destination. Oracle strongly advises against other destination targeting approaches, including default targeting. (Defaulting targeting a connection factory is fine.)
> - Changing the targets of a UDD can lead to the removal of a member destination and the consequent unintentional loss of messages.

Pausing and Resuming Message Operations on UDD Members

You can pause and resume message production, insertion, and/or consumption operations on a uniform distributed destinations, either programmatically (using JMX and the runtime MBean API) or administratively (using the WebLogic Remote Console). In this way, you can control the JMS subsystem behavior in the event of an external resource failure that would otherwise cause the JMS subsystem to overload the system by continuously accepting and delivering (and redelivering) messages.

For more information on the "pause and resume" feature, see Controlling Message Operations on Destinations.

Monitoring UDD Members

Runtime statistics for uniform distributed destination members can be monitored, as described in Monitoring JMS Statistics.

Configuring Partitioned Distributed Topics

> **Note:**
>
> Partitioned Distributed Topics are the only type of distributed topic that is supported when using cluster targeted JMS Servers or dynamic clusters. Configuration errors will be generated on an attempt to set up a Replicated Distributed Topic in these cases. If you need to replace a Replicated Distributed Topic with a Partitioned Distributed Topic, see Replacing a Replicated Distributed Topic in *Developing JMS Applications for Oracle WebLogic Server*.

The uniform distributed topic message `Forwarding Policy` specifies whether a sent message is forwarded to all members.

The valid values are:

- `Replicated`: The default. All physical topic members receive each sent message. If a message arrives at one of the physical topic members, a copy of this message is forwarded to the other members of that uniform distributed topic. A subscription on any one particular member will get a copy of any message sent to the uniform distributed topic logical name or to any particular uniform distributed topic member.

- `Partitioned`: The physical member receiving the message is the only member of the uniform distributed topic that is aware of the message. When a message is published to the logical name of a Partitioned uniform distributed topic, it will only arrive on one particular physical topic member. Once a message arrives on a physical topic member, the message is not forwarded to the rest of the members of the uniform distributed destination, and subscribers on other physical topic members do not get a copy of that message.

Most new applications will use the `Partitioned` forwarding policy in combination with a logical subscription topology on a uniform distributed topic that consists of:

- A same named physical subscription created directly on each physical member.
- A Client ID Policy of `Unrestricted`.
- A Subscription Sharing Policy of `Sharable`.

For more information on how to create and use the partitioned distributed topic, see:

- Configuring and Deploying MDBs Using Distributed Topics in *Developing Message-Driven Beans for Oracle WebLogic Server*
- Developing Advanced Pub/Sub Applications in *Developing JMS Applications for Oracle WebLogic Server*
- Load Balancing Partitioned Distributed Topics

Load Balancing Partitioned Distributed Topics

Partitioned topic publishers have the option of load balancing their messages across multiple members by tuning the connection factory `Affinity` and `Load Balance` attributes. The Unit of Order messages are routed to the correct member based on the UOO routing policy and the subscriber status.

Creating Weighted Distributed Destinations

> **Note:**
>
> Weighted Distributed Destinations (WDDs) are deprecated in WebLogic Server 10.3.4.0. Oracle strongly recommends using Uniform Distributed Destinations.
>
> You *cannot* use the WebLogic Remote Console to configure, modify, target, or delete WDD resources in JMS system modules.

For more information about the default values for all weighted distributed destination options, see the following entries in *MBean Reference for Oracle WebLogic Server*:

- DistributedQueueBean
- DistributedTopicBean

Unlike UDDs, WDD members cannot be monitored with the WebLogic Remote Console or though runtime MBeans. Also, WDD members cannot be uniformly targeted to JMS server or WebLogic Server instances in a domain. Instead, new WDD members must be manually configured on such instances, and then manually added to the WDD.

Load Balancing Messages Across a Distributed Destination

By using distributed destinations, JMS can spread or balance the messaging load across multiple destinations, which can result in better use of resources and improved response times. The JMS load-balancing algorithm determines the physical destinations that messages are sent to, as well as the physical destinations that consumers are assigned to.

Load-Balancing Options

WebLogic JMS supports two different algorithms for balancing the message load across multiple physical destinations within a given distributed destination set. You select one of these load balancing options when configuring a distributed topic or queue on the WebLogic Remote Console.

Round-Robin Distribution

In the round-robin algorithm, WebLogic JMS maintains an order of physical destinations within the distributed destination. The messaging load is distributed across the physical destinations one at a time in the order that they are defined in the WebLogic Server configuration (`config.xml`) file. Each WebLogic Server maintains an identical order, but may be at a different point within the ordering. Multiple threads of execution within a single server using a given distributed destination affect each other with respect to which physical destination a member is assigned to each time the member produces a message. Round-robin is the default algorithm and doesn't need to be configured.

For weighted distributed destinations only, if weights are assigned to any of the physical destinations in the set for a given distributed destination, then those physical destinations appear multiple times in the order.

Random Distribution

The random distribution algorithm uses the weight assigned to the physical destinations to compute a weighted distribution for the set of physical destinations. The messaging load is distributed across the physical destinations by pseudo-randomly accessing the distribution. In the short run, the load is not directly proportional to the weight. In the long run, the distribution approaches the limit of the distribution. A pure random distribution can be achieved by setting all the weights to the same value, which is typically 1.

Adding or removing a member (either administratively or as a result of a WebLogic Server shutdown/restart event) requires a recomputation of the distribution. Such events should be infrequent however, and the computation is generally simple, running in O(n) time.

Consumer Load Balancing

When an application creates a consumer, the application must provide a destination. If that destination represents a distributed destination, then WebLogic JMS must find a physical destination from which the consumer will receive messages from. The choice of which destination member to use is made by using one of the load balancing algorithms described in Load Balancing Options. The choice is made only once: when the consumer is created. From that point on, the consumer gets messages from that member only.

Producer Load Balancing

When a producer sends a message, WebLogic JMS looks at the destination to which the message is being sent. If the destination is a distributed destination, then the WebLogic JMS makes a decision as to where the message will be sent. That is, the producer sends to one of the destination members according to one of the load-balancing algorithms described in Load Balancing Options.

The producer makes such a decision each time it sends a message. However, there is no compromise of ordering guarantees between a consumer and producer, because consumers are load balanced once, and are then pinned to a single destination member.

> **Note:**
>
> If a producer attempts to send a persistent message to a distributed destination, every effort is made to first forward the message to distributed members that utilize a persistent store. However, if none of the distributed members utilize a persistent store, then the message will still be sent to one of the members according to the selected load-balancing algorithm.

Load-Balancing Heuristics

In addition to the algorithms described in Load Balancing Options, WebLogic JMS uses the following heuristics when choosing an instance of a destination.

- Transaction Affinity
- Server Affinity
- Queues with Zero Consumers
- Paused Distributed Destination Members

Transaction Affinity

When producing multiple messages within a transacted session, an effort is made to send all messages produced to the same WebLogic Server. Specifically, if a session sends multiple messages to a single distributed destination, then all of the messages are routed to the same physical destination. If a session sends multiple messages to multiple different distributed destinations, then an effort is made to choose a set of physical destinations served by the same WebLogic Server.

Server Affinity

The Server Affinity Enabled parameter on connection factories defines whether a WebLogic Server that is load balancing consumers or producers across multiple member destinations in a distributed destination set, first attempts to load balance across any other local destination members that are also running on the same WebLogic Server.

> **Note:**
>
> The Server Affinity Enabled attribute does not affect queue browsers. Therefore, a queue browser created on a distributed queue can be pinned to a remote distributed queue member even when Server Affinity is enabled.

To disable server affinity on a connection factory:

1. In the **Edit Tree**, go to **Services**, then **JMS System Resources**, then *myJMSSystemResource*.
2. From the **Navigation Tree**, as children of *myJMSSystemResource*, select the **Connection Factory** you want to configure.
3. On the **Load Balancing** page, define the **Server Affinity Enabled** field as follows:
   - If the **Server Affinity Enabled** option is *enabled* (True), then a WebLogic Server that is load balancing consumers or producers across multiple physical destinations in a distributed destination set, will first attempt to load balance across any other physical destinations that are also running on the same WebLogic Server.
   - If the **Server Affinity Enabled** option is *disabled* (False), then a WebLogic Server will load balance consumers or producers across physical destinations in a distributed destination set and disregard any other physical destinations also running on the same WebLogic Server.
4. Click **Save**.

For more information about how the Server Affinity Enabled setting affects the load balancing among the members of a distributed destination, see Distributed Destination Load Balancing When Server Affinity Is Enabled.

Queues with Zero Consumers

When load balancing consumers across multiple remote physical queues, if one or more of the queues have zero consumers, then those queues alone are considered for balancing the load. Once all the physical queues in the set have at least one consumer, the standard algorithms apply.

In addition, when producers are sending messages, queues with zero consumers are not considered for message production, unless all instances of the given queue have zero consumers.

Paused Distributed Destination Members

When distributed destinations are paused for message production or insertion, they are not considered for message production. Similarly, when destinations are paused for consumption, they are not considered for message production.

For more information about pausing message operations on destinations, see Controlling Message Operations on Destinations.

Defeating Load Balancing

Applications can defeat load balancing by directly accessing the individual physical destinations. That is, if the physical destination has no JNDI name, it can still be referenced using the `createQueue()` or `createTopic()` methods.

For instructions on how to directly access uniform and weighted distributed destination members, see Accessing Distributed Destination Members in *Developing JMS Applications for Oracle WebLogic Server*.

- Connection Factories

Connection Factories

Applications that use distributed destinations to distribute or balance their producers and consumers across multiple physical destinations, but do not want to make a load balancing decision each time a message is produced, can use a connection factory with the Load Balancing Enabled parameter disabled. To ensure a fair distribution of the messaging load among a distributed destination, the initial physical destination (queue or topic) used by producers is always chosen at random from among the distributed destination members.

To disable load balancing on a connection factory:

1. In the **Edit Tree**, go to **Services**, then **JMS System Resources**, then *myJMSSystemResource*.
2. From the **Navigation Tree**, as children of *myJMSSystemResource*, select the **Connection Factory** you want to configure.
3. On the **Load Balancing** page, define the setting of the **Load Balancing Enabled** field using the following guidelines:
   - `Load Balancing Enabled = True`

     For `Queue.sender.send()` methods, non-anonymous producers are load balanced on *every* invocation across the distributed queue members.

     For `TopicPublish.publish()` methods, non-anonymous producers are always pinned to the same physical topic for every call, irrespective of the Load Balancing Enabled setting.
   - `Load Balancing Enabled = False`

   Producers always produce to the same physical destination until they fail. At that point, a new physical destination is chosen.
4. Click **Save**.

> **Note:**
>
> Depending on your implementation, the setting of the **Server Affinity Enabled** attribute can affect load-balancing preferences for distributed destinations. See Distributed Destination Load Balancing When Server Affinity Is Enabled.

Anonymous producers (producers that do not designate a destination when created), are load-balanced each time they switch destinations. If they continue to use the same destination, then the rules for non-anonymous producers apply (as stated previously).

Distributed Destination Load Balancing When Server Affinity Is Enabled

Table 4-2 explains how the setting of a connection factory's Server Affinity Enabled parameter affects the load-balancing preferences for distributed destination members. The order of preference depends on the type of operation and whether or not durable subscriptions or persistent messages are involved.

The Server Affinity Enabled parameter for distributed destinations is different from the server affinity provided by the Default Load Algorithm attribute in the `ClusterMBean`, which is also used by the JMS connection factory to create initial context affinity for client connections.

See the Load Balancing for EJBs and RMI Objects and Initial Context Affinity and Server Affinity for Client Connections sections in *Administering Clusters for Oracle WebLogic Server*.

Table 4-2 Server Affinity Load Balancing Preferences

When the operation is	And Server Affinity Enabled is	Then load balancing preference is given to a
• `createReceiver()` for queues • `createSubscriber()` for topics	True	1. local member without a consumer 2. local member 3. remote member without a consumer 4. remote member
`createReceiver()` for queues	False	1. member without a consumer 2. member
`createSubscriber()` for topics (**Note:** nondurable subscribers)	True or False	1. local member without a consumer 2. local member
• `createSender()` for queues • `createPublisher()` for topics	True or False	There is no separate machinery for load balancing a created JMS producer. JMS producers are created on the server on which your JMS connection is load balanced or pinned. For more information about load balancing JMS connections created using a connection factory, refer to the Load Balancing for EJBs and RMI Objects and Initial Context Affinity and Server Affinity for Client Connections sections in *Administering Clusters for Oracle WebLogic Server*.

Table 4-2 (Cont.) Server Affinity Load Balancing Preferences

When the operation is	And Server Affinity Enabled is	Then load balancing preference is given to a
For persistent messages using `QueueSender.send()`	True	1. local member with a consumer and a store 2. remote member with a consumer and a store 3. local member with a store 4. remote member with a store 5. local member with a consumer 6. remote member with a consumer 7. local member 8. remote member
For persistent messages using `QueueSender.send()`	False	1. member with a consumer and a store 2. member with a store 3. member with a consumer 4. member
For nonpersistent messages using `QueueSender.send()`	True	1. local member with a consumer 2. remote member with a consumer 3. local member 4. remote member
For nonpersistent messages: • `QueueSender.send()` • `TopicPublish.publish()`	False	1. member with a consumer 2. member
`createConnectionConsumer()` for session pool queues and topics	True or False	local member *only* **Note:** Session pools are now used rarely, as they are not a required part of the Jakarta EE specification, do not support JTA user transactions, and are largely superseded by message-driven beans (MDBs), which are simpler, easier to manage, and more capable.

Distributed Destination Migration

For clustered JMS implementations that take advantage of the service migration feature, a JMS server and its distributed destination members can be migrated to another WebLogic Server instance within the cluster. Service migrations can take place due to scheduled system maintenance, as well as in response to a server failure within the cluster.

However, the target WebLogic Server may already be hosting a JMS server with all of its physical destinations. This can lead to situations where the same WebLogic Server instance hosts two physical destinations for a single distributed destination. This is permissible in the short term, because a WebLogic Server instance can host multiple physical destinations for that distributed destination. However, load balancing in this situation is less effective.

In such a situation, each JMS server on a target WebLogic Server instance operates independently. This is necessary to avoid merging of the two destination instances, and/or

disabling of one instance on both , which can make some messages unavailable for a prolonged period of time. The long-term intent, however, is to eventually re-migrate the migrated JMS server to yet another WebLogic Server instance in the cluster.

For more information about configuring JMS migratable targets, see Migration of JMS-related Services.

Distributed Destination Failover

If the server instance that is hosting the JMS connections for the JMS producers and JMS consumers should fail, then all the producers and consumers using these connections are closed and are *not* re-created on another server instance in the cluster. Furthermore, if a server instance that is hosting a JMS destination should fail, then all the JMS consumers for that destination are closed and not re-created on another server instance in the cluster.

If the distributed queue member on which a queue producer is created should fail, yet the WebLogic Server instance where the producer's JMS connection resides is still running, then the producer remains active and WebLogic JMS will fail it over to another distributed queue member, irrespective of whether the Load Balancing option is enabled.

For more information about procedures for recovering from a WebLogic Server failure, see Recovering From a Server Failure in *Developing JMS Applications for Oracle WebLogic Server*.

Configure an Unrestricted ClientID

The Client ID Policy specifies whether more than one JMS connection can use the same client ID in a cluster.

Valid values for this policy are:

- `RESTRICTED`: This is the default. Only one connection that uses this policy can exist in a cluster at any given time for a particular client ID (if a connection already exists with a given client ID, then attempts to create new connections using this policy with the same client ID fail with an exception).
- `UNRESTRICTED`: Connections created using this policy can specify any client ID, even when other restricted or unrestricted connections already use the same Client ID. When a durable subscription is created using an Unrestricted client ID, it can only be cleaned up using `weblogic.jms.extensions.WLSession.unsubscribe(Topic topic, String name)`. See Managing Subscriptions in *Developing JMS Applications for Oracle WebLogic Server*.

Oracle recommends setting the client ID policy to `Unrestricted` for new applications (unless your application architecture requires exclusive client IDs), especially if sharing a subscription (durable or non-durable). Subscriptions created with different client ID policies are always treated as independent subscriptions. See ClientIdPolicy in the *MBean Reference for Oracle WebLogic Server*.

To set the `Client ID Policy` on the connection factory using the Remote Console, go to the **Client** page in the **Edit Tree**, under **Services**: **JMS System Resources**: *myJMSSystemResource* of the **Connection Factory** you want to configure. The connection factory setting can be overridden programmatically using the `setClientIDPolicy` method of the `WLConnection` interface in the *Java API Reference for Oracle WebLogic Server*.

> **Note:**
>
> Programmatically changing (overriding) the client ID policy settings on a JMS connection runtime object is valid only for that particular connection instance and for the life of that connection. Any changes made to the connection runtime object are not persisted/reflected by the corresponding JMS connection factory configuration defined in the underlying JMS module descriptor.

For more information on how to use the client ID policy, see Developing Advanced Pub/Sub Applications in *Developing JMS Applications for Oracle WebLogic Server*.

Configure Shared Subscriptions

The Subscription Sharing Policy specifies whether subscribers can share subscriptions with other subscribers on the same connection.

Valid values for this policy are:

- `Exclusive`: This is the default. All subscribers created using this connection factory cannot share subscriptions with any other subscribers.
- `Sharable`: Subscribers created using this connection factory can share their subscriptions with other subscribers, regardless of whether those subscribers are created using the same connection factory or a different connection factory. Consumers can share a non durable subscription only if they have the same client ID and client ID policy; consumers can share a durable subscription only if they have the same client ID, client ID policy, and Subscription Name.

WebLogic JMS applications can override the Subscription Sharing Policy specified on the connection factory configuration by casting a `javax.jms.Connection` instance to `weblogic.jms.extension.WLConnection` and calling `setSubscriptionSharingPolicy(String)`.

Most applications with a Subscription Sharing Policy will use an Unrestricted client ID policy to ensure that multiple connections with the same client ID can exist.

Two durable subscriptions with the same client ID and Subscription Name are treated as two different independent subscriptions if they have a different client ID policy. Similarly, two Sharable non-durable subscriptions with the same client ID are treated as two different independent subscriptions if they have a different client ID policy.

For more information abut how to use the Subscription Sharing Policy, see Developing Advanced Pub/Sub Applications in *Developing JMS Applications for Oracle WebLogic Server*.

5

Simplified JMS Cluster and High Availability Configuration

Learn about new cluster-targeting enhancements and how they simplify JMS configuration. These enhancements make the JMS service dynamically scalable and highly available without the need for an extensive configuration process in a cluster.

This chapter includes the following sections:

- What Are the WebLogic Clustering Options for JMS?
  A WebLogic Cluster can contain individually configured servers, dynamically generated servers, or a mix of both.
- Understanding the Simplified JMS Cluster Configuration
  A cluster targeted JMS service configuration directly targets JMS service artifacts such as a JMS Server, SAF Agent, or Path Service and their associated persistent stores to the same cluster. A Messaging Bridge is also a JMS artifact that can be cluster targeted. Cluster targeting JMS artifacts is simpler and provides more HA capability than individually configuring and targeting a JMS artifact for each WebLogic server in the cluster.
- Using Persistent Stores with Cluster Targeted JMS Servers
- Targeting JMS Modules Resources
  JMS system modules support default and deployment targeting, either of these can be used to take advantage of simplified cluster configuration.
- Simplified JMS Configuration and High Availability Enhancements
  WebLogic Server supports high availability for JMS service artifacts deployed in a cluster. Both server and service failure scenarios are handled by automatically migrating an artifact's instance to other running servers. During this process, the system evaluates the overall server load and availability and moves the instances accordingly.
- Considerations and Limitations of Clustered JMS
  Before developing applications using dynamic clusters and cluster targeted JMS servers, you must consider the limitations posed by Clustered JMS.
- Best Practices for Using Cluster Targeted JMS Services
  Learn about the recommended best practices and design patterns for using cluster targeted JMS services.
- Runtime MBean Instance Naming Syntax
  The runtime MBean associated with each of the JMS artifacts, such as Persistent store, JMS server, SAFAgent, MessagingBridge and PathService distributed in a cluster are named based on the distribution policy set. This is enforced by a MBean check.

What Are the WebLogic Clustering Options for JMS?

A WebLogic Cluster can contain individually configured servers, dynamically generated servers, or a mix of both.

WebLogic Server has the following cluster types:

- Configured: A cluster where each member server is individually configured and individually targeted to the cluster. The value of the **Dynamic Cluster Size** attribute for the cluster configuration is 0. This type of cluster is also known as a Static Cluster.
- Dynamic: A cluster where all the member servers are created using a server template. These servers are referred to as dynamic servers. The value of the **Dynamic Cluster Size** attribute for the cluster configuration is greater than 0.
- Mixed: A cluster where some member servers are created using a server template (dynamic servers) and the remaining servers are manually configured (configured servers). Because a mixed cluster contains dynamic servers, the value of the **Dynamic Cluster Size** attribute for the cluster configuration is greater than 0.

For more information about using dynamic servers, see Dynamic Clusters.

Understanding the Simplified JMS Cluster Configuration

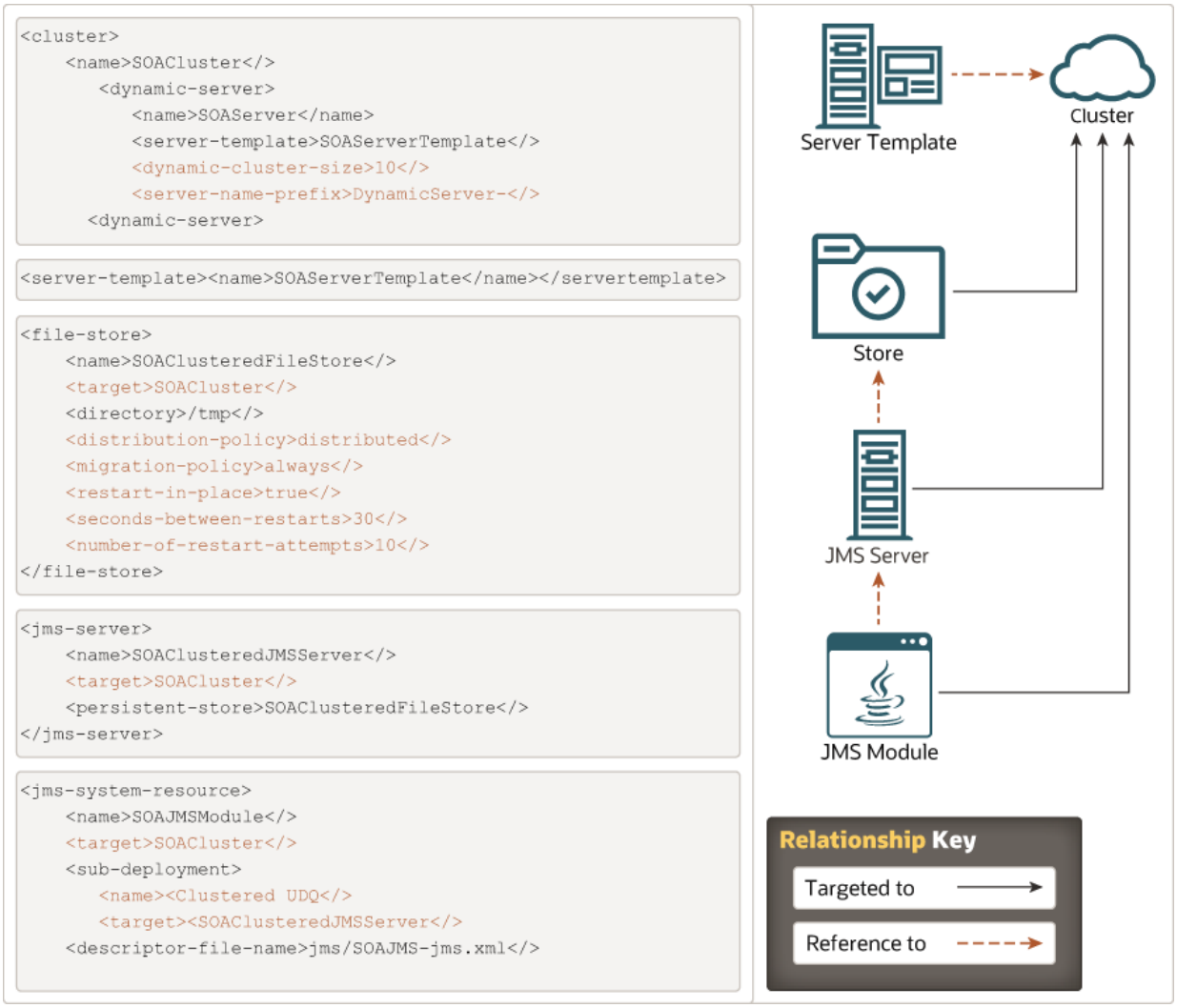
A cluster targeted JMS service configuration directly targets JMS service artifacts such as a JMS Server, SAF Agent, or Path Service and their associated persistent stores to the same cluster. A Messaging Bridge is also a JMS artifact that can be cluster targeted. Cluster targeting JMS artifacts is simpler and provides more HA capability than individually configuring and targeting a JMS artifact for each WebLogic server in the cluster.

Cluster targeted JMS service artifacts can be distributed across the cluster or singletons depending on their configured Distribution Policy. When distributed, the cluster will automatically start a new instance of the artifact (and associated store if applicable) on each new cluster member and that member becomes the preferred server for that instance. For artifacts that are not distributed, e.g. they have a singleton Distribution Policy, the system will select a single server in the cluster to start a single instance of that artifact. See Simplified JMS Configuration and High Availability Enhancements.

In the case of a dynamic or a mixed cluster, the number of instances automatically grow when the cluster size grows. To dynamically scale the size of the dynamic or mixed cluster or the dynamic servers of the mixed cluster, adjust the dynamic cluster Size attribute of your cluster configuration.

Figure 1 shows the relationship between the JMS and a dynamic cluster configuration in the `config.xml file`.

Figure 5-1 Dynamic Clustered JMS



Using Custom Persistent Stores with Cluster-Targeted JMS Service Artifacts

The custom persistent store used by the JMS service artifacts must be targeted to the same cluster with appropriate attribute values configured to take advantage of cluster enhancements. However, cluster-targeted SAF Agents and JMS Servers can also continue to use the default store available on each cluster member, which does not offer any of the new enhancements discussed in this chapter. See Simplified JMS Configuration and High Availability Enhancements.

Targeting JMS Modules Resources

JMS system modules continue to support two types of targeting, either of which can be used to take advantage of simplified cluster configuration.

- Any default targeted JMS resource in a module (a JMS resource that is not associated with a subdeployment), inherits the targeting of its parent module, and the parent module can be targeted to any type of cluster.
- Module subdeployment targets can reference clustered JMS Servers or SAF Agents for hosting regular destinations or imported destinations respectively. Using a cluster-targeted

JMS Server or a SAF Agent in a subdeployment eliminates the need to individually create and enumerate the JMS Servers or SAF Agents in the subdeployment, which is particularly useful for Uniform Distributed Destination and imported destination deployment.

