BEA WebLogic Server

Programming WebLogic Enterprise JavaBeans

BEA WebLogic Server 6.1
Document Date: February 26, 2003

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Programming WebLogic Enterprise JavaBeans

Part Number	Document Date	Software Version
N/A	June 24, 2002	BEA WebLogic Server 6.1

Contents

About This Document

1. Introducing WebLogic Server Enterprise JavaBeans

2. Designing EJBs

3. Using Message-Driven Beans

4. The WebLogic Server EJB Container and Supported Services

5. WebLogic Server Container-Managed Persistence Services

6. Packaging EJBs for the WebLogic Server Container

7. Deploying EJBs to WebLogic Server

11. weblogic-cmp-rdbms-jar.xml Document Type Definitions

About This Document

This document describes how to develop and deploy Enterprise JavaBeans (EJBs) on WebLogic Server. This document is organized as follows:

- Chapter 1, “Introducing WebLogic Server Enterprise JavaBeans,” is an overview of EJB features supported in WebLogic Server.
- Chapter 2, “Designing EJBs,” is an overview of design techniques developers can use to create EJBs.
- Chapter 3, “Using Message-Driven Beans,” explains how to develop and deploy message-driven beans in the WebLogic Server container.
- Chapter 4, “The WebLogic Server EJB Container and Supported Services,” describes the services available to the EJB with the WebLogic Services container.
- Chapter 5, “WebLogic Server Container-Managed Persistence Services,” describes the EJB container-managed persistence services available for entity EJBs in the WebLogic Server container.
- Chapter 6, “Packaging EJBs for the WebLogic Server Container,” describes the steps necessary to package EJBs for deployment to WebLogic Server.
- Chapter 7, “Deploying EJBs to WebLogic Server,” describes the process for deploying EJBs in the EJB container.
- Chapter 9, “WebLogic Server EJB Utilities,” describes the utilities, shipped with WebLogic Server, that are used with EJBs.
- Chapter 10, “weblogic-ejb-jar.xml Document Type Definitions,” describes the WebLogic-specific deployment descriptors found in the `weblogic-ejb-jar.xml` file shipped with WebLogic Server 6. 1.

- Chapter 11, “weblogic-cmp-rdbms- jar.xml Document Type Definitions,” describes the WebLogic-specific deployment descriptors found in `weblogic-cmp-rdbms-jar.xml` file, shipped with WebLogic Server 6.1.

Audience

This document is intended mainly for application developers who are interested in developing Enterprise JavaBeans (EJBs) for use in dynamic Web-based applications. Readers are assumed to be familiar with EJB architecture, XML coding, and Java programming.

e-docs Web Site

BEA WebLogic Server product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation.

How to Print the Document

You can print a copy of this document from a Web browser, one file at a time, by using the File—>Print option on your Web browser.

A PDF version of this document is available on the WebLogic Server documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Server documentation Home page, click Download Documentation and select the document you want to print.

Adobe Acrobat Reader is available at no charge from the Adobe Web site at http://www.adobe.com/.

Related Information

The BEA corporate Web site provides all documentation for WebLogic Server. However, the following information will provide you with related information that may help you when using Enterprise JavaBeans with WebLogic Server.

- For more information about Sun Microsystem's EJB Specification, see the JavaSoft EJB Specification.
- For more information about the J2EE Specification, see the JavaSoft J2EE Specification.
- For more information about SunMicrosystem's EJB deployment descriptors and descriptions, see the JavaSoft EJB Specification.
- For more information on the deployment descriptors in WebLogic Server's weblogic-ejb-jar.xml file, see Chapter 10, "weblogic-ejb-jar.xml Document Type Definitions."
- For more information on the deployment descriptors in WebLogic Server's weblogic-cmp-rdbms-jar.xml file, see Chapter 11, "weblogic-cmp-rdbms- jar.xml Document Type Definitions."
- For more information on transactions, see Programming WebLogic JTA.
- For more information about WebLogic's implementation of the JavaSoft Remote Method Invocation (RMI) specification, see the following:
  - JavaSoft Remote Method Invocation Specification
  - Programming WebLogic RMI
  - Programming RMI over IIOP

Contact Us!

Your feedback on the BEA WebLogic Server documentation is important to us. Send us e-mail at **docsupport@bea.com** if you have questions or comments. Your

comments will be reviewed directly by the BEA professionals who create and update the WebLogic Server documentation.

In your e-mail message, please indicate the software name and version you are using as well as the title and document date of your documentation.

If you have any questions about this version of BEA WebLogic Server, or if you have problems installing and running BEA WebLogic Server, contact BEA Customer Support through BEA WebSupport at http://www.bea.com. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
italics	Indicates emphasis or book titles.

Convention	Item

`monospace text`	Indicates code samples, commands and their options, data structures and their members, data types, directories, and file names and their extensions. Monospace text also indicates text that you must enter from the keyboard.
	Examples:
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

`monospace italic text`	Identifies variables in code. *Example*: `String` *`expr`*
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Example*s: LPT1 SIGNON OR
`{ }`	Indicates a set of choices in a syntax line. The braces themselves should never be typed.

`[ ]`	Indicates optional items in a syntax line. The brackets themselves should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`
	`[-l` *`file-list`*`]...`

`|`	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.

Convention	Item
. . .	Indicates one of the following in a command line: ■ That an argument can be repeated several times in a command line ■ That the statement omits additional optional arguments ■ That you can enter additional parameters, values, or other information The ellipsis itself should never be typed. *Example*: `buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...`
. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

CHAPTER

1 Introducing WebLogic Server Enterprise JavaBeans

WebLogic Server 6.1 includes an implementation of Sun Microsystems Enterprise JavaBeans (EJB) architecture as defined by Sun's EJB specification.

Note: WebLogic Server 6.1 is compliant with the Sun J2EE specification and EJB 1.1 specification. It also includes an implementation of the preliminary EJB 2.0 specification. Except where descriptions of EJB features and behaviors make specific mention of EJB 1.1 or EJB 2.0, all information in this guide relates to both implementation. You can deploy existing EJB 1.1 beans in this version of WebLogic Server. However, if you are developing new beans, we recommend that you develop EJB 2.0 beans.

The following sections provide an overview of the EJB features and introduce the changes in the WebLogic Server 6.1 Enterprise JavaBeans implementation:

Overview of Enterprise JavaBeans

Enterprise JavaBeans are reusable Java components that implement business logic and enable you to develop component-based distributed business applications. EJBs reside in an EJB container, which provides a standard set of services such as persistence, security, transactions, and concurrency. Enterprise JavaBeans are the standard for defining server-side components. WebLogic Server's implementation of the Enterprise JavaBeans component architecture is based on Sun Microsystems EJB specification.

EJB Components

An EJB consists of three main components:

- **Remote interface**. This interface exposes business logic to the client.
- **Home interface.** The EJB factory. Clients use this interface to create, find, and remove EJB instances.
- **Bean class.** This interface implements business logic.

To create an EJB, you code a distributed application's business logic into the EJB's implementation class; specify the deployment parameters in deployment descriptor files; and package the EJB into a JAR file. You can then deploy the EJB individually from a JAR file, or package it along with other EJBs and a Web application into an EAR file, which you then deploy on WebLogic Server. Client applications can locate the EJB and create an instance of the bean using the bean's home interface. The client can then invoke the methods of the EJB using the EJB's remote interface. WebLogic Server manages the EJB container and provides access to system-level services such as database management, security management, and transaction services.

Types of EJBs

The EJB specification defines the following four types of Enterprise JavaBeans:

- **Stateless session.** An instance of these non-persistent EJBs provides a service without storing an interaction or conversation state between methods. Any instance can be used for any client. Stateless session beans can use either container-managed or bean-managed transaction demarcation.
- **Stateful session.** An instance of these non-persistent EJBs maintains state across methods and transactions. Each instance is associated with a particular client. Stateful session beans can use either container-managed or bean-managed transaction demarcation.
- **Entity.** An instance of these persistent EJBs represents an object view of the data, usually rows in a database. An entity bean has a primary key as a unique identifier. Entity. An instance of these persistent EJBs represents an object view of the data, usually rows in a database. An entity bean has a primary key as a unique identifier. Entity bean persistence can be container-managed or bean-managed, but uses container-managed transaction demarcation only.
- **Message-driven.** An instance of these EJBs is integrated with the Java Message Service (JMS) to enable message-driven beans to act as a standard JMS message consumer and perform asynchronous processing between the server and the JMS message producer. The WebLogic Server container directly interacts with a message-driven bean by creating bean instances and passing JMS messages to those instances as necessary. Message-driven beans can use either container-managed or bean-managed transaction demarcation.

Note: Message driven beans are part of the Sun Microsystems EJB 2.0 specification. They are not part of the EJB 1.1 specification.

Implementation of Preliminary Specifications

The following sections describe the use of WebLogic Server with non-final implementations of Java specifications.

Preliminary J2EE Specification

WebLogic Server 6.1 is available in two different versions that do one of the following:

- Enables an implementation of advanced J2EE 1.3 features along with the J2EE 1.2 features
- Enables the J2EE 1.2 features only, which is a fully compliant implementation of the J2EE 1.2 specification

These two options comply with the rules governing J2EE. Both versions offer the same container and differ only in the APIs that are available.

Preliminary EJB 2.0 Specification

The Enterprise JavaBeans 2.0 implementation in WebLogic Server is fully supported and can be used in production. However, be advised that the Sun Microsystems EJB 2.0 specification is not yet finalized, and the WebLogic Server implementation of the EJB 2.0 architecture is based on the most current public draft of this specification. Consequently once the specification is finalized, there could be changes to the Enterprise JavaBeans 2.0 implementation in future versions of WebLogic Server. These changes may cause application code developed for WebLogic Server 6.1 to be incompatible with EJB 2.0 implementations supported in future releases.

WebLogic Server EJB 2.0 Support

WebLogic Server supports an implementation of Sun Microsystems's EJB 2.0 specification and is compliant with the Sun Microsystem's EJB 1.1 specification. In most cases, you can use EJB 1.1 beans with this version of WebLogic Server. However, in a few cases you may need to migrate existing EJB deployments from earlier versions of WebLogic Server to this version of the EJB container. If necessary, see the information on "DDConverter" on page 9-4 for instructions on converting the beans.

Sun Microsystem's EJB 2.0 specification supports the following new features:

- New type of EJB called message-driven bean that is a Java Messaging Service (JMS) consumer. See Chapter 3, “Using Message-Driven Beans,” for more information.
- New entity EJB container-managed persistence model that provides a new way of handling container-managed persistence. See Chapter 5, “WebLogic Server Container-Managed Persistence Services,” for more information.
- Model for creating container-managed relationships between entity EJBs allows you to define the relationship between the beans in the implementation classes and the deployment descriptors. See Chapter 5, “WebLogic Server Container-Managed Persistence Services,” for more information.
- New standard query language called EJB-QL which you use to query EJBs and their properties. See Chapter 5, “WebLogic Server Container-Managed Persistence Services,” for more information.
- New `ejbSelect` methods that allow an entity EJB to internally query for properties using an EJB-QL query defined in a deployment descriptor. See Chapter 5, “WebLogic Server Container-Managed Persistence Services,” for more information.
- Local interfaces for session and entity beans. EJB relationships are now based on the local interface. Any EJB that participates in a relationship must have a local interface. See Chapter 5, “WebLogic Server Container-Managed Persistence Services,” for more information.
- Home methods that allow you to execute a home business method that is not specific to a particular instance of an entity bean. You use the home interface to define one or more home methods for the entity bean See Chapter 2, “Designing EJBs,” for more information.

EJB Roles

The process of developing EJBs is divided into the following distinct roles.

Application Roles

- **Enterprise Bean Providers**—Enterprise Bean Providers produce the EJBs. Their output is the `ejb.jar` file that contains one or more EJBs. The providers use the design process documented in this guide to design the EJBs that are deployed in the WebLogic Server environment.

  For more information on the design process, see Chapter 2, "Designing EJBs."

- **Application Assemblers**—Application Assemblers combine the EJBs into deployable units such as JARs, EARs, or WARs. Their output is the JAR, EAR, or WAR file that contains the EJB and the application assembly instructions; these instructions are set by the EJB's deployment descriptors. The assemblers use the design process and the EJB deployment descriptor elements to assemble the deployment unit.

  For more information in the design process, see Chapter 2, "Designing EJBs." For more information in the assembly process, see Chapter 6, "Packaging EJBs for the WebLogic Server Container." For more information on the deployment descriptors, see Chapter 10, "weblogic-ejb-jar.xml Document Type Definitions," and Chapter 11, "weblogic-cmp-rdbms- jar.xml Document Type Definitions."

Infrastructure Roles

- **Container Providers**—Container Providers supply EJB deployment tools, container monitoring and management tools, and runtime support for deployed EJB instances. This support includes services such as transaction and security management, network distribution of clients, and scalability. The container providers use the container management process documented in this guide to provide the container.

  For more information on the container management process, see Chapter 4, "The WebLogic Server EJB Container and Supported Services."

- **Persistence Manager Providers**—Persistence Manager Providers are responsible for persistence support for the Entity EJBs in the container, if the EJB has container-managed persistence. This support is provided during deployment to generate the code that moves data between the EJB and a database. The persistence manager providers use the deploy process and

container-managed persistence (CMP) information documented in this guide to provide container-managed persistence.

For more information on container-managed persistence, see Chapter 5, "WebLogic Server Container-Managed Persistence Services." For more information on the deploy process, see Chapter 6, "Packaging EJBs for the WebLogic Server Container."

Deployment and Management Roles

- **Deployers**—Deployers, following the application assembly instructions in the deployment descriptors, deploy the EJBs contained in the JAR, EAR, or WAR file to the target environment. The target environment includes the WebLogic Server environment and the container. The deployer's output is the EJB customized for the target environment and deployed in a specific EJB container. The deployers use the deploy process documented in this guide to deploy the EJBs.

  For more information on the deploy process, see Chapter 7, "Deploying EJBs to WebLogic Server."

- **System Administrators**—System Administrators configure and administer the computing and networking infrastructure that includes WebLogic Server and the container. System Administrators use the administration process documented in the Administration Guide and the WebLogic Server online help to manage the deployed applications at runtime.

  For more information on system administrator's tasks, see the Administration Guide.

EJB Enhancements in WebLogic Server 6.1

The following EJB enhancements are new to this release of WebLogic Server.

Changed EJB Deployment Elements

For information about new and changed deployment elements in WebLogic Server 6.1, see “Changed EJB Deployment Elements in WebLogic Server 6.1” on page 10-5;

Read-Only Multicast Invalidation Support

Non-transactional entity bean caching provides a better way to invalidate or update cached data. You invalidate a read-only entity bean by using an invalidate method on your home interface. For more information on non-transactional entity bean caching, see “Read-Only Multicast Invalidation” on page 4-17.

Automatic Generated Primary Key Support

The WebLogic Server EJB container can provide automatically generated primary keys. This feature uses the native automatic key generation facilities provided by Oracle or SQLServer databases. If you are not using one of those databases, you can enable key generation through a user-designated key table. For more information on automatically generated primary keys, see “Automatic Primary Key Generation for EJB 2.0 CMP” on page 5-22.

Automatic Table Creation

You can automatically create tables based on the deployment descriptions in the deployment files and the bean class, if the table does not already exist. This feature is for use during the development phase and does not provide production quality support. However, it is very helpful for testing your designs prior to deployment in a production environment. For more information on automatic table creation, see “Automatic Table Creation” on page 5-25.

Oracle SELECT HINTS

You can pass your INDEX usage hints to the Oracle Query optimizer in WebLogic QL queries, which provide hints to the Oracle database engine. This feature is most helpful if you know that the database you are searching would benefit from these hints. For more information on Oracle SELECT HINTS, see "Using Oracle SELECT HINTS" on page 5-13.

EJB Deployment Descriptor Editor

The EJB Deployment Descriptor Editor is a an extension of the WebLogic Server Administration Console that enables you to edit the deployment descriptors for your EJBs in a graphical environment. For more information on this editor, see "Specifying and Editing the EJB Deployment Descriptors" on page 6-5 and the Administration Console online help.

ejb-client.jar Support

Use the `ejb-client.jar` file to package required classes to compile the client into one JAR file. The `ejb-client.jar` file contains the EJB interfaces necessary to call an EJB. You specify that the WebLogic EJB compiler (`weblogic.ejbc`) automatically create the `ejb-client.jar` file in the bean's deployment descriptor file. For more information on `ejb-client .jar` files, see "Specifying an ejb-client.jar" on page 6-13.

BLOB and CLOB Support

Use BLOBs and CLOBs to translate large objects into byte arrays or strings, with Oracle. For more information on BLOBs and CLOBs, see "BLOB and CLOB DBMS Column Support for the Oracle DBMS" on page 5-14.

Cascade Delete Support

The Cascade Delete feature enables you to remove entity objects. You can specify Cascade Delete for one-to-one and one-to-many relationships; many-to-many relationships are not supported. For more information on Cascade Delete, see "Cascade Delete" on page 5-15.

Local Interface Support

WebLogic Server's EJB container provides support for local interfaces. The EJB container makes the local home interface accessible to local clients through JNDI. Support for remote interfaces with container-managed persistence (CMP) relationships is still available in this release, but not recommended for new development. For more information on local interface support, see "Using the Local Client" on page 5-30.

Flushing the CMP Cache Support

You can specify that the container-managed persistence (CMP) cache be flushed before every query so that the changes show up in the results. For more information on this feature, see "Flushing the CMP Cache" on page 5-19.

Tuned CMP 1.1 Support

This release enables the EJB container to support tuned updates for container-managed persistence (CMP) 1.1 entity beans. The EJB container automatically determines and writes back to the database only those container-managed fields that have been modified in the transaction. If no fields are modified, there is no database update.This feature is enabled by default and can be disabled; however, its use is recommended for performance reasons. For more information on tuned CMP support, see "Tuned EJB 1.1 CMP Updates in WebLogic Server" on page 5-18.

EJB Developer Tools

BEA provides several tools you can use to help you create and configure EJBs.

