

Oracle® Fusion Middleware

Developing Message-Driven Beans for Oracle WebLogic Server

12c (12.1.2)

E28121-01

June 2013

This document is a resource for software developers who develop applications that include WebLogic Server Message-Driven Beans (MDBs) using the Java Platform, Enterprise Edition 6.

Oracle Fusion Middleware Developing Message-Driven Beans for Oracle WebLogic Server, 12c (12.1.2)

E28121-01

Copyright © 2007, 2013, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate failsafe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Contents

Preface	vii
Documentation Accessibility	vii
Conventions	vii
1 Understanding Message-driven Beans	
1.1 JCA-Based MDBs	1-1
2 MDB Life Cycle	
2.1 Overview	2-1
2.2 MDBs and Concurrent Processing	2-1
2.3 Limitations for Multi-threaded Topic MDBs	2-2
3 MDBs and Messaging Models	
3.1 Point-to-Point (Queue) Model: One Message Per Listener	3-1
3.2 Publish/Subscribe (Topic) Model	3-2
3.3 Exactly-Once Processing	3-3
4 Deploying MDBs	
4.1 Destination and MDBs: Collocation vs. non-Collocation	4-1
4.2 Collocated Destination/MDBs	4-1
4.3 Non-Collocated Destination/MDBs	4-2
4.4 JMS Distributed Destinations	4-3
4.5 Best Practice	4-4
5 Programming and Configuring MDBs: Main Steps	
5.1 Required JMS Configuration	5-1
5.2 Create MDB Class and Configure Deployment Elements	5-2
6 Programming and Configuring MDBs: Details	
6.1 Configuring Destination Type	6-1
6.2 Configuring Transaction Management Strategy for an MDB	6-2
6.3 Configuring MDBs for Destinations	6-3
6.3.1 Whether to Use Foreign JMS Server Mappings	6-4

6.3.2	How to Set provider-url.....	6-4
6.3.3	How to Set initial-context-factory	6-4
6.3.4	How to Set destination-jndi-name	6-4
6.3.5	How to Set connection-factory-jndi-name	6-5
6.3.6	Common Destination Scenarios: Illustrations and Key Element Settings	6-5
6.4	Configuring Message Handling Behaviors	6-8
6.4.1	Ensuring Message Receipt Order	6-8
6.4.2	Preventing and Handling Duplicate Messages	6-9
6.4.3	Redelivery and Exception Handling.....	6-10
6.5	Using the Message-Driven Bean Context.....	6-11
6.6	Configuring Suspension of Message Delivery During JMS Resource Outages.....	6-11
6.7	Manually Suspending and Resuming Message Delivery	6-12
6.8	Configuring the Number of Seconds to Suspend a JMS Connection.....	6-12
6.8.1	How the EJB Container Determines How Long to Suspend a JMS Connection	6-12
6.8.2	Turning Off Suspension of a JMS Connection.....	6-13
6.9	Configuring a Security Identity for a Message-Driven Bean	6-13
6.10	Using MDBs With Cross Domain Security	6-14
6.11	Configuring EJBs to Use Logical Message Destinations	6-14
6.11.1	Configuring Logical JMS Message Destinations for Individual MDBs.....	6-14
6.11.2	Configuring Application-Scoped Logical JMS Message Destinations.....	6-15

7 Using EJB 3.1 Compliant MDBs

7.1	Implementing EJB 3.1 Compliant MDBs	7-1
7.2	Programming EJB 3.1 Compliant MDBs.....	7-1
7.2.1	MDB Sample Using Annotations	7-3

8 Migration and Recovery for Clustered MDBs

9 Using Batching with Message-Driven Beans

9.1	Configuring MDB Transaction Batching.....	9-1
9.2	How MDB Transaction Batching Works	9-2

10 Configuring and Deploying MDBs Using JMS Topics

10.1	Supported Topic Types	10-2
10.2	The Most Commonly Used MDB Attributes	10-2
10.2.1	Setting the JMS Destination, Destination Type, and Connection Factory.....	10-3
10.2.2	Setting Subscription Durability	10-3
10.2.3	Setting Automatic Deletion of Durable Subscriptions	10-4
10.2.4	Setting Container Managed Transactions	10-4
10.2.5	Setting Message Filtering (JMS Selectors)	10-4
10.2.6	Controlling MDB Concurrency.....	10-4
10.2.7	Setting Subscription Identifiers	10-5
10.2.8	Setting Message Distribution Tuning	10-5
10.2.8.1	Setting topicMessagesDistributionMode	10-5
10.2.8.2	Setting distributedDestinationConnection	10-6
10.3	Best Practices	10-8

10.3.1	Warning about Non-Transactional MDBs in Compatibility Mode	10-8
10.3.2	Warning About Using Local RDTs with Durable MDBs.....	10-8
10.3.3	Warning about Changing Durable MDB Attributes, Topic Type, EJB Name	10-8
10.3.4	Choosing Between Partitioned and Replicated Topics	10-9
10.3.5	Choosing an MDB Topic Messages Distribution Mode.....	10-9
10.3.6	Managing and Viewing Subscriptions:	10-9
10.3.7	Handling Uneven Message Loads and/or Message Processing Delays	10-9
10.4	Configuring for Service Migration	10-10
10.5	Upgrading Applications from Previous Releases	10-10
10.6	Topic MDB Sample	10-11

11 Deployment Elements and Annotations for MDBs

A Topic Deployment Scenarios

A.1	How Configuration Permutations Determine Deployment Actions	A-1
A.2	Typical Scenarios.....	A-3
A.2.1	Standalone (Non-distributed) Topic Scenarios	A-4
A.2.1.1	One-Copy-Per-Server.....	A-4
A.2.1.2	One-Copy-Per-Application	A-4
A.2.2	Replicated Distributed Topic Scenarios	A-4
A.2.2.1	Scenario 1: Replicated DT, One Copy Per Server, Local Deployment, Local Only Consumption	A-4
A.2.2.2	Scenario 2: Replicated DT, One Copy Per Server, Local Deployment, Every Member Consumption,	A-5
A.2.2.3	Scenario 3: Replicated DT, One Copy Per Server, Remote Deployment.....	A-6
A.2.2.4	Scenario 4: Replicated DT, One Copy Per Application, Local Deployment, Local Only Consumption	A-7
A.2.2.5	Scenario 5: Replicated DT, One Copy Per Application, Local Deployment, Every Member Consumption	A-8
A.2.2.6	Scenario 6: Replicated DT One Copy Per Application, Remote Deployment	A-9
A.2.3	Partitioned Distributed Topic Scenarios	A-10
A.2.3.1	Scenario 7: Partitioned DT, One Copy Per Server, Local Deployment, Local Only Consumption	A-11
A.2.3.2	Scenario 8: Partitioned DT, One Copy Per Server, Local Deployment, Every Member Consumption	A-11
A.2.3.3	Scenario 9: Partitioned DT, One Copy Per Server, Remote Deployment.....	A-12
A.2.3.4	Scenario 9: Partitioned DT, One Copy Per Application, Local Deployment, Local Only Consumption	A-12
A.2.3.5	Scenario 11: Partitioned DT, One Copy Per Application, Local Deployment, Every Member Consumption	A-12
A.2.3.6	Scenario 12: Partitioned DT, One Copy Per Application, Remote Deployment	A-13

B Topic Subscription Identifiers

C How WebLogic MDBs Leverage WebLogic JMS Extensions

Preface

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

Documentation Accessibility

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

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit

<http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	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.

Understanding Message-driven Beans

A message-driven bean (MDB) is an enterprise bean that allows Java EE 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 Java EE component—an application client, another enterprise bean, or a Web component—or by non-Java EE 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.

When a message arrives, the container calls the message-driven bean's `onMessage` method to process the message. 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 *Developing Enterprise JavaBeans 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 *Developing Enterprise JavaBeans, Version 2.1, for Oracle WebLogic Server*.

MDB Life Cycle

This chapter 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.

This chapter includes the following sections:

- [Section 2.1, "Overview"](#)
- [Section 2.2, "MDBs and Concurrent Processing"](#)
- [Section 2.3, "Limitations for Multi-threaded Topic MDBs"](#)

2.1 Overview

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.

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 and may add and remove instances during operations based on message traffic.

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."](#)

On a WebLogic Server instance, each MDB deployment maintains one or more MDB instance pools, also known as free pools, that hold MDB instances not currently servicing requests. The maximum number of MDB instances in a 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 *Tuning Performance of Oracle WebLogic Server*.

The number of free pools associated with an MDB deployment depends on the type of destination the MDB deployment is connect to. Typically, an MDB deployment is associated with a single free pool on each WebLogic Server instance that hosts the deployment. However, on each WebLogic Server instance that hosts the deployment, an MDB deployment connected to a WebLogic JMS distributed destination might have

one free pool for each physical destination associated with the distributed destination. The number of free pools is automatically determined by the EJB container; and, for MDBs associated with a JMS destination, each MDB free pool always corresponds to a single JMS connection.

In a queue-based JMS application (point-to-point model), each MDB instance creates a single internal JMS session and corresponds to an MDB thread.

A topic-based JMS application (the publish/subscribe model) may require a single instance, may share a single JMS session between multiple instances, or may create a session for each instance. This is automatically determined by the MDB container based on the message processing pattern specified by MDB application settings, the type of topic, the work-manager, and the `max-beans-in-free-pool` setting. See [Chapter 10, "Configuring and Deploying MDBs Using JMS Topics."](#) Also see "Tuning Messaging Driven Beans" in *Tuning Performance of Oracle WebLogic Server*.

2.3 Limitations for Multi-threaded Topic MDBs

The default behavior for non-transactional topic MDBs is to multi-thread the message processing. There are some limitations when using:

- Non-transactional topic MDBs that work with foreign (non-WebLogic) topics
- Non-transactional topic MDBs that consume from a WebLogic JMS topic and process messages that have a WebLogic JMS Unit-of-Order (UOO) value

For details, see the **Caution** in [Section 10.2.6, "Controlling MDB Concurrency."](#)

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 *Developing JMS Applications for Oracle WebLogic Server*.

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

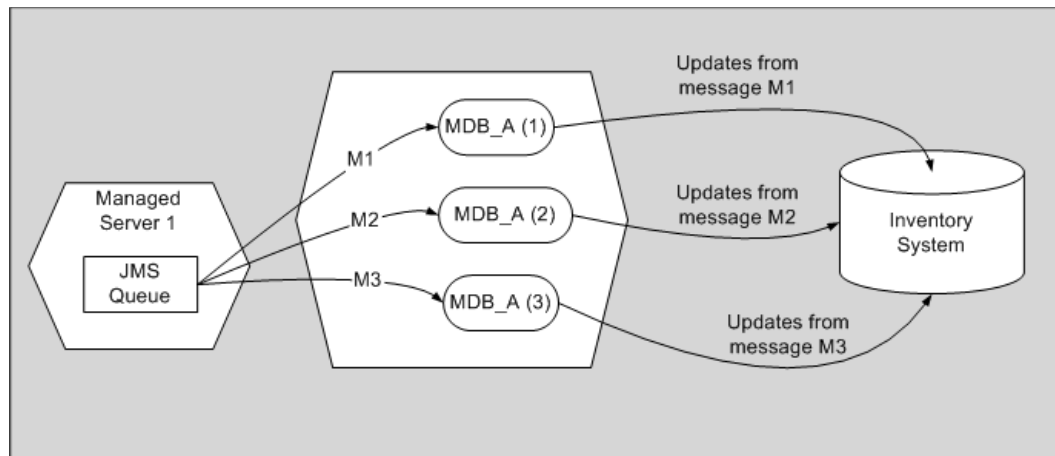
- [Section 3.1, "Point-to-Point \(Queue\) Model: One Message Per Listener"](#)
- [Section 3.2, "Publish/Subscribe \(Topic\) Model"](#)
- [Section 3.3, "Exactly-Once Processing"](#)

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).

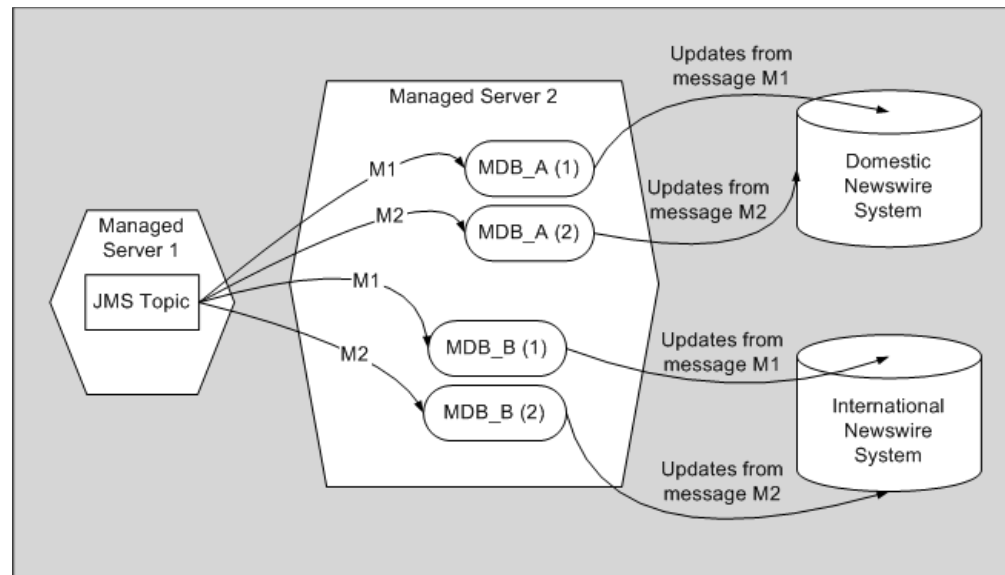
Figure 3–1 Point-to-Point Model

3.2 Publish/Subscribe (Topic) Model

In the publish/subscribe model, a JMS topic publishes a copy of each message to each logical subscription. A logical subscription may consist of one or more physical subscriptions, where each physical subscription is associated with a different member of a distributed topic. For stand-alone (non-distributed) topics, a logical subscription always consists of a single physical subscription on the topic. If an MDB goes down, that MDB will miss the message, unless the topic is a durable subscription topic. For information on durable subscriptions and for configuration instructions, see [Chapter 10.2.2, "Setting Subscription Durability."](#)

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 copy of the message is processed for each of the logical subscriptions. In this diagram, there are two logical subscriptions, where each logical subscription consists of a separate physical subscription on the single topic. MDB_A has two instances that process the messages for a single dedicated subscription. Similarly, MDB_B has two instances that process the messages for a different single dedicated subscription. Message M1 is processed by an instance of MDB_A and an instance of MDB_B. Similarly, message 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 application 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 when 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 roll back and force the message to be redelivered.