See Targeting Best Practices.

> **Note:**
>
> A module or its subdeployments cannot be directly targeted to a Dynamic cluster member server.

Using Persistent Stores with Cluster Targeted JMS Servers

The persistent store associated with a Cluster Targeted JMS server can be a custom persistent store that is targeted to the same cluster as the JMS server or, in limited circumstances, can be a default store. It is strongly recommended to always use a custom persistent store instead of a default store as this provides more high availability capabilities (such as a service migration), and this will work in all topologies including dynamic clusters and multi-tenant.

Targeting JMS Modules Resources

JMS system modules support default and deployment targeting, either of these can be used to take advantage of simplified cluster configuration.

- Any *default targeted* JMS resource in a module (a JMS resource that is not associated with a *subdeployment*) inherits the targeting of its parent module, and the parent module can be targeted to any type of cluster. Note that Oracle strongly recommends using subdeployment targeting instead of default targeting for destinations.
- Module *subdeployment* targets can reference clustered JMS servers. Using a cluster targeted JMS server in a subdeployment eliminates the need to individually enumerate individual JMS servers in the subdeployment, which is particularly useful for uniform distributed destination deployment.

See Targeting Best Practices.

> **Note:**
>
> A module or its subdeployments cannot be directly targeted to a Dynamic cluster member.

Simplified JMS Configuration and High Availability Enhancements

WebLogic Server supports high availability for JMS service artifacts deployed in a cluster. Both server and service failure scenarios are handled by automatically migrating an artifact's instance to other running servers. During this process, the system evaluates the overall server load and availability and moves the instances accordingly.

Cluster-targeting enhancements in this release of WebLogic Server eliminate many of the limitations that existed in the previous releases:

- In releases before 12.2.1.0, only JMS servers, persistent stores, and SAF Agents (partially) were allowed to target to a cluster. In 12.2.1.0 and later, the support is extended for all of the JMS service artifacts including SAF Agents, path services, and messaging bridges and for all types of clusters (Configured, Mixed, and Dynamic).
- Enhancements in 12.2.1.0 and later lets you easily configure and control the distribution behavior, as well as the JMS high availability (also known as JMS automatic service migration) behavior for all cluster targeted JMS Service artifacts. All of these configurations now exist in a single location, which is a Persistent Store for all the artifacts that depend on that store, or on the messaging bridge (which does not use the Store). This eliminates the need for migratable targets that were used in the previous releases.
- Because the *logical* JMS artifacts are targeted to clusters, the system automatically creates any "physical" instances required on a cluster member when it joins the cluster. This allows the JMS Service to automatically scale up when the cluster size grows. With optional high availability configuration, the "physical" instances can restart or migrate in the event of service failure or server failure or shutdown, making the JMS Service highly available with minimal configuration.

The primary attributes that control the scalability and high availability behavior of cluster targeted JMS services are Distribution policy and Migration policy. In addition to these policies, there are a few additional attributes that can be used for fine-tuning the high availability behavior such as restarting the instance in place (on the same server) before attempting to migrate elsewhere. These policies and attributes are described in the following sections:

- Defining the Distribution Policy for JMS Services
  The Distribution Policy setting for a custom persistent store or messaging bridge determines how the associated JMS Service artifacts (JMS Server, SAF Agent, and Path Service) are distributed in a cluster and the same setting on the Messaging Bridge determines its distribution behavior.
- Defining the Migration Policy for JMS Services
  The store Migration Policy setting controls service migration and restart behavior of cluster-targeted JMS service artifact instances.
- Additional Configuration Options for JMS Services

Defining the Distribution Policy for JMS Services

The Distribution Policy setting for a custom persistent store or messaging bridge determines how the associated JMS Service artifacts (JMS Server, SAF Agent, and Path Service) are distributed in a cluster and the same setting on the Messaging Bridge determines its distribution behavior.

The following are the options that control the distribution behavior of the JMS service artifact:

- Distributed: In this mode, the cluster automatically ensures that there is a minimum of one instance per server. When the cluster starts, the system ensures that all the messaging service instances are up if possible, and when applicable it will attempt an even distribution of the instances. In addition, all the instances will automatically try to start on their home/preferred server first. Depending on the Migration Policy, instances can automatically migrate or even fail-back as needed to ensure high availability and even load balancing across the cluster.

> **Note:**
>
> The default value for the store Distribution Policy attribute is Distributed. Distributed is the required value for SAF Agents, and is also required for cluster-targeted JMS Servers that host Uniform Distributed Destinations.

- Singleton: In this mode, a JMS server or a path service has one instance per cluster.

> **Note:**
>
> This option is required for a path service and for cluster-targeted JMS servers that host singleton or standalone (non-distributed) destinations.

Defining the Migration Policy for JMS Services

The store Migration Policy setting controls service migration and restart behavior of cluster-targeted JMS service artifact instances.

For high availability and service migration, set the migration policy as follows on the associated store:

- **Off**: This option disables migration. By default, Restart In Place is also disabled when the Migration Policy is Off.
- **Always**: This option enables the system to automatically migrate instances in all situations. This includes administrative shutdown, crashes or bad health of the hosting server or subsystem service. This option also enables service restart-in-place, which automatically tries to restart a failing store on the current hosting server JVM before trying to migrate it to another server JVM in the same cluster.
- **On-Failure**: This option enables the system to automatically migrate the instances only in case of failure or a crash (bad health) of its hosting server. The instances will not migrate when there is an administrative shutdown, instead they will restart when the server is restarted. This option also enables service restart-in-place, which automatically tries to restart a failing store on the current hosting server JVM before trying to migrate it to another server JVM in the same cluster.

Note:

- WebLogic Server provides complete in-place restart support for the JMS services regardless of targeting type, deployment scope and migration policy setting. See Service Restart In Place in *Administering the WebLogic Persistent Store*.
- JMS Service Migration and JTA Migration work independently based on their respective Migration Policy settings and are configured independently. For information on Dynamic Clusters and JTA migration policies, see Understanding the Service Migration Framework.
- To enable support for cluster-targeted JMS Service artifacts with the Always or On-Failure migration policy, you must configure Cluster Leasing. See Leasing in *Administering Clusters for Oracle WebLogic Server.*
- Note: It is a best practice to use the Database Leasing option instead of Consensus Leasing.
- When a distributed instance is migrated from its preferred server, it will try to fail back when the preferred server is restarted.

Additional Configuration Options for JMS Services

There are different store configuration options available for automatic migration and high availability of JMS services. The configuration options apply when the JMS artifact is cluster-targeted and the Migration Policy is set to `On-Failure` or `Always`, or when the Migration Policy is `Off` and Restart In Place is explicitly configured to `True`.The following table describes the different configuration properties for JMS service migration:

Table 5-1 Configuration Properties for JMS Service Migration

Property	Default Value	Description
`Restart In Place`	False when Migration Policy=Off, True otherwise	Defines how the system responds on a JMS Service failure within a healthy Oracle WebLogic Server. If a service fails and if this property is enabled, then the system first attempts to restart that store and associated service artifact on the same server before migrating to another server. **Note**: This attribute does not apply when an entire server fails. See Service Restart In Place in *Administering the WebLogic Persistent Store*.
`Seconds Between Restarts`	30	If `Restart In Place` is enabled, then this property specifies the delay, in seconds, between restarts on the same server.
`Number of Restart Attempts`	6	If `Restart In Place` is enabled, then this number determines the restart attempts the system should make before trying to migrate the artifact instance to another server.

Table 5-1 (Cont.) Configuration Properties for JMS Service Migration

Property	Default Value	Description
`Initial Boot Delay Seconds`	60	Controls how fast subsequent instances are started on a server after the first instance is started. This prevents the system from getting overloaded during startup. A value of `0` indicates that the system does not need to wait, which may lead to overload situations. The system's default value is 60 seconds.
`Failback Delay Seconds`	-1	Specifies the time to wait before failing back an artifact's instance to its preferred server. A value > 0 specifies that the time, in seconds, to delay before failing a JMS artifact back to its preferred server. A value of `0` indicates that the instance would never failback. A value of `-1` indicates that there is no delay and the instance would failback immediately.
`Partial Cluster Stability Seconds`	240	Specifies the amount of time, in seconds, to delay before a partially started cluster starts all cluster-targeted JMS artifact instances that are configured with a Migration Policy of `Always` or `On-Failure`. This delay ensures that services are balanced across a cluster even if the servers are started sequentially. A value > 0 specifies the time, in seconds, to delay before a partially started cluster starts dynamically configured services. A value of `0` specifies no delay in starting all the instances on available servers. The default delay value is `240` seconds.
`Fail Over Limit`	-1	Specify a limit for the number of cluster-targeted JMS artifact instances that can fail over to a particular JVM. A value of `-1` means there is no failover limit (unlimited). A value of `0` prevents any failovers of cluster-targeted JMS artifact instances, so no more than 1 instance will run per server (this is an instance that has not failed over). A value of `1` allows one failover instance on each server, so no more than two instances will run per server (one failed over instance plus an instance that has not failed over).

Table 5-1 (Cont.) Configuration Properties for JMS Service Migration

Property	Default Value	Description
`RebalanceEnabled`	False	Set to `True` to periodically rebalance the running cluster-targeted instances with a Migration Policy of `Always` or `On-Failure` when the system is idle and the instances are unevenly distributed. The system is considered idle when the `Partial Cluster Stability Delay` and the `Initial Boot Delay` have passed, and no instances have moved plus no server status has changed within the last two system check periods (typically 10 seconds between each check). Two is the default value. You can tune this value higher using the `Rebalance Delay Periods` on the cluster bean. The system is considered unbalanced if any running server has an instance count that is more than one higher than the instance count on any other running server. For example, when Server A has 3 instances and Server B has 1 instance. The rebalance heuristic forces all running instances that are not on their preferred server to move to their preferred server if the preferred server is running. It then finds the alphanumerically highest failed-over instance on the running server with the most instances, moves this instance to the alphanumerically least most running server with the fewest failed-over instances, and repeats this pattern until the system is no longer unbalanced. **Note**: You can override this setting for all related instances on an Oracle WebLogic Server to `true` or `false` using the `weblogic.jms.ha.RebalanceEnabledOverride` system property. For example, `-Dweblogic.jms.ha.RebalanceEnabledOverride=true`.

Considerations and Limitations of Clustered JMS

Before developing applications using dynamic clusters and cluster targeted JMS servers, you must consider the limitations posed by Clustered JMS.

The following are the limitations and other behaviors for consideration:

- There are special considerations when a SAF agent-imported destination with an Exactly-Once QoS Level forwards messages to a distributed destination that is hosted on a mixed or dynamic cluster. See Best Practices for Using Clustered JMS Services.
- WLST Offline does not support the `assign` command to target JMS servers to a dynamic cluster. Use the `get` and `set` command.
- Weighted distributed destinations (a deprecated type of distributed destination composed of a group of singleton destinations), are not supported on cluster-targeted JMS Servers.
- Replicated distributed topics (RDTs) are not supported when any member destination is hosted on a cluster-targeted JMS server. Configure a partitioned distributed topic (PDT) or

a singleton topic instead. If you are converting a configuration that already has RDTs configured, see Replacing an RDT with a PDT in *Developing JMS Applications for Oracle WebLogic Server*.

- A custom persistent store with a `Singleton` *Distribution Policy*, and an `Always` (Or On-Failure) *Migration Policy* is required for a cluster targeted JMS Server to allow Standalone destinations.
- There is no support for manually (administratively) forcing the migration or fail-back of a service instance that is generated from a cluster-targeted JMS artifact.
- A path service must be configured if there are any distributed or imported destinations that are used to host Unit-of-Order (UOO) messages. In addition, such destinations need to be configured with a path service UOO routing policy instead of the default hashed UOO routing policy because hash based UOO routing is not supported in cluster-targeted JMS. Attempts to send UOO messages to a distributed or imported destination that does not configure a path service routing policy and that is hosted on a cluster targeted JMS Service will fail with an exception.
- A *Fail Over Limit* setting only applies when the JMS artifact is cluster-targeted and the *Migration Policy* is set to `On-Failure` or `Always`.

Note that a cluster targeted path service must be configured to reference a Store that has a *Singleton Distribution Policy* and an *Always Migration Policy*.

- Interoperability and Upgrade Considerations of Cluster Targeted JMS Servers

Interoperability and Upgrade Considerations of Cluster Targeted JMS Servers

The following section provides information about interoperability and upgrade considerations when using cluster targeted JMS servers:

- JMS clients, bridges, MDBs, SAF clients, and SAF agents from previous releases can communicate with cluster targeted JMS servers.
- There are special considerations when a SAF agent-imported destination with an `Exactly-Once` QOS Level forwards messages to a distributed destination that is hosted on a Mixed or Dynamic cluster. See Best Practices for Using Clustered JMS Services.
- No conversion path is available for moving data (messages) or configurations from non-cluster targeted JMS servers to cluster targeted JMS servers, or vice versa.

The migratable target-based service migration of JMS services is supported on a Configured Cluster.

For example, a messaging configuration for JMS servers and persistent stores can target a single manually configured WebLogic Server or to a single migratable target. Similarly, SAF Agents an target a single manually configured WebLogic Server or to a single migratable target or to a Cluster.

See Automatic Migration of JMS Services.

Best Practices for Using Cluster Targeted JMS Services

Learn about the recommended best practices and design patterns for using cluster targeted JMS services.

- Prior to decreasing the dynamic-cluster-size setting of a dynamic cluster or deleting a configure server in a configured or mixed cluster, process the messages and delete the stores that are associated with a retired cluster targeted JMS server before shutting down its WebLogic Server instance. For example:
  - Pause the retiring destination instances for production.
  - Let consumer applications process the remaining messages on the paused destinations.
  - Shut down the server instance.
  - Delete any persistent store files or database tables that are associated with the retired instance.

  Alternatively, a much simpler solution is to setup the system to automatically migrate destinations on retired servers to the remaining servers:
  - Configure the stores to have a Migration Policy of On-Failure or Always (not Off).
  - Never reduce the configured dynamic-cluster-size of a dynamic cluster or delete a configured server from a configured/mixed cluster. Instead, simply don't boot the retired servers. The On-Failure or Always stores will migrate to running servers. Note that the WLST `scaleDown` command should be used only when its `updateConfiguration` option is disabled, otherwise it will reduce the cluster dynamic cluster size setting.
- Use cluster targeted stores instead of default stores for clustered targeted JMS servers and SAF Agents.
- When enabling high availability (that is, when the Migration Policy on the store is set to either On-Failure or Always), ensure that cluster leasing is configured. As a best practice, database leasing is preferred over consensus leasing. See Leasing in *Administering Clusters for Oracle WebLogic Server*.
- When configuring destinations in a module, use a subdeployment that targets a specific clustered JMS server or SAF agent instead of using default targeting. This ensures that the destination creates members on exactly the desired JMS server instances.
- When using `Exactly-Once` QOS Level SAF agents and SAF clients, as a best practice, ensure that Stores associated with the SAF Agents are configured with the migration-policy set to Always.

Also, in the event of change in the cluster size (particularly when shrinking), ensure that the backing tables (in case of JDBC store) or files (in case of FILE store) are not deleted or destroyed, so that they can be migrated over to any available cluster members for continuous availability of the SAF service.

Runtime MBean Instance Naming Syntax

The runtime MBean associated with each of the JMS artifacts, such as Persistent store, JMS server, SAFAgent, MessagingBridge and PathService distributed in a cluster are named based on the distribution policy set. This is enforced by a MBean check.

The types of Distribution Policies are:

- Distributed — Instances are automatically created and named uniquely by naming them after their home or preferred host WebLogic Server the first time when the WebLogic Server boots . The format is `<configured-name>@<server-name>`.
- Singleton — Instances will be named with its configured name along with "`01`". The format is `<configured-name>-01`. No server name is added to this instance name.

The following sections brief about the Instance Naming Syntax for persistent store:

- Instance Naming Syntax for .DAT File
- Instance Naming Syntax for .RGN File
- JDBC Store Table Name Syntax

Instance Naming Syntax for .DAT File

In case of file stores (.DAT files) that are targeted to a cluster, an instance's data files are uniquely named based on the corresponding store instance name. For example, in Distributed mode, an instance's files will be names as `<Store name>@<Server instance name> NNNNNN.DAT`, where <NNNNNN> is a number ranging from 000000–999999.

> **Note:**
>
> A single file instance may create one or more .DAT files.

Instance Naming Syntax for .RGN File

A replicated store's instance regions targeted to a cluster, are uniquely named based on the corresponding store instance name. For example, in distributed mode, a replicated store's instance will contain the syntax of `<configured Replicated Store name>@<Server instance name> NNNNNN.RGN`, where <NNNNNN> is a number ranging from `000000-999999`.

JDBC Store Table Name Syntax

The prefix of a JDBC Store's table name is configurable via its Prefix Name attribute. The suffix of a JDBC Store's table name is automatically generated and differs based on whether the store is cluster targeted. The table name must often be changed using the JDBC store PrefixName setting to ensure different JDBC stores use different tables. No two different instances of JDBC store can share the same backing table. See Table 5-2 and Table 5-3.

Table 5-2 JDBC Store Table Name Syntax for Cluster Targeted Case

PrefixName	Distribution Policy	Instance Name	Store Table Name
`myPrefix`	Distributed	`myStore@myServer`	`myPrefix_myServer_WLStore`
`myPrefix.` (ends with '.')	Distributed	`myStore@myServer`	`myPrefix._myServer_WLStore`
Not set	Distributed	`myStore@myServer`	`myServer_WLStore`
`myPrefix`	Singleton	`myStore-01`	`myPrefix_01_WLStore`
`myPrefix.` (ends with '.')	Singleton	`myStore-01`	`myPrefix.S_01_WLStore`
Not set	Singleton	`myStore-01`	`S_01_WLStore`

Table 5-3 Non-Cluster (single-server) Targeted Case

Prefix Name	Distribution Policy	Instance Name	Store Table Name
myprefix	NA	myStore	myPrefixWLStore
NA	NA	myStore	WLStore

6

Using WLST to Manage JMS Servers and JMS System Module Resources

Similar to most other WebLogic artifacts, the WebLogic Scripting Tool (WLST), a command-line scripting interface, can also be used to create and manage JMS servers and JMS system module resources for Oracle WebLogic Server.See Using the WebLogic Scripting Tool and WLST Sample Scripts in *Understanding the WebLogic Scripting Tool* for more information about the scripting tool.
This chapter includes the following sections:

- Understanding JMS System Modules and Subdeployments
  A JMS system module is described by the `jms-system-resource` MBean in the `config.xml` file.
- How to Create JMS Servers and JMS System Module Resources
  Creating JMS servers and JMS system module resources using WLST include the basic tasks of starting an edit session, creating a JMS system module, and creating a JMS server resource.
- How to Modify and Monitor JMS Servers and JMS System Module Resources
  You can modify or monitor JMS objects and attributes by using the appropriate method available from the MBean.
- Best Practices When Using WLST to Configure JMS Resources
  Learn about best practices when using WLST to configure JMS servers and JMS system module resources.

Understanding JMS System Modules and Subdeployments

A JMS system module is described by the `jms-system-resource` MBean in the `config.xml` file.

Basic components of a `jms-system-resource` MBean are:

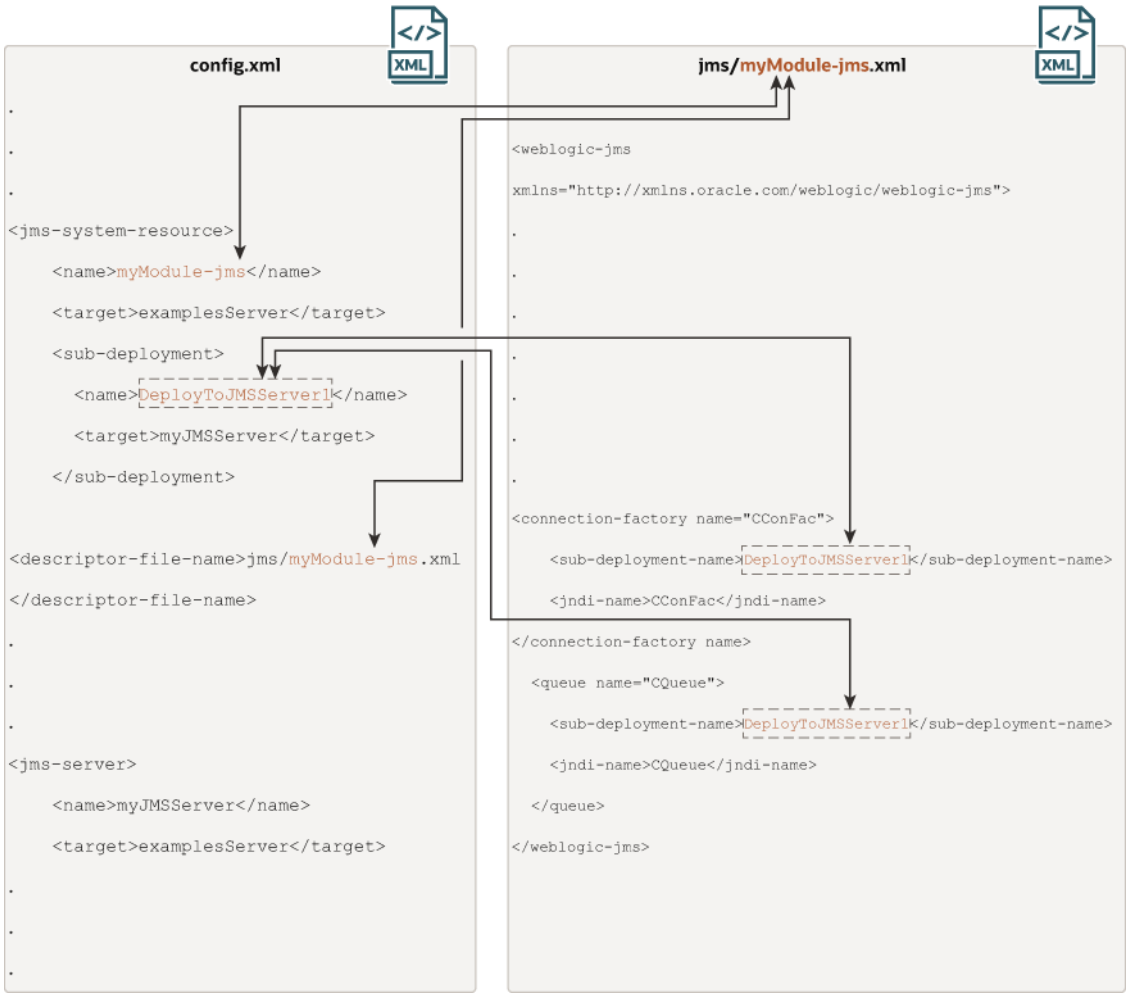
- `name`: Name of the module.
- `target`: Server, cluster, or migratable target the module is targeted to.
- `sub-deployment`: A mechanism by which JMS system module resources (such as queues, topics, and connection factories) are grouped and targeted to a server resource (such as a JMS server instance, WebLogic Server instance, or cluster).
- `descriptor-file-name`: Path and file-name of the system module file.

The JMS resources of a system module are located in a module descriptor file that conforms to the `weblogic-jmsmd.xml` schema. In Figure 6-1, the module is named `myModule-jms.xml` and it contains JMS system resource definitions for a connection factory and a queue. The `sub-deployment-name` element is used to group and target JMS resources in the *`myModule`*`-jms.xml` file to `targets` in the `config.xml`. You have to provide a value for the `sub-deployment-name` element when using WLST. For more information on subdeployments, see JMS System Module and Resource Subdeployment Targeting. In Figure 6-1, the `sub-deployment-name`

DeployToJMSServer1 is used to group and target the connection factory `CConfac` and the queue `CQueue` in the `myModule-jms` module.

For more information about how to use JMS resources, see What Are JMS Configuration Resources?.

Figure 6-1 Subdeployment Architecture



How to Create JMS Servers and JMS System Module Resources

Creating JMS servers and JMS system module resources using WLST include the basic tasks of starting an edit session, creating a JMS system module, and creating a JMS server resource.

After you have established an edit session, use the following steps to configure JMS servers and system module resources:

1. Get the WebLogic Server MBean object for the server that you want to configure resources. For example:

```
servermb=getMBean("Servers/examplesServer")
    if servermb is None:
        print '@@@ No server MBean found'
```

2. Create your system resource. For example:

```
jmsMySystemResource = create(myJmsSystemResource,"JMSSystemResource")
```

3. Target your system resource to a WebLogic Server instance. For example:

```
jmsMySystemResource.addTarget(servermb)
```

4. Get your system resource object. For example:

```
theJMSResource = jmsMySystemResource.getJMSResource()
```

5. Create resources for the module, such as queues, topics, and connection factories. For example:

```
connfact1 = theJMSResource.createConnectionFactory(factoryName)
jmsqueue1 = theJMSResource.createQueue(queueName)
```

6. Configure resource attributes. For example:

```
connfact1.setJNDIName(factoryName)
jmsqueue1.setJNDIName(queueName)
```

7. Create a subdeployment name for system resources. See Understanding JMS System Modules and Subdeployments. For example:

```
connfact1.setSubDeploymentName('DeployToJMSServer1')
jmsqueue1.setSubDeploymentName('DeployToJMSServer1')
```

8. Create a JMS server. For example:

```
jmsserver1mb = create(jmsServerName,'JMSServer')
```

9. Target your JMS server to a WebLogic Server instance or cluster. For example:

```
jmsserver1mb.addTarget(servermb)
```

10. Create a subdeployment object using the `value you provided for the sub-deployment-name` element. This step groups the system resources in `module to a sub-deployment element in the config.xml`. For example:

```
subDep1mb = jmsMySystemResource.createSubDeployment('DeployToJMSServer1')
```

11. Target the subdeployment to a server resource such as a JMS server instance, WebLogic Server instance, or cluster. For example:

```
subDep1mb.addTarget(jmsserver1mb)
```

Example 6-1 WLST Script to Create JMS System Resources

```
"""
This script starts an edit session, creates a JMS Server,
targets the jms server to the server WLST is connected to and creates
a JMS System module with a jms queue and connection factory. The
jms queues and topics are targeted using sub-deployments.
"""

import sys
from java.lang import System

print "@@@ Starting the script ..."

myJmsSystemResource = "CapiQueue-jms"
factoryName = "CConFac"
jmsServerName = "myJMSServer"
queueName = "CQueue"

url = sys.argv[1]
```

```
usr = sys.argv[2]
password = sys.argv[3]

connect(usr,password, url)
edit()
startEdit()

//Step 1
servermb=getMBean("Servers/examplesServer")
    if servermb is None:
        print '@@@ No server MBean found'

else:
    //Step 2
    jmsMySystemResource = create(myJmsSystemResource,"JMSSystemResource")

    //Step 3
    jmsMySystemResource.addTarget(servermb)

    //Step 4
    theJMSResource = jmsMySystemResource.getJMSResource()

    //Step 5
    connfact1 = theJMSResource.createConnectionFactory(factoryName)
    jmsqueue1 = theJMSResource.createQueue(queueName)

    //Step 6
    connfact1.setJNDIName(factoryName)
    jmsqueue1.setJNDIName(queueName)

    //Step 7
    jmsqueue1.setSubDeploymentName('DeployToJMSServer1')
    connfact1.setSubDeploymentName('DeployToJMSServer1')

    //Step 8
    jmsserver1mb = create(jmsServerName,'JMSServer')

    //Step 9
    jmsserver1mb.addTarget(servermb)

    //Step 10
    subDep1mb = jmsMySystemResource.createSubDeployment('DeployToJMSServer1')

    //Step 11
    subDep1mb.addTarget(jmsserver1mb)
.
.
.
```

How to Modify and Monitor JMS Servers and JMS System Module Resources

You can modify or monitor JMS objects and attributes by using the appropriate method available from the MBean.