ANT Tasks to Create Skeleton Deployment Descriptors

You can use the WebLogic ANT utilities to create skeleton deployment descriptors. These utilities are Java classes shipped with your WebLogic Server distribution. The ANT task looks at a directory containing an EJB and creates deployment descriptors based on the files it finds in the `ejb.jar` file. Because the ANT utility does not have information about all of the desired configurations and mappings for your EJB, the skeleton deployment descriptors the utility creates are incomplete. After the utility creates the skeleton deployment descriptors, you can use a text editor, an XML editor, or the EJB Deployment Descriptor Editor in the Administration Console to edit the deployment descriptors and complete the configuration of your EJB.

For more information on using ANT utilities to create deployment descriptors, see Packaging Enterprise JavaBeans at `http://e-docs.bea.com/wls/docs61/programming/packaging.html`.

EJB Deployment Descriptor Editor

The WebLogic Server Administration Console has an integrated EJB deployment descriptor editor. You must create at least a skeleton of the following deployment descriptor files that you add to the `ejb.jar` file before using this integrated editor:

- `ejb-jar.xml`
- `weblogic-ejb-jar.xml`
- `weblogic-cmp-rdbms-jar.xml`

For more information, see Web Application Deployment Descriptor Editor Help at `http://e-docs.bea.com/wls/docs61/ConsoleHelp/webservices_ddehelp.html`.

XML Editor

The XML editor is a simple, user-friendly tool from Ensemble for creating and editing XML files. It can validate XML code according to a specified DTD or XML Schema. You can use the XML editor on Windows or Solaris machines and download it from the BEA dev2dev at http://dev2dev.bea.com/resourcelibrary/utilitiestools/index.jsp.

CHAPTER

2 Designing EJBs

The following sections provide guidelines for designing WebLogic Server Enterprise JavaBeans (EJB)s. Some suggestions apply to remote object models and Remote Method Invocation (RMI) as much as they do to EJB.

- Designing Session Beans
- Designing Entity Beans
- Designing Message-Driven Beans
- Using WebLogic Server Generic Bean Templates
- Using Inheritance with EJBs
- Accessing Deployed EJBs
- Preserving Transaction Resources

Designing Session Beans

One way to design session beans is to use the model-view design. The *view* is the graph-user interface (GUI) form and the *model* is the piece of code that supplies data to the GUI. In a typical client-server system, the model lives on the same server as the view and talks to the server.

Have the model reside on the server, in the form of a session bean. (This is analogous to having a servlet providing support for an HTML form, except that a model session bean does not affect the final presentation.) There should be one model session bean instance for each GUI form instance, which acts as the form's representative on the

server. For example, if you have a list of 100 network nodes to display in a form, you might have a method called `getNetworkNodes()` on the corresponding EJB that returns an array of values relevant to that list.

This approach keeps the overall transaction time short, and requires minimal network bandwidth. In contrast, consider an approach where the GUI form calls an entity EJB finder method that retrieves references to 100 separate network nodes. For each reference, the client must go back to the datastore to retrieve additional data, which consumes considerable network bandwidth and may yield unacceptable performance.

Designing Entity Beans

Reading and writing RDBMS data via an entity bean can consume valuable network resources. Network traffic may occur between a client and WebLogic Server, as well as between WebLogic Server and the underlying datastore. Use the following suggestions to model entity EJB data correctly and avoid unnecessary network traffic.

Entity Bean Home Interface

The container provides an implementation of the home interface for each entity bean deployed in the container and it makes the home interface accessible to the clients through JNDI. An object that implements an entity beans's home interface is called an EJBHome object. The entity bean's home interface enables a client to do the following:

- Use the `create()` methods to create new entity objects within the home.
- Use the `finder()` methods to find existing entity objects within the home.
- Use the `remove()` methods to remove an entity object from the home.
- Execute a home method that is not specific to a particular entity bean instance.

Make Entity EJBs Coarse-Grained

Do not attempt to model every object in your system as an entity EJB. In particular, small subsets of data consisting of only a few bytes should never exist as entity EJBs, because the trade-off in network resources is unacceptable.

For example, cells in a spreadsheet are too fine-grained and should not be accessed frequently over a network. In contrast, logical groupings of an invoice's entries, or a subset of cells in a spreadsheet can be modeled as an entity EJB, if additional business logic is required for the data.

Encapsulate Additional Business Logic in Entity EJBs

Even coarse-grained objects may be inappropriate for modeling as an entity EJB if the data requires no additional business logic. For example, if the methods in your entity EJB work only to set or retrieve data values, it is more appropriate to use JDBC calls in an RDBMS client or to use a session EJB for modeling.

Entity EJBs should encapsulate additional business logic for the modeled data. For example, a banking application that uses different business rules for "Platinum" and "Gold" customers might model all customer accounts as entity EJBs; the EJB methods can then apply the appropriate business logic when setting or retrieving data fields for a particular customer type.

Optimize Entity EJB Data Access

Entity EJBs ultimately model fields that exist in a data store. Optimize entity EJBs wherever possible to simplify and minimize database access. In particular:

- Limit the complexity of joins against EJB data.
- Avoid long-running operations that require disk access in the datastore.
- Ensure that EJB methods return as much data as possible, so as to minimize round-trips between the client and the datastore. For example, if your EJB client must retrieve data fields, use bulk `get/setAttributes()` methods to minimize network traffic.

Designing Message-Driven Beans

A message-driven bean acts as a message consumer in the WebLogic JMS messaging system. For more information on designing message-driven beans, see Chapter 3, "Using Message-Driven Beans."

Using WebLogic Server Generic Bean Templates

For each EJB type, WebLogic Server provides a generic class that contains Java callbacks, or listeners, that are required for most EJBs. The generic classes are in the `weblogic.ejb` package:

- `GenericEnterpriseBean`
- `GenericEntityBean`
- `GenericMessageDrivenBean`
- `GenericSessionBean`

You can implement a generic bean template in a class of your own by importing the generic class into the class you are writing. This example imports the `GenericSessionBean` class into `HelloWorldEJB`:

```
import weblogic.ejb.GenericSessionBean;
...
public class HelloWorldEJB extends GenericSessionBean {
```

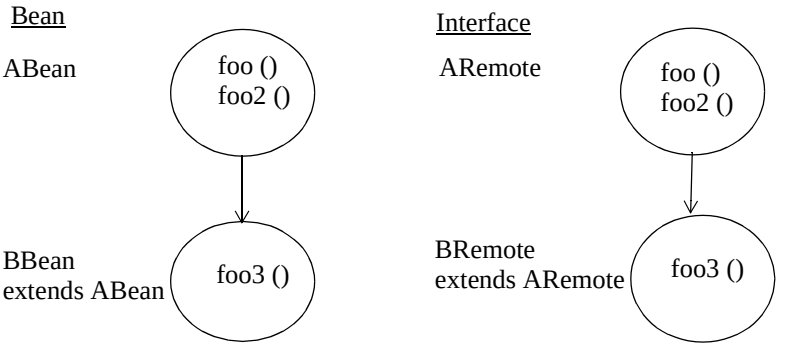
Using Inheritance with EJBs

Using inheritance may be appropriate when building groups of related beans that share common code. However, be aware of several inheritance restrictions apply to EJB implementations.

For bean-managed entity EJBs, the `ejbCreate()` method must return a primary key. Any class that inherits from the bean-managed EJB class cannot have an `ejbCreate()` method that returns a different primary key class than does the bean-managed EJB class. This restriction applies even if the new class is derived from the base EJB's primary key class. The restriction also applies to the bean's `ejbFind()` method.

Also, EJBs inheriting from other EJB implementations change the interfaces. For example, the following figure shows a situation where a derived bean adds a new method that is meant to be accessible remotely:

Figure 2-1 Derived bean (BBean) adding new method to be accessible remotely



An additional restriction is that because `AHome.create()` and `BHome.create()` return different remote interfaces, you cannot have the `BHome` interface inherit from the `AHome` interface. You can still use inheritance to have methods in the beans that are unique to a particular class, that inherit from a superclass or that are overridden in the subclass. See the EJB 1.1 subclass Child example in the and classes in the WebLogic Server distribution for an examples of inheritance.

Accessing Deployed EJBs

WebLogic Server automatically creates implementations of an EJB's home and remote interfaces that can function remotely. This means that all clients — whether they reside on the same server as the EJB, or on a remote computer — can access deployed EJBs in a similar fashion.

All EJBs must specify their environment properties using Java Naming and Directory Interface (JNDI). You can configure the JNDI name spaces of EJB clients to include the home EJBs that reside anywhere on the network — on multiple machines, application servers, or containers.

However, in designing enterprise application systems, you must still consider the effects of transmitting data across a network between EJBs and their clients. Because of network overhead, it is still more efficient to access beans from a "local" client — a servlet or another EJB — than to do so from a remote client where data must be marshalled, transmitted over the network, and then unmarshalled.

Differences Between Accessing EJBs from Local Clients and Remote Clients

One difference between accessing EJBs from local clients and remote clients is in obtaining an `InitialContext` for the bean. Remote clients obtain an `InitialContext` from the WebLogic Server `InitialContext` factory. WebLogic Server local clients generally use a `getInitialContext` method to perform this lookup, similar to the following excerpt:

Figure 2-2 Code sample of a local client performing a lookup

```
...
Context ctx = getInitialContext("t3://localhost:7001", "user1", "user1Password");
...
static Context getInitialContext(String url, String user, String password) {
   Properties h = new Properties();
   h.put(Context.INITIAL_CONTEXT_FACTORY,
      "weblogic.jndi.WLInitialContextFactory");
   h.put(Context.PROVIDER_URL, url);
   h.put(Context.SECURITY_PRINCIPAL, user);
```

```
   h.put(Context.SECURITY_CREDENTIALS, password);
}
```

Internal clients of an EJB, such as servlets, can simply create an `InitialContext` using the default constructor, as shown here:

```
Context ctx = new InitialContext();
```

Restrictions on Concurrency Access of EJB Instances

Although database concurrency is the default and recommended concurrency access option, multiple clients can use the exclusive concurrency option to access EJBs in a serial fashion. Using this exclusive option means that if two clients simultaneously attempt to access the same entity EJB instance (an instance having the same primary key), the second client is blocked until the EJB is available. For more information on the database concurrency option, see "Exclusive Locking Services" on page 4-37.

Simultaneous access to a stateful session EJB results in a `RemoteException`. This access restriction on stateful session EJBs applies whether the EJB client is remote or internal to WebLogic Server. However, you can set the `allow-concurrent-calls` option to specify that a stateful session bean instance will allow concurrent method calls.

If multiple servlet classes access a session EJB, each servlet thread (rather than each instance of the servlet class) must have its own session EJB instance. To avoid concurrent access, a JSP/servlet can use a stateful session bean in request scope.

Storing EJB References in Home Handles

Once a client obtains the `EJBHome` object for an EJB instance, you can create a handle to the home object by calling `getHomeHandle()`. `getHomeHandle()` returns a `HomeHandle` object, which can be used to obtain the home interface to the same EJB at a later time.

A client can pass the `HomeHandle` object as arguments to another client, and the receiving client can use the handle to obtain a reference to the same EJBHome object. Clients can also serialize the `HomeHandle` and store it in a file for later use.

Using Home Handles Across a Firewall

By default, WebLogic Server stores its IP address in the `HomeHandle` object for EJBs. This can cause problems with certain firewall systems. If you cannot locate `EJBHome` objects when you use home handles passed across a firewall, use the following steps:

1. Start WebLogic Server.
2. Start the WebLogic Server Administration Console.
3. From the left pane, expand the Servers node and select a server.
4. In the right pane, view the configuration information.
5. Select the Tuning tab.
6. Check the Reverse DNS Allowed box to enable reverse DNS lookups.

When you enable reverse DNS lookups, WebLogic Server stores the DNS name of the server, rather than the IP address, in EJB home handles.

Preserving Transaction Resources

Database transactions are typically one of the most valuable resources in an online transaction-processing system. When you use EJBs with WebLogic Server, transaction resources are even more valuable because of their relationship with database connections.

WebLogic Server can use a single connection pool to service multiple, simultaneous database requests. The efficiency of the connection pool is largely determined by the number and length of database transactions that use the pool. For non-transactional database requests, WebLogic Server can allocate and deallocate a connection very quickly, so that the same connection can be used by another client. However, for transactional requests, a connection becomes “reserved” by the client for the duration of the transaction.

To optimize transaction use on your system, always follow an “inside-out” approach to transaction demarcation. Transactions should begin and end at the “inside” of the system (the database) where possible, and move “outside” (toward the client application) only as necessary. The following sections describe this rule in more detail.

Allowing the Datastore to Manage Transactions

Many RDBMS systems provide high-performance locking systems for Online Transaction Processing (OLTP) transactions. With the help of Transaction Processing (TP) monitors such as Tuxedo, RDBMS systems can also manage complex decision support queries across multiple datastores. If your underlying datastore has such capabilities, use them where possible. Never prevent the RDBMS from automatically delimiting transactions.

Using Container-Managed Transactions Instead of Bean-Managed Transactions for EJBs

Your system should rarely rely on bean-managed transaction demarcation. Use WebLogic Server container-managed transaction demarcation unless you have a specific need for bean-managed transactions.

Possible scenarios where you must use bean-managed transactions are:

- You define multiple transactions from within a single method call. WebLogic Server demarcates transactions on a per-method basis.

  Note: However, instead of using multiple transactions in a single method call, it is better to break the method into multiple methods, with each of the multiple methods having its own container-managed transaction.

- You define a single transaction that “spans” multiple EJB method calls. For example, you define a stateful session EJB that uses one method to begin a transaction, and another method to commit or roll back a transaction.

  Note: Avoid this practice if possible because it requires detailed information about the workings of the EJB object. However, if this scenario is required, you must use bean-managed transaction coordination, and you must coordinate client calls to the respective methods.

Never Demarcate Transactions from Application

In general, client applications are not guaranteed to stay active over long periods of time. If a client begins a transaction and then exits before committing, it wastes valuable transaction and connection resources in WebLogic Server. Moreover, even if the client does not exit during a transaction, the duration of the transaction may be unacceptable if it relies on user activity to commit or roll back data. Always demarcate transactions at the WebLogic Server or RDBMS level where possible.

For more information on demarcating transaction see "Transaction Management Responsibilities" on page 4-29.

Always Use A Transactional Datasource for Container-Managed EJBs

If you configure a JDBC datasource factory for use with container-managed EJBs, make sure you configure a transactional datasource (`TXDataSource`) rather than a non-transactional datasource (`DataSource`). With a non-transactional datasource, the JDBC connection operates in auto commit mode, committing each insert and update operation to the database immediately, rather than as part of a container-managed transaction.

CHAPTER

3 Using Message-Driven Beans

The following sections describe how to develop message-driven beans and to deploy then on WebLogic Server. Because message-driven beans use parts of the standard JMS API, you should first become familiar with the WebLogic Java Messaging Service (JMS) before attempting to implement message-driven beans. See the Programming WebLogic JMS document for more information.

Note: Message-driven beans are an EJB 2.0 feature.

What Are Message-Driven Beans?

A message-driven bean is an EJB that acts as a message consumer in the WebLogic JMS messaging system. As with standard JMS message consumers, message-driven beans receive messages from a JMS Queue or Topic, and perform business logic based on the message contents. WebLogic Server associates a message-driven bean with a

JMS destination, such as a topic or queue, at deployment time, and WebLogic Server automatically creates and removes message-driven bean instances as needed to process incoming messages.

Differences Between Message-Driven Beans and Standard JMS Consumers

Because message-driven beans are implemented as EJBs, they benefit from several key services that are not available to standard JMS consumers. Most importantly, message-driven bean instances are wholly managed by the WebLogic Server EJB container. Using a single message-driven bean class, WebLogic Server creates multiple EJB instances as necessary to process large volumes of messages concurrently. This stands in contrast to a standard JMS messaging system, where the developer must create a `MessageListener` class that uses a server-wide session pool.

The WebLogic Server container provides other standard EJB services to message-driven beans, such as security services and automatic transaction management. These services are described in more detail in "Transaction Management" on page 4-28 and in "Using Transaction Services with Message-Driven Beans" on page 3-14.

Finally, message-driven beans benefit from the write-once, deploy-anywhere quality of EJBs. Whereas a JMS `MessageListener` is tied to specific `session pools`, Queues, or Topics, message-driven beans can be developed independently of available server resources. A message-driven bean's Queues and Topics are assigned only at deployment time, utilizing resources available on WebLogic Server.

Note: One limitation of message-driven beans compared to standard JMS listeners is that you can associate a given message-driven bean deployment with only one Queue or Topic, as described in "Invoking a Message-Driven Bean" on page 3-12. If your application requires a single JMS consumer to service messages from multiple Queues or Topics, you must use a standard JMS consumer, or deploy multiple message-driven bean classes.

Differences Between Message-Driven Beans and Stateless Session EJBs

The dynamic creation and allocation of message-driven bean instances partially mimics the behavior of stateless session EJB instances. However, message-driven beans differ from stateless session EJBs (and other types of EJBs) in several significant ways:

- Message-driven beans process multiple JMS messages asynchronously, rather than processing a serialized sequence of method calls.
- Message-driven beans have no home or remote interface, and therefore cannot be directly accessed by internal or external clients. Clients interact with message-driven beans only indirectly, by sending a message to a JMS Queue or Topic.

Note: Only the WebLogic Server container directly interacts with a message-driven bean by creating bean instances and passing JMS messages to those instances as necessary.

- WebLogic Server maintains the entire life cycle of a message-driven bean; instances cannot be created or removed as a result of client requests or other API calls.

Concurrent Processing for Topics and Queues

Message-driven beans (MDBs) support concurrent processing for both topics and queues. Previously, only concurrent processing for queues was supported.

To ensure concurrency, the container uses threads from the execute queue. The default setting for the `max-beans-in-free-pool` deployment descriptor found in the `weblogic-ejb-jar.xml` file provides the most parallelism. The only reason to change this setting would be to limit the number of parallel consumers.

Note: The maximum number of MDBs configured—via the `max-beans-in-free-pool` deployment descriptor element— to receive messages at one time cannot exceed the maximum number of execution

threads. For example, if `max-beans-in-free-pool` is set to 50 but 25 is the maximum number of execution threads allowed, only 25 of the MDBs will actually receive messages.

For more information on `max-beans-in-free-pool`, see, “max-beans-in-free-pool” on page 10-45.

