# Oracle® Fusion Middleware

Programming Message-Driven Beans for Oracle WebLogic Server

11*g* Release 1 (10.3.1)

**E15493-01**

October 2009

This document is a resource for software developers who develop applications that use message-driven beans (MDBs).

ORACLE

Oracle Fusion Middleware Programming Message-Driven Beans for Oracle WebLogic Server, 11*g* Release 1 (10.3.1)

E15493-01

# Contents

# Preface

This preface describes the document accessibility features and conventions used in this guide—*Programming Message-Driven Beans for Oracle WebLogic Server*.

## Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

### Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

### Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

### Deaf/Hard of Hearing Access to Oracle Support Services

To reach Oracle Support Services, use a telecommunications relay service (TRS) to call Oracle Support at 1.800.223.1711. An Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process. Information about TRS is available at `http://www.fcc.gov/cgb/consumerfacts/trs.html`, and a list of phone numbers is available at `http://www.fcc.gov/cgb/dro/trsphonebk.html`.

## 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

# Understanding Message-driven Beans

A message-driven bean implements loosely coupled or asynchronous business logic in which the response to a request need not be immediate. A message-driven bean receives messages from a JMS Queue or Topic, and performs business logic based on the message contents. It is an asynchronous interface between EJBs and JMS.

Throughout its life cycle, an MDB instance can process messages from multiple clients, although not simultaneously. It does not retain state for a specific client. All instances of a message-driven bean are equivalent—the EJB container can assign a message to any MDB instance. The container can pool these instances to allow streams of messages to be processed concurrently.

The EJB container interacts directly with a message-driven bean—creating bean instances and passing JMS messages to those instances as necessary. The container creates bean instances at deployment time, adding and removing instances during operation based on message traffic.

**Example:** In an on-line shopping application, where the process of taking an order from a customer results in a process that issues a purchase order to a supplier, the supplier ordering process could be implemented by a message-driven bean. While taking the customer order always results in placing a supplier order, the steps are loosely coupled because it is not necessary to generate the supplier order before confirming the customer order. It is acceptable or beneficial for customer orders to "stack up" before the associated supplier orders are issued.

A message-driven bean (MDB) is an enterprise bean that allows J2EE applications to process messages asynchronously. An MDB acts as a JMS or JCA message listener, which is similar to an event listener except that it receives messages instead of events. The messages may be sent by any J2EE component—an application client, another enterprise bean, or a Web component—or by non-J2EE applications.

These are the key features of message-driven beans:

- Clients do not access message-driven beans through interfaces. A message-driven bean has only a bean class.
- A message-driven bean's instances retain no data or conversational state for a specific client. All instances of a message-driven bean are equivalent, allowing the EJB container to assign a message to any message-driven bean instance. The container can pool these instances to allow streams of messages to be processed concurrently.
- Throughout its life cycle, a message-driven bean instance can process messages from multiple clients, although not concurrently.

When a message arrives, the container calls the message-driven bean's `onMessage` method to process the message. The `onMessage` method normally casts the message

to one of the five JMS message types and handles it in accordance with the application's business logic. The `onMessage` method may call helper methods, or it may invoke a session or entity bean to process the information in the message or to store it in a database.

A message may be delivered to a message-driven bean within a transaction context, so that all operations within the `onMessage` method are part of a single transaction. If message processing is rolled back, the message will be re-delivered.

For information about design alternatives for message-driven beans, see Section 3, "MDBs and Messaging Models."

For a description of the overall EJB development process, see *Oracle Fusion Middleware Programming Enterprise JavaBeans, Version 3.0 for Oracle WebLogic Server*

## 1.1 JCA-Based MDBs

MDBs can be configured to receive messages from JCA 1.5-compliant resource adapters, as defined by the JCA specification. To configure a MDB to use JCA, set the `resource-adapter-jndi-name` deployment descriptor.

For more information, see the JCA 1.5 specification and "resource-adapter-jndi-name" in *Oracle Fusion Middleware Programming Enterprise JavaBeans for Oracle WebLogic Server*.

# 2 MDB Life Cycle

This section describes the phases in the life cycle of a message-driven bean instance and how you can configure an MDB to control the life cycle.



## 2.1 Overview

This section describes the phases in the life cycle of a message-driven bean instance.

## 2.2 MDBs and Concurrent Processing

MDBs support concurrent processing for both topics and queues. For more information about topics and queues, see Section 3, "MDBs and Messaging Models."

WebLogic Server maintains a *free pool* where MDB instances that are not currently servicing requests reside.The number of MDB instances in the free pool is controlled by the value of the `max-beans-in-free-pool` attribute, the number of available threads in the thread pool, the type of thread pool, and sometimes other factors. See "Tuning Message-Driven Beans" in *Oracle Fusion Middleware Performance and Tuning for Oracle WebLogic Server*.

Each MDB that is deployed to a server instance creates a single JMS connection.

In a queue-based JMS application (point-to-point model), each MDB instance has its own session.

In a topic-based JMS application (the publish/subscribe model), all local instances of an MDB share a JMS session. A given message is distributed to multiple MDBs—one copy to each subscribing MDB. If multiple MDBs are deployed to listen on the same topic, then each MDB receives a copy of every message. A message is processed by one instance of each MDB that listens to the topic.

## 2.3 Limitations for Multi-threaded Topic MDBs

The default behavior for non-transactional topic MDBs is to multi-thread the message processing. In this situation, the MDB container fails to provide reproducible behavior when the topic is not a WebLogic JMS Topic, such as unexpected exceptions and acknowledgement of messages that have not yet been processed. For example, if an application throws `RuntimeException` from `onmessage`, the container may still acknowledges the message.

Oracle recommends setting `max-beans-in-free-pool` to a value of 1 in the deployment descriptor to prevent multi-threading in topic MDBs when the topic is a foreign vendor topic (not a WebLogic JMS topic).

# 3

# MDBs and Messaging Models

WebLogic Server MDBs can be used in either a point-to-point (queue) or publish/subscribe (topic) messaging model. These models are described in detail in "Understanding WebLogic JMS" in *Oracle Fusion Middleware Programming JMS for Oracle WebLogic Server*.

The following sections describe the key differences between point-to-point and publish/subscribe messaging applications.



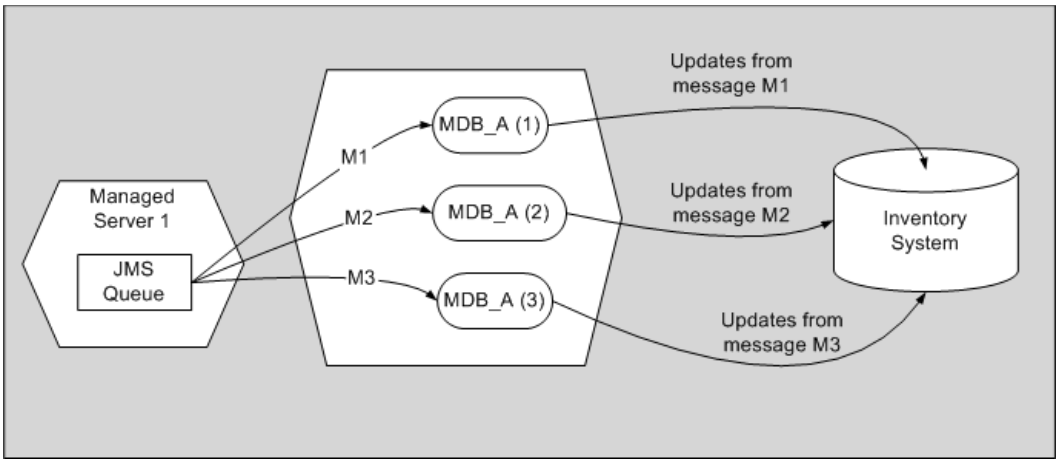
## 3.1 Point-to-Point (Queue) Model: One Message Per Listener

In the point-to-point model, a message from a JMS queue is picked up by one MDB listener and stays in the queue until processed. If the MDB goes down, the message remains in the queue, waiting for the MDB to come up again.

**Example**: A department must update its back-end inventory system to reflect items sold throughout the day. Each message that decrements inventory must be processed once, and only once. It is not necessary for messages to be processed immediately upon generation or in any particular order, but it is critical that each message be processed.

Figure 3–1 illustrates a point-to-point application. Each message is processed by single instance of MDB_A. Message "M1" is processed by MDB_A(1), "M2" is processed by MDB_A(2), and "M3" is processed by MDB_A(3).