Deploying MDBs

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

This chapter includes the following sections:

- [Section 4.1, "Destination and MDBs: Collocation vs. non-Collocation"](#)
- [Section 4.2, "Collocated Destination/MDBs"](#)
- [Section 4.3, "Non-Collocated Destination/MDBs"](#)
- [Section 4.4, "JMS Distributed Destinations"](#)

4.1 Destination and MDBs: Collocation vs. non-Collocation

You can deploy an MDB on the same server instance(s) as the JMS destination on which it listens or on separate server instance(s). 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](#) illustrates a One-Copy-Per-Server topic message distribution mode pattern. Topic patterns are discussed in more detail in [Chapter 10.2.8.1, "Setting topicMessagesDistributionMode."](#)

Figure 4–1 Collocated Destination/MDBs, Distributed Topic, One-Copy-Per-Server Pattern

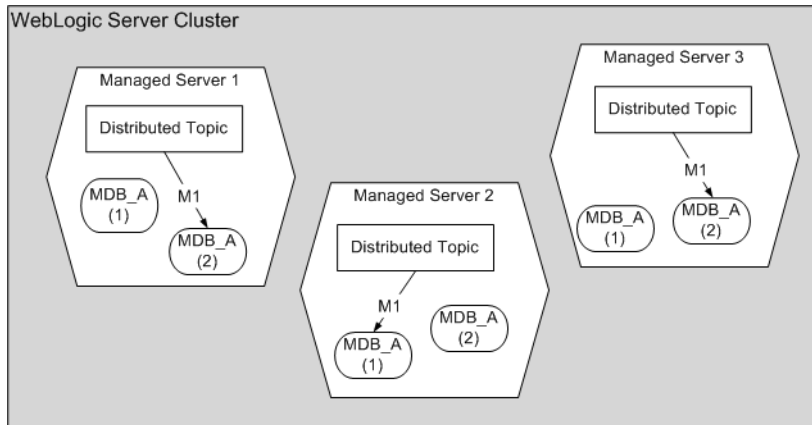
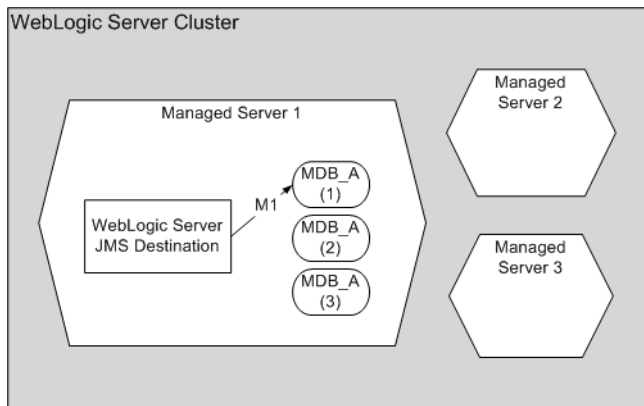


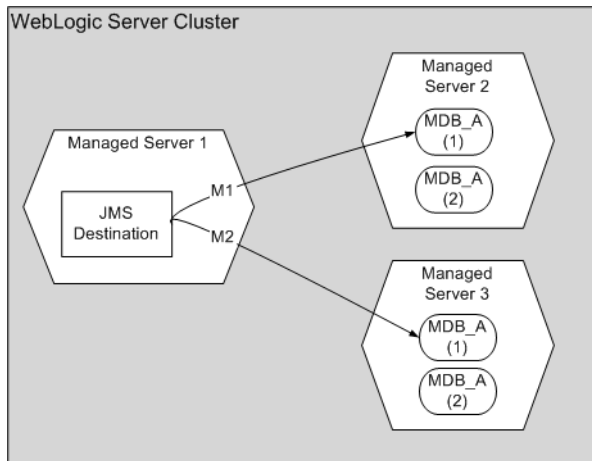
Figure 4–2 Collocated Destination/MDBs, Non-Distributed Destination



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 or the `distributedDestinationConnection` annotation. Similarly, this setting controls behavior for some topic MDB scenarios, as described in [Chapter 10, "Configuring and Deploying MDBs Using JMS Topics,"](#) and [Appendix A, "Topic Deployment Scenarios."](#)

If you deploy an MDB and the associated distributed queue to the same cluster, WebLogic Server automatically enumerates the distributed queue members and ensures that each member is serviced by at least one MDB pool. For distributed queues, there will be one MDB pool for each local member when `distributedDestinationConnection` is `LocalOnly` (the default); otherwise, for queues, when `distributedDestinationConnection` is set to `EveryMember`, each WebLogic Server instance creates multiple local MDB pools - one for each local member plus one for each remote member.

If you deploy an MDB and its associated queue to different clusters, WebLogic Server automatically enumerates the distributed queue members and ensures that each member is serviced by an MDB pool on each server in the MDB cluster. For example, if the distributed queue has three members, each JVM in the MDB cluster will create three MDB pools.

For more information about distributed topics, see [Chapter 10, "Configuring and Deploying MDBs Using JMS Topics."](#)

4.5 Best Practice

WebLogic clustering, and WebLogic JMS distributed destinations increase scalability and high availability. Oracle recommends that the machines that host a cluster have identical or similar processing power, disk space, and memory to ensure well-load-balanced message processing. Similarly, it is recommended that the WebLogic Server instances in a particular WebLogic cluster have homogenous JMS configuration and MDB deployments.

For an example, see [Figure 4-1](#). For additional information about distributed destinations, see "Using Distributed Destinations" in *Developing JMS Applications for Oracle WebLogic Server*.

Programming and Configuring MDBs: Main Steps

This chapter provides step-by-step instructions for implementing an MDB using pre-EJB 3.0 style XML descriptors to configure its behavior. For an EJB 3.1 annotation sample, see [Section 10.6, "Topic MDB Sample."](#)

For a summary of key deployment elements for MDBs, see [Chapter 11, "Deployment Elements and Annotations for MDBs."](#) For an introduction to key deployment elements for topic MDBs, see [Chapter 10, "Configuring and Deploying MDBs Using JMS Topics."](#)

This chapter contains the following sections:

- [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 access to the appropriate JMS components:

- A JMS connection factory.

A connection factory must be XA transaction (global JTA transaction) capable in order to support transactional MDBs.

The default WebLogic JMS MDB connection factory is XA-capable, is automatically generated on WebLogic clusters, and is sufficient for the majority of MDBs that consume from WebLogic JMS destinations. For information about WebLogic JMS default connection factories, see "Using a Default Connection Factory" in *Administering JMS Resources for Oracle WebLogic Server*.

For instructions about how to create a custom WebLogic JMS connection factory, see "Create connection factories in a system module" in *Oracle WebLogic Server Administration Console Online Help*.

The default behavior and configuration methods for other JMS provider connection factories 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 "Configure Messaging" in *Oracle WebLogic Server Administration Console Online 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 6.3.1, "Whether to Use Foreign JMS Server Mappings,"](#) 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

Use the following 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 invokes after creating each new instance of the MDB.
- 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 6.5, "Using the Message-Driven Bean Context,"](#).
- One `ejbRemove()` method that removes the message-driven bean instance from the free pool.

Note: Most EJB 3.1 applications implement only `javax.jms.MessageListener`, which defines a single method - `onMessage()`.

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>
```

```

        </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 6.2, "Configuring Transaction Management Strategy for an MDB."](#)
 - Destination type—The type of destination to which the MDB listens. For more information, see [Section 6.1, "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. In general, applications that follow best practices should never need to specify the `initial-context-factory` or `provider-url` fields. For instructions, see [Section 6.3, "Configuring MDBs for Destinations."](#)

4. Compile and generate the MDB class using the instructions in "Compile Java Source" in *Developing Enterprise JavaBeans, Version 2.1, for Oracle WebLogic Server*.
5. Deploy the bean on WebLogic Server using the instructions in the section "Preparing Applications and Modules for Deployment" in *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 8, "Migration and Recovery for Clustered MDBs."](#)

Programming and Configuring MDBs: Details

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

Note: This chapter uses a pre-EJB 3.0 deployment descriptor to illustrate basic MDB configuration. If you plan to use EJB 3.1 annotations, see also [Chapter 11, "Deployment Elements and Annotations for MDBs."](#) and [Chapter 7, "Using EJB 3.1 Compliant MDBs,"](#) for the equivalent settings.

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

6.1 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` or by using an annotation.

- 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`. For important additional Topic related settings see [Chapter 10, "Configuring and Deploying MDBs Using JMS Topics,"](#) and [Chapter 11, "Deployment Elements and Annotations for MDBs."](#)

- To specify a queue, set `destination-type` to `javax.jms.Queue`. For additional Queue related settings see [Chapter 11, "Deployment Elements and Annotations for MDBs."](#)

6.2 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 using descriptor elements:

- 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 3.1 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 container-level transaction management using EJB annotations:

```
import javax.ejb.TransactionAttribute;
import javax.ejb.TransactionAttributeType;
...
@TransactionAttribute(value = TransactionAttributeType.REQUIRED)
public void onMessage(Message msg) {
...
}
```

To configure bean-level transaction management using descriptor elements:

- 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://www.oracle.com/technetwork/java/jms/index.html#AUTO_ACKNOWLEDGE
 - `DUPS_OK_ACKNOWLEDGE` as described at http://www.oracle.com/technetwork/java/jms/index.html#DUPS_OK_ACKNOWLEDGE

For more information, see "Session" in *Developing JMS Applications for Oracle WebLogic Server*.

6.3 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 mappings to appear in the MDB's JNDI context. This approach is discussed in [Section 6.3.1, "Whether to Use Foreign JMS Server Mappings."](#)

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 foreign JMS server mappings. These mappings 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 foreign JMS server mappings are appropriate, and the rules for configuring the `message-driven-descriptor` in `weblogic-ejb-jar.xml`, see these sections:

- [Section 6.3.1, "Whether to Use Foreign JMS Server Mappings"](#)
- [Section 6.3.2, "How to Set provider-url"](#)
- [Section 6.3.3, "How to Set initial-context-factory"](#)
- [Section 6.3.4, "How to Set destination-jndi-name"](#)
- [Section 6.3.5, "How to Set connection-factory-jndi-name"](#)

To configure the elements in `message-driven-descriptor` for specific scenarios, see [Section 6.3.6, "Common Destination Scenarios: Illustrations and Key Element Settings."](#)

6.3.1 Whether to Use Foreign JMS Server Mappings

Using mappings 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 mappings is recommended if you use a foreign JMS provider or a remote WebLogic JMS provider. For more information on JMS mapping classes, see "Simplified Access to Remote or Foreign JMS Providers" in "Enhanced Support for Using WebLogic JMS with EJBs and Servlets" in *Developing JMS Applications for Oracle WebLogic Server*.
- If you use a mapping for either the connection factory or the destination, you must create and use mappings for each of these objects.

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

6.3.2 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 mappings, specify `provider-url`.
 - You do use mappings, do not specify `provider-url`. The URL is implicitly encoded in the mapping.

6.3.3 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 mappings, specify the initial context factory used by the JMS provider.
 - you do use mappings, do not specify `initial-context-factory`.

6.3.4 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 mappings, specify the name of the destination, as bound in the foreign provider's JNDI tree.
 - You do use mappings, specify the name Foreign Destination you set up in your local JNDI tree that corresponds the remote or foreign destination.

6.3.5 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 WebLogic Server Administration Console Online 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 mappings, specify the name of the connection factory used by the JMS provider, as bound in the remote JNDI tree.
 - You do use mappings, 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.

6.3.6 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 mappings are included.

- [Figure 6–1, "A. Destination on a Local WebLogic JMS Server"](#)
- [Figure 6–2, "B. Destination On a Remote WebLogic JMS Server—No Mappings"](#)
- [Figure 6–3, "C. Destination on Foreign JMS Server—No Mappings"](#)
- [Figure 6–4, "D. Destination on a Remote WebLogic Server or Foreign JMS Server—With Mappings"](#)

[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

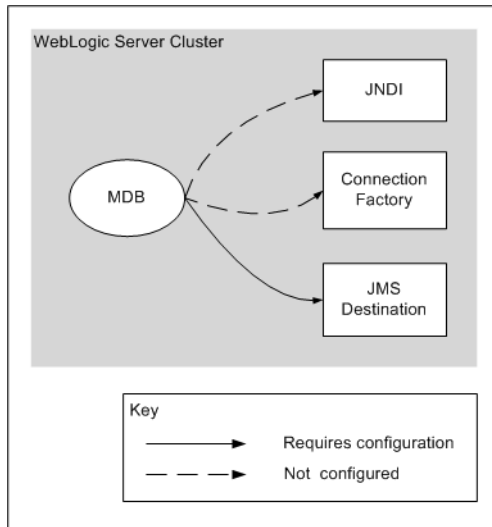


Figure 6–2 B. Destination On a Remote WebLogic JMS Server—No Mappings

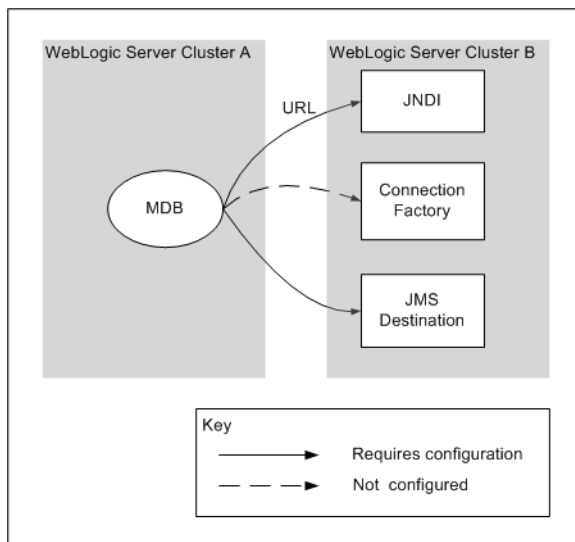


Figure 6–3 C. Destination on Foreign JMS Server—No Mappings

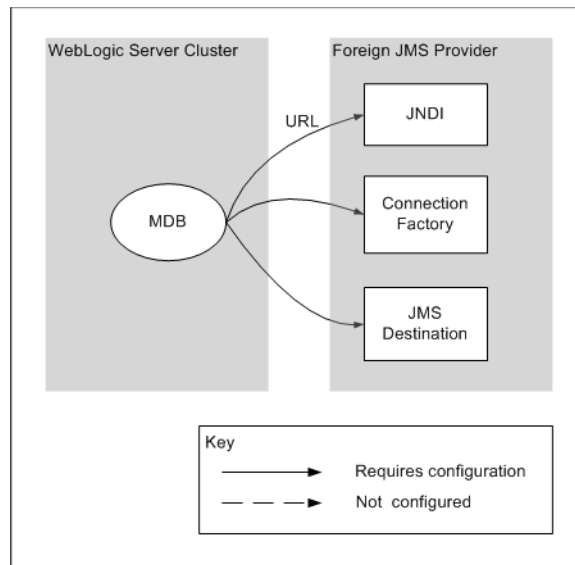


Figure 6–4 D. Destination on a Remote WebLogic Server or Foreign JMS Server—With Mappings

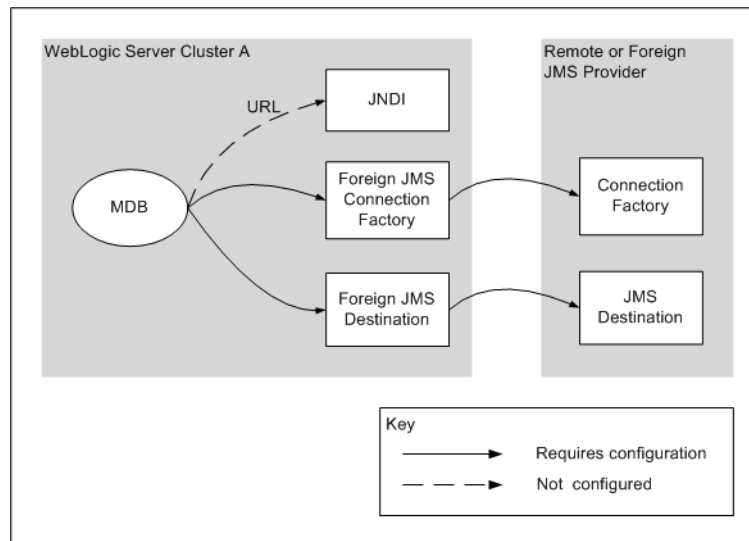


Table 6–1 Common Configuration Scenarios

Scenario	If destination is on...	Mappings 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 mappings 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 mappings 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 or Foreign JMS server	Mappings 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

6.4 Configuring Message Handling Behaviors

These topics provide guidelines for behaviors related to message delivery:

- [Section 6.4.1, "Ensuring Message Receipt Order"](#)
- [Section 6.4.2, "Preventing and Handling Duplicate Messages"](#)
- [Section 6.4.3, "Redelivery and Exception Handling"](#)

6.4.1 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.

When using WebLogic JMS destinations, Oracle recommends using the Unit-of-Order feature if strict ordering is required. This feature enforces ordering under all circumstances without requiring modification of the MDB, enables concurrent processing of sub-orderings that exist within the same destinations, and can be enabled via configuration or programmatically as appropriate. See "Using Message Unit-of-Order" in *Developing JMS Applications for Oracle WebLogic Server*.

If you are not using WebLogic destinations with unit-of-order 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. Foreign vendors have different names for the

equivalent setting. This setting controls the number of messages that a vendor may push to a consumer before the consumer completes processing of its current message.

For more information see "Ordered Redelivery of Messages" in *Developing JMS Applications for Oracle WebLogic Server*.

See the Java documentation on the Interface

`MessageListener`—`javax.jms.MessageListener.onMessage()`—for more information, at

<http://download.oracle.com/javaee/1.2.1/api/javax/jms/MessageListener.html>.