- Modify JMS objects and attributes using the set, target, untarget, and delete methods.
- Monitor JMS runtime objects using get methods.

Example 7-2 shows a sample WLST script to modify JMS objects.

See Navigating MBeans (WLST Online) in *Understanding the WebLogic Scripting Tool*.

Example 6-2 WLST Script to Modify JMS Objects

```
.
.
print '@@@ delete system resource'
jmsMySystemResource = delete("CapiQueue-jms","JMSSystemResource")
print '@@@ delete server'
jmsserver1mb = delete(jmsServerName,'JMSServer')
.
.
.
```

Best Practices When Using WLST to Configure JMS Resources

Learn about best practices when using WLST to configure JMS servers and JMS system module resources.

- Trap for Null MBean objects (such as servers, JMS servers, modules) before trying to manipulate the MBean object.
- Use a meaningful name when providing a subdeployment name. For example, the subdeployment name *DeployToJMSServer1* tells you that all subdeployments with this name are deployed to JMSServer1.

7

Interoperating with Oracle AQ JMS

Oracle WebLogic Server applications interoperate with Oracle Streams Advanced Queuing (AQ) through the JMS API using either WebLogic Server resources (Web Apps, EJBs, MDBs) or stand-alone clients.
See Introduction to Oracle Database Advanced Queuing in *Oracle Streams Advanced Queuing User's Guide*.

> **Note:**
>
> AQ-JMS integration is not supported in a mixed or dynamic cluster. No exception is thrown when this is attempted.

This chapter includes the following sections:

- Overview
  AQ JMS uses a database connection and stores JMS messages in a database accessible to an entire WebLogic Server cluster. This connection enables the use of database features and tooling for data manipulation and backup. Your WebLogic Server installation includes all the necessary classes, and no additional files are required in the WebLogic Server `classpath` to interoperate with Oracle AQ JMS.
- Using AQ Destinations as Foreign Destinations
  AQ foreign destinations must be local to the server running the application or MDBs sending or receiving messages.
- Driver Support
- Transaction Support
- Oracle Real Application Clusters
- MBean and Console Support
  Use SQL scripts or other tools for AQ administration and monitoring, such as AQ queue table creation/removal, destination creation/removal, and statistics queries.
- Configuring WebLogic Server to Interoperate with AQ JMS
- Programming Considerations
- Advanced Topics
  Learn about advanced interoperability information when WebLogic Server applications interoperate with AQ JMS.

Overview

AQ JMS uses a database connection and stores JMS messages in a database accessible to an entire WebLogic Server cluster. This connection enables the use of database features and tooling for data manipulation and backup. Your WebLogic Server installation includes all the necessary classes, and no additional files are required in the WebLogic Server `classpath` to interoperate with Oracle AQ JMS.

WebLogic AQ JMS uses the WebLogic JMS foreign server framework to allow WebLogic Server applications and stand-alone clients to lookup AQ JMS connection factories and destinations using a standard WebLogic JNDI context, and to allow applications and clients to load and invoke AQ JMS using standard Jakarta EE APIs. The required references to the database, JDBC driver, and data source are configured as part of this framework.

For applications running within the WebLogic Server's JVM:

- A configured WebLogic data source references a particular JDBC driver, pools JDBC connections, and provides connectivity to the Oracle Database hosting AQ JMS.
- A configured WebLogic foreign server references the data source.
- Local JNDI names are defined for AQ JMS connection factories and destinations as part of the WebLogic JMS foreign server configuration. These JNDI names are configured to map to existing AQ connection factories and destinations.
- In turn, WebLogic Server applications, such as MDBs, reference the local JNDI names.

Using AQ Destinations as Foreign Destinations

AQ foreign destinations must be local to the server running the application or MDBs sending or receiving messages.

An application that is running on one WebLogic Server instance cannot look up and use an AQ JMS foreign server and data source that is registered on another WebLogic Server instance. WebLogic AQ JMS uses a data source or DB connection that does not support remote connectivity. An alternative is to use a messaging bridge between AQ destinations in one domain and applications or MDBs running in another domain. See WebLogic Messaging Bridge.

Driver Support

WebLogic AQ JMS requires the Oracle thin driver to communicate with the Oracle database. Oracle OCI JDBC Driver and non-Oracle JDBC Drivers are not supported. See Supported Configurations in *What's New in Oracle WebLogic Server*.

Transaction Support

WebLogic AQ JMS supports both Global XA (JTA) transactions and local JMS transacted session transactions.Global transactions require use of XA based connection factories, while local transactions may use non-XA based JMS connection factories:

- If you select a non-XA JDBC driver, then you can only use WebLogic AQ JMS in local transactions.
- If you select an XA JDBC driver, then you can use WebLogic AQ JMS in both local and global transactions.
- This release does not support non-XA JDBC driver data sources with any of the global transaction options such as Logging Last Resource (LLR), One-Phase Commit (JTS), and Emulated Two-Phase Commit. If `Supports Global Transactions` is selected, then WebLogic Server logs a warning message.
- Global transactions are supported only with an XA JDBC driver One-Phase commit optimization. If you use the same XA capable data source for both AQ JMS and JDBC operations, then the XA transactional behavior is equivalent to having two connections in a single data source that is treated as a single resource by the transaction manager.

Therefore, if the AQ JMS and JDBC operations are called under the same JTA transaction, and no other resources are involved in the transaction, the transaction uses One-Phase Commit optimization instead of Two-Phase Commit; otherwise read-only optimization is used.

See Understanding Transactions in *Developing JMS Applications for Oracle WebLogic Server*

Oracle Real Application Clusters

WebLogic AQ JMS supports Oracle Real Application Clusters (Oracle RAC) with the use of WebLogic Multi Data Sources to provide failover in an Oracle RAC environment.

See Using WebLogic Server with Oracle RAC in *Administering JDBC Data Sources for Oracle WebLogic Server*.

> **Note:**
>
> Oracle does not recommend configuring multi data sources for `Load balancing` with AQ JMS. AQ JMS and AQ usage scenarios have natural hot spots that can cause over synchronization when the load is spread among Oracle RAC instances. Under the right circumstances, it can cause significant performance degradation

MBean and Console Support

Use SQL scripts or other tools for AQ administration and monitoring, such as AQ queue table creation/removal, destination creation/removal, and statistics queries.

Except for purposes of interoperating with AQ JMS using a JMS Foreign Server, there are no AQ JMX specific MBeans and no support for configuring AQ JMS in the WebLogic Remote Console.

Configuring WebLogic Server to Interoperate with AQ JMS

You can configure AQ JMS queues and topics in an Oracle Database and configure a JMS foreign server in WebLogic Server so that applications can lookup AQ JMS connection factories and destinations in the WebLogic JNDI context.
After you have prepared your AQ JMS queues and topics, you can perform the remaining configuration tasks using either the Remote Console or the WLST command line interface.

The following sections describe the preferred method for configuring WebLogic Server to interoperate with AQ JMS:

- Configure Oracle AQ in the Database
- Configure WebLogic Server for AQ JMS
- Additional Configuration for Interoperation with Oracle 12c Database

Configure Oracle AQ in the Database

You might find it helpful to set up your AQ JMS queues and topics in your Oracle Database before you configure WebLogic Server to integrate with AQ JMS.

For more detailed information on using and configuring AQ, see Introduction to Oracle Database Advanced Queuing in *Oracle Streams Advanced Queuing User's Guide*.

The following sections describe one configuration method:

- Create Users and Grant Permissions
- Create AQ Queue Tables
- Create a JMS Queue or Topic
- Start the JMS Queue or Topic

Create Users and Grant Permissions

Create users in the database and grant them AQ JMS permissions. Use a database user with administrator privileges to perform the following task:

- Using the Oracle SQL*Plus environment, log in with an administrator login.

  ```
  connect / as sysdba;
  ```

- Create the JMS user schema. For the following example, the user name is *jmsuser* and the password is *jmsuserpwd*.

  ```
  Grant connect, resource TO jmsuser IDENTIFIED BY jmsuserpwd;
  ```

- Grant the AQ user role to *jmsuser*.

  ```
  Grant aq_user_role TO jmsuser;
  ```

- Grant execute privileges to AQ packages.

  ```
  Grant execute ON sys.dbms_aqadm TO jmsuser;

  Grant execute ON sys.dbms_aq TO jmsuser;

  Grant execute ON sys.dbms_aqin TO jmsuser;

  Grant execute ON sys.dbms_aqjms TO jmsuser;
  ```

Create AQ Queue Tables

Each JMS queue or topic for AQ JMS is backed by an AQ queue table. Each queue table serves as a repository for JMS messages. A JMS queue or topic (see Create AQ Queue Tables) is a logical reference to the underlying AQ queue table.

AQ queue tables are created within individual JMS user schemas and can be defined using Oracle SQL*PLUS. For example:

```
connect jmsuser / jmsuserpwd;
```

Configuring an AQ queue table requires a minimum of three parameters: the name of the queue table, the payload type, and a flag for whether the AQ queue table accepts multiple consumers. For example:

```
dbms_aqadm.create_queue_table(
     queue_table=>"myQueueTable",
     queue_payload_type=>'sys.aq$_jms_text_message',
     multiple_consumers=>false
);
```

Where:

- `queue_table`: The queue table name. Mixed case is supported in Oracle Database but the name must be enclosed in double quotes. Queue table names must not be longer than 24 characters.
- `queue_payload_type`: The message type. Use `sys.aq$_jms_message` to support all JMS message interface types.
- `multiple_consumers`: Set to `false` for queues; set to `true` for topics.

For more information on creating queue tables, see CREATE_QUEUE_TABLE Procedure in *Oracle Database PL/SQL Packages and Types Reference*.

Create a JMS Queue or Topic

AQ JMS queues are the JMS administrative resource for both queues and topics. After the AQ queue table is created, you can create an AQ JMS queue within individual JMS user schemas using Oracle SQL*PLUS. For example:

```
connect jmsuser/jmsuserpwd;
```

The PL/SQL procedure for creating a queue or topic has the following form:

```
dbms_aqadm.create_queue(
     queue_name=>'userQueue',
     queue_table=>'myQueueTable'
);
```

Where:

- `queue_name` is the user-defined name for the JMS queue.
- `queue_table` must point to an existing AQ queue table.

For more information about creating queue tables, see CREATE_QUEUE Procedure in *Oracle Database PL/SQL Packages and Types Reference*.

Start the JMS Queue or Topic

Before first use, an AQ JMS queue must be started. Using the JMS user schema, execute the following PL/SQL procedure where `queue_name` represents the AQ JMS queue name.

```
connect jmsuser / jmsuserpwd
dbms_aqadm.start_queue(queue_name=>'userQueue'
```

For more information about starting queues, see START_QUEUE Procedure in *Oracle Database PL/SQL Packages and Types Reference*.

Configure WebLogic Server for AQ JMS

The following sections provide information about how to configure WebLogic Server to interoperate with AQ JMS:

- Configure a WebLogic Data Source for AQ JMS
- Configure a JMS System Module
- Configure a JMS Foreign Server
- Configure JMS Foreign Server Connection Factories
- Configure AQ JMS Foreign Server Destinations

Configure a WebLogic Data Source for AQ JMS

WebLogic Server applications (such as MDBs, EJBs, and Web apps) that use, AQ JMS configure a data source for the Oracle Database that provides the AQ JMS service. In most situations, this data source is dedicated to AQ JMS usage because it uses the JMS user and password to connect to the schema in the database. It does support multiple queues and topics if they are created in the schema used in the database connection. You can use a generic or Active GridLink (AGL) data source:

- Configuring a Generic Data Source for AQ JMS
- Configuring an AGL Data Source for AQ JMS

Configuring a Generic Data Source for AQ JMS

When configuring your data source:

- Select the appropriate Oracle Thin Driver.
- Select the driver type based on the type of transactions required for AQ JMS:
  - Select a non-XA based JDBC driver for use with AQ JMS in local transactions.
  - Select a XA based JDBC driver for use with AQ JMS in either in global transactions or in local transactions.
- When configuring a data source for non-XA drivers, do not select the `Supports Global Transactions` option. This release does not support non-XA JDBC driver data sources with any of the global transaction options such as LLR, JTS, and Emulate Two-Phase Commit. If the global transaction option is selected, the server instance logs a warning message. Global transactions are supported with XA-based JDBC drivers.
- Configure the database user name and password in the data source connection pool configuration. Identity-based connection pools are not supported.

See:

- Configuring JDBC Data Sources in *Administering JDBC Data Sources for Oracle WebLogic Server*.
- Create a Generic Data Source in the *Oracle WebLogic Remote Console Online Help*.

Configuring an AGL Data Source for AQ JMS

When configuring your data source:

- Select the Oracle Thin XA Driver for Connections.
- Configure the database user name and password in the data source connection pool configuration.

See:

- Using Active GridLink Data Sources in *Administering JDBC Data Sources for Oracle WebLogic Server*.
- Create an Active GridLink Data Source in the *Oracle WebLogic Remote Console Online Help*.

Configure a JMS System Module

Configure a dedicated JMS system module to host a JMS foreign server for AQ resources. Target the module at the WebLogic Server instances or the cluster that needs to host the foreign JNDI names. See:

- Overview of JMS Modules.
- Configure Resources for JMS System Modules in the *Oracle WebLogic Remote Console Online Help*.

Configure a JMS Foreign Server

In your JMS foreign server configuration:

- Specify `oracle.jms.AQjmsInitialContextFactory` as the JNDI initial context factory.
- Configure the JDBC data sources needed for your application environment.

See:

- Configuring Foreign Server Resources to Access Third-Party JMS Providers
- Configure Resources for JMS System Modules in the *Oracle WebLogic Remote Console Online Help*.
- Reference a Data Source

Reference a Data Source

Specify the `datasource` JNDI property which is the JNDI location of a locally bound WLS data source.

For example:

```
<foreign-server>
<initial-context-factory>oracle.jms.AQjmsInitialContextFactory</initial-context-factory>
<jndi-property>
<key>datasource</key>
<value>jdbc/aqjmsds</value>
</jndi-property>
</foreign-server>
```

The value of the `datasource` JNDI property is the name of the data source configured to access the AQ JMS Oracle Database. No other configuration information, such as security principal or credentials, is required. However, if you provide a database url, then `java.naming.security.principal` and `java.naming.security.credentials` are required.

See Configure a WebLogic Data Source for AQ JMS.

Configure JMS Foreign Server Connection Factories

After you have created a JMS foreign server, you can create JNDI mappings for the AQ JMS connection factories in the WebLogic Server JNDI tree. Unlike destinations, AQ JMS does not require connection factories to be redefined in the Oracle Database. Instead, a predefined JNDI name is specified when identifying the remote JNDI name for a connection factory. The remote JNDI name for the AQ JMS connection factory is one of those in Table 1.

Table 7-1 Remote JNDI names for AQ JMS Connection Factories

<AQ JMS Prefix Value>	JMS Interface
QueueConnectionFactory	javax.jms.QueueConnectionFactory
TopicConnectionFactory	javax.jms.TopicConnectionFactory
ConnectionFactory	javax.jms.ConnectionFactory
XAQueueConnectionFactory	javax.jms.XAQueueConnectionFactory
XATopicConnectionFactory	javax.jms.XATopicConnectionFactory
XAConnectionFactory	javax.jms.XAConnectionFactory

For example, consider two connection factories configured for an AQ JMS Foreign Server:

Table 7-2 AQ JMS Foreign Server Example Connection Factories

Local JNDI Name	RemoteJNDI Name
jms/aq/myCF	ConnectionFactory
aqjms/orderXaTopicFactory	XATopicConnectionFactory

When a WebLogic application looks up a JMS factory at `jms/aq/myCF`, the application gets the AQ JMS object that implements the JMS `javax.jms.ConnectionFactory` interface. When a WebLogic application looks up a JMS factory at `aq/orderXaTopicFactory`, the application gets the AQ JMS object that implements the JMS `javax.jms.XATopicConnectionFactory` interface.

To configure a AQ JMS foreign server connection factory, you need to:

- Specify Local and Remote JNDI names
  - The local JNDI name is the name that WebLogic uses to bind the connection factory into the WebLogic JNDI tree. The local JNDI name must be unique so that it doesn't conflict with an other JNDI name advertised on the local WebLogic Server.
  - The Remote JNDI name is the name that WebLogic passes to AQ JMS to lookup AQ JMS connection factories.

  When configuring AQ JMS for use in global transactions, use an XA based connection factory; otherwise configure a non-XA based connection factory.
- No other configuration parameters are required.

See:

- Creating Foreign Connection Factory Resources
- Configure Resources for JMS System Modules in the *Oracle WebLogic Remote Console Online Help*.

Configure AQ JMS Foreign Server Destinations

When configuring an AQ JMS foreign destination, you need to configure the following:

- Local JNDI name : The name that WLS uses to bind the destination into the WebLogic JNDI tree. The local JNDI name must be unique so that it doesn't conflict with any other JNDI names advertised on the local WebLogic Server instance.

- Remote JNDI name : The name that WLS passes to AQ JMS to do a lookup. AQ JMS requires the Remote JNDI name to be in the following syntax:
  - If the destination is a queue, then the remote JNDI name must be `Queues/<queue name>`.
  - If the destination is a topic, then the remote JNDI name must be `Topics/<topic name>`

Similar to connection factories, AQ JMS destinations require a remote JNDI name with a prefix to identify the JMS object type. There are two values for destinations:

Table 7-3 AQ JMS Prefix Value of the JMS Interface

AQ JMS Prefix Value	JMS Interface
`Queues`	`Javax.jms.Queue`
`Topics`	`javax.jms.Topic`

Unlike AQ JMS connection factory JNDI names, the value for the destination name represents the AQ JMS destination defined in the database. See Create a JMS Queue or Topic. For example, consider the two destinations configured for an AQ JMS Foreign Server in the following table:

Table 7-4 Example AQ JMS Foreign Server Destinations

Local JNDI Name	Remote JNDI Name
`jms/myQueue`	`Queues/userQueue`
`AqTopic`	`Topics/myTopic`

A WebLogic application looking up the location `jms/myQueue` references the AQ JMS queue defined by `userQueue`. Looking up the location `AqTopic` references the AQ JMS topic defined by `myTopic`.

See:

- Creating Foreign Connection Factory Resources
- Configure Resources for JMS System Modules in the *Oracle WebLogic Remote Console Online Help*.

Additional Configuration for Interoperation with Oracle 12c Database

The following section provides additional configuration that may be required when interoperating AQ JMS on a Oracle 12c Database:

- The AQ asynchronous notification feature is enabled by default with sharded queues in Oracle 12c database.

  When using the AQ sharded queues, an asynchronous receive from a Message Listener or from a Message Driven Bean (MDB) may create more JDBC connections internally based on the load. If the number of JDBC connections is restricted in the system, the client application should disable the asynchronous notification by setting the value of the system property `oracle.jms.useJmsNotification` to `false`.

  For transactional MDBs or when `-Dweblogic.mdb.message.MinimizeAQSessions` is set to true, the MDB container will internally use synchronous receivers, not an asynchronous consumer.

- To use the notification service (`-Doracle.jms.useJmsNotification=true`), set the database service name in place of SID for the JDBC URL. For example:

  `jdbc:oracle:thin:@//`*DB_HOST*`:`*DB_PORT*`/`*Database ServiceName*

- Specify a local listener by setting the `init` parameter in the `init.ora` file and restart the database. For example:

  `LOCAL_LISTENER="(ADDRESS=(PROTOCOL=tcp)(host=`*DB_HOST*`)(port=`*DB_PORT*`))`.

- Set the WebLogic Server properties to minimize the number of JMS sessions used and to control the MDB polling interval: See Settings for Message Driven Beans to Interoperate with AQ JMS

> **Note:**
>
> As of WebLogic Server 12.2.1.2, the WebLogic Server Java system properties `weblogic.mdb.message.MimizeAQSessions` and `weblogic.ejb.container.MDBDestinationPollIntervalMillis` are obsolete. Use the corresponding activation-config-property instead. The two activation-config-property overrides the respective Java system properties.

Programming Considerations

Learn about some of the programming considerations for managing WebLogic AQ JMS. The following sections include information about advanced WebLogic AQ JMS topics:

- Settings for Message Driven Beans to Interoperate with AQ JMS
- Scalability for Clustered WebLogic MDBs Listening on AQ Topics
- AQ JMS Extensions
- Resource References
- JDBC Connection Utilization
- Oracle RAC Support
- Debugging
- Performance Considerations

Settings for Message Driven Beans to Interoperate with AQ JMS

Message Driven Beans (MDBs) interoperate with AQ JMS by using a configured foreign server. See Overview.

The following are the considerations for MDBs when interoperating with AQ JMS:

- Set the following WebLogic Server properties to minimize the number of JMS sessions used and to control the MDB polling interval:
- `–Dweblogic.mdb.message.MinimizeAQSessions=true`

  This property reduces the number of AQ JMS sessions in the MDB layer and there by reduce the number of JDBC connections held by the MDB. Also, the value true for this property will be effective only when `weblogic.ejb.container.MDBDestinationPollIntervalMillis` is set to a value more than 5000 milliseconds.

```
-Dweblogic.ejb.container.MDBDestinationPollIntervalMillis=6000
```

- The message driven parameters `initial-context-factory` and `provider-url` are not supported as these parameters are supplied as part of the JMS Foreign Server configuration
- The destination type for the MDB destination in the `ejb-jar.xml` file should be configured to either: `javax.jms.Queue` or `javax.jms.Topic`
- Additional MDB configuration is required to enable container managed transactions, durable topic subscriptions, and other MDB features.

See Understanding Message-driven Beans in *Developing Message-Driven Beans for Oracle WebLogic Server*.

Scalability for Clustered WebLogic MDBs Listening on AQ Topics

Use a shared subscription to implement a one-copy-per-application messaging strategy when a clustered MDB listens to an AQ topic.

The subscribers that share a subscription need to:

- Be on different VMs.
- Use the same data source (or a data source that uses the same database user-name and password).
- Use the same subscription name.

The database username acts as the client ID for the subscription. When the client ID is the same for a same-named subscription, the subscription in AQ is shared. For more information about shared subscriptions, see Advanced Messaging Features for High Availability in *Developing JMS Applications for Oracle WebLogic Server*.

AQ JMS Extensions

AQ JMS extension APIs are supported by AQ JMS-specific classes. You can invoke the AQ JMS extensions, after casting the standard JMS objects (such as connection factories and destinations) to proprietary AQ JMS classes. For example:

```
. . .
import oracle.jms.AQjmsFactory;
. . .
ConnectionFactory myCF = (ConnectionFactory) jndiCtx.lookup("aqjms/testCF");
AQjmsFactory myCF = (AQjmsFactory) myCF;
myCF.someProprietaryAQJMSmethod(..);
. . .
```

When you use resource references for an AQ JMS connection factory, WebLogic Server wraps the underlying AQ JMS connection factory with a wrapper object. This wrapper object implements the JMS standard API, but it cannot cast it to an AQ JMS class that provides AQ JMS extension APIs. For example:

```
. . .
// Implements wrapping and cannot cast to AQ JMS
<resource-ref>
    <res-ref-name>aqjms/testCF</res-ref-name>
    <res-type>javax.jms. ConnectionFactory</res-type>
    <res-auth>Application</res-auth>
  </resource-ref>
. . .
```

To avoid the wrapping, users can specify the `java.lang.Object` as the resource type of the resource reference instead of `javax.jms.XXXConnectionFactory` in the deployment descriptor. This limitation is specific to AQ JMS, as resource references support only extensions that are exposed using Jakarta interfaces. For example:

```
. . .
// Use for AQ JMS extensions
<resource-ref>
    <res-ref-name>aqjms/testCF</res-ref-name>
    <res-type>java.lang.Object</res-type>
    <res-auth>Application</res-auth>
  </resource-ref>
. . .
```

AQ JMS does not define Jakarta interfaces for its extensions. With AQ JMS, avoiding wrapping does not disable automatic JTA transaction enlistment, nor does it prevent pooling, because AQ JMS obtains these capabilities implicitly through its embedded use of WebLogic data sources.

- Using AdtMessage

Using AdtMessage

An `AdtMessage` is a special type of AQ JMS extension that supports Abstract Data Types (ADTs). ADTs consists of a data structure and subprograms that manipulate data in an Oracle Database.

> **Note:**
>
> This is not supported with Message-Driven Beans (MDBs).

See:

- Using Oracle Java Message Service (OJMS) to Access Oracle Streams Advanced Queuing and Object Type Support in *Oracle Streams Advanced Queuing User's Guide*
- Data Abstraction in *Oracle Database PL/SQL Language Reference*

Resource References

If you use the resource references and the resource type is `javax.jms.XXXConnectionFactory`, then the WebLogic wraps the AQ JMS objects passed to a user application. If you also use the AQ JMS extension APIs, then they must be unwrapped as described in AQ JMS Extensions.

WebLogic resource reference wrappers do not automatically pool AQ JMS connections. Instead, AQ JMS server-side integration depends on data source connection pooling to mitigate the overhead of opening and closing JMS connections and sessions. WebLogic resource references disable pooling because the AQ JMS provider JMS connection factory is always pre-configured with a client identifier, which in turn, causes WebLogic resource references to disable its pooling feature.

JDBC Connection Utilization

An AQ JMS session holds a JDBC connection until the JMS session is closed, regardless of whether the connection uses a data source or a JDBC URL. Oracle recommends that you close an AQ JMS session if the session becomes idle for an extended period of time. Closing the JMS session releases the JDBC connection back to the WebLogic data source pool or releases the database and network resources for a JDBC URL.

Oracle RAC Support

The following section provides information about limitations in Oracle RAC environments:

- Oracle RAC environments require the configuration of WebLogic Multi data sources to provide AQ JMS Oracle RAC failover. See Using WebLogic Server with Oracle RAC in *Administering JDBC Data Sources for Oracle WebLogic Server*.
- Oracle RAC failover is not supported when using a WebLogic AQ JMS stand alone client for this release.