***Figure 3–1 Point-to-Point Model***




## 3.2 Publish/Subscribe (Topic) Model

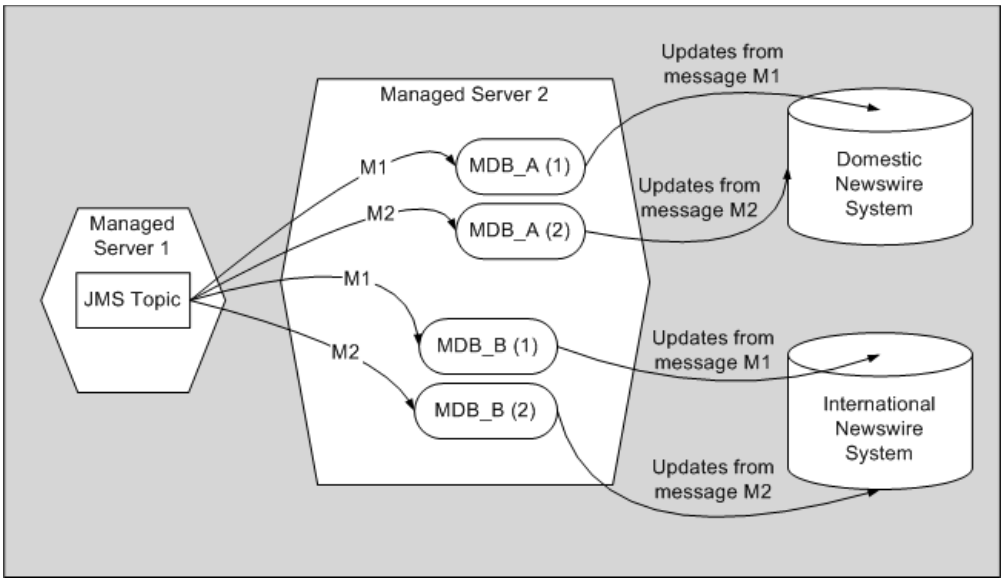
In the publish/subscribe model, a JMS topic publishes all messages to all subscribed listeners. If an MDB listener goes down, that MDB will miss the message, unless the topic is a *durable subscription* topic.

For more information on durable subscriptions and for configuration instructions, see "Setting Up Durable Subscriptions" in *Oracle Fusion Middleware Programming JMS for Oracle WebLogic Server* and Section 6.8, "Configuring Durable Topic Subscriptions."

**Example**: A financial news service broadcasts stock prices and financial stories to subscribers, such as news wire services. Each message is distributed to each subscriber.

Figure 3–2 illustrates a publish/subscribe application. In contrast to a point-to-point application, in a publish/subscribe model, a message is processed by one instance of *each* of the subscribing MDBs. Message "M1" is processed by an instance of MDB_A and an instance of MDB_B. Similarly, "M2" is processed by an instance of each of the subscribing MDBs.

***Figure 3–2 Publish/Subscribe Model***




## 3.3 Exactly-Once Processing

An MDB pool processes each message *at least once*. Potentially, a message can be processed more than once:

- If an application fails, a transaction rolls back, or the hosting server instance fails during or after the `onMessage()` method completes but before the message is acknowledged or committed, the message will be redelivered and processed again.
- Non-persistent messages are also redelivered in the case of failure, except for the case where the message's host JMS server shuts down or crashes, in which case the messages are destroyed.

To ensure that a message is processed *exactly once*, use container-managed transactions so that failures cause transactional MDB work to rollback and force the message to be redelivered.

# 4 Deploying MDBs

This section describes various approaches for deploying MDBs and the JMS destination to which the MDBs listen.

- Section 4.1, "Destination and MDBs: Collocated vs. Not Collocated"
- Section 4.2, "Collocated Destination/MDBs"
- Section 4.3, "Non-Collocated Destination/MDBs"
- Section 4.4, "JMS Distributed Destinations"

## 4.1 Destination and MDBs: Collocated vs. Not Collocated

You can deploy an MDB on the same server instance as the JMS destination to which it listens, or on a separate server instance. These alternatives are referred to as *collocation* or *non-collocation*, respectively.

## 4.2 Collocated Destination/MDBs

Collocating an MDB with the destination to which it listens keeps message traffic local and avoids network round trips. Collocation is the best choice if you use WebLogic Server JMS and want to minimize message processing latency and network traffic.

You cannot collocate the MDB and the JMS destination if you use a third-party JMS provider that cannot run on WebLogic Server, such as MQ Series.

Figure 4–1 and Figure 4–2 illustrate architectures in which the MDB application is deployed to the server instance that hosts the associated JMS destination. These architectures vary in that the first has a *distributed destination* and the second does not. In an application that uses distributed destinations, the MDB is deployed to each server instance that hosts a member of the distributed destination set. For more information about distributed destinations, see Section 4.4, "JMS Distributed Destinations." As illustrated in Figure 4–1 the message "M1" is delivered to an instance of MDB_A on each server instance where a distributed destination and MDB_A are deployed.

*Figure 4–1 Collocated Destination/MDBs, Distributed Destination*

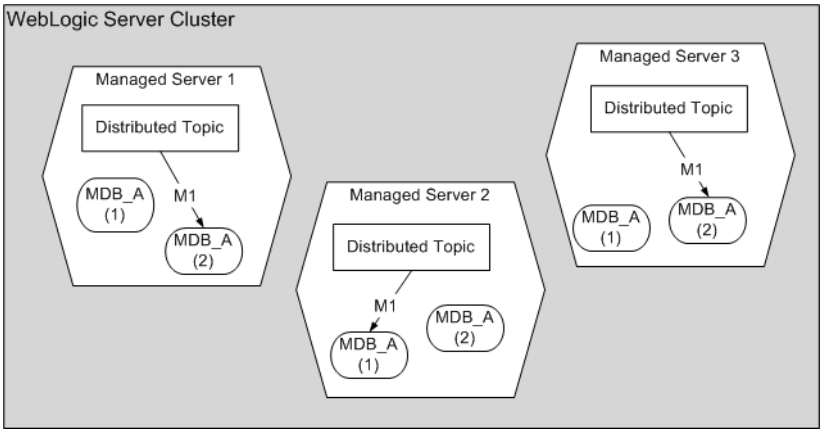
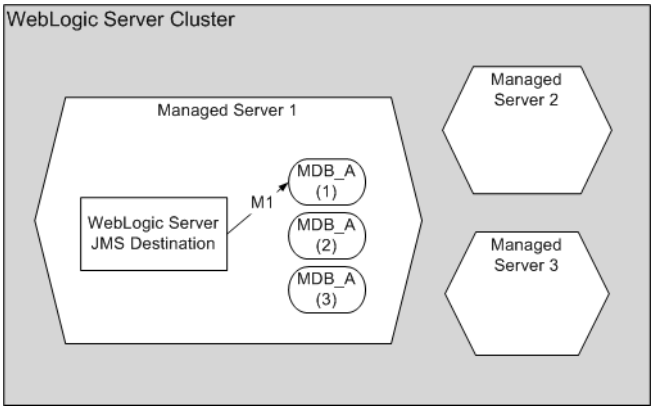


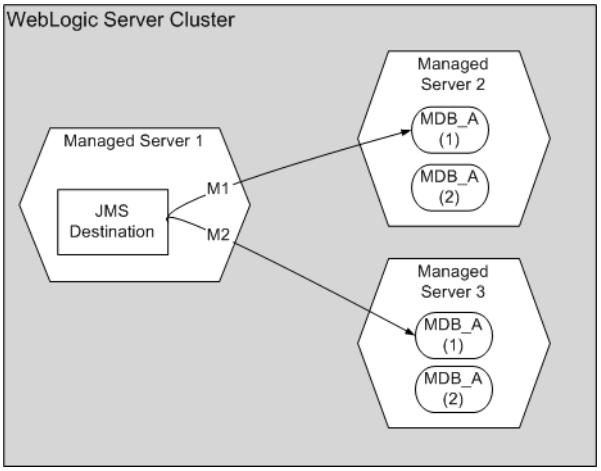

*Figure 4–2 Collocated Destination/MDBs, Non-Distributed Destination*




## 4.3 Non-Collocated Destination/MDBs

Figure 4–3 illustrates an architecture in which an MDB runs on a separate server instance than the JMS Destination to which the MDB listens.

*Figure 4–3 Non-Collocated Application, Non-Distributed Destination*



Running your MDBs on a separate server instance from the JMS Destination to which the MDB listens is suitable if:

- Your application uses a 3rd-party JMS provider, such as MQ Series.
- You want to isolate application code (the MDBs) from the JMS infrastructure. By placing JMS destinations and MDBs on separate server instances, you prevent application problems—for example, MDBs consuming all of your virtual machine's memory—from interrupting the operation of the JMS infrastructure.
- Your MDB application is very CPU-intensive. On a separate machine, your application can use 100 percent of the CPU without affecting the operation of the JMS infrastructure.
- One machine in your configuration has more processing power, disk space, and memory than other machines in the configuration.

The JMS destination and MDBs could also be deployed on non-clustered servers, servers within the same cluster, or to servers in separate clusters.

## 4.4 JMS Distributed Destinations

If your MDB application runs on a WebLogic Server cluster, you can configure multiple physical destinations (queues or topics) as a *distributed destination*, which appears to message producers and consumers to be a single destination.

If you configure a distributed destination, WebLogic JMS distributes the messaging load across available members of the distributed destination. If a member of the destination becomes unavailable due to a server failure, message traffic is re-directed to the other available physical destinations in the distributed destination set. You control whether an MDB that accesses a WebLogic distributed queue in the same cluster consumes from all distributed destination members or only those members local to the current WebLogic Server using the `distributed-destination-connection` element in the `weblogic-ejb-jar.xml` file.