6.4.2 Preventing and Handling Duplicate Messages

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

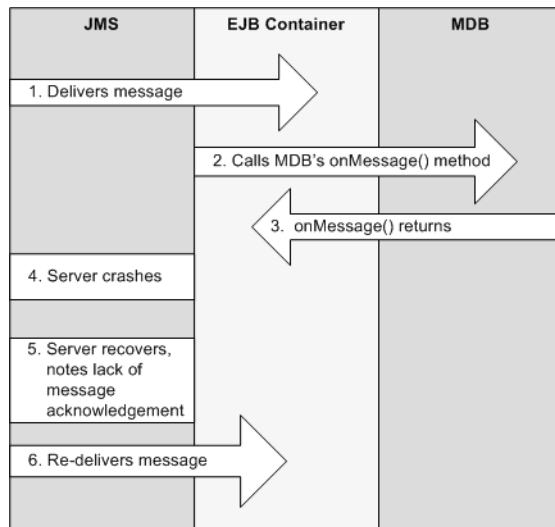
Your MDB design should allow for the likelihood of duplicate messages. Duplicate messages can be undesirable in certain cases. For example, if an MDB's `onMessage()` method includes code to debit a bank account, receiving and processing that message twice would result in the account being debited twice.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



6.4.3 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 recreated, which incurs a performance penalty. If you want to avoid the overhead of a destroy and recreate, while still throwing a runtime exception, you can use a WebLogic extension. Throw an instance of a `weblogic.ejb.NonDestructiveRuntimeException`, for example, `throw new weblogic.ejb.NonDestructiveRuntimeException("force redelivery");`

You may want to configure a 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 that do not use the Unit-of-Order feature, 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 *Developing JMS Applications for Oracle WebLogic Server*.

6.5 Using the Message-Driven Bean Context

WebLogic Server calls `setMessageDrivenContext()` to associate the MDB instance with a container context. Alternatively, EJB 3.1 MDB applications can specify an annotation that injects the MDB 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`.

6.6 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 ten 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`.

6.7 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 WebLogic Server Administration Console Online Help*.

6.8 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.xml` 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 suspend a JMS connection when the EJB container detects a JMS resource outage. The default value is 60.

6.8.1 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.

6.8.2 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.

6.9 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 *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 WebLogic Server Administration Console Online Help*.

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

6.10 Using MDBs With Cross Domain Security

MDBs may require you to configure Cross Domain Security. 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 *Developing JTA Applications for Oracle WebLogic Server*.

- You must configure Cross Domain Security in cases where an MDB listens to a distributed destination in a different domain.
- MDBs handling non-transactional messages do not require you to configure Cross Domain Security. However, you must configure Cross Domain Security for all the domains with which your process communicates, 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. You must configure Cross Domain Security for cases where an MDB listens to a distributed destination that is in a different domain.

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 *Administering Security for Oracle WebLogic Server*.

6.11 Configuring EJBs to Use Logical Message Destinations

Note: Logical destinations and application-scoped destinations are not commonly used and are for advanced users only. For most users, Oracle recommends using the methods discussed in [Section 6.3](#), "Configuring MDBs for Destinations."

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 *Developing Enterprise JavaBeans, Version 2.1, for Oracle WebLogic Server*.

6.11.1 Configuring Logical JMS Message Destinations for Individual MDBs

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 the `weblogic-ejb-jar.xml` file.

6.11.2 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 *Developing JMS Applications for Oracle WebLogic Server* and *Developing JDBC Applications 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 6.11.1, "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 the `weblogic-ejb-jar.xml` file.
- `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`.

Using EJB 3.1 Compliant MDBs

This chapter describes how to program and implement EJB 3.1 compliant MDBs:

- [Section 7.1, "Implementing EJB 3.1 Compliant MDBs"](#)
- [Section 7.2, "Programming EJB 3.1 Compliant MDBs"](#)

7.1 Implementing EJB 3.1 Compliant MDBs

To implement EJB 3.1 compliant MDBs, follow the steps described in "Overview of the EJB Development Process" in *Developing Enterprise JavaBeans for Oracle WebLogic Server*.

7.2 Programming EJB 3.1 Compliant MDBs

To program EJB 3.1 compliant MDBs, follow the steps described in "Programming the Bean File: Typical Steps" in *Developing Enterprise JavaBeans for Oracle WebLogic Server*.

You must use the `@javax.ejb.MessageDriven` annotation to declare the EJB type as message-driven. You can specify the following optional attributes:

- `messageListenerInterface`—Specifies the message listener interface, if you haven't explicitly implemented it or if the bean implements additional interfaces.

Note: The bean class must implement, directly or indirectly, the message listener interface required by the messaging type that it supports or the methods of the message listener interface. In the case of JMS, this is the `javax.jms.MessageListener` interface.

- `activationConfig`—Specifies an array of activation configuration properties that configure the bean in its operational environment.

Activation configuration properties are name-value pairs that are passed to the MDB container when an MDB is deployed. The properties can be declared in either an `ejb-jar.xml` deployment descriptor or by using the `@ActivationConfigProperty` annotation on the MDB bean class. An example using the `@ActivationConfigProperty` annotation is shown in [Example 7-1](#). An example using the `ejb-jar.xml` deployment descriptor is shown in [Example 7-2](#).

Example 7-1 Example `@ActivationConfigProperty` Code

```
. . .
@ActivationConfigProperties(
{
```

```

        @ActivationConfigProperty(
            name="connectionFactoryJndiName", value="JNDINameOfMDBSourceCF"
        ),
        @ActivationConfigProperty(
            name="initialContextFactory",
            value="weblogic.jndi.WLInitialContextFactory"
        )
    }
)
. . .

```

To set activation configuration properties in the `ejb-jar.xml` descriptor, use the `activation-config-property` element in the message-driven stanza, as shown in [Example 7-2](#).

Example 7-2 Activation Configuration Properties Set in `ejb-jar.xml`

```

<message-driven>
. . .
<activation-config>
  <activation-config-property>
    <activation-config-property-name>destinationJNDIName</activation-config-property-name>
    <activation-config-property-value>myQueue</activation-config-property-value>
  </activation-config-property>
  <activation-config-property>
    <activation-config-property-name>destinationType</activation-config-property-name>
    <activation-config-property-value>javax.jms.Queue</activation-config-property-value>
  </activation-config-property>
</activation-config>
. . .
</message-driven>
<message-driven>
. . .
<activation-config>
  <activation-config-property>
    <activation-config-property-name>destinationJNDIName</activation-config-property-name>
    <activation-config-property-value>myQueue</activation-config-property-value>
  </activation-config-property>
  <activation-config-property>
    <activation-config-property-name>destinationType</activation-config-property-name>
    <activation-config-property-value>javax.jms.Queue</activation-config-property-value>
  </activation-config-property>
</activation-config>
. . .
</message-driven>

```

Because activation configuration properties can be set in an `ejb-jar.xml` deployment descriptor or by using `activationConfigProperty` annotation properties, conflicts may result if the same name is used in both places. Conflicts may also result from using same-named properties from pre-3.0 versions of EJB or from proprietary WebLogic Server EJB annotations. Such conflicts are resolved following the following priority order, in sequence from high to low is

1. Properties set in the `weblogic-ejb-jar.xml` deployment descriptor
2. Proprietary WebLogic Server 10.0 (and later) annotations
3. `activation-config-property` properties set in the `ejb-jar.xml` deployment descriptor
4. `activationConfigProperty` annotation properties

For example, if the same property exists in the `weblogic-ejb-jar.xml` descriptor and the `ejb-jar.xml` descriptor, the one in `weblogic-ejb-jar.xml` has the higher priority and overrides the one in `ejb-jar.xml` value. Or, if the same property is set in both an `ejb-jar.xml` descriptor element and in an `activationConfigProperty` annotation, the descriptor element takes precedence and the annotation is ignored.

For more information about activation configuration properties, see "javax.ejb.ActivationConfigProperty" in *Developing Enterprise JavaBeans for Oracle WebLogic Server*. Also see [Table 11–1](#), which summarizes the activation configuration properties supported by WebLogic Server.

Note: Based on the Enterprise JavaBean specification, the `javax.ejb.ActivationConfigProperty` annotation is used for MDBs only. This annotation is not used for session or entity beans.

For detailed information on developing MDBs to support the messaging modes as described in [Section 3, "MDBs and Messaging Models,"](#) see [Section 6, "Programming and Configuring MDBs: Details."](#)

7.2.1 MDB Sample Using Annotations

[Example 7–3](#) shows a WebLogic MDB that uses a subscription to a WebLogic JMS queue (from WebLogic Server 10.3.4 or later), transactionally processes the messages, and forwards the messages to a target destination.

The MDB connects using JMS connection factory `MyCF` to receive from queue `MyQueue`. It forwards the messages to `MyTargetDest` using a connection generated from connection factory `MyTargetCF`.

Resource reference pooling note: The MDB uses a resource reference to access `MyTargetCF`. The resource reference automatically enables JMS producer pooling, as described in "Enhanced Support for Using WebLogic JMS with EJBs and Servlets" in *Developing JMS Applications for Oracle WebLogic Server*.

For a similar sample using topics instead of queues, see [Example 10–1, "Sample MDB Using Distributed Topics"](#).

Example 7–3 Sample MDB Using Distributed Queues

```
package test;
import javax.annotation.Resource;
import javax.annotation.Resource;
import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.ejb.MessageDrivenContext;
import javax.ejb.TransactionAttribute;
import javax.ejb.TransactionAttributeType;
import javax.jms.*;

@MessageDriven(
    name = "MyMDB",
    activationConfig = {
        @ActivationConfigProperty(propertyName = "destinationType",
            propertyValue = "javax.jms.Queue"),

        @ActivationConfigProperty(propertyName = "connectionFactoryJndiName",
            propertyValue = "MyCF"), // External JNDI Name
```

```
        @ActivationConfigProperty(propertyName = "destinationJndiName",
                                   propertyValue = "MyQueue") // Ext. JNDI Name
    }
)

@Resource ( {
    @Resource (name="targetCFRef",
               mappedName="MyTargetCF", // External JNDI name
               type=javax.jms.ConnectionFactory.class),

    @Resource (name="targetDestRef",
               mappedName="MyTargetDest", // External JNDI name
               type=javax.jms.Destination.class)
})

public class MyMDB implements MessageListener {

    // inject a reference to the MDB context

    @Resource
    private MessageDrivenContext mdctx;

    // cache targetCF and targetDest for re-use (performance)

    private ConnectionFactory targetCF;
    private Destination targetDest;

    @TransactionAttribute(value = TransactionAttributeType.REQUIRED)
    public void onMessage(Message message) {

        System.out.println("My MDB got message: " + message);

        // Forward the message to "MyTargetDest" using "MyTargetCF"

        Connection jmsConnection = null;

        try {
            if (targetCF == null)
                targetCF = (javax.jms.ConnectionFactory)mdctx.lookup("targetCFRef");

            if (targetDest == null)
                targetDest = (javax.jms.Destination)mdctx.lookup("targetDestRef");

            jmsConnection = targetCF.createConnection();
            Session s = jmsConnection.createSession(false, Session.AUTO_ACKNOWLEDGE);
            MessageProducer mp = s.createProducer(null);

            mp.send(targetDest, message);

        } catch (JMSEException e) {

            System.out.println("Forcing rollback due to exception " + e);
            e.printStackTrace();
            mdctx.setRollbackOnly();

        } finally {

            // Closing a connection automatically returns the connection and
            // its session plus producer to the resource reference pool.
        }
    }
}
```

```
        try { if (jmsConnection != null) jmsConnection.close(); }
        catch (JMSEException ignored) {};
    }

    // emulate 1 second of "think" time

    try { Thread.currentThread().sleep(1000); }
    catch (InterruptedException ie) {
        Thread.currentThread().interrupt(); // Restore the interrupted status
    }
}
}
```

Migration and Recovery for Clustered MDBs

WebLogic Server supports migration and recovery for clustered JMS destinations. In the event of failure, you can bring a JMS destination back online on a different JVM. You can design your cluster so that when a server instance fails, it automatically migrates the JMS destination from the failed server in the cluster to an available server instance.

In turn, any MDB deployment associated with a migrated JMS destination is automatically updated. Such an update may include closing and reinitializing MDB pools and/or reconnecting to the JMS destination

Caution: Service migration is not recommended for the following cases. In these cases, migration can result in either missing messages or duplicate message processing.

Case 1, when all of the following are true:

- The MDB `topicMessagesDistributionMode` is `One-Copy-Per-Server`
- The MDB `distributedDestinationConnection` is `LocalOnly`
- The MDB is `Durable`
- The destination is configured as the logical name of a replicated distributed topic

Case 2, when all of the following are true:

- The MDB `topicMessagesDistributionMode` is `Compatibility`
- The MDB is `Durable`
- The destination is configured as the logical name of a distributed topic

Case 3, when all of the following are true:

- The MDB `topicMessagesDistributionMode` is `One-Copy-Per-Application`
- The MDB `distributedDestinationConnection` is `LocalOnly`
- The migration target server has no MDB instance. Best practice is to target MDB deployments to the entire cluster, to avoid this problem.

For more information on topic message processing, see [Chapter 10, "Configuring and Deploying MDBs Using JMS Topics."](#)

Note: A migratable service works with clustered servers only. A WebLogic JMS destination can migrate to another server within a cluster, but cannot migrate to a different cluster.

After a WebLogic JMS destination migrates to another server, an MDB deployment, or "connection poller," reconnects to the migrated JMS destination and begins to receive messages from the JMS destination again; the MDB may also create and close pools as needed.

MDBs can be targeted to clusters or individual WebLogic Server instances, but not to migratable targets. If an MDB is running in the same cluster as a migratable destination, you must ensure that MDB is deployed everywhere that its source destination may be hosted. You can do this in two ways:

- Deploy MDBs homogeneously to the cluster. (Recommended)
- Ensure that the MDB's target set includes all WebLogic Server instances that are in the candidate lists for the migratable targets in the `config.xml` file used by the JMS servers that host the destination. For more information on configuring

migratable targets, see "Understanding Migratable Target Servers in a Cluster" in *Administering Clusters for Oracle WebLogic Server*.

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 *Administering JMS Resources for Oracle WebLogic Server*.

Using Batching with Message-Driven Beans

Within an MDB, business logic, possibly including database transactions, is performed within the `onMessage()` method. Within an EJB application, multiple MDBs can perform multiple `onMessage()` calls concurrently. If each `onMessage()` call performs a container-managed transaction, this can create a lot of overhead.

WebLogic Server provides a mechanism for grouping multiple container-managed transaction MDB `onMessage()` calls together under a single transaction. This mechanism can help increase the performance of an EJB application by implicitly grouping all of the work from different `onMessage()` calls into a single request.

For information on transaction management within MDBs, see [Section 6.2, "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.

- [Section 9.1, "Configuring MDB Transaction Batching"](#)
- [Section 9.2, "How MDB Transaction Batching Works"](#)

9.1 Configuring MDB Transaction Batching

You can enable MDB transaction batching by defining the `max-messages-in-transaction` element or using the equivalent property in `activationConfigProperty`. The 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 by increasing the value of `trans-timeout-seconds` attribute of `weblogic-ejb-jar.xml`. Alternatively, you can use `@TransactionTimeoutSeconds` annotation, as follows:

```
import weblogic.javaee.TransactionTimeoutSeconds;
```

```

...;
@TransactionTimeoutSeconds(value = 60);
...;
public class MyMDB ...

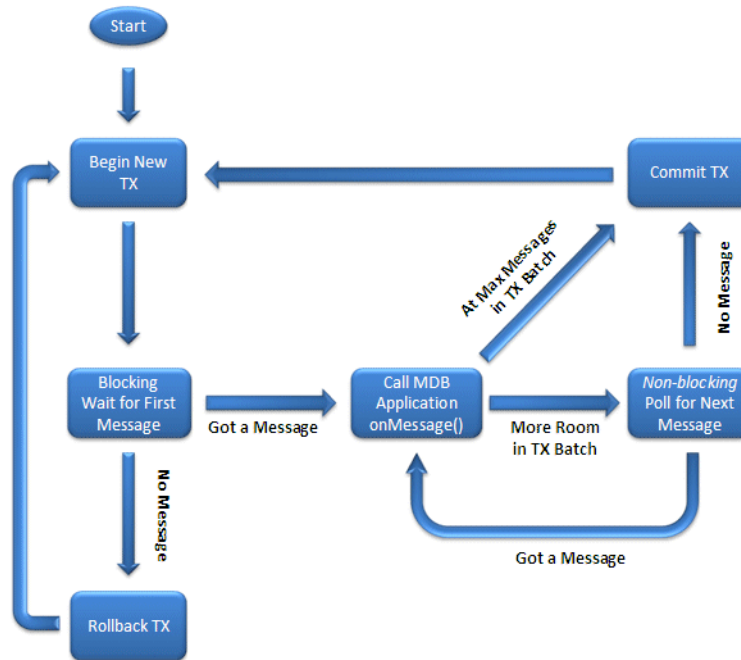
```

9.2 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 transaction for a batch. A message is added to the transaction until the number of messages in the transaction is equal to the batch size defined by `max-messages-in-transaction` or the equivalent property in `activationConfigProperty`. When the number of messages in the equals `max-messages-in-transaction` or there is no next message to be added to the transaction, the transaction is submitted for processing. See [Figure 9-1](#).