Debugging

To use AQ JMS tracing and debugging, set the following system property: `oracle.jms.traceLevel`.

The value of this property is an integer ranging from 1 to 6 where a setting of 6 provides the finest level of detail. The trace output is directed to the standard output of the running JVM.

Performance Considerations

In releases prior to Oracle RDBMS 11.2.0.2, statistics on the queue table are locked by default which causes a full table scan for each dequeue operation. To work around this issue, unlock the queue tables and collect the statistics. For example:

```
exec DBMS_STATS.UNLOCK_TABLE_STATS ('<schema>','<queue table>');

exec DBMS_STATS.gather_table_stats('<schema>','<queue table>');

exec DBMS_STATS.LOCK_TABLE_STATS ('<schema>','<queue table>');
```

Advanced Topics

Learn about advanced interoperability information when WebLogic Server applications interoperate with AQ JMS.

This section includes the following topics:

- Security Considerations
- WebLogic Messaging Bridge
- Standalone WebLogic AQ JMS Clients

Security Considerations

Stand alone clients and server-side applications have different security semantics and configuration. If security is a concern, read this section carefully and also reference the

WebLogic lock-down document for general information about how to secure a WebLogic Server or Cluster (see Lock Down WebLogic Server in *Securing a Production Environment for Oracle WebLogic Server*). The following sections outlines security considerations for this release:

- Configuring AQ Destination Security
- Access to JNDI Advertised Destinations and Connection Factories
- Controlling Access to Destinations that are Looked Up using the JMS API

Configuring AQ Destination Security

ENQUEUE and/or DEQUEUE permission must be configured for the database user in AQ to allow destination lookups as well as to allow enqueues and dequeues.

The following usernames must be given enqueue and/or dequeue permission:

- For stand-alone clients:
  - The configured JMS Foreign Server username, as specified using the `java.naming.security.principal` property.
  - For Java code that passes a `username` using the JMS ConnectionFactory API `createConnection()` method, this `username` requires permission.
- For server-side applications:
  - The `Database User Name` is configured on the WebLogic Data Source.
  - Do not give permission for a `username` specified for JDBC Data Source clients that pass a `username` using the JMS ConnectionFactory API `createConnection()` method: this `username` is a WebLogic `username`, not a database `username`.

To understand which JDBC connection credentials and permissions that are used for AQ lookups, enqueues, and dequeues, see Queue Security and Access Control in *Oracle Streams Advanced Queuing User's Guide*.

> **Note:**
>
> A permission failure while looking up a destination will manifest as a "name not found" exception thrown back to application caller, not a security exception.

Access to JNDI Advertised Destinations and Connection Factories

As described earlier, local JNDI names for connection factories and destinations must be configured as part of the JMS Foreign Server configuration task. You can optionally configure security policies on these JNDI names, so access checks occur during JNDI lookup based on the current WebLogic credentials. The current WebLogic credentials depend on the client type.

Once an application's WebLogic JNDI lookup security policy credential check passes for a destination, a JMS Foreign Server destination automatically looks up the destination resources in Oracle AQ using a JDBC connection.

For stand-alone clients, the credential used for the second part of a destination lookup process are based on the `username` and `password` that is configured on the JMS Foreign Server.

For server-side application JDBC Data Source clients, the credential used for this second destination lookup is based on the database `username` and `password` configured as part of the data source. Note that the credential used to gain access to this data source is the current WebLogic credential. It is possible to configure a WebLogic security policies on the data source. The WebLogic data source Identity Based Connection Pooling feature is not supported for this purpose.

As previously mentioned, the database credential must have AQ JMS enqueue or dequeue permission on a destination to be able to successfully look-up the destination. See Configuring AQ Destination Security.

Controlling Access to Destinations that are Looked Up using the JMS API

The JMS `QueueSession` and `TopicSession` APIs provide an alternative to JNDI for looking up destinations, named `createQueue()` and `createTopic()` respectively. See How to Lookup a Destination in *Developing JMS Applications for Oracle WebLogic Server*.

The `createQueue()` and `createTopic()` calls use the database credential associated with the JMS connection. The following sections describe how to set this credential.

- Additional Security Configuration for Standalone Clients

Additional Security Configuration for Standalone Clients

The following section is security configuration information for standalone clients:

- Network communication from a client into WebLogic occurs when establishing a JNDI initial context and when performing any subsequent JNDI lookups. To ensure secure communication and avoid plain text on the wire, use an SSL capable protocol (such as t3s or HTTPS). The credentials used for WebLogic login, as well as the JMS Foreign Server credentials that are configured for database login, are passed plain-text on the wire unless SSL is configured.
- Network communication is direct from the client to the database when communicating with AQ. This communication is controlled by the JDBC URL configuration, and is in plain text unless the JDBC URL is configured to use SSL. Stand-alone clients communicate directly with the database over a database connection when using the AQ JMS APIs, their JMS requests do not route through a WebLogic server.
- WebLogic Server user name and password: The network login from a client into WebLogic is performed as part of establishing the JNDI initial context. The user name and password properties that are optionally supplied when creating the context become the WebLogic identity (the property names are `Context.SECURITY_PRINCIPAL = "java.naming.security.principal"` and `Context.SECURITY_CREDENTIALS = "java.naming.security.credentials"` respectively). This becomes the credential that is checked for subsequent JNDI lookups. The credential is also implicitly associated with current thread, and so becomes the credential used for subsequent WebLogic operations on the same thread, but this is not the credential used for AQ JMS operations.
- The `javax.jms.ConnectionFactory createConnection()` method has an optional user name and password. For stand-alone clients, these override the context credentials that were configured as part of the JMS foreign server configuration. AQ JMS creates a database connection with the specified user identity. If `createConnection()` is called without a user-name and password, then the database connection is created using the user-name and password that was configured as part of the JMS foreign server configuration.
- Do not include a user name/password directly in the JDBC URL. Instead use the JMS Foreign Server user name and password.

- Do not configure a user name and password on the JMS Foreign Server connection factory. The resulting behavior is unsupported.
- Additional Security Configurations for Server-Side Applications

Additional Security Configurations for Server-Side Applications

The following section provides security configuration information for server-side applications.

- Do not configure a `java.naming.security.principal` or a credential on the JMS foreign server unless the same JMS foreign server is also being used to support stand-alone clients.
- Do not configure a user-name and password on the JMS foreign server connection factory. The resulting behavior is unsupported.
- Network communication from the server to the database (server-side applications) is controlled by data source configuration, and is in plain text unless the data source is configured to use SSL.
- The `javax.jms.ConnectionFactory createConnection()` method has an optional user name and password. For server-side JMS AQ applications, the method assumes the user-name is for a WebLogic user and authenticates it with the WebLogic server. This behavior deviates from other kinds of JMS AQ clients, where the user-name is instead treated as a database user. When configured with a WebLogic data source, AQ JMS delegates the authentication to the WebLogic data source and AQ JMS inherits the WebLogic user semantics.
- When an AQ JMS foreign server is configured with a WebLogic data source, the data source is exposed to general-purpose JDBC usage. Oracle recommends that you secure the data source as described in Using Roles and Policies to Secure JDBC Data Sources in *Administering JDBC Data Sources for Oracle WebLogic Server*.
- WebLogic Server user name and password: WebLogic credentials are checked when accessing secured names in JNDI, and accessing secured data sources. Server side applications automatically assume the same WebLogic credentials as the caller that called the application, or, in the case of MDBs, this credential can be configured as part of the MDB configuration.
- The WebLogic data source identity based connection pooling feature is not supported.
- JNDI context credentials: Specifying credentials as part of setting up a JNDI context within a server-side application is usually not necessary, and is not typically recommended. This creates a new credential that overrides the application's current credentials. In other words, the user-name and password properties that are optionally supplied when creating the context become the WebLogic identity and replace any current identity (the property names are `Context.SECURITY_PRINCIPAL = "java.naming.security.principal"` and `Context.SECURITY_CREDENTIALS = "java.naming.security.credentials"` respectively). The optional new credential is implicitly associated with the current thread, and so becomes the credential used for subsequent WebLogic operations on the same thread, such as JNDI lookups. The new credential is not the credential used for AQ JMS operations.

WebLogic Messaging Bridge

A WebLogic Messaging Bridge communicates with the configured source and target bridge destinations. For each mapping of a source destination to a target destination, you must configure a messaging bridge instance. Each messaging bridge instance defines the source and target destination for the mapping, a message filtering selector, a QOS, transaction semantics, and various reconnection parameters.

If you have AQ foreign destinations that are not local to the server running the application or MDBs sending and receiving messages, you must configure a messaging bridge instance on the server that is local to the AQ foreign destinations. A local database connection is used in the process of sending and receiving messages from AQ destinations.

For more information on the WebLogic Messaging Bridge, see Understanding the Messaging Bridge in *Administering the WebLogic Messaging Bridge for Oracle WebLogic Server*.

- Create a Messaging Bridge Instance

Create a Messaging Bridge Instance

The following are the major steps in creating a messaging bridge between AQ destinations are configured as foreign destinations in one domain and applications or MDBs running in another domain:

1. Create the bridge instance on the server where AQ destinations configured as foreign destinations.
2. Create source and target bridge destinations.

   Select **Other JMS** in the default Messaging Provider drop down when a Foreign AQ JMS destination is specified for a source or target destination.
3. Deploy a resource adapter.
4. Create a messaging bridge instance.

   The Messaging Bridge `Exactly-Once` quality of service requires a data source configured with the XA based JDBC driver and must use an AQ JMS connection factory that implements an XA-JMS connection factory interface. See Configure a WebLogic Data Source for AQ JMS and Configure JMS Foreign Server Connection Factories.
5. Target the messaging bridge.

The WebLogic Remote Console helps you in create a messaging bridge by deploying an appropriate resource adapter and setting the values of some attributes. Consider changing messaging bridge settings to better suit your environment. See Create a Messaging Bridge Instance in the *Oracle WebLogic Remote Console Online Help*.

Standalone WebLogic AQ JMS Clients

You can create WebLogic AQ JMS standalone clients that can look up AQ JMS connection factories and destinations defined by a JMS foreign server using a JDBC URL. The client must have the following jars on the client-side classpath: `$MW_HOME/oracle_common/modules/oracle.jdbc/aqapi.jar`, `$MW_HOME/oracle_common/modules/features/com.oracle.db.jdbc7-no-dms.jar`, and one of the following WebLogic client jars: `wlthint3client.jar` or `weblogic.jar`.

For applications running outside the WebLogic Server's JVM:

- A configured WebLogic JMS foreign server references the database's URL, as well as other JDBC driver configurations. See Configure a Foreign Server using a Database's JDBC URL.
- Local JNDI names are defined for AQ JMS connection factories and destinations as part of the WebLogic JMS Foreign Server configuration. These JNDI names are configured to map to existing AQ connection factories and destinations.

- Standalone clients reference local JNDI names. Unlike applications that run on WebLogic Server, standalone clients need to ensure that the driver and AQ client are on the classpath.
- Configure a Foreign Server using the Database's JDBC URL
- Limitations when using Standalone WebLogic AQ JMS Clients

Configure a Foreign Server using the Database's JDBC URL

Specify the `db_url`, `java.naming.security.principal` JNDI properties and a password in `jndi-properties-credentials`.

For example:

```
<foreign-server>

<initial-context-factory>oracle.jms.AQjmsInitialContextFactory</initial-context-factory>

<jndi-properties-credential-encrypted>{3DES}g8yFFu1AhP8=</jndi-properties-credential-
encrypted>

<jndi-property>
<key>java.naming.security.principal</key>
<value>j2ee</value>
</jndi-property>

<jndi-property>
<key>db_url</key>
<value>jdbc:oracle:thin:@{hostname}:{port}:{sid}</value>
</jndi-property>

</foreign-server>
```

- The value of `db_url` JNDI property is the JDBC URL used to connect to the AQ JMS Oracle Database.
- The value of the `java.naming.security.principal` is the database user-name that AQ JMS uses to connect to the database.
- `jndi-properties-credentials` contains the database password.

No other configuration properties are required.

Limitations when using Standalone WebLogic AQ JMS Clients

The following are limitations to consider when creating and using standalone WebLogic JMS clients. This release does not support:

- Use of a WebLogic AQ JMS standalone client to automatically participate in global transactions managed by WLS.
- Connection pooling for WebLogic AQ JMS standalone clients.
- Looking up JMS objects defined by an AQ JMS foreign server using a data source.

8

Monitoring JMS Statistics and Managing Messages

You can monitor and manage JMS statistics in Oracle WebLogic Server. You can create, collect, analyze, archive, and access diagnostic data generated by a running server and the applications deployed within its containers.
For WebLogic JMS, you can use the enhanced runtime statistics to monitor the JMS servers and destination resources in your WebLogic domain to see if there is a problem. If there is a problem, you can use profiling to determine which application is the source of the problem. Once you've narrowed it down to the application, you can then use JMS debugging features to find the problem within the application.

For more information on configuring JMS diagnostic notifications, debugging options, message life cycle logging, and controlling message operations on JMS destinations, see Troubleshooting WebLogic JMS.

Message administration tools in this release enhance your ability to view and browse *all* messages, and to manipulate *most* messages in a running JMS Server, using either the WebLogic Remote Console or through new public runtime APIs. These message management enhancements include message browsing (for sorting), message manipulation (such as create, move, and delete), message import and export, as well as transaction management, durable subscriber management, and JMS client connection management.

For more information about the WebLogic Diagnostic Service, see *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.

This chapter includes the following sections:

- Monitoring JMS Statistics
- Managing JMS Messages
  WebLogic JMS message monitoring and management features let you create new messages, delete selected messages, move messages to another queue, export message contents to another file, import message contents from another file, or delete all the messages from a queue.

Monitoring JMS Statistics

Once WebLogic JMS has been configured, applications can begin sending and receiving messages through the JMS API.
See Developing a Basic JMS Application in *Developing JMS Applications for Oracle WebLogic Server*.

You can monitor statistics for the following JMS resources: JMS servers, connections, queue and topic destinations, JMS server session pools, pooled connections, active sessions, message producers, message consumers, and durable subscriptions on JMS topics.

JMS statistics continue to increment as long as the server is running. Statistics are reset only when the server is restarted.

- Monitoring JMS Servers

- Monitoring Queues
- Monitoring Topics
- Monitoring Durable Subscribers for Topics
- Monitoring Uniform Distributed Queues
- Monitoring Uniform Distributed Topics
- Monitoring Pooled JMS Connections

Monitoring JMS Servers

You can monitor statistics on active JMS servers defined in your domain using the WebLogic Remote Console or through the `JMSServerRuntimeMBean`. JMS servers act as management containers for JMS queue and topic resources within JMS modules that are specifically targeted to JMS servers.

To monitor JMS servers using the Remote Console, in the **Monitoring Tree**, go to **Services**, then **Messaging**, then **JMS Servers**. Select the JMS Server for which you want to see statistics.

When monitoring JMS servers with the WebLogic Remote Console, you can also monitor statistics for active destinations, transactions, connections, and session pools.

- Monitor Cluster Targeted JMS Servers
- Monitoring Active JMS Destinations
- Monitoring Active JMS Transactions
- Monitoring Active JMS Connections, Sessions, Consumers, and Producers
- Monitoring Active JMS Session Pools

Monitor Cluster Targeted JMS Servers

Cluster targeted JMS services such as JMS Servers, SAF agents, path service, and persistent stores have multiple instances of the associated RuntimeMbeans to monitor the statistics for each respective instance in a dynamic cluster. The runtime MBeans for a JMS server and persistent store are automatically named according to the corresponding server instance name using the following pattern:

configured_JMS_service_Artifact_name@server-name

server-name is the configured server name concatenated with dynamic server instance number.

Monitoring Active JMS Destinations

You can monitor statistics on all the active destinations currently targeted to a JMS server. JMS destinations identify queue or topic destination types within JMS modules that are specifically targeted to JMS servers.

In the Remote Console, go to the **Monitoring Tree**: **Services**: **Messaging**: **JMS Servers**. Select the JMS Server for which you want to see the active destinations.

Monitoring Active JMS Transactions

You can monitor active transactions running on a JMS server.

In the Remote Console, go to the **Monitoring Tree**: **Services**: **Messaging**: **JMS Servers**. Select the JMS Server for which you want to see the active transactions.

Monitoring Active JMS Connections, Sessions, Consumers, and Producers

You can monitor statistics on all the active JMS connections to a JMS server. A JMS connection is an open communication channel to the messaging system.

In the Remote Console, go to the **Monitoring Tree**: **Services**: **Messaging**: **JMS Servers**. Select the JMS Server for which you want to see the active JMS server connections.

Using the JMS server's monitoring page, you can also monitor statistics on all the active JMS sessions, consumers, and producers on your server. A session defines a serial order for both the messages produced and the messages consumed, and can create multiple message producers and message consumers. The same thread can be used for producing and consuming messages.

Monitoring Active JMS Session Pools

You can monitor statistics on all the active JMS session pools defined for a JMS server. Session pools enable an application to process messages concurrently.

In the Remote Console, in the **Monitoring Tree**, go to **Services**: **Messaging**: **JMS Servers**: *myJMSServer*. Select **Session Pool Runtimes** to monitor the runtime statistics provided for active JMS session pools.

Monitoring Queues

You can monitor statistics on queue resources in JMS modules using the WebLogic Remote Console or through the `JMSDestinationRuntimeMBean`. A JMS queue defines a point-to-point destination type for a JMS server. Queues are used for synchronous peer communications. A message delivered to a queue will be distributed to one consumer.

In the Remote Console, in the **Monitoring Tree**, go to **Services**: **Messaging**: **JMS Servers**: *myJMSServer*. Select **Destinations**: *myDestinationResource* to monitor queue resources.

You can also use the WebLogic Remote Console to manage messages on queues, as described in Managing JMS Messages.

Monitoring Topics

You can monitor statistics on topic resources in JMS modules using the WebLogic Remote Console or through the `JMSDestinationRuntimeMBean`. A JMS topic identifies a publish/subscribe destination type for a JMS server. Topics are used for asynchronous peer communications. A message delivered to a topic will be distributed to all topic consumers.

In the Remote Console, in the **Monitoring Tree**, go to **Services**: **Messaging**: **JMS Servers**: *myJMSServer*. Select **Destinations**: *myDestinationResource* to monitor topic resources.

Monitoring Durable Subscribers for Topics

You can monitor statistics on all the durable subscribers that are running on your JMS topics using the WebLogic Remote Console or through the `JMSDurableSubscriberRuntimeMBean`. Durable subscribers allow you to assign a name to a topic subscriber and associate it with a user or application. WebLogic stores durable subscribers in a persistent file-base store or

JDBC-accessible database until the message has been delivered to the subscribers or has expired, even if those subscribers are not active at the time that the message is delivered.

You can manage durable subscribers running on topics, as described in Managing JMS Messages.

Monitoring Uniform Distributed Queues

You can monitor statistics on uniform distributed queue resources in JMS modules using the WebLogic Remote Console or through the `JMSDestinationRuntimeMBean`. A distributed queue resource is a single set of queues that are accessible as a single, logical destination to a client (for example, a distributed topic has its own JNDI name). The members of the unit are usually distributed across multiple servers within a cluster, with each member belonging to a separate JMS server.

In the Remote Console, in the **Monitoring Tree**, go to **Services**: **Messaging**: **JMS Servers**: *myJMSServer*. Select **Destinations**: *myDestinationResource* to monitor uniform distributed queue resources.

You can also use the WebLogic Remote Console to manage messages on distributed queues, as described in Managing JMS Messages.

Monitoring Uniform Distributed Topics

You can monitor statistics on uniform distributed topic resources in JMS modules using the WebLogic Remote Console or through the `JMSDestinationRuntimeMBean`. A distributed topic resource is a single set of topics that is accessible as a single, logical destination to a client (for example, a distributed topic has its own JNDI name). The members of the unit are usually distributed across multiple servers within a cluster, with each member belonging to a separate JMS server.

In the Remote Console, in the **Monitoring Tree**, go to **Services**: **Messaging**: **JMS Servers**: *myJMSServer*. Select **Destinations**: *myDestinationResource* to monitor uniform distributed topic resources.

Monitoring Pooled JMS Connections

You can monitor statistics on all the active pooled JMS connections on your server. A pooled JMS connection is a session pool used by EJBs and servlets that use a resource-reference element in their EJB or servlet deployment descriptor to define their JMS connection factories.

Managing JMS Messages

WebLogic JMS message monitoring and management features let you create new messages, delete selected messages, move messages to another queue, export message contents to another file, import message contents from another file, or delete all the messages from a queue.

- JMS Message Management Using Jakarta APIs
- Managing Transactions
- Managing Durable Topic Subscribers

JMS Message Management Using Jakarta APIs

WebLogic Java Management Extensions (JMX) enables you to access the `JMSDestinationRuntimeMBean` and `JMSDurableSubscriberRuntimeMBean` to manage messages on JMS queues and topic durable subscribers. See Accessing WebLogic Server MBeans with JMX in *Developing Custom Management Utilities Using JMX for Oracle WebLogic Server*.

In WebLogic JMS, there are various states for messages. You can use these states to help manage your messages as described in the following sections. For information on valid message states, see weblogic.jms.extensions.JMSMessageInfo in *Java API Reference for Oracle WebLogic Server*.

Managing Transactions

When a message is produced or consumed as part of a global transaction, the message is essentially locked by the transaction and remains locked until the transaction coordinator either commits or aborts the JMS branch. If the coordinator is not able to communicate the outcome of the transaction to the JMS server due to a failure, the message(s) associated with the transaction may remain pending for a long time.

The JMS server transaction management features available through the WebLogic Remote Console allow you to:

- Identify in-progress transactions for which a JMS server is a participant.
- Identify messages associated with a JMS transaction branch.
- Force the outcome of pending JMS transaction branches, either by committing them or rolling them back.
- Manage JMS client connections.

You can view all the JMS connections on a particular WebLogic Server instance and get address and port information for each process that is holding a connection. You can also terminate a connection. In the Remote Console, go to the **Monitoring Tree**: **Services**: **Messaging**: **JMS Servers**. Select the JMS Server for which you want to manage transactions.

For more information about JMS transactions, see Using Transactions with WebLogic JMS in *Developing JMS Applications for Oracle WebLogic Server*.

Managing Durable Topic Subscribers

You can view a list of durable subscribers for a given topic, browse messages associated with a subscriber, create and delete subscribers, and delete selected messages or delete all messages for a subscription.

9

Best Practices for JMS Beginners and Advanced Users

Learn about the recommended and best practices for JMS beginners and advanced JMS users of Oracle WebLogic Server.
The following topics include targeting, integration options, understanding URLs, high availability (HA), and tuning for WebLogic Server:

- Configuration Best Practices
- Targeting Best Practices
  Oracle recommends certain targeting guidelines for JMS resources.
- High Availability Best Practices
  Learn some of the best practices to achieve and leverage high availability features in WebLogic Server.
- Client Resiliency Best Practices
  Ensuring JMS resiliency generally requires attention in two main areas: infrastructure level resilience and JMS application resilience.
- Distributed Destination Best Practices
  Applications that use distributed destinations are more highly available than applications that use simple destinations because WebLogic JMS provides load balancing and failover for member destinations of a distributed destination within a cluster.
- Understanding WebLogic JMS Client Options
- Understanding WebLogic URLs
- Strict Message Ordering Best Practices
- Integrating Remote JMS Destinations
- JMS Performance and Tuning
  See the following checklist of items to consider when tuning WebLogic JMS.

Configuration Best Practices

Configuring a JMS application includes configuring persistent stores, JMS servers, JMS modules, and JMS resources within JMS modules. JMS resources within modules include JMS connection factories, JMS standalone destinations, or JMS distributed destinations.
Configuring a SAF application is very similar except that it substitutes SAF Agents for JMS Servers and imported destinations for distributed destinations.
The following sections outline the basic procedure:

- Configure JMS Servers and Persistent Stores
- Configure JMS Quotas and Paging
- Configure a JMS Module
- Configure JMS Resources
- Configure SAF Agents, Stores, and Imported Destination

Configure JMS Servers and Persistent Stores

Before you start configuring JMS servers and persistent stores, consider the following:

- Destinations, connection factories, and other JMS resources are configured separately from their host JMS servers and persistent stores. The best practice steps for configuring JMS resources will be described later.
- WebLogic JMS distributed destination features require a working WebLogic cluster.
- There are two types of clustered JMS configuration:
  - Cluster-targeted JMS servers and stores provide simplified configuration, simplified HA, and the ability to dynamically scale resources. JMS servers, stores, and distributed destination members are automatically added as the cluster size increases without any configuration change required.

    > **Note:**
    >
    > Cluster-targeted JMS is the only available JMS option if you want to leverage WebLogic dynamic clusters instead of a configured cluster. See Simplified JMS Cluster and High Availability Configuration.

  - Individually configured JMS HA consists of a set of individually configured JMS servers and stores that are in turn targeted to migratable targets within a cluster. This option primarily exists for compatibility with earlier versions.
- If you are not using a cluster, then you may want to reconsider and use a cluster of at least size one and use distributed destinations instead of standalone destinations. This helps “future-proof" your application for future HA and scaling support, as it ensures configuration and applications are easily capable of expanding to multiple servers. Leveraging a cluster is best done in very early configuration and application design stages as it is a difficult process to convert a non-clustered application and configuration into a clustered topology.

> **Note:**
>
> For high availability and scaling, Oracle recommends targeting the JMS artifacts to a cluster along with appropriate high availability configuration settings instead of using migratable targets.

Use the following steps to configure JMS servers and persistent stores in a cluster.

Steps for setting up cluster-targeted JMS:

1. Create a custom store and target it to the cluster. See Simplified JMS Cluster and High Availability Configuration. Ensure the store is configured to persist your data safely as described in Ensure Your Data is Persisted Safely.
2. Configure a JMS server and target it to the same cluster, plus configure the JMS server to reference the store that was created in step 1. Multiple JMS servers can reference the same store.

Steps for setting up individual-configured JMS in a cluster:

1. Create a custom store on each WebLogic server in the cluster. Target each store to the default migratable target on its server or to a custom migratable target. See Service Migration in *Administering Clusters for Oracle WebLogic Server*. Ensure the store is configured to persist your data safely as described in Ensure Your Data is Persisted Safely.

> **Note:**
>
> It is recommended to use custom stores that provide more flexibility in tuning and administration. In addition, the default file store is not migratable between JVMs. Only custom stores are migratable.

2. Configure a JMS server on each WebLogic server. Configure the JMS server to reference the store for the server that was created in step 1. Target the JMS server to the same target that was used for the store. Multiple JMS servers can reference the same store.

Configure JMS Quotas and Paging

Configure a message and/or byte quota on the JMS server. There is no default quota, so configuring one helps protect against out-of-memory conditions. Rule of thumb: conservatively assume that each message consumes at least 512 bytes of memory even if it is paged out.