If you deploy an MDB and the associated distributed destination to the same cluster, WebLogic Server automatically enumerates the distributed destination members and ensures that there is an MDB listening on each member.

The MDBs are homogeneously deployed to each clustered server instance. Messages are distributed across the physical destinations on the multiple server instances, and are processed in parallel. If one server instance goes down, other nodes in the cluster can continue to process messages. This architecture is a good choice if:

- Your application has high availability requirements.
- You prefer to deploy applications homogeneously so that each server runs the same set of applications.
- Your application processes a high volume of messages, or requires massively parallel processing.
- The machines in your cluster have identical or similar processing power, disk space, and memory.

For an example, see Figure 4–1. For additional information about distributed destinations, see "Using Distributed Destinations" in *Oracle Fusion Middleware Programming JMS for Oracle WebLogic Server*.

# 5

# Programming and Configuring MDBs: Main Steps

This section provides step-by-step instructions for implementing an MDB. For a summary of key deployment elements for MDBs, see Section 9.1, "message-destination-descriptor Element of the weblogic-ejb-jar.xml File."

- Section 5.1, "Required JMS Configuration"
- Section 5.2, "Create MDB Class and Configure Deployment Elements"

## 5.1 Required JMS Configuration

The steps in the following section assume that you have created the appropriate JMS components:

- A JMS connection factory—one that supports XA, if your MDBs are transactional.

  The default WebLogic MDB connection factory is XA-capable. For information about the default connection factories, see "Using a Default Connection Factory" in *Oracle Fusion Middleware Configuring and Managing JMS for Oracle WebLogic Server*. For instructions to create a custom WebLogic JMS connection factory, see "Create connection factories in a system module" in *Oracle Fusion Middleware Oracle WebLogic Server Administration Console Help*.

  The default behavior and configuration methods for other JMS providers vary. If you use a non-Oracle JMS provider, see the vendor documentation for details.

- A JMS destination

  For instructions on configuring WebLogic JMS destinations, see *Oracle Fusion Middleware Oracle WebLogic Server Administration Console Help*.

> **Note:** If your JMS provider is a remote WebLogic Server JMS provider or a foreign JMS provider, and you use the wrapper approach recommended in Section 6.7.1, "Whether to Use Wrappers," in addition to configuring the non-local JMS components, you must also configure a Foreign Connection Factory and Foreign JMS Destination in your local JNDI tree.

## 5.2 Create MDB Class and Configure Deployment Elements

Follow these steps to implement a message-driven bean:

1. Create a source file (message-driven bean class) that implements both the `javax.ejb.MessageDrivenBean` and `javax.jms.MessageListener` interfaces.

The MDB class must define the following methods:

- One `ejbCreate()` method that the container uses to create an instance of the message-driven bean in the free pool.
- One `onMessage()` method that is called by the EJB container when a message is received. This method contains the business logic that processes the message.
- One `setMessageDrivenContext{}` method that provides information to the bean instance about its environment (certain deployment descriptor values); the MDB also uses the context to access container services. See Section 6.10, "Using the Message-Driven Bean Context,".
- One `ejbRemove()` method that removes the message-driven bean instance from the free pool.

2. Declare the MDB in `ejb-jar.xml`, as illustrated in the excerpt below:

```
<ejb-jar>
    <enterprise-beans>
        <message-driven>
            <ejb-name>...</ejb-name>
            <ejb-class>...</ejb-class>
            <transaction-type>Container</transaction-type>
            <acknowledge-mode>auto_acknowledge</acknowledge-mode>
            <message-driven-destination>
                <destination-type>javax.jms.Topic</destination-type>
                <subscription-durability>Durable</subscription-durability>
            </message-driven-destination>
        </message-driven>
    </enterprise-beans>
    <assembly-descriptor>
        <container-transaction>
            <method>
                <ejb-name>...</ejb-name>
                <method-name>onMessage()</method-name>
            </method>
            <trans-attribute>Required</trans-attribute>
        </container-transaction>
    </assembly-descriptor>
</ejb-jar>
```

The key behaviors to configure are:

- Transaction management strategy—The MDB's transaction management strategy, in the `transaction-type` element. For instructions, see Section 6.3, "Configuring Transaction Management Strategy for an MDB."
- Destination type—The type of destination to which the MDB listens. For more information, see Section 6.2, "Configuring Destination Type."

3. Configure WebLogic-specific behaviors for the MDB in the `message-driven-descriptor` element of `weblogic-ejb-jar.xml`. For example:

```
<weblogic-ejb-jar>
    <weblogic-enterprise-bean>
        <ejb-name>exampleMessageDrivenA</ejb-name>
        <message-driven-descriptor>
            <pool>...</pool>
            <timer-descriptor>...</timer-descriptor>
            <destination-jndi-name>...</destination-jndi-name>
```

```
            <initial-context-factory>...</initial-context-factory>
            <provider-url>...</provider-url>
            <connection-factory-jndi-name>...</connection-factory-jndi-name>
            <jms-polling-interval-seconds>...</jms-polling-interval-seconds>
            <jms-client-id>...</jms-client-id>
            <generate-unique-jms-client-id>...</generate-unique-jms-client-id>
            <durable-subscription-deletion>...</durable-subscription-deletion>
            <max-messages-in-transaction>...</max-messages-in-transaction>
            <init-suspend-seconds>...</init-suspend-seconds>
            <max-suspend-seconds>...</max-suspend-seconds>
        </message-driven-descriptor>
    </weblogic-enterprise-bean>
</weblogic-ejb-jar>
```

The key elements to configure are those that specify how to access the destination. For instructions, see Section 6.7, "Configuring MDBs for Destinations."

4. Compile and generate the MDB class using the instructions in Compile Java Source in *Oracle Fusion Middleware Programming Enterprise JavaBeans for Oracle WebLogic Server*.

5. Deploy the bean on WebLogic Server using the instructions in the section "Preparing Applications and Modules for Deployment" in *Oracle Fusion Middleware Deploying Applications to Oracle WebLogic Server*

   If WebLogic Server cannot find an MDB's JMS destination during deployment, deployment succeeds, but WebLogic Server prints a message saying the destination was not found. The MDB bean then periodically tries to connect to its JMS queue until it succeeds. For more information, see Section 7, "Migration and Recovery for Clustered MDBs."

# 6

# Programming and Configuring MDBs: Details

The topics in this section supplement the instructions in Section 5, "Programming and Configuring MDBs: Main Steps."

- Section 6.1, "Configuring EJBs to Use Logical Message Destinations"
- Section 6.2, "Configuring Destination Type"
- Section 6.3, "Configuring Transaction Management Strategy for an MDB"
- Section 6.4, "Configuring Suspension of Message Delivery During JMS Resource Outages"
- Section 6.5, "Configuring the Number of Seconds to Suspend a JMS Connection"
- Section 6.6, "Manually Suspending and Resuming Message Delivery"
- Section 6.7, "Configuring MDBs for Destinations"
- Section 6.8, "Configuring Durable Topic Subscriptions"
- Section 6.9, "Configuring Message Handling Behaviors"
- Section 6.10, "Using the Message-Driven Bean Context"
- Section 6.11, "Configuring a Security Identity for a Message-Driven Bean"
- Section 6.12, "Using MDBs With Cross Domain Security"

## 6.1 Configuring EJBs to Use Logical Message Destinations

In this release of WebLogic Server, you can declare logical message destinations in an EJB's deployment descriptor and map the logical message destinations to actual message destinations (JMS queues or topics, or MDBs). Once you declare logical message destinations, you can then create message destination references that are linked to the logical message destinations. EJBs use the logical message destination name to perform a JNDI lookup and access the actual message destination. Logical JMS message destinations can be defined for individual MDBs or entire applications.

For information on how unresolved and unavailable message destinations are handled, see EJBs and Message Destination References in *Oracle Fusion Middleware Programming Enterprise JavaBeans for Oracle WebLogic Server*.

### 6.1.1 Configuring Logical JMS Message Destinations for Individual MDBs

In this release of WebLogic Server, you can configure logical JMS message destinations for individual MDBs.

To configure an MDB to use a logical message destination to link to an actual message destination:

1. Declare the message destination in the `message-destination-descriptor` element in `weblogic-ejb-jar.xml`.
2. Declare message destination references in the following elements in `ejb-jar.xml`:
   - `message-destination-ref`
   - `message-destination-ref-name`—the environment name used in the enterprise bean code for the message destination, relative to `java:comp/env`. For example, `<message-destination-ref>jms/StockQueue</message-destination-ref>`.
   - `message-destination-type`—the expected type of the referenced destination. For example, `<message-destination-type>javax.jms.Queue</message-destination-type>`.
   - message-destination-usage—specifies whether messages are consumed from the destination, produced for the destination, or both. For example, `<message-destination-usage>Produces<message-destination-usage>`.
   - `message-destination-link`—links the message destination reference to the actual message destination. This value must match the destination defined in `message-destination-name` in weblogic-ejb-jar.xml.