Figure 9-1 *MDB Transaction Batching Transaction Processing Flow*



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.

Configuring and Deploying MDBs Using JMS Topics

This chapter describes how to develop an MDB that automatically sets up JMS topic subscriptions and then processes subscription messages. A message that is published to a JMS topic is replicated to all subscriptions that have a matching selector filter.

A single deployed MDB may create multiple topic subscriptions and may have one or more free pools per host WebLogic Server instance. This behavior is controlled by MDB attribute settings, topic type, and whether the MDB is running on the same cluster or JVM as its topic. (For information about MDB free pools, see [Section 2.2, "MDBs and Concurrent Processing."](#))

This chapter also describes how to use topic MDBs together with WebLogic JMS distributed topics. WebLogic JMS distributed topics are logical topics that are composed of multiple physical topics, where each physical topic is hosted on a different JMS Server instance. This distributed topic capability was significantly enhanced in WebLogic Server 10.3.4 to provide increased scalability and high availability. The enhancements include direct support for remotely hosted distributed topics, for fully distributing the processing of a single logical subscription across multiple physical subscriptions, and for multiple JVMs to process messages from the same physical subscription.

This chapter is organized as follows:

- [Section 10.1, "Supported Topic Types"](#)
- [Section 10.2, "The Most Commonly Used MDB Attributes"](#)
- [Section 10.3, "Best Practices"](#)
- [Section 10.4, "Configuring for Service Migration"](#)
- [Section 10.5, "Upgrading Applications from Previous Releases"](#)
- [Section 10.6, "Topic MDB Sample"](#)

For additional information about JMS topics see:

- [Appendix A, "Topic Deployment Scenarios"](#)
- [Appendix B, "Topic Subscription Identifiers"](#)
- [Appendix C, "How WebLogic MDBs Leverage WebLogic JMS Extensions"](#)
- "Developing Advanced Pub/Sub Applications" in *Developing JMS Applications for Oracle WebLogic Server*.
- "Tuning WebLogic JMS" in *Tuning Performance of Oracle WebLogic Server*
- "Tuning Message-Driven Beans" in *Tuning Performance of Oracle WebLogic Server*.

Oracle recommends reviewing the previous chapters of this book before reading this chapter.

10.1 Supported Topic Types

WebLogic MDBs support the following types of topics:

- **Singleton topics** -- A singleton topic is either a non-distributed WebLogic JMS topic or a reference to a particular member topic of a WebLogic JMS distributed topic. The JNDI name syntax for a WebLogic JMS uniform distributed topic member is based on the name of the JMS server that hosts the member:
jms-server-name@jndi-name-of-distributed-topic.
- **Foreign provider topics** -- Non-WebLogic JMS topics are called *foreign provider topics*. MDBs treat foreign provider topics similarly to singleton topics. Such topics are typically considered to be remote.
- **Replicated distributed topics** -- WebLogic JMS distributed topics are logical topics composed of multiple physical topics, where each physical topic is hosted on a different JMS server instance in the same cluster. In releases of WebLogic Server prior to 10.3.4, each message sent to any member of a distributed topic is always automatically replicated (forwarded) to all subscriptions on all of the other members of the distributed topic. This kind of distributed topic is still supported and is now called a *replicated distributed topic* (abbreviated *RDT*).
- **Partitioned distributed topics** - WebLogic JMS distributed topics are logical topics composed of multiple physical topics, where each physical topic is hosted on a different JMS server instance in the same cluster. A *partitioned distributed topic* (abbreviated *PDT*) does not forward messages between members. Messages published to a member of a PDT are only copied to subscriptions on that member. Partitioned distributed topics are supported starting with WebLogic Server 10.3.4.

To configure a distributed topic type, you set `Partitioned` or `Replicated` as the value for the JMS distributed topic configuration attribute `JMS_Forwarding_Policy`. For more information, see "Configuring Partitioned Distributed Topics" in *Administering JMS Resources for Oracle WebLogic Server*.

10.2 The Most Commonly Used MDB Attributes

The most commonly used topic MDB attributes are:

- JMS destination and connection factory
- Subscription durability
- Container managed transactions
- Message distribution tuning

Some other useful topic MDB attributes are:

- Free pool size
- Auto-delete on undeploy
- Message filtering (JMS selectors)
- Subscription identifier

The message distribution tuning settings include the `topicMessagesDistributionMode`, `distributedDestinationConnection`, and `generate-unique-client-id` attributes.

Most attributes can be configured either by using an annotation or via descriptor XML stanzas. In addition, specific attribute names for descriptor XML stanzas and annotations are summarized in the tables in [Chapter 11, "Deployment Elements and Annotations for MDBs."](#)

10.2.1 Setting the JMS Destination, Destination Type, and Connection Factory

A topic MDB's configuration must properly specify the location of its JMS connection factory, its destination, and its destination type. Typically, this is accomplished by:

1. Specifying a topic type. In the `message-driven-destination` element of `ejb-jar.xml`, set `destination-type` to `javax.jms.Topic`. Alternatively, if using annotations, specify an `ActivationConfigProperty` with `propertyName = "destinationType"` and `propertyValue = "javax.jms.Topic"`.
2. Specifying a connection factory JNDI name and a destination JNDI name. Specifying a connection factory JNDI name is usually not necessary if the connection factory is hosted on the same cluster or server as the MDB. The default usually suffices.
3. If the destination is not located in the same cluster or server as the MDB pool, administratively configure a mapping from the remote destination and connection factory JNDI entries to local JNDI entries that match those specified in #2, above. There are alternative approaches to referencing remote resources, but the mapping approach is the Oracle-recommended best practice.

For each free pool, the MDB container creates a connection using the specified connection factory, then uses the connection to find or create one or more subscriptions on its destination, and finally uses the connection to create JMS consumers that receive the messages from the subscription(s).

For the specific names of connection factory and destination MDB attributes, as well as recommended JNDI mapping configuration, see [Section 6.3, "Configuring MDBs for Destinations."](#)

10.2.2 Setting Subscription Durability

MDBs automatically create subscriptions on JMS topics. JMS topics support two types of subscriptions: *durable* and *non-durable*.

- Non-durable subscriptions exist only for the length of time their subscribers exist. When a subscription has no more subscribers, the subscription is automatically deleted. Messages stored in a non-durable subscription are never recovered after a JMS server shut down or crash.
- Durable subscriptions make it possible for a subscriber to receive messages that are published while the subscriber application is unavailable. For each durable subscription on a topic, JMS stores a copy of each published persistent message in a file or database until it can be delivered (or until it expires), even if there are no active subscribers on the subscription at the time the message is delivered. JMS also stores a copy of each non-persistent message in each durable subscription, but such messages are not recovered if the JMS server shuts down or crashes.

Non-durable subscriptions are the default. To specify a durable subscription, in the `message-driven-destination` element of `ejb-jar.xml`, set `subscription-durability` to `Durable`. Alternatively, when using annotations, specify an `ActivationConfigProperty` with `propertyName = "subscriptionDurability"` and `propertyValue = "Durable"`.

10.2.3 Setting Automatic Deletion of Durable Subscriptions

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`.

10.2.4 Setting Container Managed Transactions

See [Section 6.2, "Configuring Transaction Management Strategy for an MDB."](#)

10.2.5 Setting Message Filtering (JMS Selectors)

JMS provides an SQL-like syntax for filtering messages based on standard JMS message header fields and message properties. In addition, WebLogic JMS supports an extension to the selector syntax that allows the specification of selectors that include XML "xpath" expressions for filtering XML messages based on their XML contents.

One way to specify a message selector is to specify it as the `propertyValue` for an `ActivationConfigProperty` with `propertyName = "messageSelector"`.

The syntax of JMS selectors is fully described in the Javadoc for the `javax.jms.Message` class. The WebLogic xpath selector extension syntax is described in "Filtering Messages" in *Developing JMS Applications for Oracle WebLogic Server*.

10.2.6 Controlling MDB Concurrency

As discussed in [Appendix A, "Topic Deployment Scenarios,"](#) an MDB deployment may create one or more MDB free pools. The `max-beans-in-free-pool` and `dispatch-policy` descriptor attributes work together to control MDB thread concurrency in an MDB free pool as follows:

- For a discussion of how to determine the number of concurrent MDBs, see "Determining the Number of Concurrent MDBs" in *Tuning Performance of Oracle WebLogic Server*.
- When an MDB `topicMessagesDistributionMode` is set to `Compatibility` and the MDB uses container-managed transactions, concurrent MDB invocations are prevented. In addition, `max-beans-in-free-pool` should be explicitly set to 1 for bean-managed transaction MDBs that are driven by a foreign (non-WebLogic) topic.

Caution:

Non-transactional Foreign Topics: Oracle recommends explicitly setting `max-beans-in-free-pool` to 1 for non-transactional MDBs that work with foreign (non-WebLogic) topics. Failure to do so may result in lost messages in the event of certain failures, such as the MDB application throwing `Runtime` or `Error` exceptions.

Unit-of-Order: Oracle recommends explicitly setting `max-beans-in-free-pool` to 1 for non-transactional `Compatibility` mode MDBs that consume from a WebLogic JMS topic and process messages that have a WebLogic JMS `Unit-of-Order` value. `Unit-of-Order` messages in this use case may not be processed in order unless `max-beans-in-free-pool` is set to 1.

See "Tuning Message-Driven Beans" in *Tuning Performance of Oracle WebLogic Server* for more information.

10.2.7 Setting Subscription Identifiers

Individual JMS topic subscriptions are created and referenced based on their "subscription identifier," which an MDB generates based on a number of MDB configuration settings. For a discussion of the syntax of generated subscription identifiers, see [Appendix B, "Topic Subscription Identifiers."](#)

10.2.8 Setting Message Distribution Tuning

This section describes how and when to use message distribution tuning settings. It contains information that applies to all topic types (singleton, foreign, and distributed). The settings include the `topicMessagesDistributionMode`, `distributedDestinationConnection`, and `generate-unique-client-id` attributes. They control where topic subscriptions are created, what the subscription identifiers are, and whether an MDB processes each published topic message only once or once per server.

For detailed descriptions and diagrams of the resulting automatically generated subscription IDs, subscription locations, and deployed MDB free pool locations, see [Appendix A, "Topic Deployment Scenarios,"](#) and [Appendix B, "Topic Subscription Identifiers."](#)

10.2.8.1 Setting `topicMessagesDistributionMode`

Use the `topicMessagesDistributionMode` setting in combination with the `distributedDestinationConnection` setting or the `generate-unique-client-id` setting to control topic message processing behavior. To set the `topicMessagesDistributionMode`, you can use the same-named `@ActivationConfigProperty` annotation or specify an `<activation-config-property>` in the `ejb-jar.xml` deployment descriptor.

The valid values for `topicMessagesDistributionMode` are:

- **One-Copy-Per-Application** -- Specifies that the MDB application as a whole receives each message published to a distributed topic once, no matter how many servers host the application. This mode works with WebLogic JMS singleton and distributed topics in WebLogic Server 10.3.4 and later.
- **One-Copy-Per-Server** -- Specifies that each deployment instance of the MDB application receives every message published to a distributed topic. This mode works with WebLogic JMS singleton and distributed topics in WebLogic Server 10.3.4 and later.
- **Compatibility** - (Default) Specifies that the MDB application handles messages from distributed topics in the same way they were handled in WebLogic Server releases prior to 10.3.4. The mode supports durable and non-durable subscriptions with foreign (non-WebLogic) topics, local replicated distributed topics (RDTs), and singleton WebLogic topics; it also supports non-durable subscriptions with a remote replicated distributed topics. See the Compatibility notes section below for more detail.

Note: Oracle recommends using the `One-Copy-Per-Application` and `One-Copy-Per-Server` modes for most new applications, except those that must consume from WebLogic JMS topics in versions of WebLogic Server prior to 10.3.4 or from foreign (non-WebLogic) topics.

The topic distribution modes support different topic types and versions with the following restrictions:

- The `One-Copy-Per-Application` and `One-Copy-Per-Server` modes work only with WebLogic singleton and distributed topics in WebLogic Server 10.3.4 and later. WebLogic MDBs log a warning and do not process messages with these modes when using a foreign (non-WebLogic) topic or when using a WebLogic topic from WebLogic Server releases prior to 10.3.4.
- `One-Copy-Per-Application` topic MDBs that are durable, that subscribe to a *local* RDT, and that use the default `LocalOnly` value for the `distributedDestinationConnection` attribute, do not support Service Migration and require that exactly one topic member be configured per WebLogic Server instance. If a service migration occurs, if there is no local topic member configured, or if more than one topic member is deployed per server, then the application may experience duplicate or lost messages and may also create abandoned subscriptions that accumulate unprocessed messages. If service migration is required, then use the `EveryMember` option for the `distributedDestinationConnection` attribute instead of the default `LocalOnly`.
- The `Compatibility` mode supports durable and non-durable subscriptions with foreign (non-WebLogic) topics, with local replicated distributed topics (RDTs) (with limitations described later), and with singleton WebLogic topics. `Compatibility` mode also supports non-durable subscriptions with a remote RDT. A deployment of a durable MDB that subscribes to the logical JNDI name of a remote RDT may succeed, but the MDB deployment will fail to connect, with `Warning` log messages. Similarly, a deployment of an MDB that subscribes to a WebLogic PDT may succeed, but the MDB deployment will fail to connect, with `Warning` log messages. See "[Notes on the Compatibility mode of topicMessagesDistributionMode](#)" below, for more detail.
- `Compatibility` mode MDBs that are durable and subscribes to a local RDT, see [Section , "Notes on the Compatibility mode of topicMessagesDistributionMode."](#)

For a detailed descriptions and diagrams of MDB generated subscriptions, subscription IDs, and free pool locations, refer to [Appendix A, "Topic Deployment Scenarios,"](#) and [Appendix B, "Topic Subscription Identifiers."](#)

10.2.8.2 Setting `distributedDestinationConnection`

To optionally fine tune the behavior of the `One-Copy-Per-Application` and `One-Copy-Per-Server` modes of `topicMessagesDistributionMode` for a local distributed topic, you can use the `distributedDestinationConnection` activation config property. Alternatively, you can use the `distributed-destination-connection` element in the `weblogic-ejb-jar.xml` deployment descriptor. The valid values are `LocalOnly` and `EveryMember`.

The `distributedDestinationConnection` setting specifies whether a WebLogic Server MDB container sets up a local MDB free pool for each subscription in the entire cluster (`EveryMember`), or local free pools only for subscriptions on members local to the current WebLogic Server (`LocalOnly` - the default).

The use of `distributedDestinationConnection` is restricted as follows: if it is specified for an MDB that subscribes to a remote cluster, a warning message is given and the option is ignored. If you try to use it in Compatibility mode, a warning is given and the option is ignored.

One reason to use `EveryMember` is that the `LocalOnly` option for durable MDBs has restrictions for local RDTs in the `One-Copy-Per-Server` mode. See [Section 10.3.2, "Warning About Using Local RDTs with Durable MDBs."](#)

Another reason to use `EveryMember` is to better handle uneven message loads or message processing delays. See [Section 10.3.7, "Handling Uneven Message Loads and/or Message Processing Delays,"](#) for advice.