> **Note:**
>
> Persistent JDBC store messages, persistent file store messages, and non-persistent messages are all fully cached in JVM memory until they page out. And any type of message has a header that remains in JVM memory even after the message is paged out.

Although JMS paging is enabled by default, verify that the default behavior is valid for your environment. See Paging Out Messages To Free Up Memory in *Tuning Performance of Oracle WebLogic Server*.

Configure a JMS Module

A JMS module encapsulates JMS connection factory and destination configuration. It is a best practice to configure a system module and system module subdeployment for each homogenous set of JMS servers within a cluster. This maintains a simple one to one correspondence between JMS module resource configuration, JMS module subdeployment, and JMS server configuration.

A homogenous set of JMS servers is either:

- A single cluster-targeted JMS server with its store's distribution-policy set to `Distributed`.
- A single cluster-targeted JMS server with its store's distribution-policy set to `Singleton`.
- A set of one or more individually configured JMS servers.

Different JMS configurations can co-exist in the same cluster scope as long as they use different configuration names and different JNDI names.

Specifically:

1. Create a system module. Target it to a single cluster (if using clusters) or a single WebLogic Server instance. You must always target the module even when leveraging subdeployments.

   > **Note:**
   >
   > System modules are the preferred way to configure JMS resources. The alternatives, deployable modules and JMS resource definitions, are not recommended. Deployable modules are a deprecated feature. System modules are strongly preferred over JMS resource definitions because specifying connection factories and destinations using a configurable system module is generally much more flexible, maintainable, and portable than hard coding configuration in an application.

2. Create exactly one subdeployment per module. Subdeployments are sometimes referred to as "advanced targeting" on the WebLogic Remote Console. A single subdeployment aids simplicity - it is much easier for third parties to understand the targeting, and it reduces the chances of making configuration errors. If a single subdeployment is not sufficient, create two modules.
3. Populate the subdeployment only with the JMS servers contained in the ‘homogenous set of JMS Servers` above (this will be a single JMS Server in a simplified cluster-targeted JMS configuration). Do not reference WebLogic Servers or a cluster in the subdeployment. Only include the JMS servers that you wish to host destinations. This ensures that when the JMS resources are configured in the next step below, they are targeted to the correct JMS servers. For modules that support non-distributed destinations, the subdeployment must only reference a single individually-configured JMS server, or a single cluster-targeted JMS server that references a store with its distribution-policy set to `Singleton`. For modules that support distributed destinations, always choose a single cluster-targeted JMS server with its distribution-policy set to `Distributed`, or a homogenous set of individually-configured JMS servers.
   If you have a mix of distributed and non-distributed destinations, use two modules each with its own subdeployment.

Configure JMS Resources

Configure your JMS resources and target them properly.

1. Create destinations and target them to a single subdeployment (called "advanced targeting" on the Remote Console). Do not use default targeting instead of a subdeployment. Note that only distributed destinations can be targeted to a subdeployment target that resolves to multiple JMS servers. If you have a mix of distributed destinations, stand alone destinations, and imported destinations, then use two modules each with its own subdeployment. See Targeting Best Practices.
2. Create and use custom connection factories instead of using default connection factories. Default connection factories are not tunable.

   In most cases, you can use default targeting with connection factories as default targeting causes the resource to inherit the module's target. For connection factories that are only used by remote clients, use the module's subdeployment target.
3. Set the connection factory unmapped resource reference mode to FailSafe instead of leaving it as its default value ReturnDefault. The value FailSafe ensures that applications

do not use unmapped resource references. See Change in Behavior of Unmapped Connection Factory Resources in *Release Notes for Oracle WebLogic Server*.

For more information about setting the unmapped resource reference mode, see Specifying the Unmapped Resource Reference Mode for Connection Factories.

Configure SAF Agents, Stores, and Imported Destination

SAF agents, their stores, and their imported destinations should follow the same best practices as JMS servers, their stores, and JMS destinations.

Targeting Best Practices

Oracle recommends certain targeting guidelines for JMS resources.

They are:

- Avoid default targeting, WebLogic server targeting, and cluster targeting with destinations. Instead use advanced targeting (subdeployment targeting) for destinations and ensure that the subdeployment references only JMS servers or SAF agents. This applies to all destination types, including non-distributed, distributed, and imported.

  Even if the current JMS Servers or SAF Agents in your domain are only used for your specific application, this is a best practice because:

  – New JMS Servers or SAF Agents that are unrelated to your current application can be introduced by other applications, web services, or 3rd-party products.

  – In the future, your application may require different destinations and different JMS Servers or SAF Agents for scalability or management purposes.
- Always use advanced targeting when configuring Web Services deployments and error queues, this includes both development and production environments.
- To use an error destination within a distributed queue, it must be targeted to the same subdeployment as its parent destination.
- In most cases, you can use default targeting with connection factories as default targeting causes the resource to inherit the module's target. For connection factories that are only used by remote clients, use the module's subdeployment target.

High Availability Best Practices

Learn some of the best practices to achieve and leverage high availability features in WebLogic Server.

This section includes the following topics:

- Develop Applications on a Cluster
- Leverage WebLogic HA Features
- Ensure Your Data is Persisted Safely

Develop Applications on a Cluster

To achieve High Availability (HA), develop applications that leverage clustered WebLogic features and use a cluster during development. This approach is best taken in the early

configuration and application design stage as it is a difficult process to change a non-clustered application into a clustered application.

Leverage WebLogic HA Features

In WebLogic JMS, a sent message is only available to consumers if the message's host JMS server instance is running. Even if a message is stored on a shared file system or database, or in a distributed destination, the only JMS server instance that can recover a message is the same instance that originally stored the message. For example, if a message is sent to a distributed queue hosted on multiple JMS server instances, then the message will be internally load balanced to exactly one of these instances and stored in that particular instance, and, if that instance later crashes, the message will not be available to consumers until the instance is restarted.

To restore message availability after a failure, WebLogic includes features for automatically restarting and/or migrating an entire WebLogic Server JVM. It also includes features for restarting a JMS server and store instance within a JVM or migrating the instance to a different JVM within the same cluster after a failure. Finally, it includes features for clustering (distributing) a destination across multiple JMS servers within the same cluster. A distributed destination helps high availability because even if some of the destination's stored messages are unavailable while they wait for their failed JMS server and store to restart or migrate, other messages for the logical destination can still be produced and consumed via a running JMS server.

HA is normally accomplished using a combination of:

- WebLogic Clustering (Dynamic or Configured Clusters). See Understanding WebLogic Server Clustering in *Administering Clusters for Oracle WebLogic Server*.
- Whole server migration and/or restart. See Whole Server Migration in *Administering Clusters for Oracle WebLogic Server*.
- WebLogic JMS server and store Restart In Place. See Service Restart In Place in *Administering the WebLogic Persistent Store*.
- JTA Service Migration between JVMs in a cluster (since JMS applications often leverage transactions). See Service Migration in *Administering Clusters for Oracle WebLogic Server*.
- JMS Service Migration between JVMs in a cluster. See Service Migration in *Administering Clusters for Oracle WebLogic Server*.
- Cluster leasing. Automatic service migration and automatic whole server migration both required setting up cluster leasing. It is a best practice to configure database leasing instead of cluster leasing. See Leasing in *Administering Clusters for Oracle WebLogic Server*.
- Distributed destinations. See Using Distributed Destinations in *Developing JMS Applications for Oracle WebLogic Server*.

Ensure Your Data is Persisted Safely

- If you are using file stores, ensure the files are located on highly available storage (such as RAID), and are centrally accessible from any machine in a cluster (via SAN, etc) or from any machine that might host a WebLogic Server after a whole server migration, so that file data can be recovered after a failure. Default file stores similarly also need to be highly available and centrally accessible if JTA is not using a JDBC TLog Store since transaction logs are stored in default file stores by default. See File Locations and Additional

Requirement for High Availability File Stores in *Administering the WebLogic Persistent Store*.

> **Note:**
>
> It is important to explicitly configure the directory of a custom or default file store when ensuring the file store uses a central directory location. Otherwise, the file store may recover its data using a directory location based on the current WebLogic Server name, and so could recover from an incorrect directory if it moves between WebLogic Servers.

- Note that paging and cache files do not need high availability and should be located on local file systems for performance reasons.
- If you are using custom database stores, transaction log JDBC stores, or cluster database leasing, then ensure their database tables are located on a highly available database solution such as Oracle RAC.
- If you have a Disaster Recovery solution that leverages async replication of data, then file base storage is not usually recommended. Instead, Oracle strongly recommends that all JMS and JTA data be stored on the same database as any applications that perform a combination of messaging and database operations. This ensures that your recovered messages and data stay synchronized even after a disaster recovery. See Using a JDBC Store and Using a JDBC TLog Store in *Administering the WebLogic Persistent Store*.

Client Resiliency Best Practices

Ensuring JMS resiliency generally requires attention in two main areas: infrastructure level resilience and JMS application resilience.

The first area is infrastructure level resilience, which is generally addressed via configuration and encompasses persistence, leveraging clusters, JVM Migration, and Service Migration as described above in High Availability Best Practices. The second area is JMS application resilience, which is generally addressed via coding best practices that apply to any JMS Provider (not just WebLogic JMS) and is described below.

On any unrecoverable client API level messaging failure with any JMS provider, it is the responsibility of the programmer to make sure that the related JMS resources are closed and recreated, and, if it's desirable, that the failed operation be retried. The recommended steps for recreating a JMS resource mirror the steps for creating one. For example, if a JMS produce or JMS consume call throws an exception, close the parent JMS Connection or JMS Context, close the initial JNDI Context (if it is still open), and finally recreate the Initial Context, re-obtain the JMS Connection Factory and Destination from JNDI, and recreate the connection or JMSContext, session, and producer or consumer.

Exception handling in server-side applications is usually simplified by taking advantage of JMS asynchronous containers or pooling:

- Error handling for synchronous applications in server side messaging can take advantage of built-in-pooling to handle the problem simply by, for example, creating a connection, session, and producer, and then closing the connection for each sent message. The pools will in turn retrieve objects from a cache or recreate them as needed. Pooling is activated transparently by using a resource reference to access a JMS connection factory, or JMS 2.0 Java EE 7 JMS Context injection. Note that even when objects are pooled via resource-reference, it is still usually a good idea to retrieve new JMS Connection Factory or Destination objects after an error (in case these objects have somehow changed). See

Enhanced Support for Using WebLogic JMS with EJBs and Servlets in *Developing JMS Applications for Oracle WebLogic Server*.

- Error handling for asynchronous 'onMessage' server side messaging is generally best handled by leveraging WebLogic MDB containers, or a JMS JCA Adapter from your framework if MDBs are not options. These containers will automatically handle failures and retry as needed. See Developing Message-Driven Beans for Oracle WebLogic Server for more information.

If an asynchronous messaging application cannot take advantage of a container, Oracle strongly recommends that the application register an 'onException' exception listener on their Connection to detect and handle failures. See the javadoc for `javax.jms.Connection.setExceptionListener()` or, if using JMS 2.0, see `javax.jms.JMSContext.setExceptionListener()`.

Note:

- There is one exception type that does not require recreating JMS client objects: a `javax.jms.ResourceAllocationException` thrown by a message producer. This exception indicates a message quota failure where the server is unable to accept a message because a quota condition has been reached.
- Applications should avoid retrying operations in a tight loop in order to avoid performing relatively expensive create operations too frequently. This is because almost all failures do not resolve immediately, and frequently destroying and creating JMS client objects such as JNDI contexts, connections, sessions, producer, and consumers in the interim can be relatively far more expensive in comparison to normal runtime operations like sending or receiving a message. In short, tight-loop-retries can consume far more resources than other operations. Oracle recommends a single immediate retry attempt, and then progressively longer intervals between retries of up to a maximum of at least a few seconds.
- It is a highly recommended best practice to pool or cache JMS resources. To simplify error handling, it is tempting to ignore this advice and write applications that always create a new JMS connection, session, etc for each JMS message. But if these items are not pooled or cached, then this approach is not recommended for the same costly reasons that are described above for a tight loop retry. Some messaging applications may see a 5X or more performance hit. If pooling is not available, Oracle recommends caching JMS objects in your application for re-use.
- One can always assume an operation succeeded if it returns success, but one cannot always assume that an operation failed if it throws an exception. For example, a message send failure can occur even if the produced message successfully made it to the server destination, but the server was unable to send an internal success response back to the sender client. Another example is that a failed commit call does not necessarily indicate that the transaction did not commit (for similar reasons as the producer example).
- The configurable WebLogic JMS Reconnect settings on JMS Connection Factories and the corresponding reconnection `weblogic.jms.extension.WLConnection` and `weblogic.jms.extension.JMSContext` APIs are deprecated. They do not handle all possible failures and so are not an effective substitute for the above resiliency best practices. They will be removed or ignored in a future release.

Distributed Destination Best Practices

Applications that use distributed destinations are more highly available than applications that use simple destinations because WebLogic JMS provides load balancing and failover for member destinations of a distributed destination within a cluster.

Distributed Queues

Distributed queues are fairly easy to apply to an arbitrary clustered queueing use case, and easiest to apply when using MDBs, the SOA JMS Adapter, or OSA Adapter to consume from them. The MDB and adapter options automatically ensure that all distributed destination members are serviced by consumers. In addition, these containers are resilient to failures, and generally provide a proven and simple way for securing and multi-threading consumer applications regardless of destination type.

Distributed Topics

Distributed Topics are best applied when you use MDBs, the SOA JMS adapter, or the OSA JMS adapter to subscribe since direct subscribers have limitations and may require use of sophisticated extensions. The MDB and adapter containers automatically setup subscriptions and consumers for you.

Distributed Topics are recommended to be setup up as “Partitioned Distributed Topics” (a type of Uniform Distributed Topic) instead of “Replicated Distributed Topics”. This is because PDTs tend to be more scalable and fully supported in all advanced features, while RDTs are not supported in a dynamic cluster, WebLogic Multi-Tenant, or cluster-targeted JMS configuration. A PDT is a Uniform Distributed Topic that has its forwarding-policy configured to be partitioned instead of replicated. If you are already using RDTs, consider replacing them with PDTs.

See Configuring and Deploying MDBs Using JMS Topics in *Developing Message-Driven Beans for Oracle WebLogic Server*, Using Distributed Destinations, and Replacing an RDT with a PDT in *Developing JMS Applications for Oracle WebLogic Server*.

Weighted Distributed Destinations

Oracle strongly recommends using Uniform Distributed Destinations in place of Weighted Distributed Destinations. Weighted Distributed Destinations are deprecated and were superseded by Uniform Distributed Destinations as of WebLogic Server 9.0. Weighted Distributed Destinations are not supported in dynamic cluster, WebLogic Multi-Tenant, or cluster-targeted-JMS configurations.

Understanding WebLogic JMS Client Options

For client applications that have a runtime environment independent of WebLogic Server, there are multiple JMS client options, including: Java, .NET, and C clients. See JMS Clients in *Developing Stand-alone Clients for Oracle WebLogic Server*.

> **Note:**
>
> WebLogic JMS clients do not directly support foreign transaction managers. Use the WebLogic TM if possible. For advanced users, the transaction subsystem Interposed Transaction Manager feature may be used as an XA resource. See Participating in Transactions Managed by a Third-Party Transaction Manager in *Developing JTA Applications for Oracle WebLogic Server*.

Understanding WebLogic URLs

Applications that are communicating with a remote WebLogic Server instance or cluster must specify a URL when creating their JNDI InitialContext objects and/or setting application attributes in order to connect to a server or a cluster.

- Do not specify URLs for applications that run on the same server or cluster as their JMS resources. When an initial context is created without specifying URLs, it automatically references the local server or cluster JNDI.
- If a URL resolves to multiple addresses, WebLogic Server clients will randomly select an address in the list to start with and then automatically try each address in turn until one succeeds.
- In production systems, consider using DNS round robin or a hardware load balancer for initial hostname resolution rather than using the multiple host/port URL notation shown in URL syntax.

> **Note:**
>
> JMS connection factories are obtained from JNDI; however, by default, they load balance their connections independently of the JNDI context address. A JMS connection factory normally load balances across the servers included in the JMS connection factory's configured target. Similar to an EJB reference, a JMS connection factory is a 'smart stub' that contains a list of server addresses implicitly obtained from domain configuration and implicitly updated as a cluster grows and shrinks. This means that even if a JNDI context's URL only references a single server in a cluster, and a cluster-targeted or default targeted connection factory is obtained from this JNDI context, the JMS connections will still load balance across the entire cluster.

- URL syntax
  Learn about the WebLogic URL syntax and how it must be used to construct a URL for an application.

URL syntax

Learn about the WebLogic URL syntax and how it must be used to construct a URL for an application.

The WebLogic URL syntax is:

```
[t3|t3s|http|https|iiop|iiops]://address[,address]...
```

Where

- `address` = `hostlist : portlist`
- `hostlist` = `hostname [,hostname]...`
- `portlist` = `portrange [+portrange]...`
- `portrange` = `port [-port]`

Use `port-port` to indicate a port range, and + to separate multiple port ranges. For example, a simple address is typically something like `t3://hostA:7001`; the address `t3://hostA,hostB:7001-7002` is equivalent to the following addresses.

- `t3://hostA,hostB:7001+7002`
- `t3://hostA:7001-7002,hostB:7001-7002`
- `t3://hostA:7001+7002,hostB:7001+7002`
- `t3://hostA:7001,hostA:7002,hostB:7001,hostB:7002`

Strict Message Ordering Best Practices

Oracle recommends using the WebLogic JMS Unit-of-Order feature when strict message delivery ordering is required. UOO is the simplest and most capable strict delivery ordering option, and it supports parallel consumption of different orderings within the same destination. Normally it requires minimal or even no changes to applications, and it is resilient to any type of failure. In addition, it works with distributed destinations, scheduled messages, delayed messages, transactions, and store-and-forward.
Note that when using JMS Unit-of-Order in combination with a distributed destination or SAF imported destination, Oracle recommends configuring the destination's unit-of-order routing policy to the Path Service option and configuring a Path Service for the cluster. The Path Service option preserves orderings even when new members are configured for the distributed destination, and is the only supported policy in a cluster-targeted JMS configuration.

See Using Message Unit-of-Order in *Developing JMS Applications for Oracle WebLogic Server*.

If strict messaging ordering is needed and the Unit-of-Order feature cannot be leveraged, see Ordered Redelivery of Messages in *Developing JMS Applications for Oracle WebLogic Server*.

Integrating Remote JMS Destinations

To integrate remote JMS destinations into a local WebLogic Server or WebLogic Cluster, it is a best practice to configure a Foreign JMS Server mapping in order to map remote JMS destination and connection factory JNDI names into the local standalone server JNDI or local cluster JNDI namespace. The destinations that can be mapped include remote WebLogic JMS destinations, Oracle AQ JMS destinations, and non-Oracle JMS destinations.
Furthermore, another best practice is to use Foreign JMS Server mappings in combination with MDBs, res-ref or context injection JMS pools, or Messaging Bridges. These components use the mapping plus any security credentials configured in the Foreign JMS Server to access remote JMS resources.

Alternatively, consider using WL JMS SAF agents to forward remote WebLogic JMS messages into a local server or cluster.

See FAQs: Integrating Remote JMS Providers in *Developing JMS Applications for Oracle WebLogic Server*.

JMS Performance and Tuning

See the following checklist of items to consider when tuning WebLogic JMS.

JMS Performance & Tuning Check List in *Tuning Performance of Oracle WebLogic Server*.

10

Troubleshooting WebLogic JMS

Learn how to troubleshoot WebLogic JMS messages and configurations in Oracle WebLogic Server.
For more information on monitoring JMS statistics and managing JMS messages, see Monitoring JMS Statistics and Managing Messages.

This chapter includes the following sections:

- Configuring Notifications for JMS
  A notification is an action that is triggered when a watch rule evaluates to `true`. JMS notifications are used to post messages to JMS topics and/or queues in response to the triggering of an associated watch.
- Debugging JMS
  After you have narrowed the problem down to a specific application, you can activate the WebLogic Server debugging features to track down the specific problem within the application.
- Message Life Cycle Logging
- JMS Message Log Content
  Each record added to the JMS Message log includes basic information such as the message ID and correlation ID for the subject message.
- Controlling Message Operations on Destinations

Configuring Notifications for JMS

A notification is an action that is triggered when a watch rule evaluates to `true`. JMS notifications are used to post messages to JMS topics and/or queues in response to the triggering of an associated watch.

In the system resource configuration file, the elements `<destination-jndi-name>` and `<connection-factory-jndi-name>` define how the message is to be delivered.

See Configuring Notifications in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.

Debugging JMS

After you have narrowed the problem down to a specific application, you can activate the WebLogic Server debugging features to track down the specific problem within the application.

The following sections describe how to enable and use the debugging features:

- Enabling Debugging
- JMS Debugging Scopes
- Messaging Kernel and Path Service Debugging Scopes
- Request Dyeing

Enabling Debugging

You can enable debugging by setting the appropriate `ServerDebug` configuration attribute to `true`. Optionally, you can also set the server `StdoutSeverity` to `Debug`.

You can modify the configuration attribute in any of the following ways.

- Enable Debugging Using the Command Line
- Enable Debugging Using the WebLogic Remote Console
- Enable Debugging Using the WebLogic Scripting Tool
- Changes to the config.xml File

Enable Debugging Using the Command Line

Set the appropriate properties on the command line. For example,

```
-Dweblogic.debug.DebugJMSBackEnd=true
-Dweblogic.log.StdoutSeverity="Debug"
```

This method is static and can only be used at server startup.

Enable Debugging Using the WebLogic Remote Console

Use the WebLogic Remote Console to set the debugging values:

1. In the **Edit Tree**, go to **Environment**, then **Servers**.
2. Select the server on which you want to enable or disable debugging.
3. Click the **Debug** tab, then the **All** subtab.
4. Select the check boxes for the debug scopes or attributes you want to enable.

   See JMS Debugging Scopes.
5. Deselect check boxes for the debug scopes or attributes you want to disable.
6. Click **Save**.

   Not all changes take effect immediately—some require a restart.

   This method is dynamic and can be used to enable debugging while the server is running.

Enable Debugging Using the WebLogic Scripting Tool

Use the WebLogic Scripting Tool (WLST) to set the debugging values. For example, the following command runs a program for setting debugging values called `debug.py`:

```
java weblogic.WLST debug.py
```

The main scope, weblogic, does not appear in the graphic; jms is a sub-scope within weblogic. Note that the fully-qualified `DebugScope` for `DebugJMSBackEnd` is `weblogic.jms.backend`.

The `debug.py` program contains the following code:

```
user='user1'
password='password'
url='t3://localhost:7001'
connect(user, password, url)
```

```
edit()
cd('Servers/myserver/ServerDebug/myserver')
startEdit()
set('DebugJMSBackEnd','true')
save()
activate()
```

Note that you can also use WLST from Java. The following example shows a Java file used to set debugging values:

```
import weblogic.management.scripting.utils.WLSTInterpreter;
import java.io.*;
import weblogic.jndi.Environment;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class test {
        public static void main(String args[]) {
       try {
              WLSTInterpreter interpreter = null;
              String user="user1";
              String pass="pw12ab";
              String url ="t3://localhost:7001";
              Environment env = new Environment();
              env.setProviderUrl(url);
              env.setSecurityPrincipal(user);
              env.setSecurityCredentials(pass);
              Context ctx = env.getInitialContext();

              interpreter = new WLSTInterpreter();
              interpreter.exec
                     ("connect('"+user+"','"+pass+"','"+url+"')");
              interpreter.exec("edit()");
              interpreter.exec("startEdit()");
              interpreter.exec
                     ("cd('Servers/myserver/ServerDebug/myserver')");
              interpreter.exec("set('DebugJMSBackEnd','true')");
              interpreter.exec("save()");
              interpreter.exec("activate()");

       } catch (Exception e) {
       System.out.println("Exception "+e);
       }
       }
}
```

Using the WLST is a dynamic method and can be used to enable debugging while the server is running.

Changes to the config.xml File

Changes in debugging characteristics, through the Remote Console, WLST, or the command line are persisted in the `config.xml` file.

This sample `config.xml` fragment shows a transaction debug scope (set of debug attributes) and a single JMS attribute.

Example 10-1 Example Debugging Stanza for JMS

```
<server>
<name>myserver</name>
```

```
<server-debug>
<debug-scope>
<name>weblogic.transaction</name>
<enabled>true</enabled>
</debug-scope>
<debug-jms-back-end>true</debug-jms-back-end>
</server-debug>
</server>
```

JMS Debugging Scopes

The following are registered debugging scopes for JMS:

- `DebugJMSBackEnd` (scope `weblogic.jms.backend`) – prints information for debugging the JMS Back End (including some information used for distributed destinations and JMS SAF).
- `DebugJMSFrontEnd` (scope `weblogic.jms.frontend`) – prints information for debugging the JMS Front End (including some information used for multicast).
- `DebugJMSCommon` (scope `weblogic.jms.common`) – prints information for debugging JMS common methods (including some information from the client JMS producer).
- `DebugJMSConfig` (scope `weblogic.jms.config`) – prints information related to JMS configuration (backend, distributed destinations, and foreign servers).
- `DebugJMSBoot` (scope `weblogic.jms.boot`) – prints some messages at boot time regarding what store the JMS server is using and its configured destinations.
- `DebugJMSDispatcher` (scope `weblogic.jms.dispatcher`) – prints information related to `PeerGone()` occurrences.
- `DebugJMSDistTopic` (scope `weblogic.jms.config`) – prints information about distributed topics, and primary bind and unbind information.
- `DebugJMSPauseResume` (scope `weblogic.jms.pauseresume`) – prints information about (backend) pause/resume destination operations.
- `DebugJMSModule` (scope `weblogic.jms.module`) – prints a lot of information about JMS module operations and message life cycle.
- `DebugJMSMessagePath` (scope `weblogic.jms.messagePath`) – prints information following a message through the message path (client, frontend, backend), including the message identifier.
- `DebugJMSSAF` (scope `weblogic.jms.saf`) – prints information about JMS SAF (store-and-forward) destinations.
- `DebugJMSCDS` (scope `weblogic.jms.CDS`) – prints detailed information about JMS "Configuration Directory Service" (used by various sub-systems to get the notification of configuration changes to the JMS resources configured in the server from within a cluster as well as across the clusters and domains).
- `DebugJMSWrappers` (scope `weblogic.jms.wrappers`) – prints information pooling and wrapping of JMS connections, sessions, and other objects, used inside an EJB or servlet using the `resource-reference` element in the deployment descriptor.

Messaging Kernel and Path Service Debugging Scopes

The following are registered debugging scopes for the messaging kernel and the Path service.

- `DebugMessagingKernel` (scope `weblogic.messaging.kernel`) : Prints information about the messaging kernel.
- `DebugMessagingKernelBoot` (scope `weblogic.messaging.kernelboot`) : Prints information about booting the messaging kernel (processing messages).
- `DebugPathSvc` (scope `weblogic.messaging.pathsvc`) : Prints limited information about some unusual conditions in the path service.
- `DebugPathSvcVerbose` (scope `weblogic.messaging.pathsvcverbose`) : Prints limited information about unusual conditions in the path service.