     For additional information on these elements or ejb-jar.xml, refer to Sun documentation.

## 6.1.2 Configuring Application-Scoped Logical JMS Message Destinations

In this release of WebLogic Server, you can configure resources for applications. Resources that are configured for entire applications are called application-scoped resources. This section describes application-scoped logical JMS destinations for an EJB application. For additional information on application-scoped resources, such as JMS and JDBC, see *Oracle Fusion Middleware Programming JMS for Oracle WebLogic Server* and *Oracle Fusion Middleware Programming JDBC for Oracle WebLogic Server*.

Application-scoped resources, such as logical JMS message destinations, for EJBs apply to all MDBs in the application. You can override application-scoped JMS for specific MDBs by configuring the MDBs to use logical message destinations to link to actual message destinations, as described in Section 6.1.1, "Configuring Logical JMS Message Destinations for Individual MDBs."

To configure application-scoped JMS for EJBs:

1. Declare the message destination in the `message-destination-descriptor` element in `weblogic-ejb-jar.xml`.
2. Declare message destination references in the following elements in `ejb-jar.xml`:

- `message-driven`
  - `message-destination-type`—the expected type of the referenced destination. For example,

<message-destination-type>javax.jms.Queue</message-destination-type>.

  - message-destination-usage—specifies whether messages are consumed from the destination, produced for the destination, or both. For example, <message-destination-usage>Produces<message-destination-usage>.
  - message-destination-link—links the message destination reference to the actual message destination. For example, <message-destination-link>ExpenseProcessingQueue<message-destination-link>. This value must match the destination defined in message-destination-name in weblogic-ejb-jar.xml.

- message-destination
  - message-destination-name—the name of the message destination. For example, <message-destination-name>ExpenseProcessingQueue<message-destination-name>. This value must match the destination defined in message-destination-name in weblogic-ejb-jar.xml.

    For additional information on these elements or ejb-jar.xml, refer to Sun documentation.

## 6.2 Configuring Destination Type

Configure the type of destination to which the MDB listens in the destination-type element in the message-driven-destination element of ejb-jar.xml.

To specify a topic, set destination-type to javax.jms.Topic. If the destination is a topic, specify subscription-durability as either Durable or NonDurable.

To specify a queue, set destination-type to javax.jms.Queue.

For more information, see Section 6.8, "Configuring Durable Topic Subscriptions."

## 6.3 Configuring Transaction Management Strategy for an MDB

An MDB can manage its own transactions or defer transaction management to the container.

To configure container-level transaction management:

- Set the transaction-type element in the message-driven element in the ejb-jar.xml file to Container.
- Set the trans-attribute element in the container-transaction element in ejb-jar.xml to Required.

---

**Note:** If transaction-type is set to Container, and trans-attribute is *not* set, the default transaction-attribute values are applied: required (for EJB 3.0 MDBs) and NotSupported (for MDBs prior to EJB 3.0). WebLogic Server allows you to deploy the MDB, and logs a compliance error. However, if you make this configuration error, the MDB *will not run transactionally*—if a failure occurs mid-transaction, updates that occurred prior to the failure will not be rolled back.

---

- To change the timeout period for the transaction, set `trans-timeout-seconds` in the `transaction-descriptor` element of `weblogic-ejb-jar.xml`. If a transaction times out, it is rolled back, and the message is redelivered. By default, transactions time out after 30 seconds. For an application with long-running transactions, it may be appropriate to increase the timeout period.

To configure bean-level transaction management:

- Set the `transaction-type` element in the `message-driven` element in the `ejb-jar.xml` file to `Bean`.
- Set the `acknowledge-mode` element to specify the desired JMS acknowledgment semantics, either one of the following:
  - `AUTO_ACKNOWLEDGE` (the default) as described at http://java.sun.com/products/jms/javadoc-102a/javax/jms/Session.html#AUTO_ACKNOWLEDGE
  - `DUPS_OK_ACKNOWLEDGE` as described at http://java.sun.com/products/jms/javadoc-102a/javax/jms/Session.html#DUPS_OK_ACKNOWLEDGE

For more information, see "Session" in *Oracle Fusion Middleware Programming JMS for Oracle WebLogic Server*.

## 6.4 Configuring Suspension of Message Delivery During JMS Resource Outages

In this release of WebLogic Server, you can configure how an MDB behaves when the EJB container detects a JMS resource outage (an MDB throwing the same exception 10 times in succession).

You can configure:

- An MDB to suspend the JMS connection and, thereby, stop receiving additional messages when the EJB container detects a JMS resource outage. If you choose this configuration option, you can specify:
  - the initial number of seconds the MDB should wait before it first resumes receiving messages.
  - the maximum number of seconds the MDB should wait before it resumes receiving messages.
- An MDB to not suspend the JMS connection when the EJB container detects a JMS resource outage.

When a JMS connection is suspended, message delivery is suspended for all JMS sessions associated with the connection. By default, when it detects a JMS resource outage, the EJB container suspends an MDB's JMS connection for `init-suspend-seconds`.

## 6.5 Configuring the Number of Seconds to Suspend a JMS Connection

You may want to suspend a JMS connection during a resource outage, which can be defined as an MDB throwing the same exception 10 times in succession.

To suspend an MDB's JMS connection, configure the following elements in the `weblogic-ejb-jar.xm` file:

- `init-suspend-seconds`—the initial amount of time, in seconds, to suspend a JMS connection when the EJB container detects a JMS resource outage. The default value is `5`.
- `max-suspend-seconds`—the maximum amount of time, in seconds, to suspended a JMS connection when the EJB container detects a JMS resource outage. The default value is `60`.

### 6.5.1 How the EJB Container Determines How Long to Suspend a JMS Connection

The EJB container uses the following algorithm, based on the `init-suspend-seconds` and `max-suspend-seconds`, to determine the amount of time a JMS connection is suspended.

When the EJB container detects a JMS resource outage:

1. The MDB's JMS connection is suspended for the amount of time specified by `init-suspend-seconds`.
2. The connection is checked. If the resource is available, go to Step 12.
3. If the value of `init-suspend-seconds` is greater than or equal to `max-suspend-seconds`, go to Step 9.
4. The amount of time used to suspend the JMS connection, represented by *Xseconds*, is calculated by multiplying the time of the previous suspension by 2.
5. The MDB's JMS connection is suspended for the amount of time specified by *Xseconds*.
6. The connection is checked. If the resource is available, go to Step 12.
7. If the value of `init-suspend-seconds` is greater than or equal to `max-suspend-seconds`, go to Step 9
8. Go to Step 4.
9. The MDB's JMS connection is suspended for the amount of time specified by `max-suspend-seconds`.
10. Check the connection. If the resource is available, go to Step 12.
11. Go to Step 9.
12. Continue processing.

### 6.5.2 Turning Off Suspension of a JMS Connection

If you do not want an MDB's JMS connection to be suspended when the EJB container detects a resource outage, set the value of `max-suspend-seconds` to `0`. When the value of `max-suspend-seconds` is `0`, the value of `init-suspend-seconds` is ignored.

## 6.6 Manually Suspending and Resuming Message Delivery

Administrators can use the Administration Console to manually suspend and resume message delivery to deployed MDBs. For information see "Suspend and resume MDB JMS connections" in *Oracle Fusion Middleware Oracle WebLogic Server Administration Console Help*.

## 6.7 Configuring MDBs for Destinations

WebLogic Server MDBs support Weblogic JMS destinations and foreign (non-Oracle) JMS provider destinations.

A *local* destination is one that runs on the same machine or in the same cluster as the MDB. A *remote* destination is one that runs on a different machine and is not a member of the same cluster as the MDB. Whether a destination is local or remote depends on whether or not it and the MDB share the same JNDI context.

To be considered local to one another, an MDB and the associated JMS destination must both run either on the same machine or within the same WebLogic Server cluster. An MDB and a JMS destination on server instances in the same WebLogic Server cluster are local to one another even if they are on separate machines, because the server instances in a WebLogic Server cluster each have a copy of the same cluster-wide JNDI tree.

Destinations that run under a non-Oracle JMS provider are referred to as *foreign*. Foreign JMS providers have their own JNDI provider and foreign JMS objects do not share the same context with a WebLogic Server MDB—unless the foreign JMS objects are configured with wrappers to appear in the MDB's JNDI context. This approach is discussed in Section 6.7.1, "Whether to Use Wrappers."

The nature of a destination—local versus remote and WebLogic JMS versus non-Oracle—governs the configuration alternatives available, and dictates to some extent how you configure these key elements in the `message-destination-descriptor` for the MDB in `weblogic-ejb-jar.xml`:

- `initial-context-factory`
- `provider-url`
- `destination-jndi-name`
- `connection-factory-jndi-name`

For foreign and remote destinations, the simplest configuration strategy is to use WebLogic Server JMS *wrappers*. Wrappers allow you to create a "symbolic link" between a JMS object in a third-party JNDI provider or in a different WebLogic Server cluster or domain, and an object in the local WebLogic JNDI tree.