Notes on the Compatibility mode of `topicMessagesDistributionMode`

- See [Section 10.2.8.1, "Setting `topicMessagesDistributionMode`,"](#) above, for a statement about supported topic types and versions.
- If you're using the `Compatibility` `topicMessagesDistributionMode` in combination with non-transactional MDBs, and the topic is a foreign (non-WebLogic) destination, or the topic is a WebLogic destination with Unit-of-Order (UOO) messages, then see [Section 10.2.6, "Controlling MDB Concurrency,"](#) for warnings.
- Set the `generate-unique-client-id` attribute to change behavior:
 - If `generate-unique-client-id` is set to `true`, each durable MDB free pool generates a unique subscriber ID. Each MDB free pool will then receive a copy of each published message. For more information see [Appendix B, "Topic Subscription Identifiers."](#) For more information about free pools, see [Section 2.2, "MDBs and Concurrent Processing,"](#) and [Appendix A, "Topic Deployment Scenarios."](#)
 - If `generate-unique-client-id` is `false` (the default), only one subscription will be created by a durable MDB, and only one MDB free pool will successfully connect to the durable subscription (the remaining MDB pools will fail to connect, log warnings, and will keep retrying).
- For durable subscription MDBs that subscribe to the logical name of a local replicated distributed topic (a local RDT), only the configuration described in [Section 10.3.2, "Warning About Using Local RDTs with Durable MDBs,"](#) below, is supported.
- For durable subscription MDBs that subscribe to the logical name of a remote replicated topic (a remote RDT):
 - A deployment of a durable MDB that subscribes to the logical JNDI name of the RDT may succeed, but the MDB deployment will fail to connect, with `Warnings` logs.
- For durable subscription MDBs that subscribe to a particular member destination of a remote replicated topic:
 - A deployment of a durable MDB that subscribes directly to a member of the RDT will succeed, and the subsequent behavior will be determined by the `generate-unique-client-id` setting, as described above. For a uniform

distributed destination, the JNDI name of a particular member is
"jms-server-name@udd-jndi-name".

- For a non-durable subscription MDB that subscribes to the logical name of a local replicated distributed topic (a local RDT), the logical name of a remote replicated distributed topic (a remote RDT), a foreign topic, or a singleton topic, each server will receive a copy of each message that is sent to the topic.
- The `distributedDestinationConnection` option does not apply to Compatibility mode. When set, a warning is given and it is ignored.

10.3 Best Practices

Consider the information in the following sections to help you configure MDBs.

10.3.1 Warning about Non-Transactional MDBs in Compatibility Mode

If you're using the `Compatibility` mode of `topicMessagesDistributionMode` in combination with non-transactional MDBs and the topic is a foreign (non-WebLogic) destination or the topic is a WebLogic destination with Unit-of-Order (UOO) messages, see [Section 10.2.6, "Controlling MDB Concurrency,"](#) above, for warnings.

10.3.2 Warning About Using Local RDTs with Durable MDBs

In `Compatibility` mode, for durable subscription MDBs that subscribe to the logical name of a local replicated distributed topic (a local RDT), only the following configuration is supported:

- Always set `generate-unique-client-id` to `true`.
- Ensure each WebLogic Server in the cluster hosts exactly one member of the RDT.
- Do not use WebLogic JMS `service-migration`. It is unsupported for this use case; but you can use "whole server migration."
- Note that each server receives a copy of each message sent to the topic. When a message arrives at one of the RDT physical topic members, the RDT automatically ensures that a copy of the message is forwarded to each of the other members of the topic.

Similarly, in `One-Copy-Per-Application` mode when `distributedDestinationConnection` is set to `LocalOnly`, for durable subscription MDBs that subscribe to the logical name of a local replicated distributed topic (a local RDT), only the following configuration is supported:

- If your configuration does not match the above recommendations, you may get nondeterministic behavior, including lost messages, duplicate messages, and stuck messages. For more information, including alternatives, see [Section 10.2.8.1, "Setting `topicMessagesDistributionMode`,"](#) above.

10.3.3 Warning about Changing Durable MDB Attributes, Topic Type, EJB Name

Changing MDB or JMS settings can cause the current messages on durable subscriptions to be deleted, or can cause existing durable subscriptions to be abandoned, deleted, or replaced in favor of new durable subscriptions. These settings include topic type, JMS selector, distribution tuning, subscription durability, `ejb-name`, and `client-id`.

Abandoned durable subscriptions continue to accumulate messages even though no MDB is processing the messages. This can eventually lead to quota exceptions or even JVM out-of-memory errors that prevent additional messages from being published to the topic.

For a discussion about locating and removing abandoned subscriptions see "Managing Durable Subscriptions" in *Developing JMS Applications for Oracle WebLogic Server*. For a discussion about subscription IDs and locations, see [Appendix B, "Topic Subscription Identifiers."](#)

10.3.4 Choosing Between Partitioned and Replicated Topics

[Section 10.1, "Supported Topic Types,"](#) above, describes the two types of WebLogic distributed topics (partitioned and replicated). In general, Oracle recommends using partitioned topics (PDTs), when available, except for these two cases:

- When replicated topic (RDT) behavior is required to interoperate with legacy applications or non-MDB applications.
- In the local RDT case in the `One-Copy-Per-Server LocalOnly` case under certain message loads. The message load determines whether the heavy forwarding overhead built into an RDT is less expensive in comparison to the increased network traffic required for the fully connected topology in the PDT `One-Copy-Per-Server` mode. In general, it is better to use a PDT for non-persistent or "lighter" persistent message loads.

To configure a distributed topic type, you set `Partitioned` or `Replicated` as the value for the WebLogic JMS Distributed Topic configuration attribute `JMS Forwarding Policy`. For more information, see "Configuring Partitioned Distributed Topics" in *Administering JMS Resources for Oracle WebLogic Server*.

10.3.5 Choosing an MDB Topic Messages Distribution Mode

Oracle recommends using the `One-Copy-Per-Application` and `One-Copy-Per-Server` modes for most new applications, except for those that must consume from WebLogic JMS topics in WebLogic Server releases prior to 10.3.4 or from foreign (non-WebLogic) topics. These two modes only work with WebLogic JMS topics in WebLogic Server 10.3.4 or later.

10.3.6 Managing and Viewing Subscriptions:

See [Appendix A, "Topic Deployment Scenarios,"](#) and [Appendix B, "Topic Subscription Identifiers,"](#) for detailed discussions of the names and location of subscriptions.

See also "Managing Durable Subscriptions" in *Developing JMS Applications for Oracle WebLogic Server*.

10.3.7 Handling Uneven Message Loads and/or Message Processing Delays

For applications with uneven message loads or unanticipated message processing delays, you may want to consider the following:

- For local distributed topics when the topic distribution mode is `One-Copy-Per-Server` or `One-Copy-Per-Application`, tune `distributedDestinationConnection` to `EveryMember`. While the `LocalOnly` option can yield significantly better performance since it avoids unnecessary network traffic, there are use cases where the `LocalOnly` optimization network savings does not outweigh the benefit of distributing

message processing for unbalanced queue loads as evenly as possible across all JVMs in a cluster. This is especially a concern when message backlogs develop unevenly throughout the cluster and message processing is expensive. In these use cases, the `LocalOnly` configuration should be avoided in favor of the `EveryMember` scenario with durable subscribers.

- Use a PDT instead of an RDT, and tune producer load balancing in the producer's connection factory configuration so that each producer's messages are evenly processed on a round-robin basis throughout the cluster. Incoming messages can be load balanced among the distributed topic members using the WebLogic JMS connection factory `Server Affinity Enabled` and `Load Balancing Enabled` attributes. Disabling affinity can increase network overhead but helps ensure that messages are evenly load balanced across a cluster. The affinity setting has no effect with RDTs. See "Load Balancing Messages Across a Distributed Destination" in *Administering JMS Resources for Oracle WebLogic Server*.
- Decrease the WebLogic JMS asynchronous message pipeline size to 1 to prevent additional messages from being pushed to an MDB thread that is already blocked processing a previous message. The default for this setting is 10, and it is configured by (a) configuring a custom WebLogic connection factory with the `Messages Maximum` attributed tuned to 1 and `XA Enabled` set to true, (b) targeting the connection factory to the same cluster that hosts the distributed topic, and (c) modifying the MDB so that it references the custom connection factory.

10.4 Configuring for Service Migration

For durable subscriptions, JMS service migration (auto or manual) is not supported once `LocalOnly` is applied on local replicated topics. Normally `LocalOnly` means the MDB deployment instance is pinned on the local distributed topic member once the distributed topic member is migrated to another server. The MDB deployment instance cannot subscribe to the same original distributed member after a restart, which may cause warning messages to be generated. Therefore, to use JMS service migration, you should configure as `EveryMember`. Whole server migration is supported for both cases.

10.5 Upgrading Applications from Previous Releases

As described throughout this chapter, new JMS features in WebLogic Server 10.3.4, such as relaxed client ID, sharable subscriptions, and partitioned durable topics, make it possible to implement and deploy MDBs that provide enhanced scalability and high availability. To take advantage of these features, you must upgrade MDB applications written for releases of WebLogic Server prior to 10.3.4.

Applications written to run on releases of WebLogic Server prior to 10.3.4 will continue to run without modification in `Compatibility` mode, which is the default setting for `topicMessagesDistributionMode`, as described in [Section 10.2.8.1, "Setting topicMessagesDistributionMode."](#)

To upgrade applications from previous releases,

1. Consider changing to a partitioned distributed topic. See [Section 10.3.4, "Choosing Between Partitioned and Replicated Topics,"](#) above.
2. Set the `topicMessagesDistributionMode` to `One-Copy-Per-Server` or `One-Copy-Per-Application` and tune the `distributedDestinationConnection` options. See [Section 10.2.8, "Setting Message Distribution Tuning,"](#) above.

Caution: Current messages are not preserved when changing out of Compatibility mode. See [Section 10.3.3, "Warning about Changing Durable MDB Attributes, Topic Type, EJB Name."](#)

10.6 Topic MDB Sample

[Example 10–1](#) shows a WebLogic MDB that uses a durable subscription to a JMS topic (in WebLogic Server 10.3.4 or later), transactionally processes the messages, and forwards the messages to a target destination.

The MDB connects using JMS connection factory `MyCF` to receive from topic `MyTopic`. It forwards the messages to `MyTargetDest` using a connection generated from connection factory `MyTargetCF`.

Resource reference pooling note: The MDB uses a resource reference to access `MyTargetCF`. The resource reference automatically enables JMS producer pooling, as described in "Enhanced Support for Using WebLogic JMS with EJBs and Servlets" in *Developing JMS Applications for Oracle WebLogic Server*.

For a similar sample using queues instead of topics, see [Example 7–3, "Sample MDB Using Distributed Queues"](#).

Example 10–1 Sample MDB Using Distributed Topics

```
package test;
import javax.annotation.Resources;
import javax.annotation.Resource;
import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.ejb.MessageDrivenContext;
import javax.ejb.TransactionAttribute;
import javax.ejb.TransactionAttributeType;
import javax.jms.*;

@MessageDriven(
    name = "MyMDB",
    activationConfig = {
        @ActivationConfigProperty(propertyName = "destinationType",
            propertyValue = "javax.jms.Topic"),

        @ActivationConfigProperty(propertyName = "subscriptionDurability",
            propertyValue = "Durable"),

        @ActivationConfigProperty(propertyName = "connectionFactoryJndiName",
            propertyValue = "MyCF"), // External JNDI Name

        @ActivationConfigProperty(propertyName = "destinationJndiName",
            propertyValue = "MyTopic"), // Ext. JNDI Name

        @ActivationConfigProperty(propertyName = "topicMessagesDistributionMode",
            propertyValue = "One-Copy-Per-Application")
    }
)

@Resources ({
    @Resource(name="targetCFRef",
        mappedName="MyTargetCF", // External JNDI name
        type=javax.jms.ConnectionFactory.class),
```

```
@Resource(name="targetDestRef",
           mappedName="MyTargetDest", // External JNDI name
           type=javax.jms.Destination.class)
})

public class MyMDB implements MessageListener {

    // inject a reference to the MDB context

    @Resource
    private MessageDrivenContext mdctx;

    // cache targetCF and targetDest for re-use (performance)

    private ConnectionFactory targetCF;
    private Destination targetDest;

    @TransactionAttribute(value = TransactionAttributeType.REQUIRED)
    public void onMessage(Message message) {

        System.out.println("My MDB got message: " + message);

        // Forward the message to "MyTargetDest" using "MyTargetCF"

        Connection jmsConnection = null;

        try {
            if (targetCF == null)
                targetCF = (javax.jms.ConnectionFactory)mdctx.lookup("targetCFRef");

            if (targetDest == null)
                targetDest = (javax.jms.Destination)mdctx.lookup("targetDestRef");

            jmsConnection = targetCF.createConnection();
            Session s = jmsConnection.createSession(false, Session.AUTO_ACKNOWLEDGE);
            MessageProducer mp = s.createProducer(null);

            mp.send(targetDest, message);

        } catch (JMSEException e) {

            System.out.println("Forcing rollback due to exception " + e);
            e.printStackTrace();
            mdctx.setRollbackOnly();

        } finally {

            // Closing a connection automatically returns the connection and
            // its session plus producer to the resource reference pool.

            try { if (jmsConnection != null) jmsConnection.close(); }
            catch (JMSEException ignored) {};

        }

        // emulate 1 second of "think" time

        try { Thread.currentThread().sleep(1000); }
        catch (InterruptedException ie) {
            Thread.currentThread().interrupt(); // Restore the interrupted status
        }
    }
}
```



```
}  
}
```

Deployment Elements and Annotations for MDBs

This chapter shows deployment elements and configuration properties that affect the behavior of MDBs. Each row in [Table 11–1](#) describes a deployment element (used in an EJB deployment descriptor) and its associated configuration property (specified in annotations). Not all elements have associated properties.

For information about using deployment descriptors vs. using annotations in MDBs, see [Section 7.2, "Programming EJB 3.1 Compliant MDBs."](#)

Note: For those elements that have an associated configuration property, Oracle recommends that you use that property instead of the element.

[Table 11–1](#) is organized as follows:

- Each element in the **Element** column is followed in parentheses by the name of the deployment descriptor in which the element is used. Elements in the `weblogic-ejb-jar.xml` descriptor link to a more complete explanation of the element in "weblogic-ejb-jar.xml Deployment Descriptor Reference" in *Developing Enterprise JavaBeans, Version 2.1, for Oracle WebLogic Server*. For elements in `ejb-jar.xml`, see the schema at http://java.sun.com/xml/ns/javaee/ejb-jar_3_0.xsd.
- Unless otherwise noted, all the properties listed in the **Configuration Property** column are activation configuration properties, that is, properties defined using `@ActivationConfigProperty` annotations or using an `activation-config-property` element in the message-driven stanza of an `ejb-jar.xml` descriptor. For more information about using `@ActivationConfigProperty`, see [Section 7.2, "Programming EJB 3.1 Compliant MDBs."](#)

Note: Based on the Enterprise JavaBean specification, the `javax.ejb.ActivationConfigProperty` annotation is used for MDBs only. This annotation is not used for session or entity beans.

- The **Configuration Property** column also lists annotations and properties that are not `@ActivationConfigProperty` properties. In those cases, the property is followed by the name of the annotation and by the `import` statement required for using that annotation.