Developing and Configuring Message-Driven Beans

When developing message-driven beans, follow the conventions described in the JavaSoft EJB 2.0 specification, and observe the general practices that result in proper bean behavior. Once you have created the message-driven bean class, configuring the bean for WebLogic Server by specify the bean’s deployment descriptor elements in the EJB XML deployment descriptor files.

To develop 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 message-driven bean class must define the following methods:

   - One `ejbCreate()` method that the container uses to create an instance of the message-driven bean on the free pool.
   - One `onMessage()` method that is called by the bean’s container when a message is received. This method contains the business logic that handles processing of the message.
   - One `ejbRemove()` method that removes the message-driven bean instance from the free pool.

   For an example of output for a message-driven bean class, see “Message-Driven Bean Class Requirements” on page 3-6

2. Specify the following XML deployment descriptor files for the message-driven bean.

- `ejb-jar.xml`
- `weblogic-ejb-jar.xml`
- `weblogic-cmp-rdbms-jar.xml`

For instructions on specifying the XML files, see "Specifying and Editing the EJB Deployment Descriptors" on page 6-5.

3. Set the `message-driven` element in the bean's `ejb-jar.xml` file to declare the bean.
4. Set the `message-driven-destination` element in the bean's `ejb-jar.xml` file to specify whether the bean is intended for a Topic or Queue.
5. Set the `subscription-durability` sub-element in the bean's `ejb-jar.xml` file when you want to specify whether an associated Topic should be durable.
6. If your bean will demarcate its own transaction boundaries, set the `acknowledge-mode` sub-element to specify the JMS acknowledgment semantics to use. This element has two possible values: `AUTO_ACKNOWLEDGE` (the default) or `DUPS_OK_ACKNOWLEDGE`.
7. If the container will manage the transaction boundaries, set the `transaction-type` element in the bean's `ejb-jar.xml` file to specify how the container must manage the transaction boundaries when delegating a method invocation to an enterprise bean's method.

The following sample shows how to specify a message-driven bean in the `ejb-jar.xml` file.

Figure 3-1 Sample XML stanza from an `ejb-jar.xml` file:

```
<enterprise-beans>
       <message-driven>
              <ejb-name>exampleMessageDriven1</ejb-name>

<ejb-class>examples.ejb20.message.MessageTraderBean</ejb-class>
              <transaction-type>Container</transaction-type>
              <message-driven-destination>
                     <destination-type>
                            javax.jms.Topic
```

```
                    </destination-type>
                </message-driven-destination>
                ...
        </message-driven>
        ...
</enterprise-beans>
```

8. Set the `message-driven-descriptor` element in the bean's `weblogic-ejb-jar.xml` file to associate the message-driven bean with a JMS destination in WebLogic Server.

The following sample shows how to specify a message-driven bean in an `weblogic-ejb-jar.xml` file.

Figure 3-2 Sample XML stanza from an `weblogic-ejb-jar.xml` file:

```
<message-driven-descriptor>
        <destination-jndi-name>...</destination-jndi-name>
</message-driven-descriptor>
```

9. Compile and generate the message-driven bean class using instructions in "Packaging EJBs into a Deployment Directory" on page 6-9.
10. Deploy the bean on WebLogic Server using the instructions in "Deploying Compiled EJB Files" on page 7-9.

The container manages the message-driven bean instances at runtime.

Message-Driven Bean Class Requirements

The EJB 2.0 specification provides detailed guidelines for defining the methods in a message-driven bean class. The following output shows the basic components of a message-driven bean class. Classes, methods, and method declarations are highlighted **`bold`**.

Figure 3-3 Sample output of basic components of message-driven beans class

```
public class MessageTraderBean implements MessageDrivenBean,
MessageListener{

       public MessageTraderBean() {...};

              // An EJB constructor is required, and it must not

              // accept parameters. The constructor must not be
declared as

              // final or abstract.

       public void ejbCreate() (...)

              //ejbCreate () is required and must not accept
parameters.

              The throws clause (if used) must not include an
application

              //exception. ejbCreate() must not be declared as
final or static.

       public void onMessage(javax.jms.Message MessageName) {...}

              // onMessage() is required, and must take a single
parameter of

              // type javax.jms.Message. The throws clause (if
used) must not

              // include an application exception. onMessage() must
not be

              // declared as final or static.

       public void ejbRemove() {...}

              // ejbRemove() is required and must not accept
parameters.

              // The throws clause (if used) must not include an
application

              //exception. ejbRemove() must not be declared as
final or static.

       // The EJB class cannot define a finalize() method

}
```

Using the Message-Driven Bean Context

WebLogic Server calls `setMessageDrivenContext()` to associate the message-driven bean instance with a container context.This is not a client context; the client context is not passed along with the JMS message. WebLogic Server provides the EJB with a container context, whose properties can be accessed from within the bean's instance by using the following methods from the `MessageDrivenContext` interface:

- `getCallerPrincipal())` – This method is inherited from the EJBContext interface and should not be called by message-driven bean instances.
- `isCallerInRole())` – This method is inherited from the EJBContext interface and should not be called by message-driven bean instances.
- `setRollbackOnly()` – The EJB can use this method only if it uses container-managed transaction demarcation.
- `getRollbackOnly()` – The EJB can use this method only if it uses container-managed transaction demarcation.
- `getUserTransaction()` – The EJB can use this method only if it uses bean-managed transaction demarcation.

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 a message-driven EJB instance yields an `IllegalStateException`.

Implementing Business Logic with onMessage()

The message-driven bean's `onMessage()` method implements the business logic for the EJB. WebLogic Server calls `onMessage()` when the EJB's associated JMS Queue or Topic receives a message, passing the full JMS message object as an argument. It is the message-driven bean's responsibility to parse the message and perform the necessary business logic in `onMessage()`.

Make sure that the business logic accounts for asynchronous message processing. For example, it cannot be assumed that the EJB receives messages in the order they were sent by the client. Instance pooling within the container means that messages are not received or processed in a sequential order, although individual `onMessage()` calls to a given message-driven bean instance are serialized.

See javax.jms.MessageListener.onMessage() for more information.

Specifying Principals and Setting Permissions for JMS Destinations

Message-driven beans connect to the JMS destination using the `run-as` principal. The `run-as` principal maps to the `run-as` element that is set in the `ejb-jar.xml` file. This setting specifies the `run-as` identity used for the execution of the message-driven bean's methods. A message-driven bean is associated with a JMS destination when you deploy the bean in the WebLogic Server container. The JMS destination can either be a queue or a topic. You specify the JMS destination by setting the `jms-destination-type` element to either `queue` or `topic` in the message-driven bean's `ejb-jar.xml` file.

Set the permissions for the bean's `run-as` principal to `receive`, as described below, when connecting message-driven beans to the JMS destinations. This allows the message-driven bean to connect to remote queues in the same domain or in another domain as long as the same principal is defined in the other domain. WebLogic Server uses the default `guest` user if you do not specify the `run-as` principal. However, whether you use the `run-as` principal or `guest`, you must assign the `receive` permission to the security principal.

To set the `receive` permission, you must first create a new access control list (ACL) or modify an existing one. are lists of Users and Groups that have permission to access the resources. Permissions are the privileges required to access resources, such as permission to read, write, send, and receive files and load servlets, and link to libraries.

Note: Do not use the `system` user for message-driven beans that connect to JMS destinations because `system` prevents the message-driven bean from connecting to a destination in another domain.

For more information on security principal users, see Defining Users.

See the following instructions to create the ACL, specify principals, and set permissions:

1. Start the WebLogic Server Administration Console.
2. Go to the Security→ACLs node in the left pane of the Administration Console.
3. In the right pane of the Administration Console, click the Create a New ACL link.

   The ACL Configuration window appears.
4. Specify the name of WebLogic Server resource that you want to protect with an ACL in the New ACL Name field.

   For example, create an ACL for a JMS destination named `topic`.
5. Click Create.
6. Click the Add a New Permission link.
7. Specify the `receive` permission for the `topic` JMS destination resource.
8. Specify the `run-as-principal` user as having the specified permission to the resource.
9. Click Apply.

Specifying Message-Driven Beans as Durable Subscribers

If you associate a message-driven bean with a topic, you can specify that the topic be durable. A durable topic subscription ensures that messages are not missed even if the server is not running. If the server is disconnected it would still receive the message and store it so that when the server is restarted it would receive the message. If you associate a message-driven bean with a topic, but do specify that topic as durable then by default, the topic will be non-durable.

To set the message-driven bean as a durable subscriber, specify the following deployment descriptor elements:

1. Set the `message-driven-destination` element in the bean's `ejb-jar.xml` file to specify whether the bean is intended for a Topic or Queue.
2. Set the `subscription-durability` sub-element in the bean's `ejb-jar.xml` file when you want to specify whether an associated Topic should be durable.
3. Set the `jms-client-id` element in the bean's `weblogic-ejb-jar.xml` file.

   For instructions on specifying the XML files, see "Specifying and Editing the EJB Deployment Descriptors" on page 6-5.

Note: If you are using message-driven beans instead of the standard JMS listeners to handle messages, be advised that a given message-driven bean is associated with only one topic. If your application requires a single JMS consumer to service messages from multiple topics or queues, you must use a standard JMS consumer or deploy multiple message-driven bean classes.

Configuring Message-Driven Beans for Foreign JMS Providers

You can configure message-driven beans to work with non-BEA JMS providers such as IBM MQSeries. For a discussion of how to configure an MDB to use a foreign provider, see "Using Foreign JMS Providers with WLS Message Driven Beans" at http://dev2dev.bea.com/products/wlserver/whitepapers/jmsproviders.jsp.

Reconnecting to a JMS Server or Foreign Service Provider

A message-driven bean listens to an associated JMS destination on either a JMS server deployed on a non-clustered WebLogic Server instance or a foreign service provider. If the connection to that destination is lost, because the server goes down, the message-driven bean attempts to reconnect to that destination at periodic intervals. You can specify the number of seconds between attempts to reconnect to the destination by setting the `jms-polling-interval-seconds` element in the bean's `weblogic-ejb-jar.xml` file.

For instructions on specifying the XML files, see “Specifying and Editing the EJB Deployment Descriptors” on page 6-5.

Handling Exceptions

Message-driven bean methods should not throw an application exception or a `RemoteException`, even in `onMessage()`. If any method throws such an exception, WebLogic Server immediately removes the EJB instance without calling `ejbRemove()`. However, from the client perspective the EJB still exists, because future messages are forwarded to a new bean instance that WebLogic Server creates.

Invoking a Message-Driven Bean

When a JMS Queue or Topic receives a message, WebLogic Server calls an associated message-driven bean as follows:

1. WebLogic Server obtains a new bean instance.

   WebLogic Server uses the `max-beans-in-free-pool` attribute, set in the `weblogic-ejb-jar.xml` file, to determine if a new bean instance is available in the free pool.

2. If a bean instance is available in the free pool, WebLogic Server uses that instance.

   If no bean instance is available in the free pool and the limit specified by `max-beans-in-free-pool`) has been reached, WebLogic Server waits until a bean instance is free. See “max-beans-in-free-pool” on page 10-45 for more information about this attribute.

   If no bean instance is located in the free pool, and the limit specified by `max-beans-in-free-pool` has not been reached, WebLogic Server creates a new instance by calling the bean’s `ejbCreate()` method and then the bean’s `setMessageDrivenContext()` to associate the instance with a container context. The bean can use elements of this context as described in “Using the Message-Driven Bean Context” on page 3-8.

3. WebLogic Server calls the bean's `onMessage()` method to implement the business logic when the bean's associated JMS Queue or Topic receives a message.

   See "Implementing Business Logic with onMessage()" on page 3-8.

Note: These instances can be pooled.

Creating and Removing Bean Instances

The WebLogic Server container calls the message-driven bean's `ejbCreate()` and `ejbRemove()` methods, to create or remove an instance of the bean class. Each message-driven bean must have at least one `ejbCreate()` and `ejbRemove()` method. The WebLogic Server container uses these methods to handle the create and remove functions when a bean instance is created, upon receipt of a message from a JMS Queue or Topic or removed, once the transaction commits.

WebLogic Server receives a message from a JMS queue or Topic

As with other EJB types, the `ejbCreate()` method in the bean class should prepare any resources that are required for the bean's operation. The `ejbRemove()` method should release those resources, so that they are freed before WebLogic Server removes the instance.

Message-driven beans should also perform some form of regular clean-up routine *outside* of the `ejbRemove()` method, because the beans cannot rely on `ejbRemove()` being called under all circumstances (for example, if the EJB throws a runtime exception).

Deploying Message-Driven Beans in WebLogic Server

Deploy the message-driven bean on WebLogic Server either when the server is first started or on a running server. For instructions on deploying the bean, see "Deploying EJBs at WebLogic Server Startup" on page 7-2 or "Deploying EJBs on a Running WebLogic Server" on page 7-3.

Using Transaction Services with Message-Driven Beans

As with other types of EJB, message-driven beans can demarcate transaction boundaries either on their own (using bean-managed transactions), or by having the WebLogic Server container manage transactions (container-managed transactions). In either case, a message-driven bean does not receive a transaction context from the client that sends a message. WebLogic Server always calls a bean's `onMessage()` method by using the transaction context specified in the bean's deployment descriptor file.

Because no client provides a transaction context for calls to a message-driven bean, beans that use container-managed transactions must be deployed with the `Required` or `NotSupported` `trans-attribute` specified for the `container-transaction` element in the `ejb-jar.xml` file.

The following sample code from the `ejb-jar.xml` file shows how to specify the bean's transaction context.

Figure 3-4 Sample XML stanza from an ejb-jar.xml file:

```
<assembly-descriptor>
       <container-transaction>
              <method>
```

```
<ejb-name>MyMessageDrivenBeanQueueTx</ejb-name>
                    <method-name>*</method-name>
                </method>
        <trans-attribute>NotSupported</trans-attribute>
        </container-transaction>
</assembly-descriptor>
```

Message Receipts

The receipt of a JMS message that triggers a call to an EJB's `onMessage()` method is not generally included in the scope of a transaction. However, it is handled differently for bean-managed and container-managed transactions.

- For EJBs that use bean-managed transactions, the message receipt is always outside the scope of the bean's transaction.
- For EJBs that use container-managed transaction demarcation, WebLogic Server includes the message receipt as part of the bean's transaction only if the bean's `transaction-type` element in the `ejb-jar.xml` file is set to `Required`.

Message Acknowledgment

For message-driven beans that use container-managed transaction demarcation, WebLogic Server automatically acknowledges a message when the EJB transaction commits. If the EJB uses bean-managed transactions, both the receipt and the acknowledgment of a message occur outside the EJB transaction context. WebLogic Server automatically acknowledges messages for EJBs with bean-managed transactions, but you can configure acknowledgment semantics using the `acknowledge-mode` deployment descriptor element defined in the `ejb-jar.xml` file.

CHAPTER

4 The WebLogic Server EJB Container and Supported Services

The following sections describe the WebLogic Server EJB container, various aspects of EJB behavior in terms of the features and services that the container provides. See to Chapter 5, “WebLogic Server Container-Managed Persistence Services,” for more information on container-managed persistence (CMP).

EJB Container

The EJB container is a runtime container for the deployed EJBs that is automatically created when WebLogic Server is started. During the entire life cycle of the entity object, from its creations to removal, it lives in the container. The EJB container provides a standard set of services, including caching, concurrency, persistence, security, transaction management, locking, environment, memory replication, environment, and clustering for the entity objects that live in the container.

You can deploy multiple entity beans in a single container. For each entity bean deployed in a container, the container provides a home interface. The home interface allows a client to create, find, and remove entity objects that belong to the entity bean as well as execute home business methods which are not specific to a particular entity bean object. A client can look up the entity bean's home interface through JNDI. The container is responsible for making the entity bean's home interface available in the JNDI name space. For instructions on looking up the home interface through JNDI, see Programming WebLogic JNDI.

EJB Life Cycle

The following sections provide information about how the container supports caching services. They describe the life cycle of EJB instances in WebLogic Server, from the perspective of the server. These sections use the term *EJB instance* to refer to the actual instance of the EJB class. *EJB instance* does not refer to the logical instance of the EJB as seen from the point of view of a client.

Entity EJB Life Cycle

WebLogic Server provides these features to improve performance and throughput for entity EJBs.

- Free pool—stores anonymous entity beans that are used for invoking finders, home methods, and creating entity beans.

- Cache—contains instances that have an identity—a primary key, or are currently enlisted in a transaction (READY and ACTIVE entity EJB instances).

The sections that follow describe the life cycle of an entity bean instance, and how the container populates and manages the free pool and cache. For an illustration of life cycle transitions, see Figure 4-1.

Initializing Entity EJB Instances (Free Pool)

If you specify a non-zero value for `initial-beans-in-free-pool`, WebLogic Server populates the pool with the specified quantity of bean instances at startup.

The default value of `initial-beans-in-free-pool` is zero. Populating the free pool at startup improves initial response time for the EJB, because initial requests for the bean can be satisfied without generating a new instance.

An attempt to obtain an entity bean instance from the free pool will always succeed, even if the pool is empty. If the pool is empty, a new bean instance is created and returned.

POOLED beans are anonymous instances, and are used for finders and home methods. The maximum number of instances the pool can contain is specified by the value of the `max-beans-in-free-pool` element in `weblogic-ejb-jar.xml`.

READY and ACTIVE Entity EJB Instances (Cache)

When a business method is called on a bean, the container obtains an instance from the pool, calls `ejbActivate`, and the instance services the method call.

A READY instance is in the cache, has an identity—an associated primary key—but is not currently enlisted in a transaction. WebLogic maintains READY entity EJB instances in least-recently-used (LRU) order.

An ACTIVE instance is currently enlisted in a transaction. After completing the transaction, the instance becomes READY, and remains in cache until space is needed for other beans.

The Current Beans in Cache field in the monitoring tab of the Administration Console displays the count of READY and ACTIVE beans.

The effect of `max-beans-in-cache`, and the quantity of instances with the same primary key allowed in the cache vary by concurrency strategy, as described in the following section, "Cache Rules Vary by Concurrency Strategy".

Cache Rules Vary by Concurrency Strategy

Table 4-1 lists, for each concurrency strategy:

- How the value of the `max-beans-in-cache` element in `weblogic-ejb-jar.xml` limits the number of entity bean instances in the cache.
- How many entity bean instances with the same primary key are allowed in the cache.