Request Dyeing

Another option for debugging is to trace the flow of an individual (typically "dyed") application request through the JMS subsystem. See Configuring the Dye Vector via the DyeInjection Monitor in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.

Message Life Cycle Logging

JMS logging is enabled by default when you create a JMS server, however, you must specifically enable it on message destinations in the JMS modules targeted to this JMS server (or on the JMS template used by destinations).For more information on WebLogic logging services, see Understanding WebLogic Logging Services in *Configuring Log Files and Filtering Log Messages for Oracle WebLogic Server*.
The message life cycle is an external view of the events that a JMS message traverses through once it has been accepted by the JMS server, either through the JMS APIs or the JMS Message Management APIs. Message life cycle logging provides an administrator with easy access to information about the existence and status of JMS messages from the JMS server viewpoint. In particular, each message log contains information about basic life cycle events such as message production, consumption, and removal.

Logging can occur on a continuous basis and over a long period of time. It can be also be used in real-time mode while the JMS server is running, or in an off-line fashion when the JMS server is down. For information about configuring message logging, see Log Messages in the *Oracle WebLogic Remote Console Online Help*.

- Events in the JMS Message Life Cycle
- Enabling JMS Message Logging

Events in the JMS Message Life Cycle

When message life cycle logging is enabled for a JMS destination, a record is added to the JMS server's message log file each time a message meets the conditions that correspond to a basic message life cycle event. The life cycle events that trigger a JMS message log entry are as follows:

- Produced : This event is logged when a message enters a JMS server using the WebLogic Server JMS API or the JMS Management API.
- Consumed: This event is logged when a message leaves a JMS server using the WebLogic Server JMS API or the JMS Management API.
- Removed : This event is logged when a message is manually deleted from a JMS server using the WebLogic Server JMS API or the JMS Management API.

- Expired : This event is logged when a message reaches the expiration time stored on the JMS server. This event is logged only once per message even though a separate expiration event occurs for each topic subscriber who received the message.
- Retry exceeded : This event is logged when a message has exceeded its redelivery retry limit. This event may be logged more than one time per message, as each topic subscriber has its own redelivery count.
- Consumer created : This event is logged when a JMS consumer is created for a queue or a JMS durable subscriber is created for a topic.
- Consumer destroyed : This event is logged when a JMS consumer is closed or a JMS durable subscriber is unsubscribed.
- Message Log Location

Message Log Location

The message log is stored under your domain directory, as follows:

$DOMAIN_HOME\servers*server_name*\logs\jmsservers*jms_server_name**jms_server_name*-jms.messages.log

where *$DOMAIN_HOME* is the root directory of your domain, typically c:\Oracle\Middleware\user_projects\domains*domain_name*.

> **Note:**
>
> JMS server name is prefixed to JMS server log location in your domain directory. For example, when you create a JMS Server *JMSServer-0*, the following log file and location are set in your domain directory: \logs\jmsservers*JMSServer-0**JMSServer-0*-jms.messages.log

Enabling JMS Message Logging

You can enable or disable JMS message logging for a queue, topic, JMS template, uniform distributed queue, and uniform distributed topic using the WebLogic Remote Console.

1. In the **Edit Tree**, go to **Services**, then **JMS Servers** or **JMS System Resources**.
2. Select the JMS Server or JMS System Resource (queue, topic, JMS template, uniform distributed queue, and uniform distributed topic) for which you want to configure logging.
3. Click the **Logging** tab and configure the logging options as needed.
4. Click **Save**.

WebLogic Java Management Extensions (JMX) lets you access the JMSSystemResourceMBean and JMSRuntimeMBean MBeans to manage JMS message logs. See Overview of WebLogic Server Subsystem MBeans in *Developing Custom Management Utilities Using JMX for Oracle WebLogic Server*.

> **Note:**
>
> Logging JMS messages for non durable subscribers is not enabled in the default configuration. To enable, set the following system property: `weblogic.jms.message.logging.logNonDurableSubscriber=true`.

You can also use the WebLogic Scripting Tool to configure JMS message logging for a JMS servers and JMS system resources. See Using WLST to Manage JMS Servers and JMS System Module Resources.

When you enable message logging, you can specify whether the log entry will include all the message header fields or a subset of them; all system-defined message properties or a subset of them; all user-defined properties or a subset of them. You may also include or exclude the body of the message. For more information about message headers and properties see Developing a Basic JMS Application in *Developing JMS Applications for Oracle WebLogic Server*.

JMS Message Log Content

Each record added to the JMS Message log includes basic information such as the message ID and correlation ID for the subject message.

You can also configure the JMS server to include additional information such as the message type and user properties.

- JMS Message Log Record Format
- Sample Log File Records
- Managing JMS Server Log Files

JMS Message Log Record Format

Except where noted, all records added to the JMS Message Life Cycle Log contain the following pieces of information in the order in which they are listed:

- Date : The date and time the message log record is generated.
- Transaction identifier : The transaction identifier for the transaction with which the message is associated
- WLS diagnostic context : A unique identifier for a request or unit of work flowing through the system. It is included in the JMS message log to provide a correlation between events belonging to the same request.
- Raw millisecond value for "Date" : To aid in troubleshooting high-traffic applications, the date and time the message log record is generated is displayed in milliseconds.
- Raw nanosecond value for "Date" : To aid in troubleshooting high-traffic applications, the date and time the message log record is generated is displayed in nanoseconds.
- JMS message ID : The unique identifier assigned to the message.
- JMS correlation ID : A user-defined identifier for the message, often used to correlate messages about the same subject.
- JMS destination name : The fully-qualified name of the destination server for the message.

- JMS message life cycle event name : The name of the message life cycle event that triggered the log entry.
- JMS user name : The name of the user who (produced? consumed? received?) the message.
- JMS message consumer identifier – This information is included in the log only when the message life cycle event being logged is the "Consumed" event, the "Consumer Created" event, or the "Consumer Destroyed" event. If the message consumed was on a queue, the log will include information about the origin of the consumer and the OAM identifier for the consumer known to the JMS server. If the consumer is a durable subscriber, the log will also include the client ID for the connection and the subscription name.

  The syntax for the message consumer identifier is as follows:

  ```
  MC:CA(…):OAMI(wls_server_name.jms.connection#.session#.consumer#)
  ```

  Where:

  – `MC` stands for message consumer
  – `CA` stands for client address
  – `OAMI` stands for OA&M identifier
  – And, when applicable, `CC` stands for connection consumer

  If the consumer is a durable subscriber, then the additional information will be shown using the following syntax:

  ```
  DS:client_id.subscription_name[message consumer identifier]
  ```

  where `DS` stands for durable subscriber.
- JMS message content : This field can be customized on a per destination basis. However, the message body will not be available.
- JMS message selector : This information is included in the log only when the message life cycle event being logged is the "Consumer Created" event. The log will show the "Selector" argument from the JMS API.

Sample Log File Records

The sample log file records that follow show the type of information that is provided in the log file for each of the message life cycle events. Each record is a fixed length, but the information included will vary depending upon relevance to the event and on whether a valid value exists for each field in the record. The log file records use the following syntax:

```
####<date_and_time_stamp> <transaction_id> <WLS_diagnostic_context>
<date_in_milliseconds> <date_in_nanoseconds> <JMS_message_id>
<JMS_correlation_id> <JMS_destination_name> <life_cycle_event_name>
<JMS_user_name> <consumer_identifier> <JMS_message_content>
<JMS_message_selector>
```

> **Note:**
>
> If you choose to include the JMS message content in the log file, note that any occurrences of the left-pointing angle bracket (<) and the right-pointing angle bracket (>) within the contents of the message will be escaped. In place of a left-pointing angle bracket you will see the string "<" and in place of the right-pointing angle bracket you will see ">" in the log file.

Consumer Created Event

```
####<May 13, 2005 4:06:33 PM EDT> <> <> <1116014793818> <345063> <> <>
 <jmsfunc!TestQueueLogging> <ConsumerCreate> <system> <MC:CA(/
10.61.6.56):OAMI(myserver.jms.connection456.session460.consumer462)> <> <>
```

Consumer Destroyed Event

```
####<May 13, 2005 4:06:33 PM EDT> <> <> <1116014793844> <40852> <> <>
<jmsfunc!TestQueueLogging> <ConsumerDestroy> <system> <MC:CA(/
10.61.6.56):OAMI(myserver.jms.connection456.session460.consumer462)> <> <>
```

Message Produced Event

```
####<May 13, 2005 4:06:43 PM EDT> <> <> <1116014803018> <693671>
<ID:<327315.1116014803000.0>> <testSendRecord> <jmsfunc!TestQueueLoggingMarker>
 <Produced> <system> <> <&lt;?xml version="1.0" encoding="UTF-8"?&gt;
&lt;mes:WLJMSMessage
 xmlns:mes="http://www.bea.com/WLS/JMS/Message"&gt;&lt;mes:Header&gt;&lt;mes:JMSCor
relationID&gt;testSendRecord&lt;/mes:JMSCorrelationID&gt;&lt;mes:JMSDeliveryMode
&gt;NON_PERSISTENT&lt;/mes:JMSDeliveryMode&gt;&lt;mes:JMSExpiration&gt;0&lt;
/mes:JMSExpiration&gt;&lt;mes:JMSPriority&gt;4&lt;/mes:JMSPriority&gt;&lt;
mes:JMSRedelivered&gt;false&lt;/mes:JMSRedelivered&gt;&lt;mes:JMSTimestamp&gt;
1116014803000&lt;/mes:JMSTimestamp&gt;&lt;mes:Properties&gt;&lt;mes:property
name="JMSXDeliveryCount"&gt;&lt;mes:Int&gt;0&lt;/mes:Int&gt;&lt;/mes:property
&gt;&lt;/mes:Properties&gt;&lt;/mes:Header&gt;&lt;mes:Body&gt;&lt;mes:Text/&gt;
&lt;/mes:Body&gt;&lt;/mes:WLJMSMessage&gt;> <>
```

Message Consumed Event

```
####<May 13, 2005 4:06:45 PM EDT> <> <> <1116014805137> <268791>
<ID:<327315.1116014804578.0>> <hello> <jmsfunc!TestQueueLogging> <Consumed> <system>
<MC:CA(/10.61.6.56):OAMI(myserver.jms.connection456.session475.consumer477)> <&lt;?xml
version="1.0" encoding="UTF-8"?&gt;
&lt;mes:WLJMSMessage
 xmlns:mes="http://www.bea.com/WLS/JMS/Message"&gt;&lt;mes:Header&gt;&lt;mes:
```

```
JMSCorrelationID&gt;hello&lt;/mes:JMSCorrelationID&gt;&lt;mes:JMSDeliveryMode
&gt;PERSISTENT&lt;/mes:JMSDeliveryMode&gt;&lt;mes:JMSExpiration&gt;0&lt;/mes:
JMSExpiration&gt;&lt;mes:JMSPriority&gt;4&lt;/mes:JMSPriority&gt;&lt;mes:
JMSRedelivered&gt;false&lt;/mes:JMSRedelivered&gt;&lt;mes:JMSTimestamp&gt;
1116014804578&lt;/mes:JMSTimestamp&gt;&lt;mes:JMSType&gt;SendRecord&lt;/mes:
JMSType&gt;&lt;mes:Properties&gt;&lt;mes:property
name="JMS_BEA_RedeliveryLimit"&gt;&lt;mes:Int&gt;1&lt;/mes:Int&gt;&lt;/
mes:property&gt;&lt;mes:
property name="JMSXDeliveryCount"&gt;&lt;mes:Int&gt;1&lt;/mes:Int&gt;&lt;/
mes:property&gt;
&lt;/mes:Properties&gt;&lt;/mes:Header&gt;&lt;mes:Body&gt;&lt;mes:Text/&gt;&lt;/
mes:Body&gt;&lt;/mes:WLJMSMessage&gt;> <>
```

Message Expired Event

```
####<May 13, 2005 4:06:47 PM EDT> <> <> <1116014807258> <445317>
<ID:<327315.1116014807234.0>> <bar> <jmsfunc!TestQueueLogging> <Expired> <<WLS
 Kernel>> <> <&lt;?xml version="1.0" encoding="UTF-8"?&gt;
&lt;mes:WLJMSMessage
 xmlns:mes="http://www.bea.com/WLS/JMS/Message"&gt;&lt;mes:Header&gt;&lt;mes:
JMSCorrelationID&gt;bar&lt;/mes:JMSCorrelationID&gt;&lt;mes:JMSDeliveryMode&gt;
PERSISTENT&lt;/mes:JMSDeliveryMode&gt;&lt;mes:JMSExpiration&gt;1116014806234
&lt;/mes:JMSExpiration&gt;&lt;mes:JMSPriority&gt;4&lt;/mes:JMSPriority&gt;&lt;
mes:JMSRedelivered&gt;false&lt;/mes:JMSRedelivered&gt;&lt;mes:JMSTimestamp&gt;
1116014807234&lt;/mes:JMSTimestamp&gt;&lt;mes:JMSType&gt;ExpireRecord&lt;/mes:
JMSType&gt;&lt;mes:Properties&gt;&lt;mes:property
name="JMS_BEA_RedeliveryLimit"&gt;&lt;mes:Int&gt;1&lt;/mes:Int&gt;&lt;/
mes:property&gt;&lt;mes:
property name="JMSXDeliveryCount"&gt;&lt;mes:Int&gt;0&lt;/mes:Int&gt;&lt;/
mes:property&gt;
&lt;/mes:Properties&gt;&lt;/mes:Header&gt;&lt;mes:Body&gt;&lt;mes:Text/&gt;&lt;/
mes:Body&gt;&lt;/mes:WLJMSMessage&gt;> <>
```

Retry Exceeded Event

```
####<May 13, 2005 4:06:53 PM EDT> <> <> <1116014813491> <394206>
<ID:<327315.1116014813453.0>> <bar> <jmsfunc!TestQueueLogging> <Retry exceeded>
<<WLS Kernel>> <> <&lt;?xml version="1.0" encoding="UTF-8"?&gt;
&lt;mes:WLJMSMessage xmlns:mes="http://www.bea.com/WLS/JMS/
Message"&gt;&lt;mes:Header&gt;&lt;mes:
JMSCorrelationID&gt;bar&lt;/mes:JMSCorrelationID&gt;&lt;mes:JMSDeliveryMode&gt;
PERSISTENT&lt;/mes:JMSDeliveryMode&gt;&lt;mes:JMSExpiration&gt;0&lt;/mes:
JMSExpiration&gt;&lt;mes:JMSPriority&gt;4&lt;/mes:JMSPriority&gt;&lt;mes:
JMSRedelivered&gt;true&lt;/mes:JMSRedelivered&gt;&lt;mes:JMSTimestamp&gt;
1116014813453&lt;/mes:JMSTimestamp&gt;&lt;mes:JMSType&gt;RetryRecord&lt;/mes:
JMSType&gt;&lt;mes:Properties&gt;&lt;mes:property
name="JMS_BEA_RedeliveryLimit"&gt;&lt;mes:Int&gt;1&lt;/mes:Int&gt;&lt;/
mes:property&gt;&lt;mes:
property name="JMSXDeliveryCount"&gt;&lt;mes:Int&gt;2&lt;/mes:Int&gt;&lt;/mes:property
&gt;&lt;/mes:Properties&gt;&lt;/mes:Header&gt;&lt;mes:Body&gt;&lt;mes:Text/&gt;
&lt;/mes:Body&gt;&lt;/mes:WLJMSMessage&gt;> <>
```

Message Removed Event

```
####<May 13, 2005 4:06:45 PM EDT> <> <> <1116014805071> <169809>
<ID:<327315.1116014804859.0>> <hello> <jmsfunc!TestTopicLogging> <Removed>
 <system> <DS:messagelogging_client.foo.SendRecordSubscriber> <&lt;?xml version="1.0"
encoding="UTF-8"?&gt;
&lt;mes:WLJMSMessage xmlns:mes="http://www.bea.com/WLS/JMS/Message"&gt;&lt;mes:
Header&gt;&lt;mes:JMSCorrelationID&gt;hello&lt;/mes:JMSCorrelationID&gt;&lt;mes:
JMSDeliveryMode&gt;PERSISTENT&lt;/mes:JMSDeliveryMode&gt;&lt;mes:JMSExpiration
```

>0</mes:JMSExpiration><mes:JMSPriority>4</mes:JMSPriority>
<mes:JMSRedelivered>false</mes:JMSRedelivered><mes:JMSTimestamp
>1116014804859</mes:JMSTimestamp><mes:JMSType>
SendRecordSubscriber</mes:JMSType><mes:Properties><mes:property
 name="JMSXDeliveryCount"><mes:Int>0</mes:Int></mes:property
></mes:Properties></mes:Header><mes:Body><mes:Text/></
mes:Body></mes:WLJMSMessage>> <>

Managing JMS Server Log Files

After you create a JMS server, you can configure criteria for moving (rotating) old log messages to a separate file. You can also change the default name of the log file.

- Rotating Message Log Files
- Renaming Message Log Files
- Limiting the Number of Retained Message Log Files

Rotating Message Log Files

You can rotate old log messages to a new file based on a specific file size or at specified intervals of time. Alternately, you can choose not to rotate old log messages; in this case, all messages will accumulate in a single file and you will have to erase the contents of the file when it becomes too large.

If you rotate old messages whenever the log file reaches a particular size you must specify a minimum file size. After the log file reaches the specified minimum size, the next time the server checks the file size it will rename the current log file and create a new one for storing subsequent messages.

If you rotate old messages at a regular interval, you must specify the time at which the first new message log file is to be created, and then specify the time interval that should pass before that file is renamed and replaced.

For more information about setting up log file rotation for JMS servers, see Rotate Log Files in the *Oracle WebLogic Remote Console Online Help*.

Renaming Message Log Files

Rotated log files are numbered in order of creation. For example, the seventh rotated file would be named `myserver.log00007`. For troubleshooting purposes, it may be useful to change the name of the log file or to include the time and date when the log file is rotated. To do this, you add `java.text.SimpleDateFormat` variables to the file name. Surround each variable with percentage (%) characters. If you specify a relative pathname when you change the name of the log file, it is interpreted as relative to the server's root directory.

Limiting the Number of Retained Message Log Files

If you rotate old message log files based on either file size or time interval, you may also wish to limit the number of log files, then this JMS server creates for storing old messages. After the server reaches this limit, it deletes the oldest log file and creates a new log file with the latest suffix. If you do not enable this option, the server will create new files indefinitely and you have to manually clean up these files.

For more information, see Configure Logs in the *Oracle WebLogic Remote Console Online Help*.

Controlling Message Operations on Destinations

WebLogic JMS configuration and runtime APIs enable you to pause and resume message production, insertion, and/or consumption operations on a JMS destination or temporary destination, on a group of destinations configured using the same template, or on all the destinations hosted by a single JMS Server, either programmatically (using JMX and the runtime MBean API) or administratively (using the WebLogic Remote Console).In this way, you can control the JMS subsystem behavior in the event of an external resource failure that would otherwise cause the JMS subsystem to overload the system by continuously accepting and delivering (and redelivering) messages.
You can boot a JMS server and its destinations in a "paused" state which prevents any message production, insertion, or consumption on those destinations immediately after boot. To resume message operation activity, the administrator can later change the state of the paused destination to "resume" normal message production, insertion, or consumption operations. In addition, new runtime options allow an administrator to change the current state of a running destination to either allow or disallow new message production, insertion, or consumption.

Definition of Message Production, Insertion, and Consumption

There are several operations performed on messages on a destination:

- Messages are produced when a producer creates and sends a new message to that destination.
- Messages are inserted as a result of in-flight work completion, as when a message is made available upon commitment of a transaction or when a message scheduled to be made available after a delay is made available on a destination.
- Messages are consumed when they are removed from the destination.

You can pause and resume any or all of these operations either at start time or during runtime, as described in the sections that follow.

Pause and Resume Logging

When message production, insertion, or consumption on a destination is successfully "paused" or "resumed" either at start time or at runtime, a message is added to the server log to indicate the same. In the event of failure to pause or resume message production, insertion, or consumption on a destination, the appropriate error/exceptions are logged.

Production Pause and Production Resume

When a JMS destination is "paused for production", new and existing producers attached to that destination are unable to produce new messages for that destination. A producer that attempts to send a message to a paused destination receives an exception that indicates that the destination is paused. When a destination is "resumed from production pause", production of new messages is allowed again. Pausing message production does not prevent the insertion of messages that are the result in-flight work.

> **Note:**
>
> For an explanation of what constitutes in-flight work, see Definition of In-Flight Work.

- Pausing and Resuming Production at Boot Time
- Pausing and Resuming Production at Runtime
- Production Pause and Resume and Distributed Destinations
- Production Pause and Resume and JMS Connection Stop/Start

Pausing and Resuming Production at Boot Time

You can pause or resume production effective at boot-time for all the destinations on a JMS server, for a group of destinations that point to the same JMS template, or for individual destinations. If you configure `production-paused-at-startup`, the next time you boot the server, message production activities will be disallowed for the specified destination(s) until you explicitly change the state to "production enabled" for that destination. If you configure production to resume, the next time you boot the server, message production activities will be allowed on the specified destination(s) until the state is explicitly changed to "production paused" for that destination.

Pausing and Resuming Production at Runtime

You can pause or resume production during runtime for all the destinations targeted on a JMS server, for a group of destinations that point to the same JMS template, or for individual destinations. The most recent configuration change always take precedence, regardless of the level at which it is made (JMS server level, JMS template level, or destination level).

Production Pause and Resume and Distributed Destinations

If a member destination is paused for production, that member destination is not considered for production by the producer. Messages are steered away to other member destinations that are available for production.

Production Pause and Resume and JMS Connection Stop/Start

Stopping or starting a JMS connection has no effect on the production pause or production resume state of a destination.

Insertion Pause and Insertion Resume

When a JMS destination is paused for "insertion", both messages inserted as a result of in-flight work and new messages sent by producers are prevented from appearing on the destination. Use insertion pause to stop all messages from appearing on a destination.

You can determine whether there is any in-flight work pending by looking at the statistics on the WebLogic Remote Console. When you pause the destination for message "insertion", messages related to in-flight work completion are made "not deliverable" and new message production operations fail. All of those messages become "invisible" to the consumers and the statistics are adjusted to reflect that the messages are no longer pending.

The "insertion" pause operation supersedes the "production" pause operation. In other words, if the destination is currently in the "production paused" state, then you can change it to the "insertion paused" state.

You must explicitly "resume" a destination for message insertion to allow in-flight messages to appear on that destination. Successful completion of the insertion "resume" operation will change the state of the destination to "insertion enabled" and all the "invisible" in-flight messages are made available.

- Pausing and Resuming Insertion at Boot Time
- Pausing and Resuming Insertion at Runtime
- Insertion Pause and Resume and Distributed Destination
- Insertion Pause and Resume and JMS Connection Stop/Start

Pausing and Resuming Insertion at Boot Time

You can pause or resume insertion effective at boot-time for all the destinations on a JMS server, for a group of destinations that point to the same JMS template, or for individual destinations. If you configure `insertion-paused-at-startup`, the next time you boot the server, message insertion and production activities will be disallowed on the specified destination(s) until you explicitly change the state to "insertion enabled" for that destination. If you configure insertion to resume, the next time you boot the server, message insertion activities will be allowed on the specified destination(s) until the state is explicitly changed to "insertion paused" for that destination.

> **Note:**
>
> Because it is possible that this operation may be configured differently at each level (for example, the JMS Server level, the JMS template level, and the destination level), there is an established order of precedence. See Order of Precedence for Boot-time Pause and Resume of Message Operations.

Pausing and Resuming Insertion at Runtime

You can pause or resume insertion during runtime for all the destinations on a JMS server, for a group of destinations that point to the same JMS template, or for individual destinations. The most recent configuration change always take precedence, regardless of the level at which it is made (JMS Server level, JMS Template level, or destination level).

Insertion Pause and Resume and Distributed Destination

If a member destination is paused for insertion, that member destination will not be considered for message forwarding. Messages will be steered away to other member destinations that are available for insertion.

Insertion Pause and Resume and JMS Connection Stop/Start

Stopping or starting a JMS Connection has no effect on the insertion pause or insertion resume state of a destination.

Consumption Pause and Consumption Resume

When a JMS destination is "paused for consumption", messages on that destination are not available for consumption. When the destination is "resumed from consumption pause", both new and existing consumers attached to that destination are allowed to consume messages on the destination again.

When the destination is paused for consumption, the destination's state is marked as "consumption paused" and all new, synchronous receive operations will block until consumption is resumed and there are messages available for consumption. All synchronous receive with blocking time-out operations will block for the specified length of time. Messages will not be delivered to synchronous consumers attached to that destination while the destination is paused for consumption.

After a successful consumption "pause" operation, the user has to explicitly "resume" the destination to allow consume operations on that destination.

Pausing and Resuming Consumption at Boot-time

You can pause or resume consumption effective at boot-time for all the destinations on a JMS server, for a group of destinations that point to the same JMS template, or for individual destinations. If you configure `consumption-paused-at-startup`, the next time you boot the server, message consumption activities will be disallowed on the specified destination(s) until you explicitly change the state to "consumption enabled" for that destination. If you configure consumption to resume, the next time you boot the server, message consumption activities will be allowed on the specified destination(s) until the state is explicitly changed to "consumption paused" for that destination.

Pausing and Resuming Consumption at Runtime

You can pause or resume consumption during runtime for all the destinations on a JMS server, for a group of destinations that point to the same JMS template, or for individual destinations. The most recent configuration change always take precedence, regardless of the level at which it is made (JMS Server level, JMS Template level, or destination level).

Consumption Pause and Resume and Queue Browsers

Queue Browsers are special type of consumers that are only allowed to "peek" into queue destinations. A browse operation on a destination paused for consumption is perfectly legitimate and is allowed.

Consumption Pause and Resume and Distributed Destination

Member destinations that are currently paused for consumption are not considered by the consumer load balancing algorithm.

Consumption Pause and Resume and Message-Driven Beans

Pausing a destination for consumption prevents a message-driven bean (MDB) from getting any messages from its associated destination. This feature gives you more flexible control over the delivery of messages delivery to MDBs from the individual destination level as opposed to using connection start/stop. In other words, if you use the consumption pause/resume feature, then you can share the JMS connection among the multiple MDBs and still be able to prevent message delivery to selected MDBs by pausing the associated destination for consumption.

For more information about using MDBs, see Configuring Suspension of Message Delivery During JMS Resource Outages in Developing Message-Driven Beans for Oracle WebLogic Server.

Consumption Pause and Resume and JMS Connection Stop/Start

The JMS connection stop/start feature determines whether a consumer can successfully call the receiving APIs or not. The consumption pause/resume feature on a destination determines whether or not the receive call gets any messages from the destination or not. Stopping or starting a consumer's connection does not have any effect on the destination's consumption pause state.