Table 11-1 Deployment Elements and Annotations for MDBs

Element	Configuration Property	Description	Allowable Values	Default
acknowledge-mode (ejb-jar.xml)	acknowledgeMode	Notifies the JMS provider that the message was received and processed. The acknowledgement mode is ignored if using container-managed transactions. (The acknowledgement is performed in the context of the transaction.)	<ul style="list-style-type: none"> ■ AUTO_ ■ ACKNOWLEDG E - the message is acknowledged immediately ■ DUPES_OK_ ■ ACKNOWLEDG E - the acknowledgement may be delayed, allowing duplicate messages to be received 	AUTO_
connection-factory-jndi-name (weblogic-ejb-jar.xml)	connectionFactoryJndiName	The JNDI name of the JMS ConnectionFactory that the MDB looks up to create its queues and topics. See Section 6.3.5, "How to Set connection-factory-jndi-name."	Valid JNDI name	weblogic.jms.MessageDrivenBeanConnectionFactory
connection-factory-resource-link (weblogic-ejb-jar.xml)	connectionFactoryResourceLink	Maps to a resource within a JMS module defined in <code>ejb-jar.xml</code> to an actual JMS Module Reference in WebLogic Server. Rarely used.	Valid resource within a JMS module	n/a
destination-jndi-name (weblogic-ejb-jar.xml)	destinationJndiName	The JNDI name used to associate an MDB with an actual JMS queue or topic deployed in the WebLogic Server JNDI tree. See Section 6.3.4, "How to Set destination-jndi-name."	Valid JNDI name	n/a
destination-resource-link (weblogic-ejb-jar.xml)	destinationResourceLink	Maps to a resource within a JMS module defined in <code>ejb-jar.xml</code> to an actual JMS Module Reference in WebLogic Server. Rarely used.	Valid resource within a JMS module	n/a
dispatch-policy (weblogic-ejb-jar.xml)	n/a	This optional element allows you to specify a particular WorkManager for the bean. See "Tuning Message-Driven Beans" in <i>Tuning Performance of Oracle WebLogic Server</i> .	Valid execute queue name	n/a
distributed-destination-connection (weblogic-ejb-jar.xml)	distributedDestinationConnection	Specifies whether an MDB that accesses a WebLogic JMS distributed destination (topic or queue) in the same cluster consumes from all distributed destination members or only those members local to the current WebLogic Server instance. May not apply to all use cases. See Section 4.4, "JMS Distributed Destinations," and Chapter 10, "Configuring and Deploying MDBs Using JMS Topics."	<ul style="list-style-type: none"> ■ LocalOnly ■ EveryMember 	LocalOnly
durable-subscription-deletion (weblogic-ejb-jar.xml)	durableSubscriptionDeletion	Indicates whether you want durable topic subscriptions to be automatically deleted when an MDB is undeployed or removed.	<ul style="list-style-type: none"> ■ True ■ False 	False

Table 11–1 (Cont.) Deployment Elements and Annotations for MDBs

Element	Configuration Property	Description	Allowable Values	Default
generate-unique-jms-client-id (weblogic-ejb-jar.xml)	See the generateUniqueClientID attribute of the weblogic.javaee.JMSClientID annotation.	Indicates whether or not you want the EJB container to generate a unique client-id for every instance of an MDB. This setting should be used only when topicMessagesDistributionMode is set to Compatibility (the default). See Chapter 10, "Configuring and Deploying MDBs Using JMS Topics."	<ul style="list-style-type: none"> ■ True ■ False 	False
initial-beans-in-free-pool (weblogic-ejb-jar.xml)	n/a	Sets the initial size of the free pool. WebLogic Server populates the free pool with the specified number of bean instances for every bean class at startup. Populating the free pool in this way improves initial response time for the MDB, because initial requests for the bean can be satisfied without generating a new instance.	0 to maxBeans	0
initial-context-factory (weblogic-ejb-jar.xml)	initialContextFactory	The initial context factory that the EJB container uses to create its connection factories. See Section 6.3.3, "How to Set initial-context-factory."	Valid name of an initial context factory	weblogic.jndi.WLInitialContextFactory
init-suspend-seconds (weblogic-ejb-jar.xml)	initSuspendSeconds	The initial number of seconds to suspend an MDB's JMS connection when the EJB container detects a JMS resource outage. See Section 6.6, "Configuring Suspension of Message Delivery During JMS Resource Outages."	Any integer	5
jms-client-id (weblogic-ejb-jar.xml)	jmsClientId	The client ID for the MDB when it connects to a JMS destination. Optional. Used for durable subscriptions to JMS topics. For more information, see Appendix B, "Topic Subscription Identifiers."	n/a	Depends on the topicMessageDistributionMode activation config property and possibly on generate-unique-client-id. See Appendix B, "Topic Subscription Identifiers")
jms-polling-interval-seconds (weblogic-ejb-jar.xml)	jmsPollingIntervalSeconds	The number of seconds between attempts by the EJB container to reconnect to a JMS destination that has become unavailable. See Section 8, "Migration and Recovery for Clustered MDBs."	Any integer	10 seconds
max-beans-in-free-pool (weblogic-ejb-jar.xml)	n/a	The maximum number of bean instances in an MDB free pool. The actual number of instances is also limited by thread pool size as well as other factors. See "Tuning Message-Driven Beans" in <i>Tuning Performance of Oracle WebLogic Server</i> .	0 to maxBeans	1000
max-messages-in-transaction (weblogic-ejb-jar.xml)	maxMessagesInTransaction	Specifies the maximum number of messages that can be in a transaction for this MDB.	All positive integers	n/a

Table 11–1 (Cont.) Deployment Elements and Annotations for MDBs

Element	Configuration Property	Description	Allowable Values	Default
max-suspend-seconds (weblogic-ejb-jar.xml)	maxSuspendSeconds	The maximum number of seconds to suspend an MDB's JMS connection when the EJB container detects a JMS resource outage. See Section 6.6, "Configuring Suspension of Message Delivery During JMS Resource Outages."	Any integer	60
message-destination-type (ejb-jar.xml)	destinationType	Specifies the type of the JMS destination—the Java interface expected to be implemented by the destination.	<ul style="list-style-type: none"> ■ javax.jms.Queue ■ javax.jms.Topic 	n/a
messages-maximum (weblogic-ejb-jar.xml)	messagesMaximum	<p>The maximum number of messages that can exist for an asynchronous session, which have not yet been passed to the message listener. A value of -1 indicates that there is no limit on the number of messages. In this case, however, the limit is set to the amount of remaining virtual memory.</p> <p>When the number of messages reaches the specified value, the following occurs:</p> <ul style="list-style-type: none"> ■ For multicast sessions, new messages are discarded according to the specified overrun policy, and a <code>DataOverrunException</code> is thrown. ■ For non-multicast sessions, new messages are flow-controlled, or retained on the server until the application can accommodate the messages. <p>For multicast sessions, when a connection is stopped, messages will continue to be accumulated, but only until the specified maximum value is reached. Once this value is reached, messages will be discarded based on the overrun policy.</p>	-1 and $1-2^{63-1}$	10
message-selector (ejb-jar.xml)	messageSelector	A string used by a client to specify, by header field references and property references, the messages it is interested in. Only messages whose header and property values match the selector are delivered	Conditional expression using message properties, or message header	Null
messaging-type (ejb-jar.xml)	n/a	Rarely used.	javax.jms.MessageListener	n/a
provider-url (weblogic-ejb-jar.xml)	providerURL	The URL provider to be used by the <code>InitialContext</code> . Typically, this is the <code>host:port</code> . See Section 6.3.2, "How to Set provider-url."	Valid URL	Null
resource-adapter-jndi-name (weblogic-ejb-jar.xml)	resourceAdapterJndiName	For JCA-driven MDBs, identifies the resource adapter from which this MDB receives messages.	n/a	n/a
security-role-assignment (weblogic-ejb-jar.xml)	n/a	Maps application roles in the <code>ejb-jar.xml</code> file to the names of security principals available in WebLogic Server.	n/a	n/a

Table 11–1 (Cont.) Deployment Elements and Annotations for MDBs

Element	Configuration Property	Description	Allowable Values	Default
start-mdb-with-application (weblogic-application.xml)	n/a	Controls when MDBs start processing messages. When set to <code>true</code> , 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. Set to <code>false</code> to defer message processing until after WebLogic Server opens its listen port.	<ul style="list-style-type: none"> ■ True ■ False 	False
subscription-durability (ejb-jar.xml)	subscriptionDurability	Specifies whether a JMS topic subscription is <code>Durable</code> or <code>NonDurable</code> . For more information, see Section 10.2.2, "Setting Subscription Durability."	<ul style="list-style-type: none"> ■ Durable ■ NonDurable 	NonDurable
n/a	topicMessagesDistributionMode	Sets the distribution mode for topic messages. See Chapter 10, "Configuring and Deploying MDBs Using JMS Topics."	<ul style="list-style-type: none"> ■ One-Copy-Per-Application ■ One-Copy-Per-Server ■ Compatibility 	Compatibility
transaction-type (ejb-jar.xml)	See trans-attribute	Specifies an enterprise bean's transaction management type. For more information, see Section 6.2, "Configuring Transaction Management Strategy for an MDB."	<ul style="list-style-type: none"> ■ Bean ■ Container 	Container
trans-attribute (ejb-jar.xml)	TransactionAttribute property of <code>@TransactionAttribute</code> , for example: <pre>import javax.ejb.TransactionAttribute @TransactionAttribute(TransactionAttributeType.REQUIRED)</pre>	Specifies how the container must manage the transaction boundaries when delegating a method invocation to an enterprise bean's business method. Note: If the bean is specified as using container-managed transaction demarcation, either the <code>REQUIRED</code> or the <code>NOT_SUPPORTED</code> transaction attribute must be used for the message listener methods, and either the <code>REQUIRED</code> , <code>REQUIRES_NEW</code> , or the <code>NOT_SUPPORTED</code> transaction attribute for timeout callback methods. For more information, see Section 6.2, "Configuring Transaction Management Strategy for an MDB."	<ul style="list-style-type: none"> ■ Required ■ NotSupported ■ Supports ■ RequiresNew ■ Mandatory ■ Never 	Required
trans-timeout-seconds (weblogic-ejb-jar.xml)	<code>@TransactionTimeoutSeconds</code> <pre>import weblogic.javaee.TransactionTimeoutSeconds</pre>	The maximum duration for an EJB's container-initiated transactions, in seconds, after which the transaction is rolled back. See Section 6.2, "Configuring Transaction Management Strategy for an MDB."	0 to max	If the transaction timeout is not specified or is set to 0, the transaction timeout configured for the domain is used. If a timeout is not configured for the domain, the default is 30.
use81-style-polling (weblogic-ejb-jar.xml)	use81StylePolling	Enables backwards compatibility for WebLogic Server version 8.1-style polling.	<ul style="list-style-type: none"> ■ True ■ False 	False

Topic Deployment Scenarios

The following sections describe MDB deployment actions for various topic MDB configurations. The actions include where and how many MDB free pools are created, where and how many subscriptions are created, and how the subscribers work together to achieve a given messaging consumption pattern.

These sections do not cover details about legacy behavior which occurs when the `topicMessagesDistributionMode` is set to `Compatibility`, when the topics are foreign (non-WebLogic) topics, or when the topics are WebLogic JMS topics from WebLogic Server releases prior to 10.3.4.

For help determining the right scenario (permutation) for your application, including suggested settings, see [Chapter 10, "Configuring and Deploying MDBs Using JMS Topics."](#)

A.1 How Configuration Permutations Determine Deployment Actions

The following settings determine how WebLogic MDBs that consume from WebLogic JMS topics (from WebLogic Server 10.3.4 or later) create instances of MDB free pools, subscription naming, subscription locations, and how messages are distributed to those MDB pool instances:

- The topic location (in the same cluster or server as the MDB deployment or on a remote cluster or server).
- The topic type (singleton WebLogic topic, `Replicated` or `Partitioned` distributed topic).
- The `subscriptionDurability` setting.
- The `topicMessagesDistributionMode` and `distributedDestinationConnection` settings.

[Table A-1](#) describes possible configuration permutations and corresponding deployment actions. The first two columns describe the configuration permutations, and the last two columns describe the resulting deployment. The columns are as follows:

- **topicMessagesDistributionMode** -- The value of the `topicMessagesDistribution` configuration option, that is, `One-Copy-Per-Server` or `One-Copy-Per-Application`. The legacy `Compatibility` mode is not covered in this table.
- **Topic Type Permutation** -- Options include the following:
 - **Local** or **Remote** -- Whether the topic is deployed to the same cluster or server as the MDB (Local) or to a different cluster or server (Remote).

- **PDT, RDT, or Singleton WebLogic JMS topic** -- The type of topic: partitioned distributed topic (PDT), replicated distributed topic (RDT), or singleton WebLogic JMS topic.
- **EveryMember or LocalOnly** -- The value of `distributedDestinationConnection`. Specifies whether the MDB that accesses a `Local` distributed topic in the same cluster consumes from all distributed topic members or only from those local to the current server. If neither `EveryMember` nor `LocalOnly` is specified, the permutation applies regardless of how `distributedDestinationConnection` is set.

For example, the topic type permutation "Local RDT LocalOnly" means "An MDB is deployed to the same cluster (Local) as the replicated topic (RDT), and the MDB is configured to consume only from members of the topic on the same WebLogic Server as the MDB (LocalOnly)."

- **Each Server Subscribes to...** -- The number of MDB pools a WebLogic Server instance creates, and the members of the distributed topic to which the MDB instances subscribe. For example,
 - "Each server subscribes to ... All members" means "the container creates one local MDB pool for each member of the distributed topic."
 - "Each server subscribes to ... All local members" means "the container creates one MDB pool for each of the members that are running on the same server, and each MDB pool subscribes to one of those members."
- **MDB Pools Per Server** -- The number of MDB deployment instances on each server in the cluster (and thereby the number of connections to the distributed topic members). M = the number of distributed topic members ($M=1$ for standalone topics).

Table A-1 Configuration Permutations and Their Resulting Deployment Actions

MDB Configuration		Deployment Actions	
<code>topicMessagesDistributionMode</code>	Topic Type Permutation	Each Server Subscribes to...	MDB Pools per Server
<code>One-Copy-Per-Server</code>	<ul style="list-style-type: none"> ■ RDT Local LocalOnly¹ 	One of the local members	One
<code>One-Copy-Per-Server</code>	<ul style="list-style-type: none"> ■ RDT Remote (Non-durable only)² 	One of the remote members	One

Table A-1 (Cont.) Configuration Permutations and Their Resulting Deployment Actions

MDB Configuration		Deployment Actions	
topicMessagesDistributionMode	Topic Type Permutation	Each Server Subscribes to...	MDB Pools per Server
One-Copy-Per-Server	<ul style="list-style-type: none"> ▪ PDT Local EveryMember³ ▪ PDT Remote² ▪ RDT Local EveryMember ▪ RDT Remote (durable subscriptions only)² ▪ Singleton WebLogic JMS (M=1) 	All members	M
One-Copy-Per-Application	<ul style="list-style-type: none"> ▪ PDT Local LocalOnly ▪ RDT Local LocalOnly 	All local members	One per local member
One-Copy-Per-Application	<ul style="list-style-type: none"> ▪ PDT Local EveryMember ▪ PDT Remote² ▪ RDT Local EveryMember ▪ RDT Remote² ▪ Singleton WebLogic JMS (M=1) 	All members	M

¹ The "One-Copy-Per-Server, RDT, Local, LocalOnly" permutation *is not supported* for durable subscription cases in some configuration topologies (See details in [Section A.2.2.1, "Scenario 1: Replicated DT, One Copy Per Server, Local Deployment, Local Only Consumption."](#))

² For remote distributed topics, WebLogic Server always creates subscriptions to every topic member except for non-durable subscriptions in the "One-Copy-Per-Server, Replicated Distributed Topic, Remote" permutation. In that case, only one subscription to one of the remote members is created. (See [Section A.2.2.3, "Scenario 3: Replicated DT, One Copy Per Server, Remote Deployment."](#))

³ The LocalOnly setting is always automatically replaced with EveryMember in the "One-Copy-Per-Server, Partitioned Distributed Topic, Local" permutation. (See [Section A.2.3.1, "Scenario 7: Partitioned DT, One Copy Per Server, Local Deployment, Local Only Consumption."](#))

A.2 Typical Scenarios

The following sections show possible deployment scenarios of an MDB application:

- [Standalone \(Non-distributed\) Topic Scenarios](#)
- [Replicated Distributed Topic Scenarios](#)
- [Partitioned Distributed Topic Scenarios](#)

Images and labels used in the figures presented in the scenarios are explained in [Table A-2](#):

Table A-2 Explanation of Images and Text Used in Scenarios


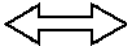


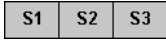
Image or Text	Explanation
	Messages published to a distributed topic.
	Messages are duplicated, and copies are forwarded to other members of the topic. This indicates that the topics are replicated distributed topics.
DT Member <i>n</i>	Member of a distributed topic.

Table A-2 (Cont.) Explanation of Images and Text Used in Scenarios

Image or Text	Explanation
MDB Pool	An MDB free bean pool.
	A subscription. The MDB on one end of the arrow listens for and consumes messages from the topic on the other end of the arrow.
	Shared subscription.
	Non-shared subscription. S1 is Managed Server 1's subscription, S2 is Managed Server 2's subscription, etc.

A.2.1 Standalone (Non-distributed) Topic Scenarios

Standalone topic scenarios are as follows.

A.2.1.1 One-Copy-Per-Server

On each WebLogic Server instance that hosts the MDB application, an MDB pool is created for the topic, whether the topic is running in the same cluster or in a different cluster. For an MDB cluster of N nodes, N MDB pools are created. Each MDB pool creates an individual subscription on the topic, and subscribers from different MDB pools do not share the same subscription.