Table 4-1 Entity EJB Caching Behavior by Concurrency Type

Concurrency Option	What is the effect of `max-beans-in-cache` on the number of bean instances in the cache?	How many instances with same primary key can exist in cache simultaneously?
`Exclusive`	`max-beans-in-cache` = number of ACTIVE bean + number of READY instances.	one
`Database`	The cache can contain up to `max-beans-in-cache` ACTIVE bean instances *and* up to max-beans-in-cache READY bean instances.	multiple
`ReadOnly`	`max-beans-in-cache` = number of ACTIVE bean + number of READY instances.	one

Removing Beans from Cache

READY entity EJB instances are removed from the cache when the space is needed for other beans. When a READY instance is removed from cache, `ejbPassivate` is called on the bean, and the container will try to put it back into the free pool.

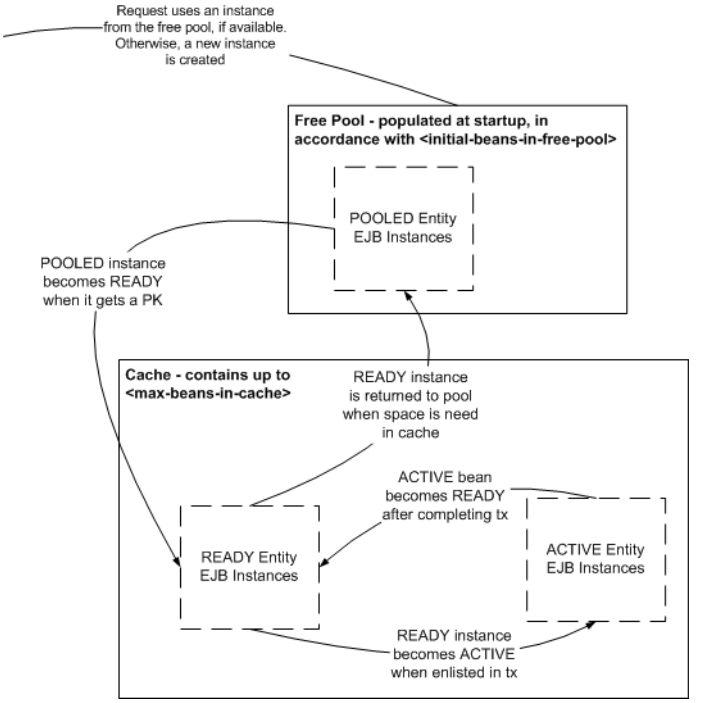
When the container tries to return an instance to the free pool and the pool already contains `max-beans-in-free-pool` instances, the instance is discarded.

ACTIVE entity EJB instances will not be removed from cache until the transaction they are participating in commits or rolls back, at which point they will become READY, and hence eligible for removal from the cache.

Entity EJB Life Cycle Transitions

Figure 4-1 illustrates the Entity EJB free pool and cache, and the transitions that occur throughout an instance's life cycle.

Figure 4-1 Entity Bean Life Cycle

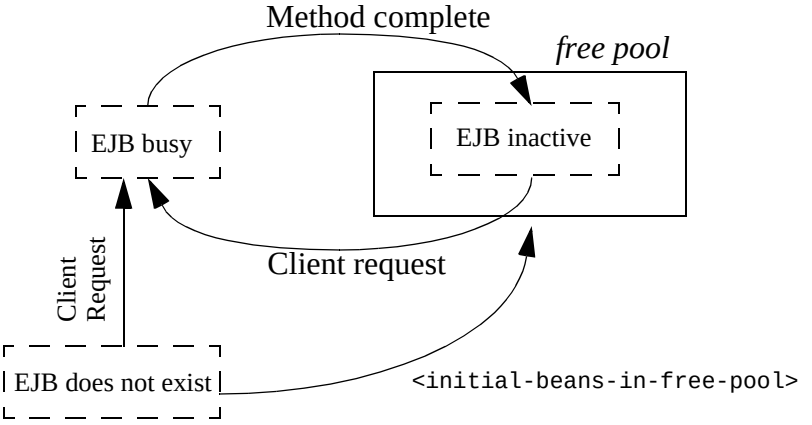


Stateless Session EJB Life Cycle

WebLogic Server uses a free pool to improve performance and throughput for stateless session and message-driven EJBs. The free pool stores unbound stateless session EJBs. Unbound EJBs are instances of a stateless session EJB class that are not processing a method call.

The following figure illustrates the WebLogic Server free pool, and the processes by which stateless EJBs enter and leave the pool. Dotted lines indicate the state of the EJB from the perspective of WebLogic Server.

Figure 4-2 WebLogic Server free pool showing stateless session EJB life cycle



Initializing Stateless Session EJB Instances

By default, no stateless session EJB instances exist in WebLogic Server at startup time. As clients access individual beans, WebLogic Server initializes new instances of the EJB class.

To configure WebLogic Server to populate the free pool with inactive EJB instances EJB at startup, specify the desired quantity in the `initial-beans-in-free-pool` deployment element, in the `stateful-session-descriptor` stanza of `weblogic-ejb-jar.xml`. This can improve initial response time when clients access

EJBs, because initial client requests can be satisfied by activating the bean from the free pool (rather than initializing the bean and then activating it). By default, `initial-beans-in-free-pool` is set to 0.

Note: The maximum size of the free pool is limited by available memory, the number of execute threads, or the value of the `max-beans-in-free-pool` deployment element.

Activating and Pooling Stateless Session EJBs

When a client calls a method on a stateless EJB, WebLogic Server obtains an instance from the free pool, or initializing and activating a new instance, if necessary. The EJB remains active for the duration of the client's method call. After the method completes, the EJB instance is returned to the free pool. Because WebLogic Server unbinds stateless session beans from clients after each method call, the actual bean class instance that a client uses may be different from invocation to invocation.

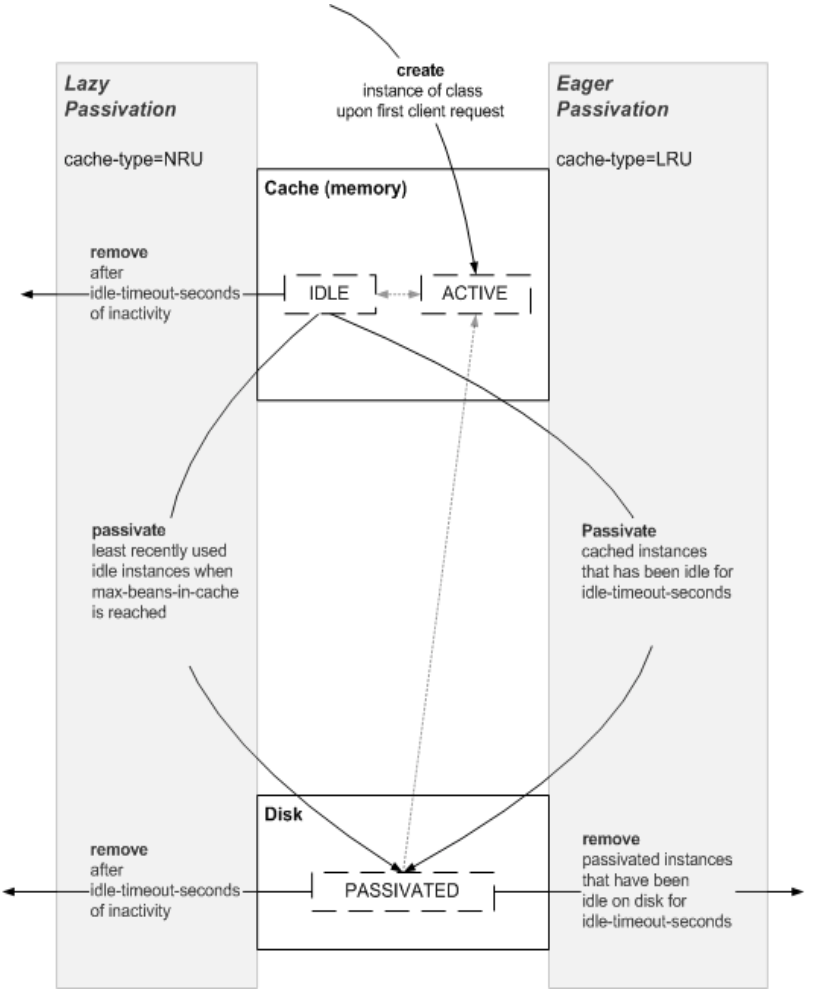
If all instances of an EJB class are active and `max-beans-in-free-pool` has been reached, new clients requesting the EJB class will be blocked until an active EJB completes a method call. If the transaction times out (or, for non-transactional calls, if five minutes elapse), WebLogic Server throws a `RemoteException`.

Stateful Session EJB Life Cycle

WebLogic Server uses a cache of bean instances to improve the performance of stateful session EJBs. The cache stores active EJB instances in memory so that they are immediately available for client requests. The cache contains EJBs that are currently in use by a client and instances that were recently in use. Stateful session beans in cache are bound to a particular client.

The following figure illustrates the WebLogic Server cache, and the processes by which stateful EJBs enter and leave the cache.

Figure 4-3 Stateful Session EJB Life Cycle



Stateful Session EJB Creation

No stateful session EJB instances exist in WebLogic Server at startup. Before a client begins accessing a stateful session bean, it creates a new bean instance to use during its session with the bean. When the session is over the instance is destroyed. While the session is in progress, the instance is cached in memory.

Stateful Session EJB Passivation

Passivation is the process by which WebLogic Server removes an EJB instance from cache while preserving its state on disk. While passivated, EJBs are not in memory and are not immediately available for client requests, as they are when in the cache.

The EJB developer must ensure that a call to the `ejbPassivate()` method leaves a stateful session bean in a condition where WebLogic Server can serialize its data and passivate the bean's instance. During passivation, WebLogic Server attempts to serialize any fields that are not declared `transient`. This means that you must ensure that all non-`transient` fields represent serializable objects, such as the bean's remote or home interface. EJB 2.1 specifies the field types that are allowed.

The rules that govern the passivation of stateful session beans vary, based on the value of the beans `cache-type` element, which can be:

- `LRU`—least recently used, referred to as eager passivation.
- `NRU`—not recently used, referred to as lazy passivation

The `idle-timeout-seconds` and `max-beans-in-cache` elements also affect passivation and removal behaviors, based on the value of `cache-type`.

Eager Passivation (LRU)

When you configure eager passivation for a stateful session bean by setting cache-type to `LRU`, the container:

- Passivates instances to disk:
  - as soon as an instance has been inactive for `idle-timeout-seconds`, regardless of the value of `max-beans-in-cache`.
  - when `max-beans-in-cache` is reached, even though `idle-timeout-seconds` has not expired.
- Removes a passivated instance from disk after it has been inactive for `idle-timeout-seconds` after passivation. This is referred to as a *lazy remove*.

Lazy Passivation (NRU)

When lazy passivation is configured by setting cache-type to `NRU`, the container avoids passivating beans, because of the associated systems overhead—pressure on the cache is the only event that causes passivation or eager removal of beans. The container:

- Removes a bean instance from cache when `idle-timeout-seconds` expires, and does not passivate it to disk. This is referred to as a *eager remove*. An eager remove ensures that an inactive instance does not consume memory or disk resources.
- Passivates instances to disk when `max-beans-in-cache` is reached, even though `idle-timeout-seconds` has not expired.

Managing EJB Cache Size

For a discussion of managing cache size to optimize performance in a production environment see "Setting EJB Cache Size" in *WebLogic Server Performance and Tuning*.

Specifying the Persistent Store Directory for Passivated Beans

When a stateful session bean is passivated, its state is stored in a file system directory known as the *persistent store directory*. The persistent store directory contains one subdirectory for each passivated bean.

The persistent store directory is created by default in the root directory of your WebLogic Server installation, for example:

```
D:\releases\610\pstore\
```

The path to the persistence store is:

`WLHOME``\persistent-store-dir`

where:

- *`WLHOME`*—the directory where WebLogic Server is installed, for example:

  ```
  D:\releases\610\
  ```

- *`persistent-store-dir`*—the value of the of the `persistent-store-dir` element in the `<stateful-session-descriptor>` stanza of `weblogic-ejb-jar.xml`. If no value is specified for `persistent-store-dir`, the directory is named `pstore` by default.

The persistent store directory contains a subdirectory for each passivated bean. The subdirectory name is comprised of the bean's home and JNDI name. For example:

```
D:\releases\610\pstore\statefulSessionful.TraderHome
```

Concurrent Access to Stateful Session Beans

In accordance with the EJB 2.0 specification, simultaneous access to a stateful session EJB results in a `RemoteException`. This access restriction on stateful session EJBs applies whether the EJB client is remote or internal to WebLogic Server. To override this restriction and configure a stateful session bean to allow concurrent calls, set the `allow-concurrent-calls` deployment element.

If multiple servlet classes access a stateful session EJB, each servlet thread (rather than each instance of the servlet class) must have its own session EJB instance. To prevent concurrent access, a JSP/servlet can use a stateful session bean in request scope.

Comparing the Performance of Stateless Session Beans to BMP EJBs

To improve performance, we recommend that you use stateless session beans or CMP (container-managed persistent) entity beans instead of BMP (bean-managed persistent) entity beans for retrieving data. Tests have shown that because BMP entity beans can not cache data during a finder query, the performance of stateless session beans may be as much as 90 percent greater than BMP entity beans. For example, a BMP entity bean that returns 100 beans from a finder query does one JDBC call to create bean references when the finder query is run and then one JDBC call per bean, to load each bean as each is accesses by the client. This means that the finder query for the BMP entity bean does a total of 101 calls to the database. By comparison, the stateless session bean does just one JDBC call, so it's performance is much faster.

The fact that BMP entity beans do not scale very well is a known performance limitation.

In addition, CMP entity beans can cache data during a finder query. So as with the stateless session bean, only one query is performed.

ejbLoad() and ejbStore() Behavior for Entity EJBs

WebLogic Server reads and writes the persistent fields of entity EJBs using calls to `ejbLoad()` and `ejbStore()`. By default, WebLogic Server calls `ejbLoad()` and `ejbStore()` in the following manner:

1. A transaction is initiated for the entity EJB. The client may explicitly initiate a new transaction and invoke the bean, or WebLogic Server may initiate a new transaction in accordance with the bean's method transaction attributes.
2. WebLogic Server calls `ejbLoad()` to read the most current version of the bean's persistent data from the underlying datastore.
3. When the transaction commits, WebLogic Server calls `ejbStore()` to write persistent fields back to the underlying datastore.

This simple process of calling `ejbLoad()` and `ejbStore()` ensures that new transactions always use the latest version of the EJB's persistent data, and always write the data back to the datastore upon committing. In certain circumstances, however, you may want to limit calls to `ejbLoad()` and `ejbStore()` for performance reasons. Alternately, you may want to call `ejbStore()` more frequently to view the intermediate results of uncommitted transactions.

WebLogic Server provides several deployment descriptor elements in the `weblogic-ejb-jar.xml` and `weblogic cmp-rdbms-jar.xml` files that enable you to configure `ejbLoad()` and `ejbStore()` behavior.

Using db-is-shared to Limit Calls to ejbLoad()

WebLogic Server's default behavior of calling `ejbLoad()` at the start of each transaction works well for environments where multiple sources may update the datastore. Because multiple clients (including WebLogic Server) may be modifying an EJB's underlying data, an initial call to `ejbLoad()` notifies the bean that it needs to refresh its cached data and ensures that it works against the most current version of the data.

In the special circumstance where only a single WebLogic Server instance ever accesses a particular EJB, calling `ejbLoad()` by default is unnecessary. Because no other clients or systems update the EJB's underlying data, WebLogic Server's cached version of the EJB data is always up-to-date. Calling `ejbLoad()` in this case simply creates extra overhead for WebLogic Server clients that access the bean.

To avoid unnecessary calls to `ejbLoad()` in the case of a single WebLogic Server instance accessing a particular EJB, WebLogic Server provides the `db-is-shared` deployment parameter. By default, `db-is-shared` is set to "true" for each EJB in the bean's `weblogic-ejb-jar.xml` file, which ensures that `ejbLoad()` is called at the start of each transaction. Where only a single WebLogic Server instance ever accesses an EJB's underlying data, you can set `db-is-shared` to "false" in the bean's `weblogic-ejb-jar.xml` file if the concurrency option is set to Exclusive. When you deploy an EJB with `db-is-shared` set to "false," the single instance of WebLogic Server calls `ejbLoad()` for the bean if:

- The EJB does not exist in the cache
- The EJB's transaction rolls back

Restrictions and Warnings for db-is-shared

Setting `db-is-shared` to "false" overrides WebLogic Server's default `ejbLoad()` container-managed-persistence behavior, regardless of whether the EJB's underlying data is updated by one WebLogic Server instance or multiple clients. If you incorrectly set `db-is-shared` to "false" and multiple clients (database clients, other WebLogic Server instances, and so forth) update the bean data, you run the risk of losing data integrity.

Do not set `db-is-shared` to "false" if you set the entity bean's `concurrency strategy` to the "Database" option. If you do, WebLogic Server will ignore the `db-is-shared` setting.

With database locking specified, the EJB container continues to cache instances of entity bean classes. However, the container does not cache the intermediate state of the EJB instance between transactions. Instead, WebLogic Server calls `ejbLoad()` for each instance at the beginning of a transaction to obtain the latest EJB data. This means that setting `db-is-shared` to "false" which prevents WebLogic Server from calling `ejbload()` at the beginning of each transaction is invalid.

Also, due to caching limitations, you cannot set `db-is-shared` to "false" in a WebLogic Server cluster.

Using is-modified-method-name to Limit Calls to ejbStore() (EJB 1.1 Only)

This method is no longer required.

Note: The `is-modified-method-name` deployment descriptor element applies to EJB 1.1 container-managed-persistence (CMP) beans only. This element is found in the `weblogic-ejb-jar.xml` file. However, it is not longer required for EJB 2.0. WebLogic Server CMP implementation automatically detects modifications of CMP fields and writes only those changes to the underlying datastore. We recommend that you do not use `is-modified-method-name` with bean-managed-persistence (BMP) because you would need to create both the `is-modified-method-name` element. and the `ejbstore` method.

By default, WebLogic Server calls the `ejbStore()` method at the successful completion (commit) of each transaction. `ejbStore()` is called at commit time regardless of whether the EJB's persistent fields were actually updated. WebLogic Server provides the `is-modified-method-name` element for cases where unnecessary calls to `ejbStore()` may result in poor performance.