For more information on when wrappers are appropriate, and the rules for configuring the `message-driven-descriptor` in `weblogic-ejb-jar.xml`, see these sections:

- Section 6.7.1, "Whether to Use Wrappers"
- Section 6.7.2, "How to Set provider-url"
- Section 6.7.3, "How to Set initial-context-factory"
- Section 6.7.4, "How to Set destination-jndi-name"
- Section 6.7.5, "How to Set connection-factory-jndi-name"

To configure the elements in `message-driven-descriptor` for specific scenarios, see Section 6.7.6, "Common Destination Scenarios: Illustrations and Key Element Settings."

### 6.7.1 Whether to Use Wrappers

Using wrappers means configuring a Foreign Connection Factory and a Foreign Destination that correspond to remote JMS objects (either non-Oracle or WebLogic JMS) as entries in your local JNDI tree.

- The use of wrappers is recommended if you use a foreign JMS provider or a remote WebLogic JMS provider. For more information on JMS wrapper classes, see "Simplified Access to Remote or Foreign JMS Providers" in "Enhanced Support for Using WebLogic JMS with EJBs and Servlets" in *Oracle Fusion Middleware Programming JMS for Oracle WebLogic Server*.
- If you use a wrapper for either the connection factory or the destination, you must use a wrappers for *both* of these objects.

Whether or not you use the wrapper classes determines how you configure the `initial-context-factory` and `destination-jndi-name`, as described below.

### 6.7.2 How to Set provider-url

`provider-url` specifies the URL of the JNDI service used by the JMS provider for the destination to which the MDB listens.

- If the JMS provider is local to the MDB (by definition, WebLogic JMS), do not specify `provider-url`.
- If the JMS provider is remote, whether WebLogic JMS or a foreign provider, and:
  - You do not use wrappers, specify `provider-url`.
  - You do use wrappers, do not specify `provider-url`. The URL is implicitly encoded in the wrapper.

### 6.7.3 How to Set initial-context-factory

`initial-context-factory` identifies the class that implements the initial context factory used by the JMS provider.

- If your JMS provider is WebLogic JMS, whether local or remote, do not specify `initial-context-factory`.
- If your JMS provider is foreign, and
  - you do not use wrappers, specify the initial context factory used by the JMS provider.
  - you do use wrappers, do not specify `initial-context-factory`.

### 6.7.4 How to Set destination-jndi-name

`destination-jndi-name` identifies the JNDI name of destination to which the MDB listens.

- If your JMS provider is local, specify the name bound in the local JNDI tree for the destination.
- If your JMS provider is foreign and:
  - You do not use wrappers, specify the name of the destination, as bound in the foreign provider's JNDI tree.
  - You do use wrappers, specify the name Foreign Destination you set up in your local JNDI tree that corresponds the remote or foreign destination.

### 6.7.5 How to Set connection-factory-jndi-name

`connection-factory-jndi-name` identifies the JNDI name of the connection factory used by the JMS provider.

- If your JMS provider is local, do not specify `connection-factory-jndi-name` unless you have configured a custom connection factory that the MDB will use.

  Custom connection factories are used when the default WebLogic Server connection factory does not satisfy your application requirements. For example, you might configure a custom connection factory in order to specify a particular desired value for the `MessagesMaximum` attribute. The procedure for configuring a connection factory is described in "Configure connection factories" in *Oracle Fusion Middleware Oracle WebLogic Server Administration Console Help*.

  **Note:** If you configure a custom JMS connection factory for an MDB, be sure to set the `Acknowledge Policy` attribute to `Previous`, and that the `UserTransactionsEnabled` attribute is enabled.

- If your JMS provider is remote or foreign, and:
  - You do not use wrappers, specify the name of the connection factory used by the JMS provider, as bound in the remote JNDI tree.
  - You do use wrappers, specify the Foreign Connection Factory you set up in your local JNDI tree that corresponds to the remote or foreign JMS provider's connection factory.

### 6.7.6 Common Destination Scenarios: Illustrations and Key Element Settings

The figures in this section illustrate common destination configurations. For remote and foreign destinations, scenarios with and without wrappers are included.

- Section 6–1, "A. Destination on a Local WebLogic JMS Server"
- Section 6–2, "B. Destination On a Remote WebLogic JMS Server—No Wrappers"
- Section 6–3, "C. Destination on Foreign JMS Server—No Wrappers"
- Section 6–4, "D. Destination on a Remote WebLogic Server or Foreign JMS Server—With Wrappers"

Table 6–1 summarizes how to configure the elements in the `message-driven-descriptor` element of `weblogic-ejb-jar.xml` for each scenario.