If the consumer's connection is "started" from the "stopped" state, synchronous receive operations might block or time-out if the destination is currently paused for consumption. Asynchronous consumers do not receive any messages if the associated destination is in a "consumption paused" state.

Definition of In-Flight Work

In-flight Work Associated with Producers

The following types of messages are inserted on a destination as a result of in-flight work associated with message producers:

- Unborn Messages : Messages that are created by the producer with "birth time" (TimeToDeliver) set in the future. Until delivered, unborn messages are counted as "pending" messages in the destination statistics and are not available for consumption.
- Uncommitted Messages : Messages that are produced as part of a transaction (using either user transaction or transacted session) and have not yet been either committed or

rolled back. Until the transaction has been completed, uncommitted messages are counted as "pending" messages in the destination statistics and are not available for consumption.

- Quota Blocking Send : Messages that, if initially prevented from reaching a destination due to a quota limit, will block for a specific period of time while waiting for the destination to become available. The message may exceed the message quota limit, the byte quota limit, or both quota limits on the destination. While blocking, these messages are invisible to the system and are not counted against any of the destination statistics.

In-flight Work Associated with Consumers

The following types of messages are inserted on a destination as a result of in-flight work associated with message consumers.

- Unacknowledged (CLIENT ACK PENDING) Messages : Messages that have been received by a client and are awaiting acknowledgement from the client. These are "pending messages" which are removed from the destination or system when the acknowledgement is received.
- Uncommitted Messages : Messages that have been received by a client within a transaction which has not yet been committed or rolled back. When the client successfully commits the transaction, the messages are removed from the system.
- Rolled-back Messages : Messages that are put back on a destination because of the successful rollback of a transaction.

  These messages might or might not be ready for redelivery to the clients immediately, depending on the redelivery parameters (that is, RedeliveryDelay and/or RedeliveryDelayOverride and RedeliveryLimit) configured on the associated connection factory and destination, or whether rollback requests are internally processed asynchronously. Consequently, a message that is involved in consumption operation subject to a rollback request may not be visible to a consumer `receiveNoWait()` call if the call is made immediately after the rollback request.

  If there is a redelivery delay configured, then, for the duration of that delay, the messages are not available for redelivery and the messages are counted as "pending" in the destination statistics. After the delay period, if the redelivery limit has not been exceeded, then they are delivered and are counted as "current" messages in the destination statistics. If the redelivery limit has been exceeded, then the messages are moved to the error destination, if one has been configured, or are deleted, if no error destination has been configured.

  Rollbacks can affect the order in which messages are processed. A rolled back message can be redelivered after subsequent messages in the same queue or subscription are processed. If strict message ordering is required, see Using Message Unit-of-Order in *Developing JMS Applications for Oracle WebLogic Server*.
- Recovered Messages : Messages that appear on the queue because of an explicit call to session "recover" by the client. These messages are similar to the Rolled-back Messages discussed above.
- Redelivered Messages : Messages that reappear on the destination because of an unsuccessful delivery attempt to the client. These messages are similar to the Rolled-back Messages discussed previously.

Order of Precedence for Boot Time Pause and Resume of Message Operations

You can pause and resume destinations at boot time by setting attributes at several different levels:

- If you are using a JMS server to host a group of destinations, you can pause or resume message operations on the entire group of destinations.
- If you are using a JMS template to define the attribute values of groups of destinations, you can pause or resume message operations on all of the destinations in a group.
- You can pause and resume message operations on a single destination.

If the values at each of these levels are not in agreement at boot-time, the following order of precedence is used to determine the behavior of the message operations on the specified destination(s). For each of the attributes used to configure pausing and resumption of message operations:

1. If the hosting JMS server for the destination has the attribute set with a valid value, then that value determines the state of the destination at boot time. Server-level settings have first precedence.
2. If the hosting JMS server does not have the attribute set with a valid value, then the value of the attribute on the destination level has second highest precedence and determines the state of the destination at boot time.
3. If neither the hosting JMS server nor the destination has the attribute set with a valid value, then the value of the attribute on the JMS template determines the state of the destination at boot time.
4. If the attribute has not been set at any of the three levels, then the value is assumed to be "false".

Security

The administrative user or group can override the current state of a destination irrespective of whether the destination's state is currently being controlled by other users.

If two non-administrative users are trying to control the state of the destination, then the following rules apply.

1. Only a user who belongs to the same group as the user who changed the state of the destination to "paused" is allowed to "resume" the destination to the normal operation.
2. If the state change is attempted by two different users who belong to two different

A

JMS Resource Definition Elements Reference

Oracle WebLogic Server lets you configure JMS resources using the elements for destinations and connections factories. These JMS resource definitions allow an application to be deployed into a Jakarta EE environment with minimal administrative configuration.

> **Note:**
>
> Oracle generally recommends using system modules instead of definitions to define JMS resources. System module configured resources can be added, removed, and modified without requiring a change, rebuild, and redeploy of an application. See Configure a JMS System Module.

The following sections describe the Jakarta Messaging resource definition elements and properties:

- Defining JMS Resources Using Jakarta EE Resource Definitions
- JMS Connection Factory Definition Elements and Properties
- JMS Destination Definition Elements and Properties

Defining JMS Resources Using Jakarta EE Resource Definitions

You can define the JMS resources using the `@JMSConnectionFactoryDefinition` and `@JMSDestinationDefinition` annotations, or the `<jms-destination>` and `<jms-connection-factory>` elements as defined in the Java EE 7 Platform Specification.These JMS resource definitions allow an application to be deployed into a Jakarta EE environment with minimal administrative configuration.

> **Note:**
>
> JMS resources that are configured within the scope of an application cannot be modified after the application is deployed. In addition, there is no direct option to delete destinations that are created in the application module. Undeploying the application does not delete the destination. For these reasons, it is recommended that you configure connection factories and destinations using a JMS system module instead. See JMS System Module Configuration.

- Resource Definitions Using Annotations
- Resource Definitions in the Deployment Descriptor
- Considerations and Best Practices for Using JMS Resource Definitions

Resource Definitions Using Annotations

You can define the annotations `@JMSConnectionFactoryDefinition` and `@JMSDestinationDefinitions` inside the web module, EJB module, or an application client module. You can also define resources using the annotations inside these classes:

- Classes defined in the libraries defined by the <library-directory> element of the `application.xml` deployment descriptor
- Classes in a .jar file referenced by the Class-path entry of a EJB jar file or a .war file

After it is defined, a resource can be referenced by a component using the lookup element of the @Resource annotation or using the look-up element of the resource-ref deployment descriptor element..

Example A-1 Example: JMS Connection Factory Definition

```
@JMSConnectionFactoryDefinition(   name="java:app/MyJMSConnectionFactory",
interfaceName="javax.jms.QueueConnectionFactory").
```

Example A-2 Example: JMS Destination Definition

```
@JMSDestinationDefinition(   name="java:app/MyJMSQueue",
interfaceName="javax.jms.Queue",   destinationName="myQueue1")
```

For more information about the elements and properties that you can use with the resource definitions, see JMS Resource Definition Elements Reference

Resource Definitions in the Deployment Descriptor

Instead of using annotations, you can define resources using the `<jms-destination>` and `<jms-connection-factory>` elements in the deployment descriptor.

JMS Connection Factory Definition

You can define a JMS destination resource using the `jms-connection-factory` element in the `ejb-jar.xml` or `web.xml` deployment descriptors. It creates the connection factory and binds it to the appropriate naming context based on the namespace specified.

The following example defines a connection factory that is bound to JNDI at the location `java:app/MyJMSConnectionFactory`:

```
<jms-connection-factory>   <description>Sample JMS ConnectionFactory
definition</description>   <name>java:app/MyJMSConnectionFactory</name>
<interface-name>javax.jms.QueueConnectionFactory</interface-name>
<user>scott</user>   <password>tiger</password>   <client-id>MyClientId</
client-id>   <property>     <name>Property1</name>     <value>10</value>  </
property>   <property>     <name>Property2</name>     <value>20</value>   </
property>   <transactional>false</transactional>   <max-pool-size>30</max-
pool-size>   <min-pool-size>20</min-pool-size> </jms-connection-factory>
```

For more information about the `jms-connection-factory` element and its attributes, see the schema at http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/javaee_8.xsd

JMS Destination Definition

You can define a JMS destination resource using the `jms-destination` element in the `ejb-jar.xml` or web.xml deployment descriptors. It creates the destination and binds it to the appropriate naming context based on the namespace specified.

The following example defines a queue destination myQueue1 that is bound to JNDI at the location `java:app/MyJMSDestination`:

```
<jms-destination>   <description>JMS Destination definition</description>
<name>java:app/MyJMSDestination</name>   <interface-name>javax.jms.Queue</
interface-name>   <destination-name>myQueue1</destination-name>
<property>     <name>Property1</name>     <value>10</value>   </property>
<property>     <name>Property2</name>     <value>20</value>   </property> </
jms-destination>
```

For more information about the jms-destination element and its attributes, see the schema at http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/javaee_8.xsd

Considerations and Best Practices for Using JMS Resource Definitions

The following are the considerations for resources that are defined using the new annotations:

- Define a JMS resource in any of the following allowed JNDI namespaces:
  - `java:comp`
  - `java:module`
  - `java:app`
  - `java:global`
- When the JNDI name starts with java: but does not start with any of the prior namespace prefixes, that results in an exception during deployment.
- When the JNDI name does not start with `java:`, the destination or the connection factory is defined in the `java:comp/env namespace`. For example, a JNDI name of the format jms/myDestination is considered to be the same as `java:comp/env/jms/myDestination`.
- WebLogic Server allows an application to define a destination in `java:app` and `java:global` namespaces in the `application.xml` deployment descriptor or by using an annotation on a class in the application package other than a class within a web module, an EJB module, or an application client module.
- The destination defined using `@JMSDestinationDefinition` is internally created as uniform distributed destinations (queues or topics). By default, the value of the Forwarding Policy option for a uniform distributed topic that is created using `@JMSDestinationDefinition with interfaceName` of type `javax.jms.Topic` is 'Partitioned'. You can specify the value as 'Replicated' in the application deployment. See Best Practices for Distributed Topics in *Developing JMS Applications for Oracle WebLogic Server.*
- When an application that defines a queue using `@JMSDestinationDefinition` is undeployed, the persistent store preserves the messages that are not received or consumed, and those messages are available for consumers of the destination when you redeploy the application.

For more information, see Annotation Type JMSConnectionFactoryDefinition and Annotation Type JMSDestinationDefinition in *Java(TM) EE7 Specification APIs.*.

JMS Connection Factory Definition Elements and Properties

You can configure the connection factory in an application module using a JMS resource definition.

This section explains the elements of the resource definition as mentioned in *Java(TM) EE7 Specification APIs document.* See Annotation Type JMSConnectionFactoryDefinition.

The following example defines a connection factory using the `@JMSConnectionFactoryDefinition` annotation:

```
@JMSConnectionFactoryDefinition(
    name="java:global/jms/demoConnectionFactory",
    className= "javax.jms.ConnectionFactory",
    description="Sample description for Connection Factory",
    clientId="client_Id01",
    transactional=true,
    properties= {"UserName=myuser", "delivery-mode=Persistent", "load-
balancing-enabled=true"}
```

For the element definitions in the JMS application module, see the weblogic-jms.xsd schema available at http://xmlns.oracle.com/weblogic/weblogic-jms/1.8/weblogic-jms.xsd .

Table A-1 @JMSConnectionFactoryDefinition Elements

Element in @JMSConnectionFactoryDefinition Elements	Equivalent Element in the JMS Application Module	Type	Description
`name`	`jndi-name`	String	The JNDI name of the JMS connection factory that is being defined. **NOTE**: This is a mandatory element.
`description`	`notes`	String	A description of the connection factory.
`clientId`	`clientId`	String	The client ID to be used for the connection.
`transactional`	`transaction-paramsxa-connection-factory-enabled`	Boolean	Specifies whether the connections participate in transactions. Set this property to `false` if connections should not participate in transactions. . The default is `True`.
`resourceAdapter`	None	String	The name of the resource adapter.
`className`	None	String	The value is ignored.
`InterfaceName`	None	String	The value is ignored.
`maxPoolSize`	None	integer	The value is ignored.
`minPoolSize`	None	integer	The value is ignored.
`user`	None	String	The value is ignored.

Table A-1 (Cont.) @JMSConnectionFactoryDefinition Elements

Element in @JMSConnectionFactoryDefinition Elements	Equivalent Element in the JMS Application Module	Type	Description
`password`	None	String	The value is ignored.

JMS Connection Factory Properties

> **Note:**
>
> The properties defined in this section apply only when the connection factory definition does not specify an adapter. When an adapter is specified, properties that are unique to the adapter are considered.

Table A-2 @JMSConnectionFactoryDefinition Properties

@JMSConnectionFactoryDefinition Property	Equivalent Element in the JMS Application Module	Type	Description
`name`	`name`	Sting	A name for the connection factory. When not specified, the name is `<module_name>_cf`, where `<module_name>`represents the name of the JMS application module where the resource is defined.
`jms-server-name`	`sub-deployment-name`	Sting	The JMS server to which the resource is targeted. If the `jms-server-name` specified by the application does not exist in the scope of the deployment, then the deployment fails with an exception indicating the reason for the failure.
`default-targeting-enabled`	`default-targeting-enabled`	boolean	Set this property to `true` when `jms-server-name` is not specified. This value is ignored when `jms-server-name` is specified.

Table A-3 @JMSConnectionFactoryDefinition Properties for Default Message Delivery

@JMSConnectionFactoryDefinition Property	Equivalent default-delivery-params Element in the JMS Application Module	Type	Description
`default-delivery-mode`	`default-delivery-mode`	String	The delivery mode assigned to all messages sent by a producer using the connection factory that is being defined. Accepted values: `Persistent` or `Non-persistent`
`default-time-to-deliver`	`default-time-to-deliver`	String	The delay time, in milliseconds, between when a message is produced and when it is made visible on its destination
`default-time-to-live`	`default-time-to-live`	Long	The maximum length of time, in milliseconds, that a message exists. This value is used for messages when a priority is not explicitly defined.
`default-priority`	`default-priority`	integer	The default priority used for messages when a priority is not explicitly defined.
`default-redelivery-delay`	`default-redelivery-delay`	Long	The delay time, in milliseconds, before rolled back or recovered messages are redelivered.
`default-compression-threshold`	`default-compression-threshold`	integer	The number of bytes for a serialized message body so any message that exceeds this limit triggers message compression when the message is sent or received by the JMS message producer or consumer
`default-unit-of-order`	`default-unit-of-order`	String	The default Unit-of-Order producer for this connection factory. Options are: System-generated and User-Generated.
`send-timeoutr`	`send-timeoutr`	Long	The maximum length of time, in milliseconds, that a sender waits when there is not enough available space (no quota) on a destination to accommodate the message being sent.

Table A-4 @JMSConnectionFactoryDefinition Client Properties

@JMSConnectionFactoryDefinition Property	Equivalent client-params element in the JMS Application Module	Type	Description
`client-id-policy`	`client-id-policy`	String	Whether more than one JMS connection can use the same client ID. Valid options are `Restricted` and `Unrestricted`.

Table A-4 (Cont.) @JMSConnectionFactoryDefinition Client Properties

@JMSConnectionFactoryDefinition Property	Equivalent client-params element in the JMS Application Module	Type	Description
`subscription-sharing-policy`	`subscription-sharing-policy`	String	The Subscription Sharing Policy on the connection. Valid options are `Exclusive` and `Sharable`.
`acknowledge-policy`	`acknowledge-policy`	String	The client acknowledge policy for non transacted sessions that use the `CLIENT_ACKNOWLEDGE` mode.Valid options are `All` and `Previous`.
`allow-close-in-onMessage`	`allow-close-in-onMessage`	Boolean	Whether the connection factory creates message consumers that allow a `close`() method to be issued within its `onMessage`() method call. **NOTE**: The default value of this property is `false` for a connection factory that is created by using the JMS Connection Factory Definition. If connection factory is created in a JMS system module or in a application-scoped module, the default value of `allow-close-in-onMessage` will be `true`.
`messages-maximum`	`messages-maximum`	int	The maximum number of messages that can exist for an asynchronous session and that have not yet been passed to the message listener
`multicast-overrun-policy`	`multicast-overrun-policy`	String	The policy to be used when the number of outstanding multicast messages reaches the value specified in `messages-maximum` and some messages must be deleted. Valid options are `KeepOld` and `KeepNew`.
`synchronous-prefetch-mode`	`synchronous-prefetch-mode`	String	Whether a synchronous consumer will prefetch messages (that is, messages sent from the server to the client) in one server access. Valid options are `enabled`, `disabled` or `topicSubscriberOnly`..

Table A-4 (Cont.) @JMSConnectionFactoryDefinition Client Properties

@JMSConnectionFactoryDefinition Property	Equivalent client-params element in the JMS Application Module	Type	Description
`reconnect-policy`	`reconnect-policy`	String	Which types of JMS clients are explicitly and implicitly refreshed after a lost network connection with a server or upon a server restart. Valid options are `none`, `producer` or `all`.
`reconnect-blocking-millis`	`reconnect-blocking-millis`	Long	The maximum length of time, in milliseconds, that any synchronous JMS calls block the calling thread before giving up on a JMS client reconnect in progress.
`total-reconnect-period-millis`	`total-reconnect-period-millis`	Long	The maximum length of time, in milliseconds, that JMS clients (particularly asynchronous consumers) continue to try to reconnect to the server after either the initial network disconnect or the last synchronous call attempt, whichever occurred most recently, before giving up retrying.

Table A-5 @JMSConnectionFactoryDefinition Transaction Properties

@JMSConnectionFactoryDefinition Property	Equivalent transaction-params element in the JMS Application Module	Type	Description
`transaction-timeout`	`transaction-timeout`	Long	The timeout value (in seconds) for all transactions on connections created with the connection factory. Set this property only when the `transactional` element is set to `true`.

Table A-6 @JMSConnectionFactoryDefinition Flow Control Properties

@JMSConnectionFactoryDefinition Property	Equivalent flow-control-params Element in the JMS Application Module	Type	Description
`flow-minimum`	`flow-minimum`	integer	The maximum number of messages-per-second allowed for a producer that is experiencing a threshold condition. When a producer is flow controlled it is never be allowed to go faster than the FlowMaximum messages per second.
`flow-maximum`	`flow-maximum`	integer	The minimum number of messages-per-second allowed for a producer that is experiencing a threshold condition. This is the lower boundary of a producer's flow limit. That is, WebLogic JMS does not further slow down a producer whose message flow limit is at its FlowMinimum.
`flow-interval`	`flow-interval`	integer	The adjustment period of time, in seconds, when a producer adjusts its flow from the FlowMaximum number of messages to the FlowMinimum amount, or reverse.
`flow-steps`	`flow-steps`	integer	The number of steps used when a producer is adjusting its flow from the Flow Maximum amount of messages to the Flow Minimum amount, or vice versa.
`flow-control-enabled`	`flow-control-enabled`	Boolean	Whether a producer created using a connection factory allows flow control.
`one-way-send-mode`	`one-way-send-mode`	String	Whether message producers created using this connection factory are allowed to do one-way message sends to improve typical non-persistent, non-transactional messaging performance. Valid options are `enabled`, `disabled`, or `topicOnly`.

Table A-6 (Cont.) @JMSConnectionFactoryDefinition Flow Control Properties

@JMSConnectionFactoryDefinition Property	Equivalent flow-control-params Element in the JMS Application Module	Type	Description
`one-way-send-window-size`	`one-way-send-window-size`	integer	The maximum number of sent messages per window when One-Way Send Mode is set to allow queue senders and/or topic publishers to make `one-way` sends. The window size determines when a two-way message is required to regulate the producer before it can continue making additional one-way sends.

Table A-7 @JMSConnectionFactoryDefinition Load Balancing Properties

@JMSConnectionFactoryDefinition Property	Equivalent load-balancing-params Element in the JMS Application Module	Type	Description
`load-balancing-enabled`	`load-balancing-enabled`	Boolean	Whether non-anonymous producers created through a connection factory are load balanced within a distributed destination on a per-call basis.
`server-affinity-enabled`	`server-affinity-enabled`	Boolean	Whether a server instance that is load balancing consumers or producers across multiple members destinations of a distributed destination, first attempts to load balance across any other physical destinations that are also running on the same server instance.

Table A-8 @JMSConnectionFactoryDefinition Load Balancing Properties

@JMSConnectionFactoryDefinition Property	Equivalent security-params Element in the JMS Application Module	Type	Description
`attach-jmsx-user-id`	`attach-jmsx-user-id`	Boolean	Whether non-anonymous producers created through a connection factory are load balanced within a distributed destination on a per-call basis.

JMS Destination Definition Elements and Properties

You can configure a JMS destination resource in an application module by using a JMS resource definition.

This section describes the elements of the resource definition as mentioned in Defining JMS Resources Using Jakarta EE Resource Definitions.

The following example defines a destination using the `@JMSDestinationDefinition` annotation:

```
@JMSDestinationDefinition(
name="java:global/jms/demoDestination",
interfaceName="javax.jms.Queue",
className= "javax.jms.Queue",
description="Sample description for Queue",
destinationName="myQueue",
properties= {"default-unit-of-order=true", "time-to-deliver=Persistent",
"attach-sender=always"}
```

For more information, see Annotation Type JMSDestinationDefinition.

For the element definitions in the JMS application module, see the http://xmlns.oracle.com/weblogic/weblogic-jms/1.8/weblogic-jms.xsd

Table A-9 @JMSDestinationDefinition Elements

Element in @JMSDestinationDefinition	Equivalent Element in the JMS Application Module	Type	Description
`name`	`jndi-name`	String	The JNDI name of the JMS destination resource that is being defined. **NOTE**: This is a mandatory element.
`interfaceName`	`uniform-distributed-queue` or `uniform-distributed-topic`.	String	The fully qualified name of the JMS destination interface. Valid options are `javax.jms.Queue` or `javax.jms.Topic`. The JMS destination definition is converted to a JMS module with entity of type `uniform-distributed-queue` or `uniform-distributed-topic` based on the value specified for `interfaceName`. **NOTE**: This is a mandatory element.
`description`	`notes`	String	A description for the JMS destination that is being defined.

Table A-9 (Cont.) @JMSDestinationDefinition Elements

Element in @JMSDestinationDefinition	Equivalent Element in the JMS Application Module	Type	Description
`destination name`	`name`	String	The runtime MBean name of the queue or topic. When not specified, the destination name is `<module_name>_<queue/topic>`, where `<module_name>` represents the name of the JMS application module where the resource is defined.
`className`	None	-	This value is ignored.
`resourceAdapter`	None	-	This value is ignored.

Table A-10 @JMSDestinationDefinition Properties

@JMSDestinationDefinition Property	Equivalent Element in the JMS Application Module	Type	Description
`jms-server-name`	`sub-deployment-name`	String	The JMS server to which the resource is targeted. If the `jms-server-name` specified by the application does not exist in the scope of the deployment, then the deployment will fail with an exception indicating the reason for the failure.
`default-targeting-enabled`	`default-targeting-enabled`	Boolean	Set this property to `true` when `jms-server-name` is not specified. This value is ignored when `jms-server-name` is specified.

Table A-11 @JMSDestinationDefiition Threshold Properties

@JMSDestinationDefinition Property	Equivalent thresholds Element in the JMS Applicaton Module	Type	Description
`bytes-high`	`bytes-high`	Long	The upper threshold (total number of bytes in this destination) that triggers logging or flow control events.
`bytes-low`	`bytes-low`	Long	The lower threshold (total number of bytes in this destination) that triggers logging or flow control events.

Table A-11 (Cont.) @JMSDestinationDefiition Threshold Properties

@JMSDestinationDefinition Property	Equivalent thresholds Element in the JMS Applicaton Module	Type	Description
`messages-high`	`messages-high`	Long	The upper threshold (total number of messages in this destination) that triggers logging or flow control events.
`messages-low`	`messages-low`	Long	The lower threshold (total number of messages in this destination) that triggers logging or flow control events.

Table A-12 @JMSDestinationDefinition Message Delivery Override Properties

@JMSDestinationDefinition Property	Equivalent delivery-params-overrides Element in the JMS Application Module	Type	Description
`delivery-mode`	`delivery-mode`	String	The delivery mode assigned to all messages that arrive at the destination regardless of the DeliveryMode specified by the message producer. Valid options are `Persistent` or `Non-Persistent`.
`time-to-deliver`	`time-to-deliver`	String	The delivery delay, either in milliseconds or as a schedule, between when a message is produced and when it is made visible on its target distributed destination.
`time-to-live`	`time-to-live`	Long	The time-to-live assigned to all messages that arrive at the destination, regardless of the TimeToLive value specified by the message producer.
`priority`	`priority`	Integer	The priority assigned to all messages that arrive at the destination, regardless of the Priority specified by the message producer.
`redelivery-delay`	`redelivery-delay`	Long	The delay, in milliseconds, before rolled back or recovered messages are redelivered, regardless of the RedeliveryDelay specified by the consumer and/or connection factory.

Table A-13 @JMSDestinationDefinition Message Delivery Failure Properties

@JMSDestinationDefinition Property	Equivalent delivery-failure params Element in the JMS Application Module	Type	Description
`redelivery-limit`	`redelivery-limit`	Integer	The number of redelivery tries a message can have before it is moved to the error destination.
`expiration-policy`	`expiration-policy`	String	The message Expiration Policy to be used when an expired message is encountered on a destination. Valid options are Discard or `Log` or `Redirect`.
`expiration-logging-policy`	`expiration-logging-policy`	String	The information about the message is logged when the Expiration Policy is set to `Log`.