To use `is-modified-method-name`, EJB providers must first develop an EJB method that "cues" WebLogic Server when persistent data has been updated. The method must return "false" to indicate that no EJB fields were updated, or "true" to indicate that some fields were modified.

The EJB provider or EJB deployment descriptors then identify the name of this method by using the value of the `is-modified-method-name` element. WebLogic Server calls the specified method name when a transaction commits, and calls `ejbStore()` only if the method returns "true." For more information on this element, see "is-modified-method-name" on page 10-37.

Warning for is-modified-method-name

Using the `is-modified-method-name` element can improve performance by avoiding unnecessary calls to `ejbStore()`. However, it places a greater burden on the EJB developer to identify correctly when updates have occurred. If the specified `is-modified-method-name` returns an incorrect flag to WebLogic Server, data integrity problems can occur, and they may be difficult to track down.

If entity EJB updates appear "lost" in your system, start by ensuring that the value for all `is-modified-method-name` elements return "true" under every circumstance. In this way, you can revert to WebLogic Server's default `ejbStore()` behavior and possibly correct the problem.

Using delay-updates-until-end-of-tx to Change ejbStore() Behavior

By default, WebLogic Server updates the persistent store of all beans in a transaction only at the completion (commit) of the transaction. This generally improves performance by avoiding unnecessary updates and repeated calls to `ejbStore()`.

If your datastore uses an isolation level of `READ_UNCOMMITTED`, you may want to allow other database users to view the intermediate results of in-progress transactions. In this case, the default WebLogic Server behavior of updating the datastore only at transaction completion may be unacceptable. To do this, set `delay-updates-until-end-of-tx` to "false."

You can disable the default behavior by using the `delay-updates-until-end-of-tx` deployment descriptor element. This element is set in the `weblogic-ejb-jar.xml` file. When you set this element to "false," WebLogic Server calls `ejbStore()` after each method call, rather than at the conclusion of the transaction.

Note: Setting `delay-updates-until-end-of-tx` to false does not cause database updates to be "committed" to the database after each method invoke; they are only sent to the database. Updates are committed or rolled back in the database only at the conclusion of the transaction.

Setting Entity EJBs to Read-Only

Entity EJBs can also use the `read-only` concurrency strategy to modify basic `ejbLoad()` and `ejbStore()` behavior. The following sections describe how the EJB container supports the concurrency service.

You specify the read-only cache strategy by editing the `concurrency-strategy` element in the `weblogic-ejb-jar.xml` deployment file. For instructions on how to edit the deployment descriptors, see "Specifying and Editing the EJB Deployment Descriptors" on page 6-5.

Read-Only Concurrency Strategy

You can use the `read-only` concurrency strategy for entity EJBs that are never modified by an EJB client, but they can be updated periodically by an external source. For example, a `read-only` entity EJB can represent a stock quote for a particular company; the quote is updated externally to the WebLogic Server system.

WebLogic Server never calls `ejbStore()` for a `read-only` entity EJB. `ejbLoad()` is called initially when the EJB is created; afterwards, WebLogic Server calls `ejbLoad()` only at intervals defined by the `read-timeout-seconds` deployment parameter.

Restrictions for Read-Only Concurrency Strategy

Entity EJBs using the `read-only` concurrency strategy must observe the following restrictions:

- They cannot require updates to the EJB data, because WebLogic Server never calls `ejbStore()` for read-only entity EJBs.
- Their transaction attributes must be set to `NotSupported` (the beans cannot rely on a transaction).
- The EJB's method calls must be idempotent. See "Session EJBs in a Cluster" on page 4-23 for more information.

- Because the bean's underlying data may be updated by an external source, calls to `ejbLoad()` are governed by the deployment parameter, `read-timeout-seconds`.

Read-Only Multicast Invalidation

Read-only multicast invalidation is an efficient means of to invalidating cached data.

Invalidate a read-only entity bean by calling the following `invalidate()` method on either the `CachingHome` or `CachingLocalHome` interface:

Figure 4-4 Sample code showing CachingHome and CachingLocalHome interfaces

```
package weblogic.ejb;

public interface CachingHome {

    public void invalidate(Object pk) throws RemoteException;
    public void invalidate (Collection pks) throws RemoteException;
    public void invalidateAll() throws RemoteException;

public interface CachingLocalHome {

    public void invalidate(Object pk) throws RemoteException;
    public void invalidate (Collection pks) throws RemoteException;
    public void invalidateAll() throws RemoteException

}
```

The following example codes shows how to cast the home to CachingHome and then call the method:

Figure 4-5 Sample code showing how to cast the home and call the method

```
import javax.naming.InitialContext;
import weblogic.ejb.CachingHome;

Context initial = new InitialContext();
Object o = initial.lookup("CustomerEJB_CustomerHome");
CustomerHome customerHome = (CustomerHome)o;

CachingHome customerCaching = (CachingHome)customerHome;
customerCaching.invalidateAll();
```

When the `invalidate()` method is called, the read-only entity beans are invalidated in the local server, and a multicast message is sent to the other servers in the cluster to invalidate their cached copies. The next call to an invalidated read-only entity bean causes `ejbLoad` to be called. `ejbLoad()` reads the most current version of the persistent data from the underlying datastore

WebLogic Server calls the `invalidate()` method after the transaction update has completed. If the invalidation occurs during a transaction update, the previous version may be read if the isolation level does not permit reading uncommitted data.

Standard Read-Only Entity Beans

WebLogic Server continues to support the standard read-only entity beans with the `read-timeout` element set in the deployment descriptor. If the `ReadOnly` option is selected in the `concurrency strategy` element and the `read-timeout-seconds` element is set in the `weblogic-ejb-jar.xml` file, when a read-only bean is invoked, WebLogic Server checks whether the cached data is older than the `read-timeout` setting. If it is, the bean's `ejbLoad` is called. Otherwise, the cached data is used. So, previous versions of read-only entity beans will work in this version of WebLogic Server.

Read-Mostly Pattern

WebLogic Server does not support a read-mostly cache strategy setting in `weblogic-ejb-jar.xml`. However, if you have EJB data that is only occasionally updated, you can create a "read-mostly pattern" by implementing a combination of `read-only` and `read-write` EJBs.

For an example of the read-mostly pattern, see the Read Mostly example in your WebLogic Server distribution:

```
wlserver6.1\samples\examples\ejb\extensions\readMostly
```

WebLogic Server provides an automatic `invalidate()` method for the Read-Mostly pattern. With this pattern, Read-Only entity bean and a Read-Write entity bean are mapped to the same data. To read the data, you use the Read-Only entity bean; to update the data, you use the Read-Write entity bean.

In a read-mostly pattern, a `read-only` entity EJB retrieves bean data at intervals specified by the `read-timeout-seconds` deployment descriptor element specified in the `weblogic-ejb-jar.xml` file. A separate `read-write` entity EJB models the same data as the `read-only` EJB, and updates the data at required intervals.

When creating a read-mostly pattern, use the following suggestions to reduce data consistency problems:

- For all `read-only` EJBs, set `read-timeout-seconds` to the same value for all beans that may be updated in the same transaction.
- For all `read-only` EJBs, set `read-timeout-seconds` to the smallest timeframe that yields acceptable performance levels.
- Ensure that all `read-write` EJBs in the system update only the smallest portion of data necessary; avoid beans that write numerous, unchanged fields to the datastore at each `ejbStore()`.
- Ensure that all `read-write` EJBs update their data in a timely fashion; avoid involving `read-write` beans in long-running transactions that may span the `read-timeout-seconds` setting for their `read-only` counterparts.

Note: In a WebLogic Server cluster, clients of the `read-only` EJB benefit from using cached EJB data. Clients of the `read-write` EJB benefit from true transactional behavior, because the `read-write` EJB's state always matches the state of its data in the underlying datastore. See "Entity EJBs in a Cluster" on page 4-26 for more information.

Read-Write Cache Strategy

The `read-write` strategy defines the default caching behavior for entity EJBs in WebLogic Server.

For `read-write` EJBs, WebLogic Server loads EJB data into the cache at the beginning of each transaction, or as described in "Using db-is-shared to Limit Calls to ejbLoad()" on page 4-12. WebLogic Server calls `ejbStore()` at the successful commit of a transaction, or as described under "Using is-modified-method-name to Limit Calls to ejbStore() (EJB 1.1 Only)" on page 4-14.

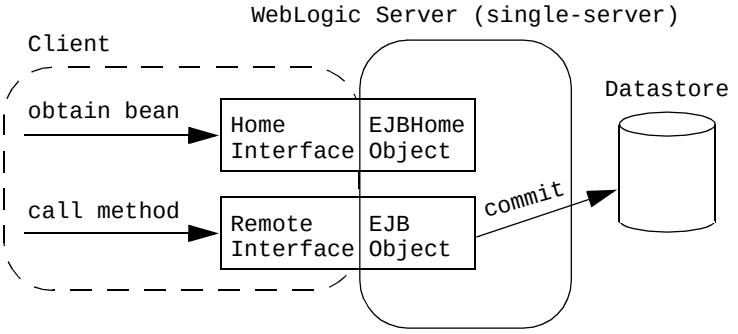
EJBs in WebLogic Server Clusters

This section providers information on how the EJB container supports clustering services. It describes the behavior of EJBs and their associated transactions in a WebLogic Server cluster, and explains key deployment descriptors that affect EJB behavior in a cluster.

EJBs in a WebLogic Server cluster use modified versions of two key structures: the Home object and the EJB object. In a single server (unclustered) environment, a client looks up an EJB through the EJB's home interface, which is backed on the server by a corresponding Home object. After referencing the bean, the client interacts with the bean's methods through the remote interface, which is backed on the server by an EJB object.

The following figure shows EJB behavior in a single server environment.

Figure 4-6 Single server behavior



Note: Failover of EJBs work only between a *remote* client and the EJB.

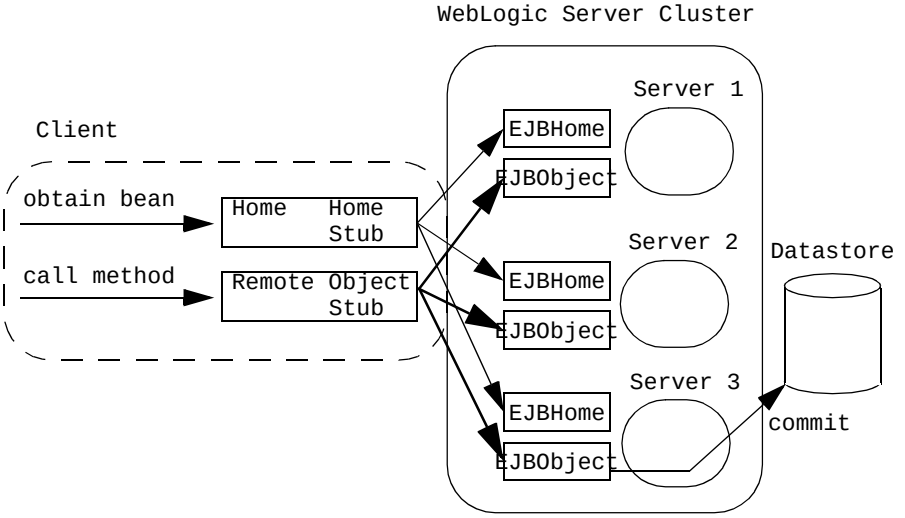
Clustered EJBHome Objects

In a WebLogic Server cluster, the server-side representation of the Home object can be replaced by a cluster-aware "stub." The cluster-aware home stub has knowledge of EJB Home objects on all WebLogic Servers in the cluster. The clustered home stub provides load balancing by distributing EJB lookup requests to available servers. It can also provide failover support for lookup requests, because it routes those requests to available servers when other servers have failed.

All EJB types — stateless session, stateful session, and entity EJBs — can have cluster-aware home stubs. Whether or not a cluster-aware home stub is created is determined by the `home-is-clusterable` deployment element in `weblogic-ejb-jar.xml`. If you set this element to "true" (the default), `ejbc` calls the `rmic` compiler with the appropriate options to generate a cluster-aware home stub for the EJB.

The following figure shows EJB behavior in a WebLogic Server clustered environment. For an illustration of EJBs in a clustered server environment, see Figure 4-7.

Figure 4-7 Clustered server environment



Clustered EJBObjects

In a WebLogic Server cluster, the server-side representation of the EJBObject can also be replaced by a replica-aware EJBObject stub. This stub maintains knowledge about all copies of the EJBObject that reside on servers in the cluster. The EJBObject stub can provide load balancing and failover services for EJB method calls. For example, if a client invokes an EJB method call on a particular WebLogic Server and the server goes down, the EJBObject stub can failover the method call to another, running server.

Whether or not an EJB can use a replica-aware EJBObject stub depends on the type of EJB deployed and, for entity EJBs, the cache strategy selected at deployment time.

Session EJBs in a Cluster

This section describes cluster capabilities and limitations for stateful and stateless session EJBs.

Stateless Session EJBs

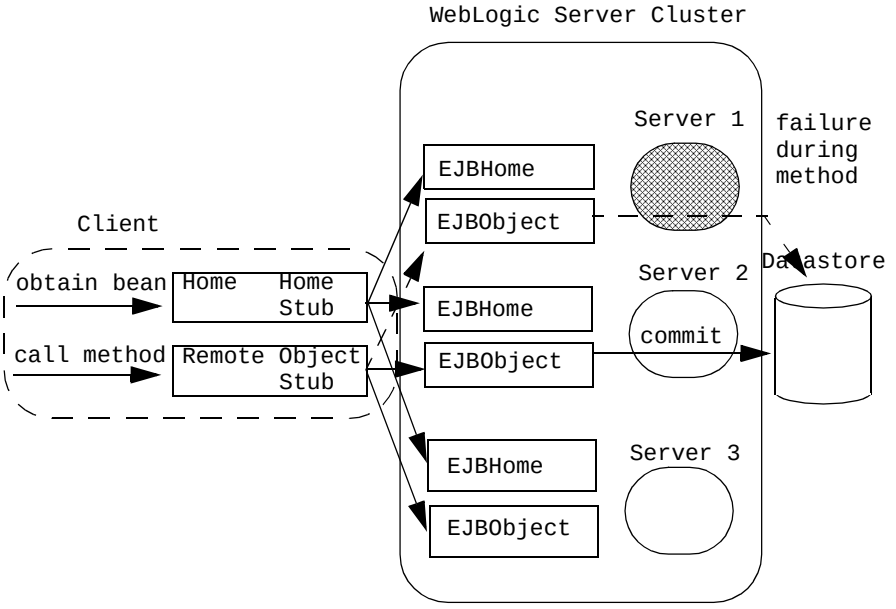
Stateless session EJBs can have both a cluster-aware home stub and a replica-aware EJBObject stub. By default, WebLogic Server provides failover services for EJB method calls, but only if a failure occurs *between* method calls. For example, failover is automatically supported if a failure occurs after a method completes, or if the method fails to connect to a server. When failures occur while an EJB method is in progress, WebLogic Server does not automatically fail over from one server to another.

This default behavior ensures that database updates within an EJB method are not "duplicated" due to a failover scenario. For example, if a client calls a method that increments a value in a datastore and WebLogic Server fails over to another server before the method completes, the datastore would be updated twice for the client's single method call.

If methods are written in such a way that repeated calls to the same method do not cause duplicate updates, the method is said to be "idempotent." For idempotent methods, WebLogic Server provides the `stateless-bean-methods-are-idempotent` deployment property. If you set this property to "true" in `weblogic-ejb-jar.xml`, WebLogic Server assumes that the method is idempotent and *will* provide failover services for the EJB method, even if a failure occurs during a method call.

The following figure show a stateless session EJBs in a WebLogic Server clustered environment.

Figure 4-8 Stateless session EJBs in a clustered server environment



Stateful Session EJBs

To enable stateful session EJBs to use cluster-aware home stubs, set `home-is-clusterable` to "true." This provides failover and load balancing for stateful EJB lookups. Stateful session EJBs configured this way use replica-aware EJBObject stubs. For more information on in-memory replication for stateful session EJBs, see "In-Memory Replication for Stateful Session EJBs" on page 4-25.

In-Memory Replication for Stateful Session EJBs

The following sections describe how the EJB Container supports replication services. The WebLogic Server EJB container supports clustering for stateful session EJBs. Whereas in WebLogic Server 5.1 only the `EJBHome` object is clustered for stateful session EJBs, the EJB container can also replicate the state of the EJB across clustered WebLogic Server instances.

Replication support for stateful session EJBs is transparent to clients of the EJB. When a stateful session EJB is deployed, WebLogic Server creates a cluster-aware `EJBHome` stub and a replica-aware `EJBObject` stub for the stateful session EJB. The `EJBObject` stub maintains a list of the primary WebLogic Server instances on which the EJB instance runs, as well as the name of a secondary WebLogic Server to use for replicating the bean's state.

Each time a client of the EJB commits a transaction that modifies the EJB's state, WebLogic Server replicates the bean's state to the secondary server instance. Replication of the bean's state occurs directly in memory, for best performance in a clustered environment.

Should the primary server instance fail, the client's next method invocation is automatically transferred to the EJB instance on the secondary server. The secondary server becomes the primary WebLogic Server for the EJB instance, and a new secondary server handles possible additional failovers. Should the EJB's secondary server fail, WebLogic Server enlists a new secondary server instance from the cluster.

Clients of a stateful session EJB are therefore guaranteed to have quick access to the latest committed state of the EJB, except under the special circumstances described in "Limitations of In-Memory Replication" on page 4-26.

Requirements and Configuration for In-Memory Replication

To replicate the state of a stateful session EJB in a WebLogic Server cluster, make sure that the cluster is homogeneous for the EJB class. In other words, deploy the same EJB class to every WebLogic Server instance in the cluster, using the same deployment descriptor. In-memory replication is not supported for heterogeneous clusters.

By default, WebLogic Server does not replicate the state of stateful session EJB instances in a cluster. This models the behavior released with WebLogic Server Version 6.0. To enable replication, set the `replication-type` deployment parameter in the `weblogic-ejb-jar.xml` deployment file to `InMemory`.