***Figure 6–1 A. Destination on a Local WebLogic JMS Server***

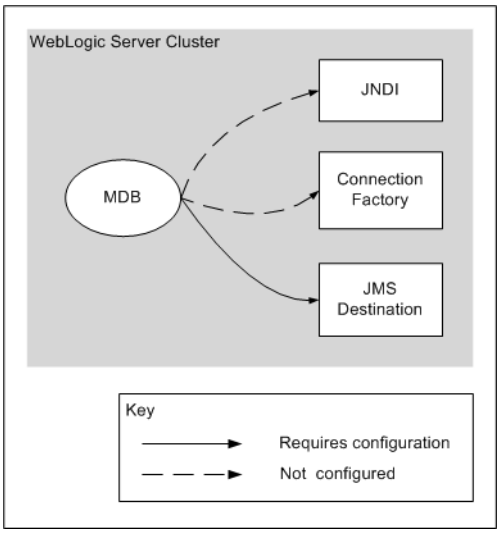



***Figure 6–2 B. Destination On a Remote WebLogic JMS Server—No Wrappers***

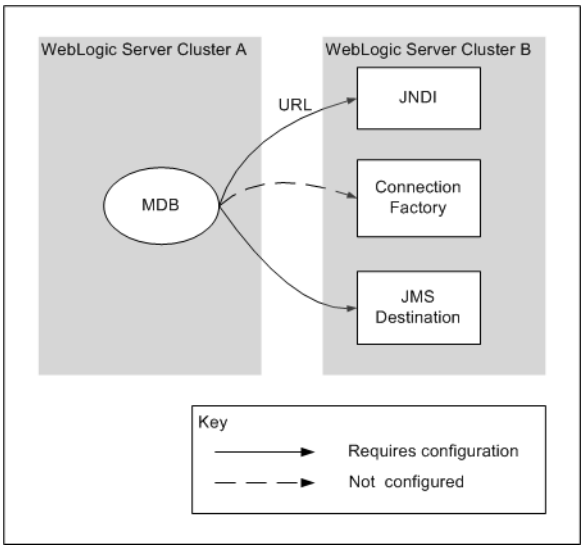


***Figure 6–3 C. Destination on Foreign JMS Server—No Wrappers***

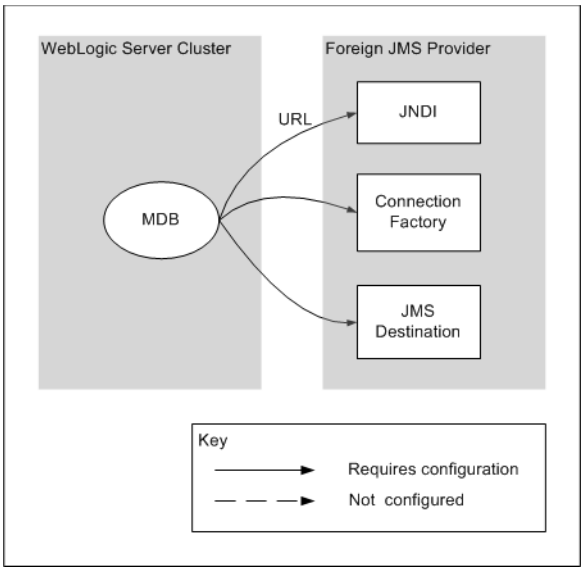



***Figure 6–4 D. Destination on a Remote WebLogic Server or Foreign JMS Server—With Wrappers***

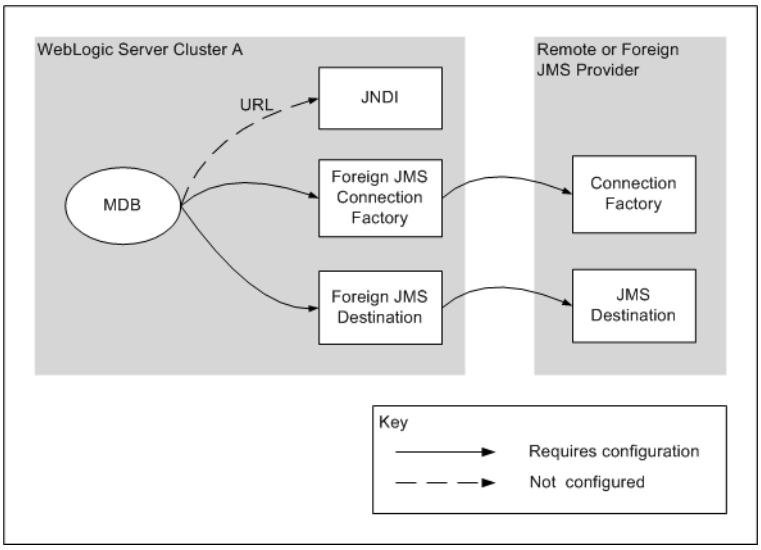


*Table 6–1 Common Configuration Scenarios*

| Scen. | If destination is on... | Wrappers Configured? | destination-jndi-name | initial-context-factory | provider-url | connection-factory -jndi-name |
|---|---|---|---|---|---|---|
| A | Local WebLogic JMS server | Not applicable for local WebLogic JMS server | Name of the local destination, as bound in local JNDI tree | Do not specify | Do not specify | Specify only if using a custom connection factory |
| B | Remote WebLogic JMS Server | No wrappers configured | Name of the remote destination, as bound in the remote JNDI tree | Do not specify | URL or cluster address for the remote WebLogic JMS Server | Specify only if using a custom connection factory on the remote provider |
| C | Foreign JMS Provider | No wrappers configured | Name of the remote destination, as bound in the remote JNDI tree | Name of remote initial context factory, as bound in remote JNDI tree | URL to access the foreign JMS provider | JNDI name of foreign connection factory |
| D | Remote Weblogic JMS Server<br>or<br>Foreign JMS server | Wrappers configured | The name of the Foreign Destination—as bound in your local JNDI tree —that maps to the remote or foreign destination | Do not specify | Do not specify | The name of the Foreign Connection Factory—as bound in your local JNDI tree —that maps to the remote or foreign connection factory |

# 6.8 Configuring Durable Topic Subscriptions

Durable subscriptions allow an MDB to receive messages that were delivered to a topic while the MDB was not connected.

## 6.8.1 Configuring a Durable Topic Subscription for a Non-Clustered Server

Follow these instructions to configure a durable topic subscription for an MDB that is deployed to non-clustered server instances.

1. Configure the `destination-type` in `ejb-jar.xml` to `javax.jms.Topic`.
2. In the `message-driven-destination` element of `ejb-jar.xml`, set:
   - `destination-type` to `javax.jms.Topic`
   - `subscription-durability` to `Durable`.
3. Configure the `ClientId` for the MDB, as desired:

   If you configure your own connection factory to suit specific application requirements, as described in "Configure connection factories" in *Oracle Fusion Middleware Oracle WebLogic Server Administration Console Help*, you can define the `ClientID` for the MDB when you configure the connection factory.

   If you set up your connection factory and do not assign it a `ClientID`, or if you use a default connection factory, the MDB uses the value of `jms-client-id` in `weblogic-ejb-jar.xml` as its client ID. If `jms-client-id` is not specified, the default value is the `ejb-name` for the MDB.

## 6.8.2 Configuring a Durable Topic Subscription for a Cluster

In a cluster, a JMS durable subscription is uniquely identified by the combination of an MDB's:

- *connection ID*—`ClientId` for the connection factory, and is unique within a cluster.
- *subscription ID*—the MDB's `jms-client-id`. The subscription ID must be unique on its topic, hence an MDB with a durable topic subscription cannot run on multiple server instances in a cluster. After a first instance of the MDB starts on a server instance in the cluster, an additional instance of the EJB can deploy successfully on another clustered server, but when the MDB starts, a conflict is detected and that instance of the MDB fails to fully connect to JMS.

To allow a durable subscriber MDB to run on multiple server instances in a cluster, where each MDB instance receives a copy of each topic message, each MDB instance should be deployed with a unique `jms-client-ID` or, if `jms-client-ID` is not specified, `ejb-name`.

In a local cluster, durable subscribers subscribe directly to a distributed destination topic. The MDB deploys only where there is a distributed destination on the server.

In a remote cluster, if durable subscribers subscribe directly to a distributed destination topic, the MDB is deployed once on every distributed destination member.

MDB can subscribe to a physical destination using one of the following methods:

- Configure each distributed destination topic physical destination with a unique JNDI name and configure each durable subscriber MDB pool with a matching `destination-jndi-name`, or
- Configure each physical destination with the same JNDI name, and:
  - Set the `LocalJNDIName` attribute for each physical destination. For more information, see "JMS Topic->Configuration->General" in *Oracle Fusion Middleware Oracle WebLogic Server Administration Console Help*.
  - Ensure that there is only one physical destination per server instance.

### 6.8.3 Configuring Automatic Deletion of Durable Topic Subscriptions

In this release of WebLogic Server, you can configure an MDB to automatically delete a durable topic subscription when the MDB is undeployed or deleted from a server. To configure an MDB to automatically delete durable topic subscriptions, set `durable-subscription-deletion` to `True`; by default, `durable-subscription-deletion` is set to `False`.

## 6.9 Configuring Message Handling Behaviors

These topics provide guidelines for behaviors related to message delivery:

- Section 6.9.1, "Ensuring Message Receipt Order"
- Section 6.9.2, "Preventing and Handling Duplicate Messages"
- Section 6.9.3, "Redelivery and Exception Handling"

### 6.9.1 Ensuring Message Receipt Order

Make sure that the MDB's business logic allows for asynchronous message processing. Do not assume that MDBs receive messages in the order the client issues them. To ensure that receipt order matches the order in which the client sent the message, you must do the following:

- Set `max-beans-in-free-pool` to `1` for the MDB. This ensures that the MDB is the sole consumer of the message.
- If your MDBs are deployed on a cluster, deploy them to a single node in the cluster, as illustrated in Figure 6–5.

To ensure message ordering in the event of transaction rollback and recovery, configure a custom connection factory with `MessagesMaximum` set to 1, and ensure that no redelivery delay is configured. For more information see "Ordered Redelivery of Messages" in *Oracle Fusion Middleware Programming JMS for Oracle WebLogic Server*.

See Sun's documentation on the Interface `MessageListener`—`javax.jms.MessageListener.onMessage()`—for more information, at `http://java.sun.com/j2ee/sdk_1.2.1/techdocs/api/javax/jms/MessageListener.html`.

### 6.9.2 Preventing and Handling Duplicate Messages

A JMS producer expects an MDB to acknowledge received messages. If the MDB receives the message, but fails to send an acknowledgement, the JMS producer re-sends the same message.

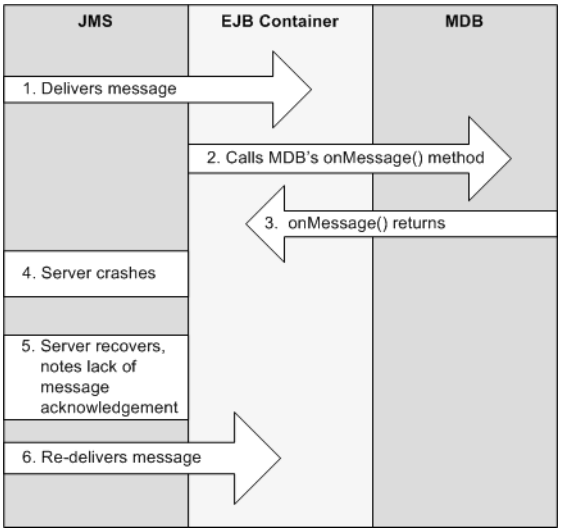
Your MDB design should allow for the likelihood of duplicate messages. Duplicate messages can be undesirable in certain cases. For example, if an MDB's `onMessage()` method includes code to debit a bank account, receiving and processing that message twice would result in the account being debited twice. In addition, re-sending messages consumes more processing resources.

The best way to prevent delivery of duplicate messages is to use container-managed transactions. In a container-managed transaction, message receipt and acknowledgement occur within the transaction; either both happen or neither happens. However, while this provides more reliability than using bean-managed transactions, performance can be compromised because container-managed transactions use more CPU and disk resources.

If the MDB manages its own transactions, your `onMessage()` code must handle duplicate messages, as receipt and acknowledgement occur outside of a transaction. In some applications, receipt and processing of duplicate messages is acceptable. In other cases, such as the bank account scenario described above, if a transaction is bean-managed, the bean code must prevent processing of duplicate messages. For example, the MDB could track messages that have been consumed in a database.

Even if an MDB's `onMessage()` method completes successfully, the MDB can still receive duplicate messages if the server crashes between the time `onMessage()` completes and the time the container acknowledges message delivery. Figure 6–5 illustrates this scenario.

***Figure 6–5 Server Crash Between Completion of onMessage() and Container Delivery Acknowledgement***




### 6.9.3 Redelivery and Exception Handling

If an MDB is consuming a message when an unexpected error occurs, the MDB can throw a system exception that causes JMS to resend, delay, and then resend or give up, depending on how JMS is configured.

To force message redelivery for a transactional MDB, use the bean context to call `setRollbackOnly()`.

To force message redelivery for any MDB—transactional or non-transactional—you can throw an exception derived from the RuntimeException or Error thrown by the MDB. This causes the MDB instance to be destroyed and re-created, which incurs a performance penalty.

Configure the redelivery delay based on what type of task the MDB's `onMessage()` method is performing. In some cases, redelivery should be instantaneous, for example, in an application that posts breaking news to a newswire service. In other cases, for example, if the MDB throws an exception because the database is down, redelivery should not occur immediately, but after the database is back up.