Table A-14 @JMSDestinationDefinition Message Logging Properties

@JMSDestinationDefinition Property	Equivalent message-logging-params Element in the JMS Application Module	Type	Description
`message-logging-enabled`	`message-logging-enabled`	Boolean	Whether the module logs information about the message life cycle.
`message-logging-format`	`message-logging-format`	String	The information about the message is logged. This property is defined as shown in the following example: For more information about the valid values for this property, see the description for the `MessageLoggingFormat` attribute of `MessageLoggingParamsBean` in *MBean Reference for Oracle WebLogic Server.*

Table A-15 @JMSDestinationDefinition Advanced Configuration Properties

@JMSDestinationDefinition Property	Equivalent Element in the JMS Application Module	Type	Description
`load-balancing-policy`	`load-balancing-policy`	String	How messages are distributed to the members of this destination Valid options are `Round-Robin` and `Random`

Table A-15 (Cont.) @JMSDestinationDefinition Advanced Configuration Properties

@JMSDestinationDefinition Property	Equivalent Element in the JMS Application Module	Type	Description
`production-paused-at-startup`	`production-paused-at-startup`	Boolean	Whether new message production is paused on a destination at startup
`insertion-paused-at-startup`	`insertion-paused-at-startup`	Boolean	Whether new message insertion is paused on a destination at startup.
`consumption-paused-at-startup`	`consumption-paused-at-startup`	Boolean	Whether consumption is paused on a destination at startup.
`default-unit-of-order`	`default-unit-of-order`	Boolean	Specifies whether WebLogic Server creates a system-generated unit-of-order name based on the domain, JMS server, and destination name.
`unit-of-order-routing`	`unit-of-order-routing`	String	Determines how a distributed destination member is selected as the destination for a message that is part of a unit-of-order. Valid options are `Hash` and `PathService`.
`attach-sender`	`attach-sender`	String	Whether messages landing on this destination should attach the credential of the sending user. Valid options are supports, `always`, and `never`.
`jms-create-destination-identifier`	`jms-create-destination-identifier`	String	A reference name for a destination or a member of a distributed destination that provides a way to look-up that destination without JNDI using `javax.jms.Session createQueue or createTopic.`
`saf-export-policy`	`saf-export-policy`	String	Whether a user can send messages to a destination using Store-and-Forward. Valid options are `All` and `None`.
`messaging-performance-preferenc`	`messaging-performance-preferenc`	Integer	How long destinations are willing to wait to create full batches of available messages (if at all) for delivery to consumers.
`unit-of-work-handling-policy`	`unit-of-work-handling-policy`	String	Whether the Unit-of-Work (UOW) feature is enabled for this destination. Valid options are `PassThrough` and `SingleMessageDelivery`.

Table A-15 (Cont.) @JMSDestinationDefinition Advanced Configuration Properties

@JMSDestinationDefinition Property	Equivalent Element in the JMS Application Module	Type	Description
`incomplete-work-expiration-time`	`incomplete-work-expiration-time`	Integer	The maximum length of time, in milliseconds, before undelivered messages in an incomplete UOW are expired.

> **Note:**
>
> These properties are applicable only when the `interfaceName` element of `@JMSDestinationDefinition` is set to `javax.jms.Queue`.

Table A-16 @JMSDestinationDefinition Properties for Queue Destinations

@JMSDestinationDefinition Property	Equivalent Element in the JMS Application Module	Type	Description
`forward-delay`	`forward-delay`	Integer	The number of seconds after which a uniform distributed queue member with no consumers wait before forwarding its messages to other uniform distributed queue members that do have consumers.
`reset-delivery-count-on-forward`	`reset-delivery-count-on-forward`	Boolean	Whether or not the delivery count is reset during message forwarding between distributed queue members.

> **Note:**
>
> These properties are applicable only when the `interfaceName` element of `@JMSDestinationDefinition` is set to `javax.jms.Topic`.

Table A-17 @JMSDestinationDefinition Properties for Topic Destinations

@JMSDestinationDefinition Property	Equivalent Element in the JMS Application Module	Type	Description
`forwarding-policy`	`forwarding-policy`	String	The uniform distributed topic message Forwarding Policy specifies whether or not a sent message is forwarded to all members. Valid options are `Partitioned` and `Replicated`. When a destination is created by using JMS destination definition, the default value of this property is `Partitioned`. The default value is `Replicated` if the destination is created otherwise.
`multicast-address`	`multicast-address`	String	The address used by the topic to transmit messages to multicast consumers.
`multicast-time-to-live`	`multicast-time-to-live`	Integer	The Time-To-Live value used for multicasting, which specifies the number of routers that the message can traverse en route to the consumers.
`multicast-port`	`multicast-port`	Integer	The port used by the topic to transmit messages to multicast consumers.

B

Configuring JMS Application Modules for Deployment (Deprecated)

Learn how to configure JMS application modules for deployment in Oracle WebLogic Server. This includes JMS application modules packaged with a Java EE enterprise application and globally-available, standalone application modules.

> **Note:**
>
> WebLogic JMS Application Modules for Deployment are deprecated, including packaged and standalone modules. Support for JMS Application Modules will be removed in a future release. Oracle recommends creating required JMS configuration using system modules.

This chapter includes the following sections:

- Methods for Configuring JMS Application Modules
  All JMS resources that can be configured in a JMS system module can be configured and managed as deployable application modules, similar to standard Java EE modules.
- JMS Schema
- Packaging JMS Application Modules In an Enterprise Application
  JMS application modules can be packaged as part of an Enterprise Application Archive (EAR), as a *packaged module*. Packaged modules are bundled with an EAR or exploded EAR directory, and are referenced in the `weblogic-application.xml` descriptor.
- Deploying Standalone JMS Application Modules
- Generating Unique Runtime JNDI Names for JMS Resources
  JMS resources, such as connection factories and destinations, are configured with a JNDI name. The runtime implementations of these resources are then bound into JNDI using the given names. WebLogic Server facilitates to generate the JNDI name dynamically instead of using a static JNDI name for these resources.

Methods for Configuring JMS Application Modules

All JMS resources that can be configured in a JMS system module can be configured and managed as deployable application modules, similar to standard Java EE modules.

> **Note:**
>
> JMS resources that are configured within the scope of an application cannot be modified after the application is deployed. In addition, there is no direct option to delete destinations that are created in the application module. Undeploying the application does not delete the destination. For these reasons, it is recommended that you configure connection factories and destinations using a JMS system module instead. See JMS System Module Configuration.

Deployed JMS application modules are owned by the developer who created and packaged the module, rather than the administrator who deploys the module; therefore, the administrator has more limited control over deployed resources.

For example, administrators can modify (override) only certain properties of the resources specified in the module using the deployment plan (JSR-88) at the time of deployment, but they cannot dynamically add or delete resources. As with other Java EE modules, configuration changes for an application module are stored in a deployment plan for the module, leaving the original module untouched.

Application developers can use these tools to create and deploy (target) system resources:

- Create a JMS system module, as described in JMS System Module Configuration and then copy the resulting XML file to another directory and rename it, using `-jms.xml` as the file suffix.
- Create application modules in an enterprise-level IDE or another development tool that supports editing of XML files, then package the JMS modules with an application and pass the application to a WebLogic Administrator to deploy.
- Use Java EE connection factory and destination dDefinitions which implicitly create an application module based on the source code annotations. See Defining JMS Resources Using Java EE Resource Definitions.

JMS Schema

In support of the modular deployment model for JMS resources in WebLogic Server 9.*x* or higher, Oracle provides a schema for defining WebLogic JMS resources: `weblogic-jms.xsd`. When you create JMS modules (descriptors), the modules must conform to this schema. IDEs and other tools can validate JMS modules based on the schema. The `weblogic-jms.xsd` schema is available online at `http://xmlns.oracle.com/weblogic/weblogic-jms/1.7/weblogic-jms.xsd` .
For an explanation of the JMS resource definitions in the schema, see the corresponding system module beans in the System Module MBeans folder of the *MBean Reference for Oracle WebLogic Server*. The root bean in the JMS module that represents an entire JMS module is named JMSBean.

Packaging JMS Application Modules In an Enterprise Application

JMS application modules can be packaged as part of an Enterprise Application Archive (EAR), as a *packaged module*. Packaged modules are bundled with an EAR or exploded EAR directory, and are referenced in the `weblogic-application.xml` descriptor.

The packaged JMS module is deployed along with the Enterprise Application, and the resources defined in this module can optionally be made available only to the enclosing application (i.e., as an *application-scoped* resource). Such modules are particularly useful

when packaged with EJBs (especially MDBs) or Web Applications that use JMS resources. Using packaged modules ensures that an application always has the required resources and simplifies the process of moving the application into new environments.

Creating Packaged JMS Application Modules

You create packaged JMS modules using an enterprise-level IDE or another development tool that supports editing of XML descriptor files. You then deploy and manage standalone modules using JSR 88-based tools, such as the `weblogic.Deployer` utility.

Packaged JMS Application Module Requirements

Inside the EAR file, a JMS module must meet the following criteria:

- Conforms to the `http://xmlns.oracle.com/weblogic/weblogic-jms/1.7/weblogic-jms.xsd` schema
- Uses `-jms.xml` as the file suffix (for example, *`MyJMSDescriptor`*`-jms.xml`)
- Uses a name that is unique within the WebLogic domain and a path that is relative to the root of the Java EE application

Main Steps for Creating Packaged JMS Application Modules

To configure a packaged JMS module:

1. If necessary, create a JMS server to target the JMS module to.
2. Create a JMS system module and configure the necessary resources, such as queues or topics.
3. The system module is saved in the `config\jms` subdirectory of the domain directory, with a "`-jms.xml`" suffix.
4. Copy the system module to a new location, and then:
   a. Give the module a unique name within the domain namespace.
   b. Delete the `JNDI-Name` attribute to make the module *application-scoped* to only the application.
5. Add references to the JMS resources in the module to all applicable Java EE application component's descriptor files, as described in Referencing a Packaged JMS Application Module In Deployment Descriptor Files in *Developing JMS Applications for Oracle WebLogic Server*.
6. Package all application modules in an EAR, as described in Packaging an Enterprise Application With a JMS Application Module.
7. Deploy the EAR, as described in Deploying a Packaged JMS Application Module.

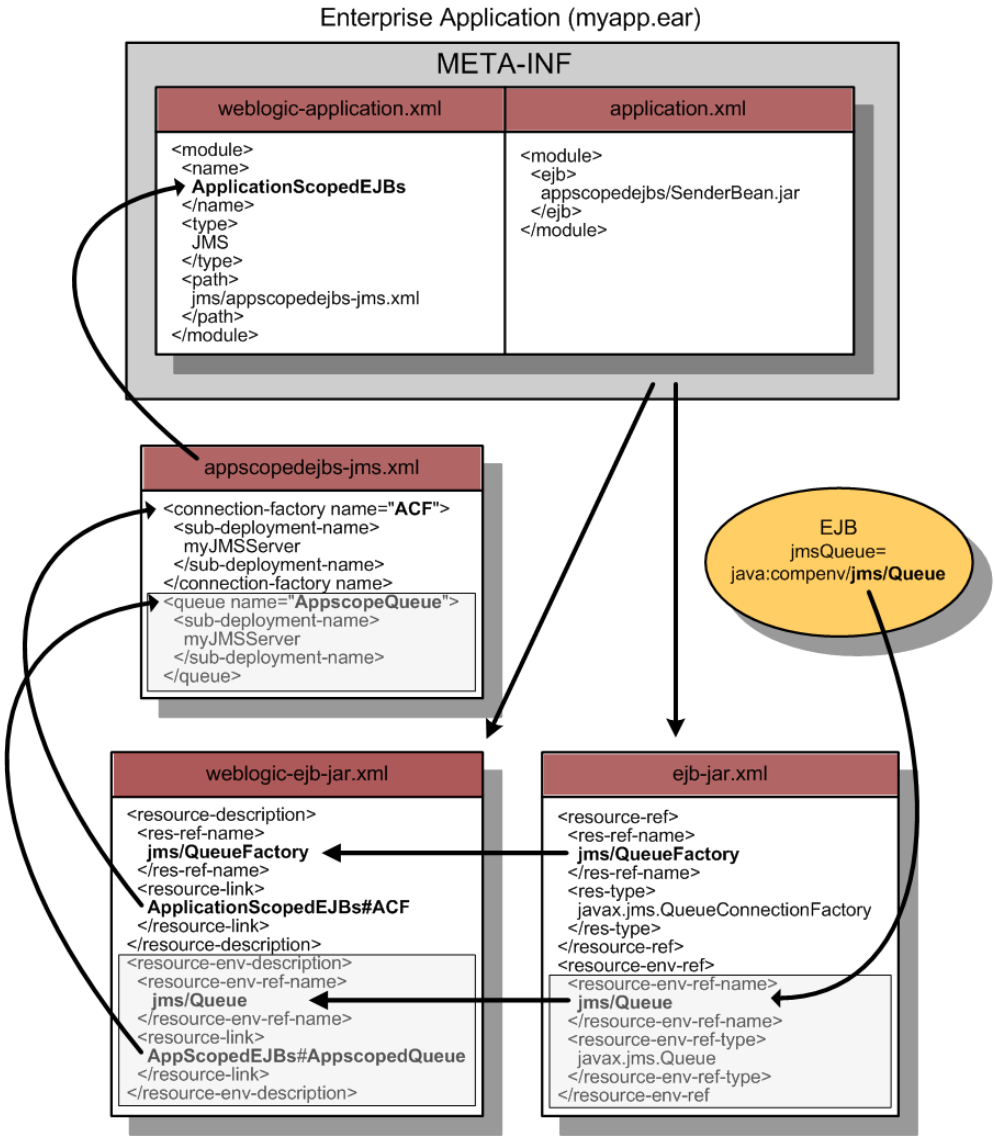
Sample of a Packaged JMS Application Module in an EJB Application

The following code snippet is an example of the packaged JMS module, `appscopedejbs-jms.xml`, referenced by the descriptor files in Figure B-1 below.

```
<weblogic-jms xmlns="http://xmlns.oracle.com/weblogic/weblogic-jms">
  <connection-factory name="ACF">
  </connection-factory>
  <queue name="AppscopeQueue">
  </queue>
</weblogic-jms>
```

Figure B-1 illustrates how a JMS connection factory and queue resources in a packaged JMS module are referenced in an EJB EAR file.

Figure B-1 Relationship Between a JMS Application Module and Descriptors in an EJB Application



- Packaged JMS Application Module References In weblogic-application.xml
- Packaged JMS Application Module References In ejb-jar.xml
- Packaged JMS Application Module References In weblogic-ejb-jar.xml

Packaged JMS Application Module References In weblogic-application.xml

When including JMS modules in an enterprise application, you must list each JMS module as a module element of type JMS in the `weblogic-application.xml` descriptor file packaged with the application, and a path that is relative to the root of the application. For example:

```
<module>
  <name>AppScopedEJBs</name>
  <type>JMS</type>
  <path>jms/appscopedejbs-jms.xml</path>
</module>
```

Packaged JMS Application Module References In ejb-jar.xml

If EJBs in your application use connection factories through a JMS module packaged with the application, you must list the JMS module as a `res-ref` element and include the `res-ref-name` and `res-type` parameters in the `ejb-jar.xml` descriptor file packaged with the EJB. This way, the EJB can lookup the JMS Connection Factory in the application's local context. For example:

```
<resource-ref>
  <res-ref-name>jms/QueueFactory</res-ref-name>
  <res-type>javax.jms.QueueConnectionFactory</res-type>
</resource-ref>
```

The `res-ref-name` element maps the resource name (used by `java:comp/env`) to a module referenced by an EJB. The `res-type` element specifies the module type, which in this case, is `javax.jms.QueueConnectionFactory`.

If EJBs in your application use Queues or Topics through a JMS module packaged with the application, you must list the JMS module as a `resource-env-ref` element and include the `resource-env-ref-name` and `resource-env-ref-type` parameters in the `ejb-jar.xml` descriptor file packaged with the EJB. This way, the EJB can lookup the JMS Queue or Topic in the application's the local context. For example:

```
<resource-env-ref>
  <resource-env-ref-name>jms/Queue</resource-env-ref-name>
  <resource-env-ref-type>javax.jms.Queue</resource-env-ref-type>
</resource-env-ref>
```

The `resource-env-ref-name` element maps the destination name to a module referenced by an EJB. The `res-type` element specifies the name of the Queue, which in this case, is `javax.jms.Queue`.

Packaged JMS Application Module References In weblogic-ejb-jar.xml

You must list the referenced JMS module as a `res-ref-name` element and include the `resource-link` parameter in the `weblogic-ejb-jar.xml` descriptor file packaged with the EJB.

```
<resource-description>
  <res-ref-name>jms/QueueFactory</res-ref-name>
  <resource-link>AppScopedEJBs#ACF</resource-link>
</resource-description>
```

The `res-ref-name` element maps the connection factory name to a module referenced by an EJB. `In the resource-link` element, the JMS module name is followed by a pound (#) separator character, which is followed by the name of the resource inside the module. So for

this example, the JMS module *AppScopedEJBs* containing the connection factory *ACF*, would have a name *AppScopedEJBs#ACF*.

Continuing the example above, the `res-ref-name` element also maps the Queue name to a module referenced by an EJB. And in the `resource-link` element, the queue *AppScopedQueue*, would have a name *AppScopedEJBs#AppScopedQueue*, as follows:

```
<resource-env-description>
  <resource-env-ref-name>jms/Queue</resource-env-ref-name>
  <resource-link>AppScopedEJBs#AppScopedQueue</resource-link>
</resource-env-description>
```

Packaging an Enterprise Application With a JMS Application Module

You package an application with a JDBC module as you would any other enterprise application. See Packaging Applications Using wlpackage in *Developing Applications for Oracle WebLogic Server*.

Deploying a Packaged JMS Application Module

The deployment of packaged JMS modules follows the same model as all other components of an application: individual modules can be deployed to a single server, a cluster, or individual members of a cluster.

A recommended best practice for other application components is to use the `java:comp/env` JNDI environment to retrieve references to JMS entities, as described in Referencing a Packaged JMS Application Module in Deployment Descriptor Files in *Developing JMS Applications for Oracle WebLogic Server*. (However, this practice is not required.)

By definition, packaged JMS modules are included in an enterprise application, and therefore are deployed when you deploy the enterprise application. For more information about deploying applications with packaged JMS modules, see Deploying Applications Using wldeploy in *Developing Applications for Oracle WebLogic Server*.

Deploying Standalone JMS Application Modules

You can deploy and manage the standalone JMS application modules using the `weblogic.Deployer` utility.This section describes how to create, deploy, and manage standalone JMS application modules:

About Standalone JMS Modules

A JMS application module can be deployed by itself as a *standalone module*, in which case the module is available to the server or cluster targeted during the deployment process. JMS modules deployed in this manner can be reconfigured using the `weblogic.Deployer` utility but are not available through JMX or WLST.

However, standalone JMS modules are available using the basic JSR-88 deployment tool provided with WebLogic Server plug-ins (without using WebLogic Server extensions to the API) to configure, deploy, and redeploy Java EE applications and modules to WebLogic Server. For information about WebLogic Server deployment, see Understanding WebLogic Server Deployment in *Deploying Applications to Oracle WebLogic Server*.

JMS modules deployed in this manner are called *standalone modules*. Depending on how they are targeted, the resources inside standalone JMS modules are globally available in a cluster or locally on a server instance. Standalone JMS modules promote sharing and portability of JMS resources. You can create a JMS module and distribute it to other developers. Standalone JMS modules can also be used to move JMS information between domains, such as between the development domain and the production domain, without extensive manual JMS reconfiguration.

Creating Standalone JMS Application Modules

You can create JMS standalone modules using an enterprise-level IDE or another development tool that supports editing XML descriptor files. You then deploy and manage standalone modules using WebLogic Server tools, such as the `weblogic.Deployer` utility.

- Standalone JMS Application Module Requirements
- Main Steps for Creating Standalone JMS Application Modules

Standalone JMS Application Module Requirements

A standalone JMS module must meet the following criteria:

- Conforms to the `http://xmlns.oracle.com/weblogic/weblogic-jms/1.7/weblogic-jms.xsd` schema
- `Uses "-jms.xml"` as the file suffix (for example, *`MyJMSDescriptor`*`-jms.xml`)
- Uses a name that is unique within the WebLogic domain (cannot conflict with JMS system modules)

Main Steps for Creating Standalone JMS Application Modules

To configure a standalone JMS module:

1. If necessary, create a JMS server to which to target the JMS module.
2. Create a JMS system module and configure the necessary resources, such as queues or topics.
3. The system module is saved in the `config\jms` subdirectory of the domain directory, with a `-jms.xml` suffix.
4. Copy the system module to a new location and then:
   a. Give the module a unique name within the domain namespace.
   b. To make the module *globally available*, uniquely rename the `JNDI-Name` attributes of the resources in the module.
   c. If necessary, modify any other values that can be tuned , such as destination thresholds or connection factory flow control parameters.
5. Deploy the module, as described in Deploying Standalone JMS Application Modules.

Sample of a Simple Standalone JMS Application Module

The following code snippet is an example of simple standalone JMS module.

```
<weblogic-jms xmlns="http://xmlns.oracle.com/weblogic/weblogic-jms">
  <connection-factory name="exampleStandAloneCF">
    <jndi-name>exampleStandAloneCF</jndi-name>
  </connection-factory>
  <queue name="ExampleStandAloneQueue">
    <jndi-name>exampleStandAloneQueue</jndi-name>
  </queue>
</weblogic-jms>
```

Deploying Standalone JMS Application Modules

The command line for using the `weblogic.Deployer` utility to deploy a standalone JMS module (using the example above) would be:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
-password weblogic \
-name ExampleStandAloneJMS \
-targets examplesServer \
-submoduletargets
ExampleStandaloneQueue@examplesJMSServer,ExampleStandaloneCF@examplesServer \
-deploy ExampleStandAloneJMSModule-jms.xml
```

For information about deploying standalone JMS modules, see Deploying JDBC, JMS, and WLDF Application Modules in *Deploying Applications to Oracle WebLogic Server*.

When you deploy a standalone JMS module, an `app-deployment` entry is added to the `config.xml` file for the domain. For example:

```
<app-deployment>
  <name>standalone-examples-jms</name>
  <target>MedRecServer</target>
  <module-type>jms</module-type>
  <source-path>C:\modules\standalone-examples-jms.xml</source-path>
  <sub-deployment>
  ...
  </sub-deployment>
  <sub-deployment>
  ...
  </sub-deployment>
</app-deployment>
```

Note that the `source-path` for the module can be an absolute path or it can be a relative path from the *`domain`* directory. This differs from the `descriptor-file-name` path for a system resource module, which is relative to the *`domain`*`\config` directory.

Tuning Standalone JMS Application Modules

JMS resources deployed within *standalone modules* can be reconfigured using the `weblogic.Deployer` utility, as long as the resources are considered bindable (such as JNDI names), or tunable (such as destination thresholds). However, standalone resources are not available through WebLogic JMX APIs or WebLogic Scripting Tool (WLST).

However, standalone JMS modules are available using the basic JSR-88 deployment tool provided with WebLogic Server plug-ins (without using WebLogic Server extensions to the API)

to configure, deploy, and redeploy Java EE applications and modules to WebLogic Server. For information about WebLogic Server deployment, see Understanding WebLogic Server Deployment in *Deploying Applications to Oracle WebLogic Server*.

Additionally, standalone resources cannot be dynamically added or deleted with any WebLogic Server utility and must be redeployed.

Generating Unique Runtime JNDI Names for JMS Resources

JMS resources, such as connection factories and destinations, are configured with a JNDI name. The runtime implementations of these resources are then bound into JNDI using the given names. WebLogic Server facilitates to generate the JNDI name dynamically instead of using a static JNDI name for these resources.

In some cases, it is impossible or inconvenient to provide a static JNDI name for these resources. An example of such a situation is when JMS resources are defined in a JMS module within an application library. In this case, the library can be referenced from multiple applications, each of which receives a copy of the application library (and the JMS module it contains) when they are deployed. If you were to use static JNDI names for the JMS resources in this case, then all applications that refer to the library would attempt to bind the same set of JNDI resources at the same static JNDI name.

Therefore, the first application to deploy successfully binds the JMS resources into JNDI, but subsequent application deployments fail with exceptions indicating that the JNDI names are already bound.

To avoid this problem, WebLogic Server provides a facility to dynamically generate a JNDI name for the following types of JMS resources:

- Connection factory
- Destination (queue and topic)
- Weighted distributed destination (deprecated)
- Weighted distributed destination members
- Uniform distributed destination

The facility to generate unique names is based on placing a special character sequence called ${APPNAME} in the JNDI name of the previously mentioned JMS resources. If you include ${APPNAME} in the JNDI name element of a JMS resource (either in the JMS module descriptor, or the `weblogic-ejb-jar.xml` descriptor), then the actual JNDI name used at runtime will have the ${APPNAME} string replaced with the effective application ID (name and possibly version) of the application hosting the JMS resource.

> **Note:**
>
> The ${APPNAME} facility does not imply that you can define your own variables and substitute their values into the JNDI name at runtime. The string `${APPNAME}` is treated specially by the JMS implementation, and no other strings of the form `${<some name>}` have any special meaning.

- Unique Runtime JNDI Name for Local Applications
- Unique Runtime JNDI Name for Application Libraries

Unique Runtime JNDI Name for Local Applications

In the case of JMS modules in a local application, at runtime ${`APPNAME`} becomes the name or ID of the application. For example:

```
<jndi-name>${APPNAME}/jms/MyConnectionFactory</jndi-name>
```

When deployed within an application called *`MyApp`*, it would result in a runtime JNDI name of:

```
MyApp/jms/MyConnectionFactory
```

Unique Runtime JNDI Name for Application Libraries

In the case of JMS modules in an application library, at runtime ${APPNAME} becomes the name/ID of the application which refers to the library (not the name of the library). For example:

```
<jndi-name>${APPNAME}/jms/MyConnectionFactory</jndi-name>
```

When deployed within an application library called *`MyAppLib`*, and referenced from an application called *`MyApp`*, it would result in a runtime JNDI name of:

```
MyApp/jms/MyConnectionFactory
```

Unique Runtime JNDI Name for Standalone JMS Modules

In the case of JMS modules deployed as stand-alone modules, at runtime ${APPNAME} becomes the name/ID of the stand-alone module. For example:

```
<jndi-name>${APPNAME}/jms/MyConnectionFactory</jndi-name>
```

When deployed within a stand-alone JMS module *`MyJMSModule`*, it would result in a runtime JNDI name of:

```
MyJMSModule/jms/MyConnectionFactory
```

Where to Use the ${APPNAME} String

The ${APPNAME} string can be used anywhere you refer to the JNDI name of a JMS resource. For example, in the:

- `jndi-name` or `local-jndi-name` element of `connection-factory` elements in the JMS module descriptor.
- `jndi-name` or `local-jndi-name` element of `queue` or `topic` elements in the JMS module descriptor.
- `jndi-name` element of `distributed-queue` or `distributed-topic` elements in the JMS module descriptor.
- `jndi-name` element of `uniform-distributed-queue` or `uniform-distributed-topic` elements in the JMS module descriptor.
- `destination-jndi-name` element of `message-destination-descriptor` elements in the `weblogic-ejb-jar.xml` descriptor.

> **Note:**
>
> WebLogic EJB also supports the use of the ${APPNAME} string.

- `jndi-name` element of `weblogic-enterprise-bean` elements in the `weblogic-ejb-jar.xml` descriptor.

Example Use-Case

In a single-server environment, WebLogic Integration Worklist uses application-scoped JMS resources (For example, queues and connection factories) to support its modular deployment goals. Application-scoped JMS allows WebLogic Integration to have an application library define the EJBs, JMS resources, and so on needed by worklist, and then have users simply include worklist into their application by adding a `library-ref` to their application. However, this prevents the worklist user from scaling those destinations to the cluster from an application library.

In a clustered environment, users can now substitute the ${`APPNAME}` string for the queue's JNDI name at runtime to make the global JNDI names for the queues unique. This way, the JMS ${`APPNAME`} parameter is replaced at runtime with the application name of the host application being merged to the application library.