Figure 4-9 XML sample enabling replication

```
<stateful-session-clustering>

       ...

       <replication-type>InMemory</replication-type>

</stateful-session-clustering>
```

Limitations of In-Memory Replication

By replicating the state of a stateful session EJB, clients are generally guaranteed to have the last committed state of the EJB, even if the primary WebLogic Server instance fails. However, in the following rare failover scenarios, the last committed state may not be available:

- A client commits a transaction involving a stateful EJB, but the primary WebLogic Server fails before the EJB's state is replicated. In this case, the client's next method invocation works against the previous committed state.
- A client creates an instance of a stateful session EJB and commits an initial transaction, but the primary WebLogic Server fails before the EJB's initial state can be replicated. The client's next method invocation fails to locate the bean instance, because the initial state could not be replicated. The client needs to recreate the EJB instance, using the clustered `EJBHome` stub, and restart the transaction.
- Both the primary and secondary servers fail. The client needs to recreate the EJB instance and restart the transaction.

Entity EJBs in a Cluster

As with all EJB types, entity EJBs can utilize cluster-aware home stubs once you set `home-is-clusterable` to "true." The behavior of the EJBObject stub depends on the `cache-strategy` deployment element in `weblogic-ejb-jar.xml`.

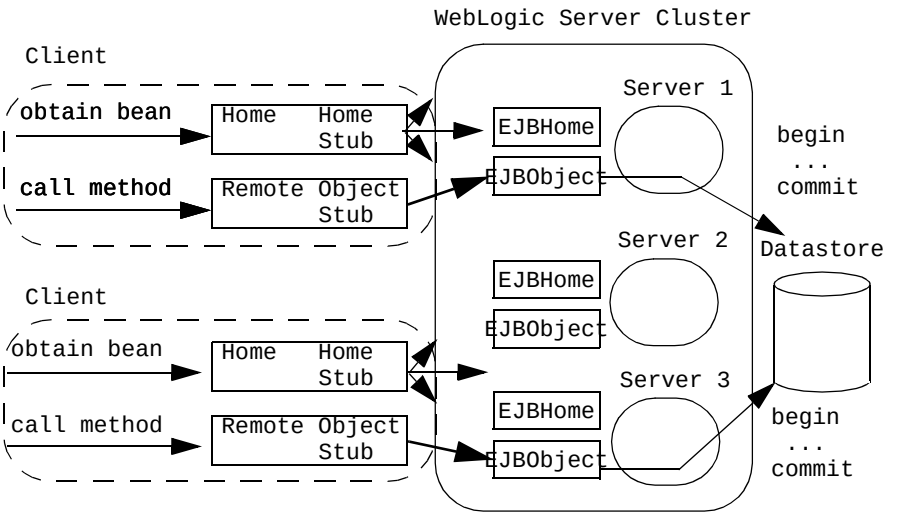
Read-Write Entity EJBs in a Cluster

`read-write` entity EJBs in a cluster behave similarly to entity EJBs in a non-clustered system, in that:

- Multiple clients can use the bean in transactions.
- `ejbLoad()` is always called at the beginning of each transaction (because the `db-is-shared` deployment parameter cannot be set to "false" in a cluster).
- `ejbStore()` behavior is governed by the rules described in "ejbLoad() and ejbStore() Behavior for Entity EJBs" on page 4-12.

Figure 4-10 shows read-write entity EJBs in a WebLogic Server clustered environment. The three arrows on Home Stub point to all three servers and show multiple client access.

Figure 4-10 Read-write entity EJBs in a clustered server environment



Note: In the preceding figure, the set of three arrows for both home stubs refers to the EJBHome on each server.

`read-write` entity EJBs support automatic failover on a safe exception, if `home-is-clusterable` is set to true. For example, failover is automatically supported if there is a failure after a method completes, or if the method fails to connect to a server.

Cluster Address

When you configure a cluster, you supply a cluster address that identifies the Managed Servers in the cluster. The cluster address is used in entity and stateless beans to construct the host name portion of URLs. If the cluster address is not set, EJB handles may not work properly. For more information on cluster addresses, see Using WebLogic Server Clusters.

Transaction Management

The following sections provide information on how the EJB container supports transaction management services. They describe EJBs in several transaction scenarios. EJBs that engage in distributed transactions (transactions that make updates in multiple datastores) guarantee that all branches of the transaction commit or roll back as a logical unit.

The current version of WebLogic Server supports Java Transaction API (JTA), which you can use to implement distributed transactional applications.

Also, two-phase commit is supported for both 1.1 and 2.0 EJBs. The **two-phase commit protocol** is a method of coordinating a single transaction across two or more resource managers. It guarantees data integrity by ensuring that transactional updates are committed in all participating databases, or are fully rolled back out of all the databases, reverting to the state prior to the start of the transaction. For more information on using transactions and the two-phase commit protocol, see Introducing Transactions.

Transaction Management Responsibilities

Session EJBs can rely on their own code, their client's code, or the WebLogic Server container to define transaction boundaries. EJBs can use container- or client-demarcated transaction boundaries, but they cannot define their own transaction boundaries unless they observe certain restrictions.

- In **bean-managed transactions**, the EJB manages the transaction demarcation. If bean- or client-managed transactions are required, you must provide the java code and use the `javax.transaction.UserTransaction` interface. The EJB or client can then access a `UserTransaction` object through JNDI and specify transaction boundaries with explicit calls to `tx.begin()`, `tx.commit()`, `tx.rollback()`. See "Using javax.transaction.UserTransaction" on page 4-29 for more information on defining transaction boundaries.
- In **container-managed transactions**, the WebLogic Server EJB container manages the transaction demarcation. For EJBs that use container-managed transactions (or EJBs that mix container and bean-managed transactions) you can use several deployment elements to control the transactional requirements for individual EJB methods. For more information about the deployment descriptors, see *Programming WebLogic EJB*.

Note: If the EJB provider does not specify a transaction attribute for a method in the `ejb-jar.xml` file, WebLogic Server uses the *`supports`* attribute by default.

The sequence of transaction events differs between container-managed and bean-managed transactions.

Using javax.transaction.UserTransaction

To define transaction boundaries in EJB or client code, you must obtain a `UserTransaction` object and begin a transaction *before* you obtain a Java Transaction Service (JTS) or JDBC database connection. If you start a transaction after obtaining a database connection, the connection has no relationship to the new transaction, and there are no semantics to "enlist" the connection in a subsequent transaction context. If a JTS connection is not associated with a transaction context, it operates similarly to a standard JDBC connection that has autocommit equal to `true`, and updates are automatically committed to the datastore.

Once you create a database connection within a transaction context, that connection becomes "reserved" until the transaction either commits or rolls back. To maintain performance and throughput for your applications, always ensure that your transaction completes quickly, so that the database connection can be released and made available to other client requests. See "Preserving Transaction Resources" on page 2-8 for more information.

Note: You can associate only a single database connection with an active transaction context.

Restriction for Container-Managed EJBs

You cannot use the `javax.transaction.UserTransaction` method within an EJB that uses container-managed transactions.

Transaction Isolation Levels

The method for setting the transaction isolation level differs according to whether your application uses bean-managed or container-managed transaction demarcation. The following sections examine each of these scenarios.

Setting User Transaction Isolation Levels

You set the isolation level for user transactions in the beans java code. When the application runs, the transaction is explicitly started. See Figure 4-11 for a code sample of how to set the level.

Figure 4-11 Sample Java Code setting user transaction isolation levels

```
import javax.transaction.Transaction;
import java.sql.Connection
import weblogic.transaction.TxHelper:
import weblogic.transaction.Transaction;
import weblogic.transaction.TxConstants;

User Transaction tx = (UserTransaction)

ctx.lookup("javax.transaction.UserTransaction");

//Begin user transaction
```

```
        tx.begin();

//Set transaction isolation level to TRANSACTION_READ_COMMITED

Transaction tx = TxHelper.getTransaction();
        tx.setProperty (TxConstants.ISOLATION_LEVEL, new Integer
         (Connection.TRANSACTION_READ_COMMITED));

//perform transaction work

        tx.commit();
```

Setting Container-Managed Transaction Isolation Levels

You set the isolation level for container-managed transactions in the `transaction-isolation` element of the `weblogic-ejb-jar.xml` deployment file. WebLogic Server passes this value to the underlying database. The behavior of the transaction depends both on the EJB's isolation level setting and the concurrency control of the underlying persistent store. For more information on setting container-managed transaction isolation levels, see Programming WebLogic JTA.

Limitations of TRANSACTION_SERIALIZABLE

Many datastores provide limited support for detecting serialization problems, even for a single user connection. Therefore, even if you set `transaction-isolation` to `TRANSACTION_SERIALIZABLE`, you may experience serialization problems due to the limitations of the datastore.

Refer to your RDBMS documentation for more details about isolation level support.

Special Note for Oracle Databases

Oracle uses optimistic concurrency. As a consequence, even with a setting of `TRANSACTION_SERIALIZABLE`, Oracle does not detect serialization problems until commit time. The message returned is:

```
java.sql.SQLException: ORA-08177: can't serialize access for this
transaction
```

Even if you use the TRANSACTION_SERIALIZABLE setting for an EJB, you may receive exceptions or rollbacks in the EJB client if contention occurs between clients for the same rows. To avoid these problems, make sure that the code in your client application catches and examines the SQL exceptions, and take you take the appropriate action to resolve the exceptions, such as restarting the transaction.

With WebLogic Server, you can set the isolation level for transactions to TRANSACTION_READ_COMMITTED_FOR_UPDATE for methods on which this option is defined. When set, every SELECT query from that point on will have FOR_UPDATE added to require locks on the selected data. This condition remains in effect until the transaction does a COMMIT or ROLLBACK.

Distributing Transactions Across Multiple EJBs

WebLogic Server does support transactions that are distributed over multiple datasources; a single database transaction can span multiple EJBs on multiple servers. You can explicitly enable support for these types of transactions by starting a transaction and invoking several EJBs. Or, a single EJB can invoke other EJBs that implicitly work within the same transaction context. The following sections describe these scenarios.

Calling Multiple EJBs from a Single Transaction Context

In the following code fragment, a client application obtains a UserTransaction object and uses it to begin and commit a transaction. The client invokes two EJBs within the context of the transaction. The transaction attribute for each EJB is set to Required:

Figure 4-12 Beginning and committing a transaction

```
import javax.transaction.*;

...

u = (UserTransaction)
jndiContext.lookup("javax.transaction.UserTransaction");

u.begin();

account1.withdraw(100);

account2.deposit(100);
```

```
u.commit();

...
```

In the above code fragment, updates performed by the "account1" and "account2" EJBs occur within the context of a single `UserTransaction`. The EJBs commit or roll back as a logical unit. This is true regardless of whether "account1" and "account2" reside on the same WebLogic Server, multiple WebLogic Servers, or a WebLogic Server cluster.

The only requirement for wrapping EJB calls in this manner is that both "account1" and "account2" must support the client transaction. The beans' `trans-attribute` element must be set to `Required`, `Supports`, or `Mandatory`.

Encapsulating a Multi-Operation Transaction

You can also use a "wrapper" EJB that encapsulates a transaction. The client calls the wrapper EJB to perform an action such as a bank transfer. The wrapper EJB responds by starting a new transaction and invoking one or more EJBs to do the work of the transaction.

The "wrapper" EJB can explicitly obtain a transaction context before invoking other EJBs, or WebLogic Server can automatically create a new transaction context, if the EJB's `trans-attribute` element is set to `Required` or `RequiresNew`. The `trans-attribute` element is set in the `ejb-jar.xml` file. All EJBs invoked by the wrapper EJB must be able to support the transaction context (their `trans-attribute` elements must be set to `Required`, `Supports`, or `Mandatory`).

Distributing Transactions Across EJBs in a WebLogic Server Cluster

WebLogic Server provides additional transaction performance benefits for EJBs that reside in a WebLogic Server cluster. When a single transaction utilizes multiple EJBs, WebLogic Server attempts to use EJB instances from a single WebLogic Server instance, rather than using EJBs from different servers. This approach minimizes network traffic for the transaction.

In some cases, a transaction can use EJBs that reside on multiple WebLogic Server instances in a cluster. This can occur in heterogeneous clusters, where all EJBs have not been deployed to all WebLogic Server instances. In these cases, WebLogic Server uses a multitier connection to access the datastore, rather than multiple direct connections. This approach uses fewer resources, and yields better performance for the transaction.

However, for best performance, the cluster should be homogeneous — all EJBs should reside on all available WebLogic Server instances.

Delay-Database-Insert-Until

By default, the database insert occurs after the client calls the `ejbPostCreate` method. To delay having WebLogic Server insert the new bean, use the `delay-database-insert-until` element in the `weblogic-cmp-rdbms-jar.xml` file to specify the precise time at which a new bean that uses RDBMS CMP is inserted into the database.

Delaying the database insert until after `ejbPostCreate` is required when a `cmr-field` is mapped to a `foreign-key column` that does not allow null values. In this case, set the `cmr-field` to a non-null value in `ejbPostCreate` before the bean is inserted into the database.

Note: The `cmr-fields` may not be set during a `ejbCreate` method call, before the primary key of the bean is known.

BEA recommend that you specify the delay the database insert until after `ejbPostCreate` if the `ejbPostCreate` method modifies the persistent fields of the bean. Doing so yields better performance by avoiding an unnecessary store operation.

For maximum flexibility, avoid creating related beans in their `ejbPostCreate` method. The creation of these additional instances may make delaying the database insert impossible if database constraints prevent related beans from referring to a bean that has not yet been created.

The allowed values for the `delay-database-insert-until` element are:

- `ejbCreate` - This method performs a database insert immediately after `ejbCreate`.
- `ejbPostCreate` - This method performs an insert immediately after `ejbPostCreate` (default).

Figure 4-13 Sample xml specifying the delay-database-insert until element

```
<delay-database-insert-until>ejbPostCreate</delay-database-insert
-until> -->
```

Resource Factories

The following sections provide information on how the EJB container supports resource services. In WebLogic Server, EJBs can access JDBC connection pools by directly instantiating a JDBC pool driver. However, it is recommended that you instead bind a JDBC datasource resource into the WebLogic Server JNDI tree as a resource factory.

Using resource factories enables the EJB to map a resource factory reference in the EJB deployment descriptor to an available resource factory in a running WebLogic Server. Although the resource factory reference must define the type of resource factory to use, the actual name of the resource is not specified until the bean is deployed.

The following sections explain how to bind JDBC datasource and URL resources to JNDI names in WebLogic Server.

Note: WebLogic Server also supports JMS connection factories.

Setting Up JDBC Datasource Factories

Follow these steps to bind a `javax.sql.DataSource` resource factory to a JNDI name in WebLogic Server. Note that you can set up either a transactional or non-transactional JDBC datasource as necessary:

1. Set up a JDBC connection pool in the Administration Console. See Managing JDBC Connectivity in the Administration Guide for more information.
2. Start WebLogic Server.
3. Start WebLogic Server Administration Console.
4. In the Console, click the Services node and expand JDBC.
5. Select Data Sources and choose the Configure a new JDBC Data Source option.
6. Enter the Name, JNDI Name, and Pool Name. Check to enable Row Prefetch if you if you want to prefetch rows between client and WebLogic Server for each resultSet and then specify the Row Prefetch Size and Stream Chunk Size.

a. *For non-transactional JDBC data sources*, enter the full WebLogic Server JNDI name to bind to the datasource and the name of the WebLogic Server connection pool.

b. *For transactional JDBC data sources*, enter the full WebLogic Server JNDI name to bind to the transactional datasource and the name of the WebLogic Server connection pool.

For more information on configuring transactional and non-transactional data sources, see Configure a JDBC Data Source.

7. Click Create to save the new JDBC Data Source.
8. Bind the JNDI name of the datasource to the EJB's local JNDI environment by doing one of the following:

   Map an existing EJB resource factory reference to the JNDI name by directly editing the `resource-description` element in the `weblogic.ejb-jar.xml` deployment file. See "Specifying and Editing the EJB Deployment Descriptors" on page 6-5 for instructions on editing deployment descriptors.

Setting Up URL Connection Factories

To set up a URL connection factory in WebLogic Server, bind a URL string to a JNDI name using these instructions:

1. In a text editor, open the `config.xml` file for the instance of the WebLogic Server you are using and set the `URLResource` attribute for the following `config.xml` elements:
   - `WebServer`
   - `VirtualHost:`
2. Set the `URLResource` attribute for the `WebServer` element using the following syntax:

   ```
   <WebServer URLResource="weblogic.httpd.url.testURL=http://
   localhost:7701/testfile.txt" DefaultWebApp="default-tests"/>
   ```

3. Set the `URLResource` attribute for the `VirtualHost` element, when virtual hosting is required, using the following syntax:

```
<VirtualHostName=guestserver" targets="myserver,test_web_server
"URLResource="weblogic.httpd.url.testURL=http://
localhost:7701/testfile.txt" VirtualHostNames="guest.com"/>
```

4. Save the changes in the `config.xml` file and reboot WebLogic Server

Locking Services for Entity EJBs

The following sections describe how the EJB Container supports locking services. The WebLogic Server container supports both the database locking and exclusive locking mechanisms. The default and *recommended* mechanism for EJB 1.1 and EJB 2.0 beans is database locking.

Exclusive Locking Services

Exclusive locking was the default in WebLogic Server 5.1 and 4.5.1. This method of locking provides reliable access to EJB data, and avoids unnecessary calls to `ejbLoad()` to refresh the EJB instance's persistent fields. However, exclusive locking does not provide the best model for concurrent access to the EJB's data. Once a client has locked an EJB instance, other clients are blocked from the EJB's data even if they intend only to read the persistent fields.

The EJB container in WebLogic Server can use exclusive locking mechanism for entity EJB instances. As clients enlist an EJB or EJB method in a transaction, WebLogic Server places an exclusive lock on the EJB instance for the duration of the transaction. Other clients requesting the same EJB or method are blocked until the current transaction completes.

Database Locking Services

Database locking is the default for WebLogic Server 6.1. It improves concurrent access for entity EJBs. The WebLogic Server container defers locking services to the underlying database. Unlike exclusive locking, the underlying data store can provide finer granularity for locking EJB data, and deadlock detection.

With the database locking mechanism, the EJB container continues to cache instances of entity EJB classes. However, the container does not cache the intermediate state of the EJB instance between transactions. Instead, WebLogic Server calls `ejbLoad()` for each instance at the beginning of a transaction to obtain the latest EJB data. The request to commit data is subsequently passed along to the database. The database, therefore, handles all lock management and deadlock detection for the EJB's data.