---

**Note:** For fully ordered MDBs, do not set a redelivery delay.

---

For instructions on configuring a redelivery delay, and other JMS exception handling features that can be used with MDB see "Managing Rolled Back, Recovered, Redelivered, or Expired Messages" in *Oracle Fusion Middleware Programming JMS for Oracle WebLogic Server*.

## 6.10 Using the Message-Driven Bean Context

WebLogic Server calls `setMessageDrivenContext()` to associate the MDB instance with a container context. This is not a client context; the client context is not passed along with the JMS message.

To access the container context's properties from within the MDB instance, use the following methods from the `MessageDrivenContext` interface:

- `getCallerPrincipal())`—Inherited from the `EJBContext` interface and should not be called by MDB instances.
- `isCallerInRole()`—Inherited from the `EJBContext` interface and should not be called by MDB instances.
- `setRollbackOnly()`—Can only be used by EJBs that use container-managed transactions.
- `getRollbackOnly()`— Can only be used by EJBs that use container-managed transactions.
- `getUserTransaction()`—Can only be used by EJBs that use bean-managed transaction demarcations.

---

**Note:** Although `getEJBHome()` is also inherited as part of the `MessageDrivenContext` interface, message-driven beans do not have a home interface. Calling `getEJBHome()` from within an MDB instance causes an `IllegalStateException`.

---

## 6.11 Configuring a Security Identity for a Message-Driven Bean

When a message-driven bean (MDB) receives messages from a JMS queue or topic, the EJB container uses a Credential Mapping provider and a credential map to obtain the security identity—username and password—to use when establishing the JMS connection and to execute the onMessage() method. This credential mapping occurs only once, when the MDB is started.

Once the EJB container is connected, the JMS provider uses the established security identity to retrieve all messages.

To configure a security identity for an MDB:

1. Create a WebLogic user for the MDB. See "Users, Groups, and Security Roles" in *Oracle Fusion Middleware Securing Resources Using Roles and Policies for Oracle WebLogic Server*. Assign the user the username and password that the non-Oracle JMS provider requires to establish a JMS connection.
2. In the `ejb-jar.xml` deployment descriptor, define a `run-as` identity for the MDB:

```
<security-identity>
    <run-as>
        <role-name>admin</role-name>
    </run-as>
</security-identity>
```

3. To create the `security-identity`, you must also define the `security-role` inside the `assembly-descriptor` element in `ejb-jar.xml`, as shown below.

```
<assembly-descriptor>
    <security-role>
        <role-name>jmsrole</role-name>
    </security-role>
    ....
</assembly-descriptor>
```

4. In the `weblogic-ejb-jar.xml` deployment descriptor, map the `run-as` identity to the user defined in Step 2, as shown below:

```
<security-role-assignment>
```

```
    <role-name>admin</role-name>
    <principal-name>username</principal-name>
</security-role-assignment>
```

where `username` is the username for the user created in step 1.

5. If the JMS provider is not WebLogic JMS, configure the credential mapper as described in "Create EJB component credential mappings" in *Oracle Fusion Middleware Oracle WebLogic Server Administration Console Help*.

---

**Note:** If the JMS provider is WebLogic JMS, it is *not* necessary to configure a credential mapper.

---

## 6.12 Using MDBs With Cross Domain Security

MDBs do not require you to configure Cross Domain Security. However, you should consider the following guidelines when implementing MDBs:

- If your MDBs must handle transactional messages, you must correctly configure either Cross Domain Security or Security Interop Mode for all participating domains.

  Keep all the domains used by your process symmetric with respect to Cross Domain Security configuration and Security Interop Mode. Because both settings are set at the domain level, it is possible for a domain to be in a mixed mode, meaning the domain has both Cross Domain Security and Security Interop Mode set. For more information, see "Configuring Domains for Inter-Domain Transactions" in *Oracle Fusion Middleware Programming JTA for Oracle WebLogic Server*.

- MDBs handling non-transactional messages do not require you to configure Cross Domain Security. However, you will need to configure Cross Domain Security for all the domains your process communicates with if Cross Domain Security is configured on one domain and the membership of the Distributed Destination that the MDB listens to in any domain changes.

  A best practice is to keep all the domains used by your process symmetric, with respect to Cross Domain Security configuration— that is, all domains use Cross Domain Security (or are on the appropriate exception lists) or none of the domains have Cross Domain Security configured. See "Configuring Security for a WebLogic Domain" in *Oracle Fusion Middleware Securing Oracle WebLogic Server*.

# 7

# Migration and Recovery for Clustered MDBs

WebLogic Server supports migration and recovery for clustered MDB applications. In the event of failure, you can bring a JMS destination and MDBs back online. Design your application so that when a server instance fails, your application automatically migrates the JMS destination and its associated MDBs from the failed server in the cluster to an available server instance.

> **Note:** An MDB can use the migratable service with clustered servers only. The migratable service cannot span multiple clusters.

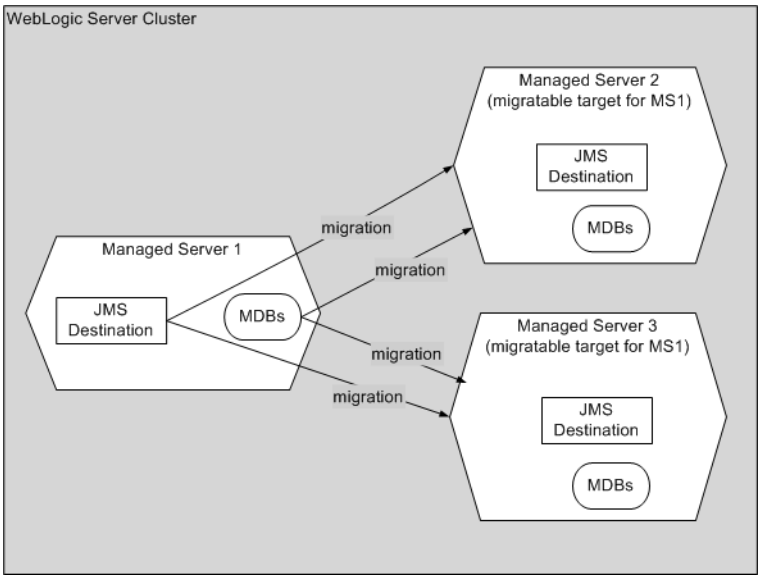
After an MDB application migrates to another server, it reconnects to the migrated JMS destination and begins to receive messages from the JMS destination again.

MDBs do not have migratable targets. Instead, an MDB automatically detects the JMS Server migration target during deployment, and uses that as its migratable target. You must ensure that MDB are deployed everywhere that a JMS Server is deployed. You can do this in two ways:

- Deploy MDBs homogeneously to the cluster.
- Match an MDB's targets to the JMS migratable target list in the `config.xml` file for the server instance's cluster. The MDB target server name must match the JMS migratable target list, or MDB migration will fail. For more information on configuring migratable targets, see "Understanding Migratable Target Servers in a Cluster" in *Oracle Fusion Middleware Using Clusters for Oracle WebLogic Server*.

In Figure 7–1, Managed Servers 1, 2 and 3 are in the same cluster. Managed Server 2 and Managed Server 3 are configured as Managed Server 1's migratable targets. In the event that Managed Server 1 fails, the JMS destination and MDB migrate to the next available Managed Server. If a Managed Server on the target list is unavailable, the destination and MDBs migrate to the next available Managed Server on the target list. For instance if Managed Server 2 is unavailable with Managed Server 1 fails, the JMS destination and MDB application migrate to Managed Server 3.

***Figure 7–1 Migration of JMS Destination***




For instructions on implementing the migratable service and for background information on WebLogic JMS migration and recovery services for clustered architectures, see "JMS as a Migratable Service within a Cluster" in *Oracle Fusion Middleware Configuring and Managing JMS for Oracle WebLogic Server*.

# 8

# Using Batching with Message-Driven Beans

Within an MDB, business logic, including database transactions, is performed within the `onMessage()` method. Within an EJB application, multiple MDBs can perform multiple `onMessage()` calls. If each `onMessage()` call performs a database transaction, this can create a lot of overhead where each call requires its own database connection.

WebLogic Server provides a mechanism for grouping `onMessage()` calls together as a single transaction. This mechanism can help increase database performance of an EJB application by grouping all of the transactions into a single I/O request. Grouping transactions allows requires fewer transaction logs.

For information on transaction management within MDBs, see Section 6.3, "Configuring Transaction Management Strategy for an MDB."

> **Note:** Transaction batching is not effective for all MDB applications. For example, database deadlocks can occur in an application where an MDB makes multiple calls to a database. Using the transaction batching feature will cause the MDB to lock more rows per transaction which can lead to database deadlocks.



## 8.1 Configuring MDB Transaction Batching

You can enable MDB transaction batching by defining the `max-messages-in-transaction` element. This element is part of the `message-driven-descriptor` element of the `weblogic-ejb-jar.xml` deployment descriptor.