A.2.1.2 One-Copy-Per-Application

On each WebLogic Server instance that hosts the MDB application, an MDB pool is created for the topic, whether the topic is running in the same cluster or in a different cluster. For an MDB cluster of N nodes, N MDB pools are created. All subscribers created by the MDB pools of the same MDB application share the same subscription.

A.2.2 Replicated Distributed Topic Scenarios

With replicated distributed topics, all physical topic members receive each message sent. When a message arrives at one of the physical topic members, a copy of the message is automatically internally forwarded to the other members of the topic.

The following are the possible deployment scenarios for a replicated distributed topic:

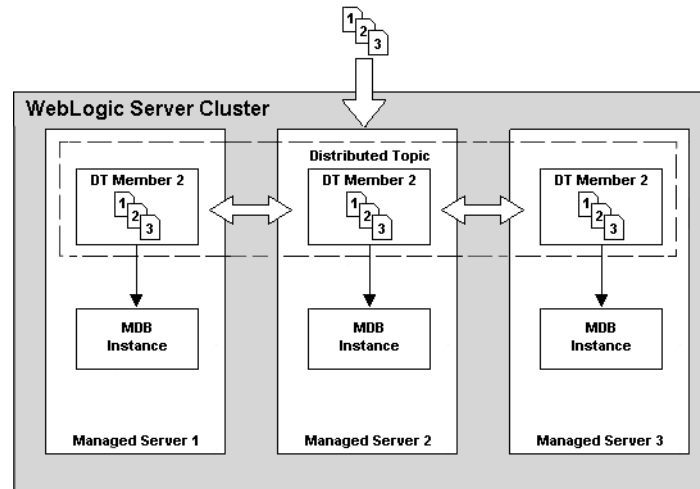
- Scenario 1: Replicated DT, One Copy Per Server, Local Deployment, Local Only Consumption
- Scenario 2: Replicated DT, One Copy Per Server, Local Deployment, Every Member Consumption,
- Scenario 3: Replicated DT, One Copy Per Server, Remote Deployment
- Scenario 4: Replicated DT, One Copy Per Application, Local Deployment, Local Only Consumption
- Scenario 5: Replicated DT, One Copy Per Application, Local Deployment, Every Member Consumption
- Scenario 6: Replicated DT One Copy Per Application, Remote Deployment

A.2.2.1 Scenario 1: Replicated DT, One Copy Per Server, Local Deployment, Local Only Consumption

Figure A-1 shows the following configuration:

- Replicated distributed topic
- `topicMessagesDistributionMode = One-Copy-Per-Server`.
- The MDB and the topic are deployed in the same (local) cluster.
- `distributedDestinationConnection = LocalOnly`.

Figure A–1 Scenario 1: Replicated DT, One Copy Per Server, Local Deployment, Local Only Consumption



In this scenario:

- Copies of messages are forwarded to other servers in the cluster by the RDT.
- One MDB pool is created on each server in the local cluster.
- Each MDB pool listens to one of the distributed topic member on the same server.

This approach can yield higher performance than "RDT, One Copy Per Server, Local Deployment, EveryMember," because all messaging is local (it avoids transferring messages over network calls) and still ensures that all distributed topic members are serviced by MDB consumers. However for some use cases, the `EveryMember` alternative may work better, based on the trade-offs discussed in [Section 10.3.7, "Handling Uneven Message Loads and/or Message Processing Delays."](#)

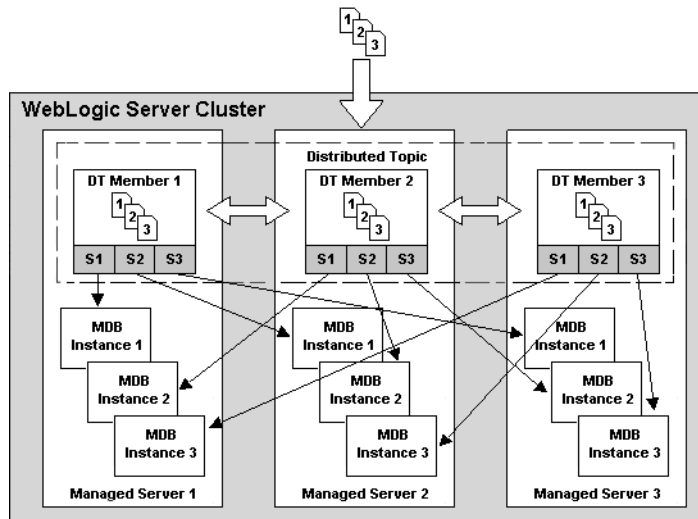
This scenario does not work correctly for durable subscriptions when there are multiple members on the same server, when there are no members on any of the local servers that host the MDB application, or when JMS service migration (auto or manual) is involved.

A.2.2.2 Scenario 2: Replicated DT, One Copy Per Server, Local Deployment, Every Member Consumption,

[Figure A–2](#) shows the following configuration:

- Replicated distributed topic
- `topicMessagesDistributionMode = One-Copy-Per-Server`.
- The MDB and the topic are deployed in the same (local) cluster.
- `distributedDestinationConnection = EveryMember`.

Figure A–2 Scenario 2: Replicated DT, One Copy Per Server, Local Deployment, Every Member Consumption



In this scenario:

- Copies of messages are forwarded to other servers in the cluster by the RDT, but these copies are filtered out (ignored) by the MDB subscriptions.
- An MDB pool is created for each distributed topic member on each server in the local cluster.
- Each MDB pool listens to one of the distributed topic members in the cluster.
- Each WebLogic Server instance that hosts the MDB application listens to all members of the distributed topic.
- Each server's subscribers on the same member of the DT have their own independent subscriptions. In other words, subscribers from different servers to the same member do not share any subscriptions.

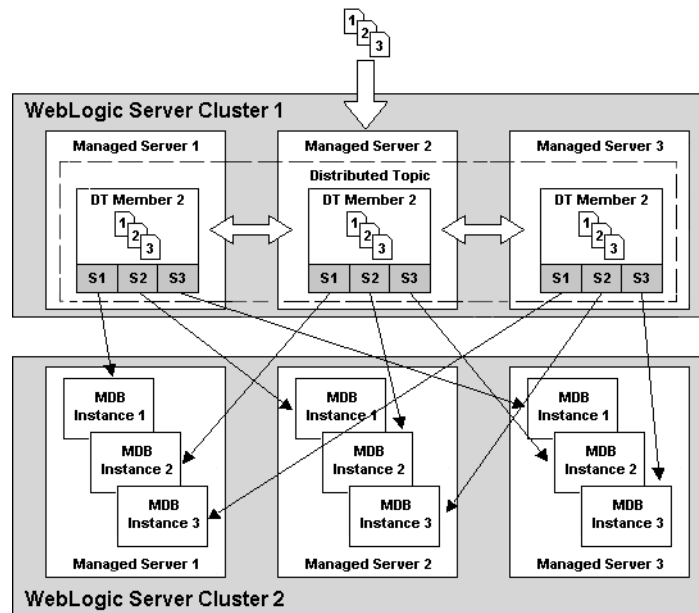
This configuration yields high flexibility and is good for an application where an RDT is required, but it cannot be guaranteed that there will be exactly one member per server, for example due to migration.

This configuration does not give the best performance in comparison to [Scenario 1: Replicated DT, One Copy Per Server, Local Deployment, Local Only Consumption](#), especially for a static environment where no migration is involved and there is one and only one member of the distributed topic on each managed server. Applications where no migration is involved and where there is one and only one member of the distributed topic on each managed server can use Scenario 1.

A.2.2.3 Scenario 3: Replicated DT, One Copy Per Server, Remote Deployment

Figure A–3 shows the following configuration:

- Replicated distributed topic
- Durable subscription
- `topicMessagesDistributionMode = One-Copy-Per-Server`.
- The MDB and the topic are deployed in different (remote) clusters.
- `distributedDestinationConnection` ignored for remote deployments.

Figure A-3 Scenario 3: Replicated DT, One Copy Per Server, Remote Deployment


In this scenario:

- Copies of messages are forwarded to other servers in the cluster by the RDT, but these copies are filtered out (ignored) by the MDB subscriptions.
- An MDB pool for each distributed topic member is created on each server in the remote cluster.
- Each WebLogic Server instance that hosts the MDB application listens to all members of the distributed topic (one local pool for each remote member).
- Each server's subscribers on the same member of the DT have their own independent subscription. In other words, subscribers from different servers to the same member do not share any subscriptions.

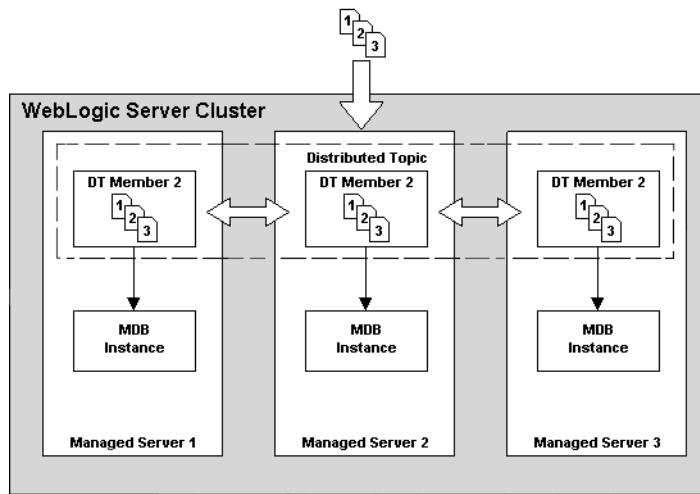
Note that this is the behavior for durable cases. For non-durable cases, each WebLogic Server instance creates a single MDB pool which connects to one of the members (any member) as an optimization.

A.2.2.4 Scenario 4: Replicated DT, One Copy Per Application, Local Deployment, Local Only Consumption

Figure A-4 shows the following configuration:

- Replicated distributed topic
- `topicMessagesDistributionMode = One-Copy-Per-Application.`
- The MDB and the topic are deployed in the same (local) cluster.
- `distributedDestinationConnection = LocalOnly.`

Figure A-4 Scenario 4: Replicated DT, One Copy Per Application, Local Deployment, Local Only Consumption



In this scenario:

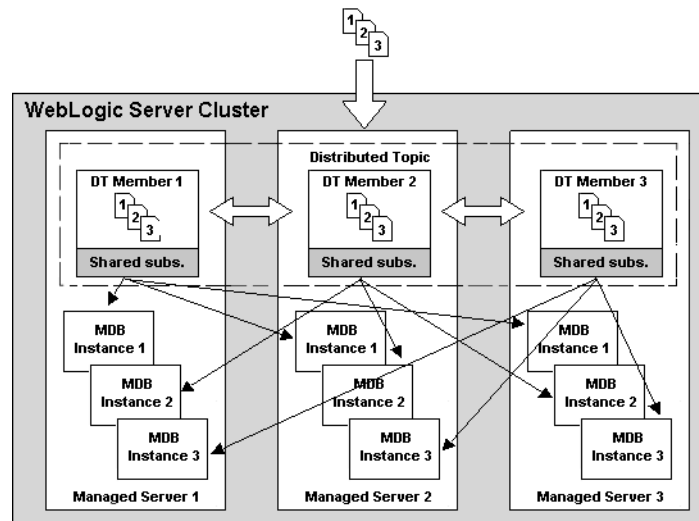
- Copies of messages are forwarded to other servers in the cluster by the RDT, but these copies are filtered out (ignored) by the MDB subscriptions.
- One MDB pool is created on each server in the local cluster for each local member (Figure A-5 shows a configuration where each WebLogic Server instance hosts only one member. When there are multiple members on the same local WebLogic Server instance, multiple MDB pools are created on the server).
- A message is given to only one MDB pool.
- All subscribers on the same member share the same subscription.

A.2.2.5 Scenario 5: Replicated DT, One Copy Per Application, Local Deployment, Every Member Consumption

Figure A-5 shows the following configuration:

- Replicated distributed topic
- `topicMessagesDistributionMode = One-Copy-Per-Application`.
- The MDB and the topic are deployed in the same (local) cluster.
- `distributedDestinationConnection = EveryMember`.

Figure A-5 Scenario 5: Replicated DT, One Copy Per Application, Local Deployment, Every Member Consumption



In this scenario:

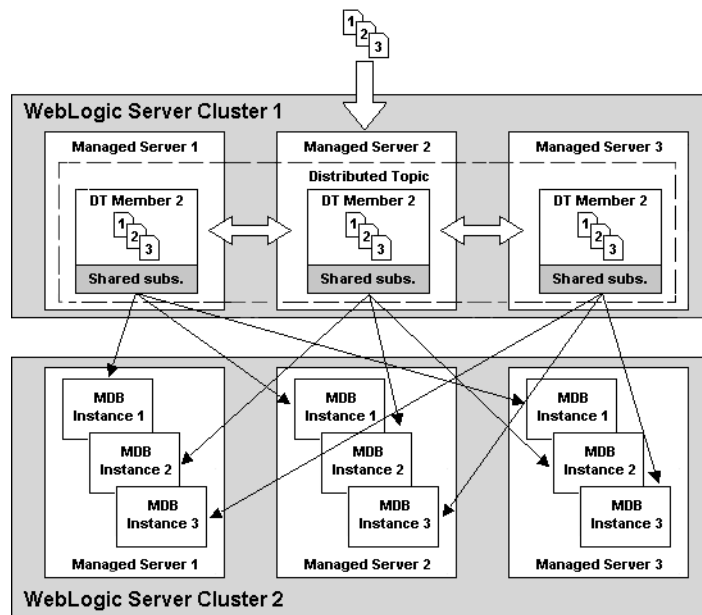
- Copies of messages are forwarded to other servers in the cluster by the RDT, but these copies are filtered out (ignored) by the MDB subscriptions.
- One MDB pool is created on each server in the local cluster for each member.
- A message is given to only one MDB pool.
- All subscribers on the same member share the same subscription.

A.2.2.6 Scenario 6: Replicated DT One Copy Per Application, Remote Deployment

Figure A-6 shows the following configuration:

- Replicated distributed topic
- `topicMessagesDistributionMode = One-Copy-Per-Application.`
- The MDB and the topic are deployed in different (remote) clusters.
- `distributedDestinationConnection` ignored for remote deployments.

Figure A-6 Scenario 6: Replicated DT, One Copy Per Application, Remote Deployment



In this scenario:

- Copies of messages are forwarded to other servers in the cluster by the RDT, but these copies are filtered out (ignored) by the MDB subscriptions.
- One MDB pool is created on each server in the local cluster for each member in the remote cluster.
- A message is given to only one MDB pool.
- All subscribers on the same member share the same subscription.

A.2.3 Partitioned Distributed Topic Scenarios

With partitioned topics:

- The distributed topic member receiving the message is the only member that is aware of the message. The message is not forwarded to other members, and subscribers on other members do not get a copy of the message.
- Incoming messages can be load balanced among the distributed topic members using the JMS *Affinity* and *Load Balance* attributes. See "Load Balancing Partitioned Distributed Topics" in *Administering JMS Resources for Oracle WebLogic Server*.

The following are the possible deployment scenarios for a partitioned distributed topic:

- [Scenario 7: Partitioned DT, One Copy Per Server, Local Deployment, Local Only Consumption](#)
- [Scenario 8: Partitioned DT, One Copy Per Server, Local Deployment, Every Member Consumption](#)
- [Scenario 9: Partitioned DT, One Copy Per Server, Remote Deployment](#)
- [Scenario 9: Partitioned DT, One Copy Per Application, Local Deployment, Local Only Consumption](#)

- Scenario 11: Partitioned DT, One Copy Per Application, Local Deployment, Every Member Consumption
- Scenario 12: Partitioned DT, One Copy Per Application, Remote Deployment

A.2.3.1 Scenario 7: Partitioned DT, One Copy Per Server, Local Deployment, Local Only Consumption

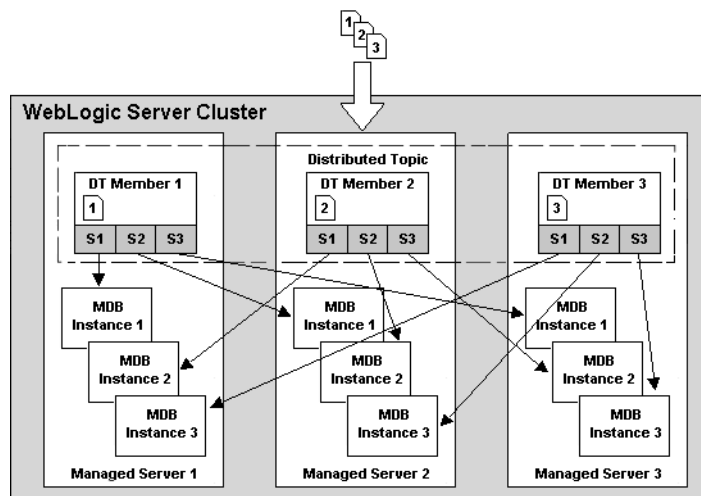
The setting of `distributedDestinationConnection` is ignored for this scenario and a warning message is logged. The setting is forced to `EveryMember` instead. The behavior becomes the same as the "EveryMember" case (see "Scenario 8: Partitioned DT, One Copy Per Server, Local Deployment, Every Member Consumption").