Deferring locks to the underlying database improves throughput for concurrent access to entity EJB data, while also providing deadlock detection. However, using database locking requires more detailed knowledge of the underlying datastore's lock policies, which can reduce the EJB's portability among different systems.

Setting Up Database Locking

You specify the locking mechanism used for an EJB by setting the `concurrency-strategy` deployment parameter in `weblogic-ejb-jar.xml`. You set `concurrency-strategy` at the individual EJB level, so that you can mix locking mechanisms within the EJB container.

The following excerpt from `weblogic-ejb-jar.xml` shows an EJB that uses database locking.

Figure 4-14 Database locking sample

```
<entity-descriptor>
        <entity-cache>
        ...
        <concurrency-strategy>Database</concurrency-strategy>
        </entity-cache>
        ...
</entity-descriptor>
```

If you do not specify `concurrency-strategy`, WebLogic Server performs database locking for entity EJB instances, as described in "Database Locking Services" on page 4-37.

CHAPTER

5 WebLogic Server Container-Managed Persistence Services

The following sections describe the new container-managed persistence (CMP) services available with the WebLogic Server EJB container.

Overview of Container Managed Persistence Services

WebLogic Server's container is responsible for providing a uniform interface between the EJB and the server. The container creates new instances of the EJBs, manages these bean resources, and provides persistent services such as, transactions, security, concurrency, and naming at runtime. In most cases, pre WebLogic Server 6.1 EJBs run in the container. However, see the Migration Guide for information on when you would need to migrate your bean code. See "DDConverter" on page 9-4 for instructions on using the conversion tool.

WebLogic Server's container-managed persistence (CMP) model handles persistence of CMP entity beans automatically at runtime by synchronizing the EJB's instance fields with the data in the database.

EJB Persistence Services

WebLogic Server provides persistence services for entity beans. An entity EJB can save its state in any transactional or non-transactional persistent storage ("bean-managed persistence"), or the container can save the EJB's non-transient instance variables automatically ("container-managed persistence"). WebLogic Server allows both choices and a mixture of the two.

If an EJB will use container-managed persistence, you specify the type of persistence services that the EJB uses in the `weblogic-ejb-jar.xml` deployment file. High-level definitions for automatic persistence services are stored in the persistence-type and persistence-use elements. The `persistence-type` element defines one or more automatic services that the EJB can use. The `persistence-use` element defines which service the EJB uses at deployment time.

Automatic persistence services use additional deployment files to specify their deployment descriptors, and to define entity EJB finder methods. For example, WebLogic Server RDBMS-based persistence services obtain deployment descriptors and finder definitions from a particular bean using the bean's `weblogic-cmp-rdbms-jar.xml` file, described in "Using WebLogic Server RDBMS Persistence" on page 5-3.

Third-party persistence services cause other file formats to configure deployment descriptors. However, regardless of the file type, you must reference the configuration file in the `persistence-type` and `persistence-use` elements in `weblogic-ejb-jar.xml`.

Note: Configure container-managed persistence beans with a connection pool with maximum connections greater than 1. WebLogic Server's container-managed persistence service sometimes needs to get two connections simultaneously.

Using WebLogic Server RDBMS Persistence

To use WebLogic Server RDBMS-based persistence service with your EJBs, create a dedicated XML deployment file and define the persistence elements for each EJB that will use container-managed persistence. If you use WebLogic Server's utility, DDConverter to create this file, it is named `weblogic-cmp-rdbms-jar.xml`. If you create the file from scratch, you can save it to a different filename. However, you must ensure that the `persistence-type` and `persistence-use` elements in `weblogic-ejb-jar.xml` refer to the correct file.

`weblogic-cmp-rdbms-jar.xml` defines the persistence deployment descriptors for EJBs using WebLogic Server RDBMS-based persistence services.

In each `weblogic-cmp-rdbms-jar.xml` file you define the following persistence options:

- EJB connection pools or data source for EJB 2.0 CMP
- EJB field to database element mappings
- Query Language
  - WebLogic Query Language (WLQL) for EJB 1.1 CMP
  - WebLogic EJB-QL with WebLogic QL extension for EJB 2.0 CMP (optional)

- Finder method definitions (CMP 1.1)
- Foreign key mappings for relationships
- WebLogic Server-specific deployment descriptors for queries

Writing for RDBMS Persistence for EJB 1.1 CMP

Clients use finder methods to query and receive references to entity beans that fulfill query conditions. This section describes how to write finders for WebLogic-specific 1.1 EJBs that use RDBMS persistence. EJB QL, a portable query language, is used to define finder queries for 2.0 EJBs with container-managed persistence. For more information about on EJB QL, see "Using EJB QL for EJB 2.0" on page 5-10.

WebLogic Server provides an easy way to write finders. The EJB provider writes the method signature of a finder in the `EJBHome` interface, and defines the finder's query expressions in the `ejb-jar.xml` deployment file.

`ejbc` creates implementations of the finder methods at deployment time, using the queries in `ejb-jar.xml`.

The key components of a finder for RDBMS persistence are:

- The finder method signature in `EJBHome`.
- A `query` stanza defined within `ejb-jar.xml`.
- An optional `finder-query` stanza within `weblogic-cmp-rdbms-jar.xml`.

The following sections explain how to write EJB finders using XML elements in WebLogic Server deployment files.

Finder Signature

EJB providers specify finder method signatures using the form `find`*`MethodName`*`()`. Finder methods defined in `weblogic-cmp-rdbms-jar.xml` must return a Java collection of EJB objects or a single object.

Note: EJB providers can also define a `findByPrimaryKey(primkey)` method that returns a single object of the associated EJB class.

finder-list Stanza

The `finder-list` stanza associates one or more finder method signatures in `EJBHome` with the queries used to retrieve EJB objects. The following is an example of a simple `finder-list` stanza using WebLogic Server RDBMS-based persistence:

```
<finder-list>
   <finder>
       <method-name>findBigAccounts</method-name>
       <method-params>
          <method-param>double</method-param>
        </method-params>
        <finder-query><![CDATA[(> balance $0)]]>
      </finder-query>
   </finder>
</finder-list>
```

Note: If you use a non-primitive data type in a `method-param` element, you must specify a fully qualified name. For example, use `java.sql.Timestamp` rather than `Timestamp`. If you do not use a qualified name, `ejbc` generates an error message when you compile the deployment unit.

finder-query Element

The `finder-query` element defines the WebLogic Query Language (WLQL) expression used to query EJB objects from the RDBMS. WLQL uses a standard set of operators against finder parameters, EJB attributes, and Java language expressions. See "Using WebLogic Query Language (WLQL) for EJB 1.1 CMP" on page 5-6 for more information on WLQL.

Note: Always define the text of the `finder-query` value using the XML `CDATA` attribute. Using `CDATA` ensures that any special characters in the WLQL string do not cause errors when the finder is compiled.

A CMP finder can load all beans using a single database query. So, 100 beans can be loaded with a single database round trip. A bean-managed persistence (BMP) finder must do one database round trip to get the primary key values of the beans selected by the finder. As each bean is accessed, another database access is also typically required, assuming the bean wasn't already cached. So, to access 100 beans, a BMP might do 101 database accesses.

Using WebLogic Query Language (WLQL) for EJB 1.1 CMP

WebLogic Query Language (WLQL) for EJB 1.1 CMP allows you to query 1.1 entity EJBs with container-managed persistence. In the `weblogic-cmp-rdbms-jar.xml` file, each `finder-query` stanza must include a WLQL string that defines the query used to return EJBs. Use WLQL for EJBs and their corresponding deployment files that are based on the EJB 1.1 specification.

Note: For queries to 2.0 EJBs, see "Using EJB QL for EJB 2.0" on page 5-10. Using the weblogic-ql query completely overrides the ejb-ql query.

Syntax

WLQL strings use the prefix notation for comparison operators:

```
(operator operand1 operand2)
```

Additional WLQL operators accept a single operand, a text string, or a keyword.

Operators

The following are valid WLQL operators.

Operator	Description	Sample Syntax
`=`	Equals	`(= operand1 operand2)`
`<`	Less than	`(< operand1 operand2)`
`>`	Greater than	`(> operand1 operand2)`
`<=`	Less than or equal to	`(<= operand1 operand2)`
`>=`	Greater than or equal to	`(>= operand1 operand2)`
`!`	Boolean not	`(! operand)`
`&`	Boolean and	`(& operand)`
`|`	Boolean or	`(| operand)`
`like`	Wildcard search based on % symbol in the supplied *`text_string`*	`(like` *`text_string`*`%)`
`isNull`	Value of single operand is null	`(isNull operand)`
`isNotNull`	Value of single operand is not null	`(isNotNull operand)`
`orderBy`	Orders results using specified database columns **Note:** Always specify a database column name in the `orderBy` clause, rather than a persistent field name. WebLogic Server does not translate field names specified in `orderBy`.	`(orderBy '`*`column_name`*`')`
`desc`	Orders results in descending order. Used only in combination with `orderBy`.	`(orderBy '`*`column_name`*` desc')`

Operands

Valid WLQL operands include:

- Another WLQL expression
- A container-managed field defined elsewhere in the `weblogic-cmp-rdbms-jar.xml` file

  Note: You cannot use RDBMS column names as operands in WLQL. Instead, use the EJB attribute (field) that maps to the RDBMS column, as defined in the attribute-map in `weblogic-cmp-rdbms-jar.xml`.

- A finder parameter or Java expression identified by `$`*n*, where *n* is the number of the parameter or expression. By default, `$`*n* maps to the *n*th parameter in the signature of the finder method. To write more advanced WLQL expressions that embed Java expressions, map `$`*n* to a Java expression.

  Note: The *$n* notation is based on an array that begins with 0, *not* 1. For example, the first three parameters of a finder correspond to `$0`, `$1`, and `$2`. Expressions need not map to individual parameters. Advanced finders can define more expressions than parameters.

Examples of WLQL Expressions

The following example code shows excerpts from the `weblogic-cmp-rdbms-jar.xml` file that use basic WLQL expressions.

- This example returns all EJBs that have the `balance` attribute greater than the `balanceGreaterThan` parameter specified in the finder. The finder method signature in EJBHome is:

```
public Enumeration findBigAccounts(double balanceGreaterThan)
     throws FinderException, RemoteException;
```

  The sample `<finder>` stanza is:

```
<finder>
     <method-name>findBigAccounts</method-name>
     <method-params>
```

```
        <method-param>double</method-param>
    </method-params>
    <finder-query><![CDATA[(> balance $0)]]></finder-query>
</finder>
```

Note that you must define the `balance` field n the attribute map of the EJB's persistence deployment file.

Note: Always define the text of the `finder-query` value using the XML `CDATA` attribute. Using `CDATA` ensures that any special characters in the WLQL string do not cause errors when the finder is compiled.

- The following example shows how to use compound WLQL expressions. Also note the use of single quotes (') to distinguish strings:

  ```
  <finder-query><![CDATA[(& (> balance $0) (! (= accountType
  'checking')))]]></finder-query>
  ```

- The following example finds all the EJBs in a table. It uses the sample finder method signature:

  ```
  public Enumeration findAllAccounts()
      throws FinderException, RemoteException
  ```

  The sample `<finder>` stanza uses an empty WLQL string:

  ```
  <finder>
      <method-name>findAllAccounts</method-name>
      <finder-query></finder-query>
  </finder>
  ```

- The following query finds all EJBs whose `lastName` field starts with "M":

  ```
  <finder-query><![CDATA[(like lastName M%)]]></finder-query>
  ```

- This query returns all EJBs that have a null `firstName` field:

  ```
  <finder-query><![CDATA[(isNull firstName)]]></finder-query>
  ```

- This query returns all EJBs whose balance field is greater than 5000, and orders the beans by the database column, id:

  ```
  <finder-query><![CDATA[WHERE >5000 (orderBy 'id' (> balance
  5000))]]></finder-query>
  ```

- This query is similar to the previous example, except that the EJBs are returned in descending order:

```
<finder-query><![CDATA[(orderBy 'id desc' (>
))]]></finder-query>
```

Using EJB QL for EJB 2.0

EJB Query Language (QL) is a portable query language that defines finder methods for 2.0 entity EJBs with container-managed persistence. Use this SQL-like language to select one or more entity EJB objects or fields in your query. Because of the declaration of CMP fields in a deployment descriptor, you can create queries in the deployment descriptor for any finder method other than `findByPrimaryKey()`. `findByPrimaryKey` is automatically handled by the container. The search space for an EJB QL query consists of the EJB's schema as defined in `ejb-jar.xml` (the bean's collection of container-managed fields and their associated database columns).

EJB QL Requirement for EJB 2.0 Beans

The deployment descriptors must define each finder query for EJB 2.0 entity beans by using an EJB QL query string. You cannot use WebLogic Query Language (WLQL) with EJB 2.0 entity beans. WLQL is intended for use with EJB 1.1 CMP.

Migrating from WLQL to EJB QL

If you have used previous versions of WebLogic Server, your container-managed entity EJBs may use WLQL for finder methods. This section provides a quick reference to common WLQL operations. Use this table to map the WLQL syntax to EJB QL syntax.

Sample WLQL Syntax	Equivalent EJB QL Syntax
`(= operand1 operand2)`	`WHERE operand1 = operand2`

Sample WLQL Syntax	Equivalent EJB QL Syntax
(< operand1 operand2)	WHERE operand1 < operand2
(> operand1 operand2)	WHERE operand1 > operand2
(<= operand1 operand2)	WHERE operand1 <= operand2
(>= operand1 operand2)	WHERE operand1 >= operand2
(! operand)	WHERE NOT operand
(& expression1 expression2)	WHERE expression1 AND expression2
(| expression1 expression2)	WHERE expression1 OR expression2
(like *text_string*%)	WHERE operand LIKE *'text_string%'*
(isNull operand)	WHERE operand IS NULL
(isNotNull operand)	WHERE operand IS NOT NULL

Using EJB 2.0 WebLogic QL Extension for EJB QL

WebLogic Server has an SQL-like language, called WebLogic QL, that extends the standard EJB QL. This language works with the finder expressions and is used to query EJB objects from the RDBMS. You define the `query` in the `weblogic-cmp-rdbms-jar.xml` deployment descriptor using the `weblogic-ql` element.

There must be a query element in the `ejb-jar`.file that corresponds to the `weblogic-ql` element in the `weblogic-cmp-rdbms-jar.xml` file. However, the `weblogic-cmp-rdbms-jar.xml` query element overrides the `ejb-jar.xml` query element.

SELECT DISTINCT

The EJB WebLogic QL extension SELECT DISTINCT tells your database to filter duplicate queries. Using SELECT DISTINCT means that the EJB container's resources are not used to sort through duplicated results when SELECT DISTINCT is specified in the EJB QL query.

If you specify a `sql-select-distinct` element with the value `TRUE` in a `weblogic-ql` element's XML stanza for an EJB 2.0 CMP bean, then the generated SQL STATEMENT for the database query will contain a DISTINCT clause.

You specify the `sql-select-distinct` element in the `weblogic-cmp-rdbms-jar.xml` file. However, you cannot specify `sql-select-distinct` if you are running an isolation level of `READ_C0MMITED_FOR_UPDATE` on an Oracle database. This is because a query on Oracle cannot have both a `sql-select-distinct` and a `READ_C0MMITED_FOR_UPDATE`. If there is a chance that this isolation level will be used, for example in a session bean, do not use the `sql-select-distinct` element.

ORDERBY

The EJB WebLogic QL extension ORDERBY is a keyword that works with the Finder method to specify the CMP field selection sequence for your selections.

Figure 5-1 WebLogic QL ORDERBY extension showing order by id.

```
ORDERBY
   SELECT OBJECT(A) from A for Account.Bean
      ORDERBY A.id
```

Note: ORDERBY defers all sorting to the DBMS. Thus, the order of the retrieved result depends on the particular DBMS installation on top of which the bean is running.

Using Oracle SELECT HINTS

WebLogic Server supports an EJB QL extension that allows you to pass INDEX usage hints to the Oracle Query optimizer. With this extension, you can provide a hint to the database engine. For example, if you know that the database you are searching can benefit from an ORACLE_SELECT_HINT, you can define an ORACLE_SELECT_HINT clause that will take ANY string value and then insert that String value after the SQL SELECT statement as a hint to the database.

To use this option, declare a query that uses this feature in the `weblogic-ql` element. This element is found in the `weblogic-cmp-rdbms-jar.xml` file. The `weblogic-ql` element specifies a query that contains a WebLogic specific extension to the EJB-QL language.

The WebLogic QL keyword and usage is as follows:

```
SELECT OBJECT(a) FROM BeanA AS a WHERE a.field > 2 ORDERBY a.field
SELECT_HINT '/*+ INDEX_ASC(myindex) */'
```

This statement generates the following SQL with the optimizer hint for Oracle:

```
SELECT /*+ INDEX_ASC(myindex) */ column1 FROM .... (etc)
```

In the WebLogic QL ORACLE_SELECT_HINT clause, whatever is between the single quotes (`''`) is what gets inserted after the SQL SELECT. It is the query writer's responsibility to make sure that the data within the quotes makes sense to the Oracle database.

"get" and "set" Method Restrictions

WebLogic Server uses a series of accessor methods. The names of these methods begin with *set* and *get*. WebLogic Server uses these methods to read and modify container-managed fields. These container-generated classes must begin with "get" or "set" and use the actual name of a persistent field defined in `ejb-jar.xml`. The methods are also declared as `public`, `protected`, and `abstract`.

BLOB and CLOB DBMS Column Support for the Oracle DBMS

WebLogic Server supports Oracle Binary Large Object (BLOB) and Character Large Object (CLOB) DBMS columns with EJB CMP. BLOBs and CLOBs are data types used for efficient storage and retrieval of large objects. CLOBs are string or char objects; BLOBs are binary or serializable objects such as pictures that translate into large byte arrays.

BLOBs and CLOBs map a string variable, a value of `OracleBlob` or `OracleClob`, to a BLOB or CLOB column. WebLogic Server maps CLOBs only to the data type `java.lang.string`. At this time, no support is available for mapping `char` arrays to a CLOB column.