`max-messages-in-transaction` defines the batch size WebLogic Server uses to process `onMessage()` transactions. However, increasing the batch size can increase latency. You should start with a small value, 5 for example. You can increase this value as your application performance allows.

When using MDB batching, more messages are processed per transaction. This may cause more transactions to time out since more work is being performed in each transaction. You can increase the transaction timeout be increasing the value of `trans-timeout-seconds` attribute of `weblogic-ejb-jar.xml`.

## 8.2 How MDB Transaction Batching Works

MDB transaction batching does not require any changes to application code. As far as the application is concerned, individual messages are still processed one by one. There is no application level message list.

Internally, WebLogic Server creates a queue to manage the transactions. Each message is queued until the number of messages in the queue is equal to the batch size defined by `max-messages-in-transaction`. However, if there is no next message to be queued, the current messages in the queue are submitted for processing.

If an individual `onMessage()` call fails, then the entire batch is rolled back. If the failure was due to a transaction timeout, as defined in the `trans-timeout-seconds` attribute of `weblogic-ejb-jar.xml`, the MDB container temporarily reduces the batch size and attempts to process the transactions with smaller batches.

If failure occurs for another reason, the MDB reprocesses each message within the failed batch as an individual transaction. This avoids the scenario where an individual `onMessage()` call can permanently hang an entire batch.

# 9 Deployment Elements for MDBs

This section lists key deployment elements that affect the behavior of MDBs:

- Section 9.1, "message-destination-descriptor Element of the weblogic-ejb-jar.xml File"
- Section 9.2, "ejb Element of the weblogic-application.xml File"
- Section 9.3, "message-driven Element of the ejb-jar.xml File"

## 9.1 message-destination-descriptor Element of the weblogic-ejb-jar.xml File

Table 9–1 summarizes the deployment elements in the `message-destination-descriptor` element of `weblogic-ejb-jar.xml`.

***Table 9–1 weblogic-ejb-jar.xml Deployment Elements for MDBs***

| Element | Description | Default |
|---|---|---|
| `connection-factory-jndi-name` | The JNDI name of the JMS ConnectionFactory that the message-driven EJB should look up to create its queues and topics. See Section 6.7.5, "How to Set connection-factory-jndi-name." | `weblogic.jms.MessageDriven.BeanConnection.Factory` |
| `connection-factory-resource-link` | Maps to a resource within a JMS module defined in `ejb-jar.xml` to an actual JMS Module Reference in WebLogic Server. | n/a |
| `destination-jndi-name` | The JNDI name used to associate a MDB with an actual JMS Queue or Topic deployed in the WebLogic Server JNDI tree. See Section 6.7.4, "How to Set destination-jndi-name." | n/a |
| `destination-resource-link` | Maps to a resource within a JMS module defined in `ejb-jar.xml` to an actual JMS Module Reference in WebLogic Server. | n/a |
| `dispatch-policy` | This optional element allows you to specify a particular WorkManager for the bean. | n/a |
| `distributed-destination-connection` | Specifies whether an MDB that accesses a WebLogic JMS distributed queue in the same cluster consumes from all distributed destination members or only those members local to the current Weblogic Server. | `LocalOnly` (only consumes from members local to the current WebLogic Server) |
| `durable-subscription-deletion` | Indicates whether you want durable topic subscriptions to be automatically deleted when an MDB is undeployed or removed. | `False` |

***Table 9–1 (Cont.) weblogic-ejb-jar.xml Deployment Elements for MDBs***

| Element | Description | Default |
|---|---|---|
| `generate-unique-jms-client-id` | Indicates whether or not you want the EJB container to generate a unique client-id for every instance of an MDB. Enabling this flag makes it easier to deploy durable MDBs to multiple server instances in a WebLogic Server cluster. | `False` |
| `init-suspend-seconds` | The initial number of seconds to suspend an MDB's JMS connection when the EJB container detects a JMS resource outage. See Section 6.4, "Configuring Suspension of Message Delivery During JMS Resource Outages." | `5` |
| `initial-beans-in-free-pool` | The number of inactive instances of an MDB that exist in WebLogic Server when it is started. | 0 |
| `initial-context-factory` | The initial contextFactory that the EJB container uses to create its connection factories. See Section 6.7.3, "How to Set initial-context-factory." | `weblogic.jndi.WLInitialContextFactory` |
| `jms-client-id` | The client id for the a message-driven bean associated with a durable subscriber topic. See "Defining the Client ID" in *Oracle Fusion Middleware Programming JMS for Oracle WebLogic Server*. | *`ejb-name`* of the EJB |
| `jms-polling-interval-seconds` | The number of seconds between attempts by the EJB container to reconnect to a JMS destination that has become unavailable. See Section 7, "Migration and Recovery for Clustered MDBs." | 10 seconds |
| `max-beans-in-free-pool` | The maximum size of the free pool of inactive MDBs. | 1000 |
| `max-messages-in-transaction` | Specifies the maximum number of messages that can be in a transaction for this MDB. | n/a |
| `max-suspend-seconds` | The maximum number of seconds to suspend an MDB's JMS connection when the EJB container detects a JMS resource outage. See Section 6.4, "Configuring Suspension of Message Delivery During JMS Resource Outages." | `60` |
| `pool` | Configures the behavior of the WebLogic Server free pool for message-driven EJBs. | n/a |
| `provider-url` | The URL provider to be used by the InitialContext. Typically, this is the host:port. See Section 6.7.2, "How to Set provider-url." | `t3://localhost:7001` |
| `resource-adapter-jndi-name` | Identifies the resource adapter that this MDB receives messages from. | n/a |
| `security-role-assignment` | Maps application roles in the `ejb-jar.xml` file to the names of security principals available in WebLogic Server. | n/a |

*Table 9–1 (Cont.) weblogic-ejb-jar.xml Deployment Elements for MDBs*

| Element | Description | Default |
|---|---|---|
| `timer-descriptor` | An EJB timer object. For more information, see Programming the EJB Timer Service in *Oracle Fusion Middleware Programming Enterprise JavaBeans for Oracle WebLogic Server*. | n/a |
| `trans-timeout-seconds` | The maximum duration for an EJB's container-initiated transactions, in seconds, after which the transaction is rolled back. See Section 6.3, "Configuring Transaction Management Strategy for an MDB." | `30` |
| `use81-style-polling` | Enables backwards compatibility for WLS Version 8.1-style polling. | False |

## 9.2 ejb Element of the weblogic-application.xml File

Table 9–2 lists key deployment elements for MDBs in the `ejb` element of `weblogic-application.xml`.

*Table 9–2 weblogic-application.xml Elements for MDBs*

| Element | Description | Default |
|---|---|---|
| `start-mdbs-with-application` | Controls when MDBs start processing messages. With default setting of true, an MDB starts processing messages as soon as it is deployed, even if WebLogic Server has not completed booting. This can cause an MDB application to access uninitialized services or applications during boot up and, therefore, to fail.<br>Set to false to defer message processing until after WebLogic Server opens its listen port. | `false` |

## 9.3 message-driven Element of the ejb-jar.xml File

Table 9–3 lists key J2EE deployment elements for MDBs that you configure in the `message-driven` element of `ejb-jar.xml`.

*Table 9–3 Key J2EE Deployment Elements for MDBs*

| Element | Description | Allowable Values |
|---|---|---|
| `acknowledge-mode` | Specifies JMS message acknowledgment semantics for the `onMessage` method of a message-driven bean that uses bean managed transaction demarcation. | ▪ `AUTO_ACKNOWLEDGE`<br>▪ `DUPS_OK_ACKNOWLEDGE` |
| `activation-config` | Defines information about the expected configuration properties of the message-driven bean in its operational environment. This may include information about message acknowledgement, message selector, expected destination type, and so on.<br>The configuration information is expressed in terms of name/value configuration properties. | The properties that are recognized for a particular message-driven bean are determined by the messaging type. |
| `destination-type` | Specifies the type of the JMS destination—the Java interface expected to be implemented by the destination. | ▪ `javax.jms.Queue`<br>▪ `javax.jms.Topic` |
| `message-destination` | Specifies the type of the destination. | Specified by the Java interface expected to be implemented by the destination. |
| `messaging-type` | Specifies the message listener interface of the message-driven bean. | Valid message listener interface. |

***Table 9–3 (Cont.) Key J2EE Deployment Elements for MDBs***

| Element | Description | Allowable Values |
|---|---|---|
| `subscription-durability` | Specifies whether a JMS topic subscription is intended to be durable or nondurable. | ■ `Durable`<br>■ `NonDurable` |
| `transaction-type` | Specifies an enterprise bean's transaction management type.<br>**Note**: If transaction-type is set to Container, trans-attribute must be set to Required. | ■ `Bean`<br>■ `Container` |
| `trans-attribute` | Specifies how the container must manage the transaction boundaries when delegating a method invocation to an enterprise bean's business method.<br>Set to `Required` for container-managed transactions. For more information, see Section 5.6.5, "Configuring Transaction Management Strategy for an MDB." | ■ `Required`<br>■ `NotSupported` |