A.2.3.2 Scenario 8: Partitioned DT, One Copy Per Server, Local Deployment, Every Member Consumption

Figure A-7 shows the following configuration:

- Partitioned distributed topic
- `topicMessagesDistributionMode = One-Copy-Per-Server`.
- The MDB and the topic are deployed in the same (local) cluster.
- `distributedDestinationConnection = EveryMember`.

Figure A-7 Scenario 8: Partitioned DT, One Copy Per Server, Local Deployment, Every Member Consumption



In this scenario:

- Messages are distributed individually to the distributed topic members. Messages are not duplicated or copied to other members in the cluster.
- An MDB pool is created for each distributed topic member on each server in the local cluster.
- Each server's subscribers on the same member of the DT have their own independent subscription. In other words, subscribers from a particular server to the same member do not share any subscriptions with subscribers from another server.

A.2.3.3 Scenario 9: Partitioned DT, One Copy Per Server, Remote Deployment

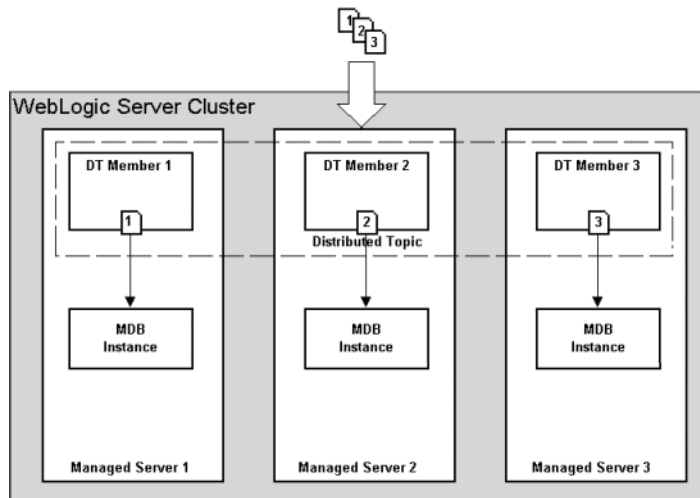
The details of this scenario are the same as the previous one except that the MDB deployment and the PDT are in different clusters.

A.2.3.4 Scenario 9: Partitioned DT, One Copy Per Application, Local Deployment, Local Only Consumption

Figure A-8 shows the following configuration:

- Partitioned distributed topic
- `topicMessagesDistributionMode = One-Copy-Per-Application.`
- The MDB and the topic are deployed in the same (local) cluster.
- `distributedDestinationConnection = LocalOnly.`

Figure A-8 Scenario 9: Partitioned DT, One Copy Per Application, Local Deployment, LocalOnly Consumption



In this scenario:

- Messages are distributed individually to the distributed topic members. Messages are not duplicated or copied to other members in the cluster.
- One MDB pool is created on each server in the local cluster for each local member (Figure A-9 shows a configuration where each WebLogic Server hosts only one member. When there are multiple members on the same local WebLogic Server, multiple MDB pools are created on each WebLogic Server instance).

This scenario is the recommended configuration for `One-Copy-Per-Application` and `Local PT` for high performance. However for some use cases, the `EveryMember` alternative may work better, based on the trade-offs discussed in [Section 10.3.7, "Handling Uneven Message Loads and/or Message Processing Delays."](#)

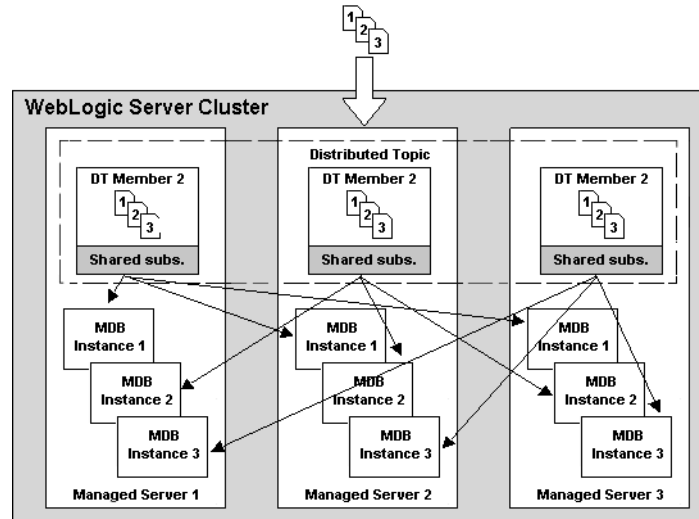
A.2.3.5 Scenario 11: Partitioned DT, One Copy Per Application, Local Deployment, Every Member Consumption

Figure A-10 shows the following configuration:

- Partitioned distributed topic
- `topicMessagesDistributionMode = One-Copy-Per-Application.`

- The MDB and the topic are deployed in the same (local) cluster.
- `distributedDestinationConnection = EveryMember`.

Figure A-9 Scenario 11: Partitioned DT, One Copy Per Application, Local Deployment, Every Member Consumption



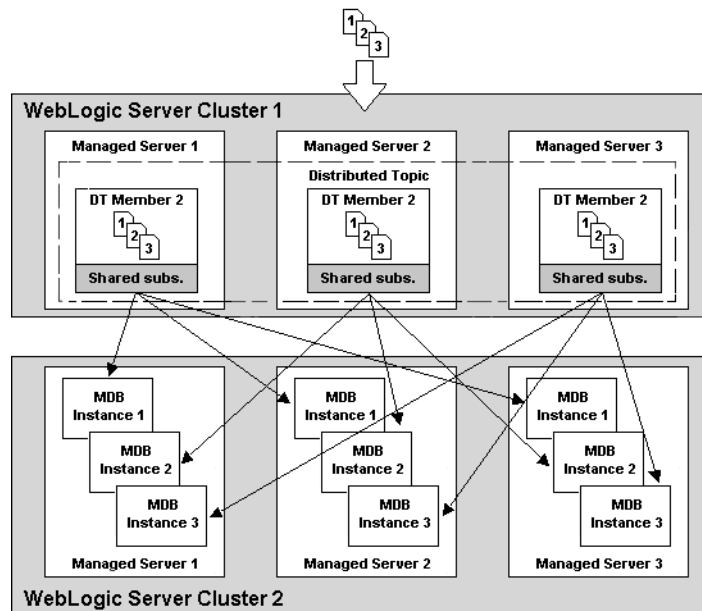
For a partitioned distributed topic, one copy per application, local deployment, it is better to use `LocalOnly` consumption for most use cases, as shown in [Scenario 9: Partitioned DT, One Copy Per Application, Local Deployment, Local Only Consumption](#). However the trade-offs discussed in [Section 10.3.7, "Handling Uneven Message Loads and/or Message Processing Delays,"](#) apply here.

A.2.3.6 Scenario 12: Partitioned DT, One Copy Per Application, Remote Deployment

[Figure A-10](#) shows the following configuration:

- Partitioned distributed topic
- `topicMessagesDistributionMode = One-Copy-Per-Application`.
- The MDB and the topic are deployed in different (remote) cluster.
- `distributedDestinationConnection` ignored for remote deployments.

Figure A-10 Scenario 12: Partitioned DT, One Copy Per Application, Remote Deployment



In this scenario:

- Messages are distributed individually to the distributed topic members. Messages are not duplicated or copied to other members in the cluster.
- Subscriptions are created on all the distributed topic members automatically and dynamically.
- A message is given to only one MDB pool.
- All subscribers from the same MDB application on the same member share the same subscription.

Topic Subscription Identifiers

In JMS, a subscription is identified and located based on (a) the topic with which it is associated, (b) the connection "Client ID" string that is specified for the connection that is used to access the subscription, and (c), if using durable subscriptions, the subscription name that is specified when the durable subscription is created.

Furthermore, in WebLogic JMS, a subscription is also identified by (d) the "Client ID Policy" option. If two WebLogic JMS subscription references on the same physical topic have the same Client ID and subscription name, then the references resolve to a single subscription if the Client ID Policy is also the same, but they resolve to two different subscriptions if the Client ID Policies are different.

A WebLogic MDB container generates a, b, c, and d automatically, based on the following settings:

- `ejb-name`
- `jms-client-id`
- `topicMessagesDistributionMode`
- `distributedDestinationConnection`
- `generate-unique-client-id`
- `subscriptionDurability`
- Other elements of the MDB deployment and JMS configurations

The last four settings, above, apply only to Compatibility mode MDBs.

[Table B-1](#) summarizes how the settings are used to generate subscription IDs:

Table B-1 How Subscription IDs are Generated

Setting	ClientID	Subscription Name for the Durable Subscription Case	Client ID Policy for WebLogic Topics
topicMessagesDistributionMode = One-Copy-Per-Application	jmsClientIDBase	ejb-name	Unrestricted
topicMessagesDistributionMode = One-Copy-Per-Server	jmsClientIDBase + " + currentDomainName + " + currentServerName	ejb-name	Unrestricted

Key:

- `jms-client-id` = an optional MDB attribute string set by the MDB descriptor or an annotation; alternatively (but rarely), the `jms-client-id` can be set by changing the MDB to reference a custom JMS connection factory that in turn has a `client-id` configured
- `ejb-name` = the name of the EJB
- `jmsClientIDBase` = `jms-client-id` (if specified by user) or `ejb-name` (if `jms-client-id` is not specified)
- `currentDomainName` = the name of the WebLogic domain that runs the MDB
- `currentServerName` = the name of the WebLogic Server that the MDB is running on
- `uniqueKey` = a string that contains some of the MDB deployment elements, possibly the `currentServerName`, plus, if the destination is a WebLogic destination that is hosted by a JMS server that is using a migratable target, then it includes this migratable target name.
- `DDMemberName` = the name of a distributed destination member; or, alternatively, the destination name if the topic is a singleton or a distributed destination in releases of WebLogic Server prior to 10.3.4.

Table B-1 (Cont.) How Subscription IDs are Generated

Setting	ClientID	Subscription Name for the Durable Subscription Case	Client ID Policy for WebLogic Topics
topicMessagesDistributionMode = Compatibility generateUniqueClientID = true distributedDestinationOnConnection = LocalOnly subscriptionDurability = Durable ¹	jmsClientIDBase + " + currentDomainName + " + uniqueKey	Same as the ClientId	Restricted
Same as previous row, except: distributedDestinationOnConnection = EveryMember	jmsClientIDBase + " + currentDomainName + " + uniqueKey + " + DDMemberName	Same as the ClientId	Restricted
topicMessagesDistributionMode = Compatibility generateUniqueClientID = false subscriptionDurability = Durable ¹	jmsClientIDBase	Same as the ClientId	Restricted

Key:

- `jms-client-id` = an optional MDB attribute string set by the MDB descriptor or an annotation; alternatively (but rarely), the `jms-client-id` can be set by changing the MDB to reference a custom JMS connection factory that in turn has a `client-id` configured
- `ejb-name` = the name of the EJB
- `jmsClientIDBase` = `jms-client-id` (if specified by user) or `ejb-name` (if `jms-client-id` is not specified)
- `currentDomainName` = the name of the WebLogic domain that runs the MDB
- `currentServerName` = the name of the WebLogic Server that the MDB is running on
- `uniqueKey` = a string that contains some of the MDB deployment elements, possibly the `currentServerName`, plus, if the destination is a WebLogic destination that is hosted by a JMS server that is using a migratable target, then it includes this migratable target name.
- `DDMemberName` = the name of a distributed destination member; or, alternatively, the destination name if the topic is a singleton or a distributed destination in releases of WebLogic Server prior to 10.3.4.

¹ Non-durable `Compatibility` mode MDBs do not set a Client-ID or Subscription-Name, and use the default Restricted Client ID Policy.

Client ID uniqueness is enforced as follows:

- *For foreign (non-WebLogic) JMS vendors:* Some JMS vendors prevent more than one connection from specifying the same connection Client ID. (An exception is thrown on an attempt to create the second connection.) This limitation in turn can prevent more than one free pool from using the same Client ID, because each free

pool creates a single JMS connection with potentially the same Client ID as other free pool connections. After a first free pool 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 attempts to create a JMS connection, a Client ID conflict is detected and that instance of the MDB fails to fully connect to JMS.

- *For WebLogic JMS:* For WebLogic JMS in releases of WebLogic Server prior to 10.3.4, JMS connections were restricted so only one connection with the same Client ID could exist in the scope of a cluster. However, for WebLogic Server 10.3.4 and later, WebLogic JMS connection factories or connections can optionally set a Client ID Policy to control this restriction. With a Client ID Policy of RESTRICTED, the pre-10.3.4 behavior remains in effect, while with a Client ID Policy of UNRESTRICTED, this limitation is lifted. For more information, see *Developing JMS Applications for Oracle WebLogic Server*. Unrestricted client IDs make it possible for multiple WebLogic subscriber connections and subscriptions to share the same client ID. Both `One-Copy-Per-Server` and `One-Copy-Per-Application` Topic Message Distribution Modes set the `ClientIDPolicy` to `Unrestricted`. Note that if two WebLogic JMS subscription references on the same physical topic have the same Client ID and durable subscription name, then the references resolve to a single subscription if the Client ID Policy is also the same, but they resolve to two different subscriptions if the Client ID Policies are different.

How WebLogic MDBs Leverage WebLogic JMS Extensions

The MDB deployment scenarios described in [Appendix A, "Topic Deployment Scenarios."](#) take advantage of the following JMS features:

- **Shared subscriptions** -- Shared subscriptions allow multiple subscribers to share one subscription, even when the subscribers are created from different MDB servers. All subscribers that share the same subscription collectively process all of the messages published to the topic. Each message is processed by only one of the subscribers. For example, if there are two subscribers, S1 and S2, and three messages, M1, M2, and M3, S1 might receive M1 and M2 (but not M3) and, then, S2 would receive M3 (but not M1 and M2).

This enables applications to employ "round-robin" distributed or parallel processing of a single subscription's topic messages. MDBs can create multiple subscribers on the same subscription identifier, whether it is durable or non-durable. For more information about the JMS Subscription Sharing Policy, see "Configure Shared Subscriptions" in *Administering JMS Resources for Oracle WebLogic Server*.

- **Unrestricted Client IDs** -- Unrestricted Client IDs allow multiple concurrently active connections to use the same Client ID. The JMS `clientID` identifies a JMS connection and is used to identify a durable subscription on that connection. Setting the `clientID` to `Unrestricted` allows you to create multiple physical subscriptions, with the same name, on different destinations. This allows subscriptions with the same name to exist on different members of the same distributed topic, and together, these subscriptions can be treated as a single logical subscription. For more information, see "Configure an Unrestricted Client ID" in *Administering JMS Resources for Oracle WebLogic Server*.

The `topicMessagesDistributionMode` defines permutations of the JMS attributes `SubscriptionSharingPolicy` and `ClientIdPolicy` (set on the connection factory), to control how messages are distributed to distributed topics. WebLogic Server sets those values as shown in [Table C-1](#).

Table C-1 Relationships Between `topicMessagesDistributionMode` Settings and Settings on JMS Connection Factory

<code>topicMessagesDistributionMode</code>	<code>SubscriptionSharingPolicy</code>	<code>ClientIdPolicy</code>
One-Copy-Per-Server or One-Copy-Per-Application	Sharable	Unrestricted
Compatibility (replicated distributed topics and foreign topics only)	Exclusive	Restricted

If the settings on the connection factory are not these values, WebLogic Server overrides them and gives a warning message. If WebLogic Server cannot override the values for any reason, it throws an exception, and the MDB cannot process any messages unless the administrator changes the settings on the JMS connection factory. You cannot programmatically set these attributes on the connection factory directly. Instead, use `topicMessagesDistributionMode`, and the MDB deployment will set the values on the connection instances.