To enable BLOB/CLOB support:

1. In the bean class, declare the variable.
2. Edit the XML by declaring the `dbms-column-type` deployment descriptor in the weblogic-cmp-rdbms jar.xml file.
3. Create the BLOB or CLOB in the Oracle database.

Using BLOB or CLOB may slow performance because of the size of the BLOB or CLOB object.

Specifying a BLOB Using the Deployment Descriptor

The following XML code shows how to specify a BLOB object using the `dbms-column` element in `weblogic-cmp-rdbms-jar-xml` file.

Figure 5-2 Specifying a BLOB object

```
<field-map>
       <cmp-field>photo</cmp-field>
       <dbms-column>PICTURE</dbms-column>
       <dbms_column-type>OracleBlob</dbms-column-type>
</field-map>
```

Controlling Serialization of cmp-fields Mapped to OracleBlobs

By default, when WebLogic Server writes and reads a cmp-field of type `byte[]` that is mapped to an OracleBlob, it serializes and deserializes the field, respectively.

If WebLogic Server reads a BLOB that was written directly to the database by another program, errors can result, because the container assumes that the data is serialized.

To specify that the data is not serialized, compile the EJB with this flag:

```
java -Dweblogic.byteArrayIsSerializedToOracleBlob=false
weblogic.ejbc std_ejb.jar ejb.jar
```

Specifying a CLOB Using the Deployment Descriptors

The following XML code shows how to specify a CLOB object using the `dbms-column` element in the `weblogic-cmp-rdbms-jar-xml` file.

Figure 5-3 Specifying a CLOB object

```
<field-map>
       <cmp-field>description</cmp-field>
       <dbms-column>product_description</dbms-column>
       <dbms_column-type>OracleClob</dbms-column-type>
</field-map>
```

Cascade Delete

Use the cascade delete mechanism to remove entity bean objects. When cascade delete is specified for a particular relationship, the lifetime of one entity object depends on another. You can specify cascade delete for one-to-one and one-to-many relationships; many-to-many relationships are not supported. The `cascade delete()` method uses the delete features in WebLogic Server, and the `database cascade delete()` method instructs WebLogic Server to use the underlying database's built-in support for cascade delete.

To enable this feature, you must recompile the bean code for the changes to the deployment descriptors to take effect.

Use one of the following two methods to enable cascade delete.

Cascade Delete Method

With the `cascade delete()` method you use WebLogic Server to remove objects. If an entity is deleted and the `cascade delete` element is specified for a related entity bean, then the removal is cascaded and any related entity bean objects are also removed.

To specify cascade delete, use the `cascade-delete` element in the `ejb-jar.xml` deployment descriptor elements. This is the default method. Make no changes to your database settings, and WebLogic Server will cache the entity objects for removal when the cascade delete is triggered.

Specify cascade delete using the `cascade-delete` element in the `ejb-jar.xml` file as follows:

Figure 5-4 Specifying a cascade delete

```
<ejb-relation>
       <ejb-relation-name>Customer-Account</ejb-relation-name>
       <ejb-relationship-role>
              <ejb-relationship-role-name>Account-Has-Customer
              </ejb-relationship-role-name>
              <multiplicity>one</multiplicity>
              <cascade-delete/>
       </ejb-relationship-role>
</ejb-relation>
```

Note: This `cascade delete()` method can only be specified for a `ejb-relationship-role` element contained in an `ejb-relation` element if the other `ejb-relationship-role` element in the same `ejb-relation` element specifies a `multiplicity` attribute with a value of `one`.

Database Cascade Delete Method

The `database cascade delete()` method allows an application to take advantage of a database's built-in cascade delete support, and possibly improve performance. If the `db-cascade-delete` element is not already specified in the `weblogic-cmp-rdbms-jar.xml` file, do not enable any of the database's cascade delete functionality, because this will produce incorrect results in the database.

The `db-cascade-delete` element in the `weblogic-cmp-rdbms-jar.xml` file specifies that a cascade delete operation will use the built-in cascade delete facilities of the underlying DBMS. By default, this feature is turned off and the EJB container removes the beans involved in a cascade delete by issuing an individual SQL DELETE statement for each bean.

If `db-cascade-delete` element is specified in the `weblogic-cmp-rdbms-jar.xml`, the `cascade-delete` element must be specified in the `ejb-jar.xml`.

When `db-cascade-delete` is enabled, additional database table setup is required. For example, the following setup for the Oracle database table will cascade delete all of the employees if the `dept` is deleted in the database.

Figure 5-5 Oracle table setup for cascade delete

```
CREATE TABLE dept
    (deptno   NUMBER(2) CONSTRAINT pk_dept PRIMARY KEY,
     dname    VARCHAR2(9) );
CREATE TABLE emp
    (empno    NUMBER(4) PRIMARY KEY,
     ename    VARCHAR2(10),
     deptno  NUMBER(2)   CONSTRAINT fk_deptno
             REFERENCES dept(deptno)
             ON DELETE CASCADE );
```

Tuned EJB 1.1 CMP Updates in WebLogic Server

EJB container-managed persistence (CMP) automatically support tuned updates because the container receives `get` and `set` callbacks when container-managed EJBs are read or written. Tuning EJB 1.1 CMP beans helps improve their performance.

WebLogic Server now supports tuned updates for EJB 1.1 CMP. When `ejbStore` is called, the EJB container automatically determines which container-managed fields have been modified in the transaction. Only modified fields are written back to the database. If no fields are modified, no database updates occur.

With previously versions of WebLogic Server, you could to write an `isModified` method that notified the container whether the EJB 1.1 CMP bean had been modified. `isModified` is still supported in WebLogic Server, but we recommend that you no longer use `isModified` methods and instead allow the container to determine the update fields.

This feature is enabled for EJB 2.0 CMP, by default. To enable tuned EJB 1.1 CMP updates, make sure that you set the following deployment descriptor element in the `weblogic-cmp-rdbms-jar.xml` file to `true`.

```
<enable-tuned-updates>true</enable-tuned-updates>
```

You can disable tuned CMP updates by setting this deployment descriptor element as follows:

```
<enable-tuned-updates>false</enable-tuned-updates>
```

In this case, `ejbStore` always writes all fields to the database.

Flushing the CMP Cache

Updates made by a transaction must be reflected in the results of queries, finders, and `ejbSelects` issued during the transactions. Because this requirement can slow performance, a new option enables you to specify that the cache be flushed before the query for the bean is executed.

If this option is turned off, which is the default behavior, the results of the current transactions are not reflected in the query. If this option is turned on, the container flushes all changes for cached transactions written to the database before executing the new query. This way, the changes show up in the results.

To enable this option, in `weblogic-cmp-rdbms-jar.xml` file set the `include-updates` element to `true`.

Figure 5-6 Specifying that results of transactions be reflected in the query

```
<weblogic-query>
       <query-method>
       <method-name>findBigAccounts</method_name>
       <method-params>
              <method-param>double</method-param>
       </method-params>
       </query-method>
       <weblogic-ql>WHERE BALANCE>10000 ORDERBY NAME</weblogic-ql>
       <include-updates>true</include-updates>
</weblogic-query>
```

The default is `false`, which provides the best performance. Updates made to the cached transaction are reflected in the result of a query; no changes are written to the database, and you do not see the changes in the query result.

Whether you use this feature depends on whether performance is more important than current and consistent data.

Primary Keys

The primary key is an object that uniquely identifies an entity bean within its home. The container must be able to manipulate the primary key of an entity bean. Each entity bean class may define a different class for its primary key, but multiple entity beans can use the same primary key class. The primary key is specified in the deployment descriptor for the entity bean. You can specify a primary key class for an entity bean with container-managed persistence by mapping the primary key to either a single field or to multiple fields in the entity bean class.

Every entity object has a unique identity within its home. If two entity objects have the same home and the same primary key, they are considered identical. A client can invoke the `getPrimaryKey()` method on the reference to an entity object's remote interface to determine the entity object's identity within its home. The object identify associated with the a reference does not change during the lifetime of the reference. Therefore, the `getPrimaryKey()` method always returns the same value when called on the same entity object reference. A client that knows the primary key of an entity object can obtain a reference to the entity object by invoking the `findByPrimaryKey(key)` method on the bean's home interface.

Primary Key Mapped to a Single CMP Field

In the entity bean class, you can have a primary key that maps to a single CMP field. You use the `primkey-field` element, a deployment descriptor in the `ejb-jar.xml` file, to specify the container-managed field that is the primary key. The `prim-key-class` element must be the primary key field's class.

Primary Keys Class That Wraps Single or Multiple CMP Fields

You can have a primary key class that maps to single or multiple fields. The primary key class must be `public`, and have a `public` constructor with no parameters. You use the `prim-key-class` element, a deployment descriptor in the `ejb-jar.xml` file to specify the name of the entity bean's primary key class. You can only specify the

the class name in this deployment descriptor element. All fields in the primary key class must be declared public. The fields in the class must have the same name as the primary key fields in the `ejb-jar.xml` file.

Hints for Using Primary Keys

Some hints for using primary keys with WebLogic Server include:

- Do not make the primary key class a container-managed field.

  Although `ejbCreate` specifies the primary key class as a return type:

- Do not construct a new primary key class with an `ejbCreate`. Instead, allow the container to create the primary key class internally.

- Set the values of the primary key cmp-fields using the `setXXX` methods within the `ejbCreate` method.

- Do not use a cmp field of the type `BigDecimal` as a primary key field for CMP beans. The `boolean BigDecimal.equals (object x)` method considers two `BigDecimal` equal only if they are equal in value and scale. This is because there are differences in precision between the Java language and different databases. For example, the method does not consider `7.1` and `7.10` to be equal. Consequently, this method will most likely return false or cause the CMP bean to fail.

  If you need to use `BigDecimal` as the primary key, you should:

  a. Implement a primary key class.

  b. In this primary key class, implement the `boolean equal (Object x)` method.

  c. In the equal method, use `boolean BigDecimal.compareTo(BigDecimal val)`.

Mapping to a Database Column

WebLogic Server supports mapping a database column to a `cmp-field` and a `cmr-field` concurrently. The `cmp-field` is read-only in this case. If the `cmp-field` is a primary key field, specify that the value for the field be set when the `create()` method is invoked by using the `setXXX` method for the `cmp-field`.

Automatic Primary Key Generation for EJB 2.0 CMP

WebLogic Server supports an automatic primary key generation feature for container-managed persistence (CMP).

Note: This feature is supported for the EJB 2.0 CMP container only, there is no automatic primary key generation support for EJB 1.1 CMP. For 1.1 beans, you must use bean-managed-persistence (BMP.)

Generated key support is provided in two ways:

- **Using DBMS primary key generation**. A set of deployment descriptors are specified at compile time to generate container code that is used in conjunction with a supported database to provide key generation support.

  With this option, the container defers all key generation to the underlying database. To enable this feature, you specify the name of the supported DBMS and the generator name, if required by the database. The CMP code handles all details of implementing this feature.

  For more information on this feature, see "Specifying Primary Key Support for Oracle" on page 5-23 and "Specifying Primary Key Support for Microsoft SQL Server" on page 5-24.

- **Using Bean Provider Designated Named Sequence table**. A user-named and user-created database table has a schema specified by WebLogic Server. The container uses this table to generate the keys.

  With this option, you name a table that holds the current primary key value. The table consists of a single row with a single column as defined by the following statement:

  ```
  CREATE table_name (SEQUENCE int)
  INSERT into table_name VALUES (0)
  ```

Note: For instructions on creating a table in Oracle, use the Oracle database documentation.

In the `weblogic-cmp-rdbms-jar.xml` file, set the `key_cache_size` element to specify how many primary key values a database SELECT and UPDATE will

fetch at one time. The default value of `key_cache_size` is `1`. BEA recommends that you set this element to a value of `>1`, to minimize database accesses and to improve performance. For more information in this feature, see "Specifying Primary Key Named Sequence Table Support" on page 5-24.

At this time, WebLogic Server only provides DBMS primary key generation support for Oracle and Microsoft SQL Server. However, you can use named/sequence tables with other unsupported databases. Also, this feature is intended for use with simple (non-compound) primary keys.

Note: The key field must be declared to be of type `java.lang.Integer` in the abstract 'get' and 'set' methods of the bean.

Valid Key Field Values

In the abstract 'get' and 'set' methods of the bean, you can declare the field to be either of the following two types:

- java.lang.Integer
- java.lang.Long (this capability is only available if you have installed Service Pack 3 or higher for WebLogic Server 6.1)

Specifying Primary Key Support for Oracle

Generated primary key support for Oracle databases uses Oracle's `SEQUENCE` feature. This feature works with a `Sequence` entity in the Oracle database to generate unique primary keys. The Oracle `SEQUENCE` is called when a new number is needed.

Once the `SEQUENCE` already exists in the database, you specify automatic key generation in the XML deployment descriptors. In the `weblogic-cmp-rdbms-jar.xml` file, you specify automatic key generation as follows:

Figure 5-7 Specifying automatic key generation for Oracle

```
<automatic-key-generation>
       <generator-type>ORACLE</generator-type>
       <generator_name>test_sequence</generator-name>
```

```
        <key-cache-size>10</key-cache-size>
</automatic-key-generation>
```

Specify the name of the `ORACLE SEQUENCE` to be used, using the `generator-name` element. If the `ORACLE SEQUENCE` was created with a `SEQUENCE INCREMENT` value, then you must specify a `key-cache-size`. This value must match the Oracle `SEQUENCE INCREMENT` value. If these two values are different, then you will most likely have duplicate key problems.

Specifying Primary Key Support for Microsoft SQL Server

Generated primary key support for Microsoft SQL Server databases uses SQL Server's `IDENTITY` column. When the bean is created and a new row is inserted in the database table, SQL Server automatically inserts the next primary key value into the column that was specified as an `IDENTITY` column.

Note: For instructions on creating a table in Microsoft SQL Server, see the Microsoft SQL Server database documentation.

Once the `IDENTITY` column is created in the database table, you specify automatic key generation in the XML deployment descriptors. In the `weblogic-cmp-rdbms-jar.xml` file, you specify automatic key generation as follows:

Figure 5-8 Specifying automatic key generation for Microsoft SQL

```
<automatic-key-generation>
        <generator-type>SQL_SERVER</generator-type>
</automatic-key-generation>
```

The `generator-type` element lets you specify the primary key generation method that you want to use.

Specifying Primary Key Named Sequence Table Support

Generated primary key support for unsupported databases uses a `Named SEQUENCE TABLE` to hold key values. The table must contain a single row with a single column that is an integer, `SEQUENCE INT`. This column will hold the current sequence value.

Note: For instructions on creating the table, see the documentation for the specific database product.

Once the NAMED_SEQUENCE_TABLE exists in the database, you specify automatic key generation by using the XML deployment descriptors in the weblogic-cmp-rdbms-jar.xml file, as follows:

Figure 5-9 Specifying automatic key generation for named sequence table support

```
<automatic-key-generation>
       <generator-type>NAMED_SEQUENCE_TABLE</generator-type>
       <generator_name>MY_SEQUENCE_TABLE_NAME</generator-name>
       <key-cache-size>100</key-cache-size>
</automatic-key-generation>
```

Specify the name of the SEQUENCE TABLE to be used, with the generator-name element. Using the key-cache-size element, specify the optional size of the key cache that tells you how many keys the container will fetch in a single DBMS call.

For improved performance, BEA recommends that you set this value to >1, a number greater than one. This setting reduces the number of calls to the database to fetch the next key value.

Also, it is recommended that you define one NAMED SEQUENCE table per bean type. Beans of different types should not share a common NAMED SEQUENCE table. This reduces contention for the key table.

Automatic Table Creation

You can specify that WebLogic Server automatically create tables based on the descriptions in the XML deployment descriptor files and the bean class, if the table does not already exist. Tables are created for all beans and relationship join tables, if the relationships in the JAR files have joins. You explicitly turn on this feature by defining it in the deployment descriptors per each RDBMS deployment, for all beans in the JAR file.

WebLogic Server makes a best attempt to create the new table. However, if based on the descriptions in the deployment files, the field cannot be successfully mapped to an appropriate column type in the database, the `TABLE CREATION` fails, an error is thrown, and you must create the table yourself.

Automatic table creation is not recommended for use in a production environment. It is better suited for the development phase of design and prototype work. A production environment may require the use of more precise table schema definitions, for example; the declaration of foreign key constraints.

To define automatic table creation:

1. In the `weblogic-cmp-rdbms-jar.xml` file, set the `create-default-dbms-tables` element to `True` to explicitly turn on automatic table creation for all beans in the JAR file.
2. Use the following syntax:

   ```
   <create-default-dbms-tables>True</create-default-dbms-tables>
   ```

Because automatic table creation may not map every Java field type successfully to your target database, the following list is provided to give you an idea of the type of mapping you can expect to see.

Table 5-1 Java Field Types

Java Type	DBMS Column Type
boolean	INTEGER
byte	INTEGER
char	CHAR
double	DOUBLE PRECISION
float	FLOAT
int	INTEGER
long	INTEGER
short	INTEGER
java.lang.string	VARCHAR (150)

Java Type	DBMS Column Type
java.lang.BigDecimal	DECIMAL (38, 19)
java.lang.Boolean	INTEGER
java.lang.Byte	INTEGER
java.lang.Character	CHAR (1)
java.lang.Double	DOUBLE PRECISION
java.lang.Float	FLOAT
java.lang.Integer	INTEGER
java.lang.Long	INTEGER
java.lang.Short	INTEGER
java.sql.Date	DATE
java.sql.Time	DATE
java.sql.Timestamp	DATETIME
byte[ ]	RAW (1000)
Any serializable Class that is not a valid SQL type:	RAW (1000)

Container-Managed Relationships

The entity bean relies on container-managed persistence to generate the methods that perform persistent data access for the entity bean instances. The generated methods transfer data between entity bean instances and the underlying resource manager. Persistence is handled by the container at runtime. The advantage of using container-managed persistence is that the entity bean can be logically independent of the data source in which the entity is stored. The container manages the mapping between the logical and physical relationships at runtime and manages their referential integrity.

Persistent fields and relationships make up the entity bean's abstract persistence schema. The deployment descriptors indicate that the entity bean uses container-managed persistence, and these descriptors are used as input to the container for data access.

Entity beans can have relationships with other beans. You specify relationships in the `ejb-jar.xml` file and `weblogic-cmp-rdbms-jar.xml`. You specify container-managed field mappings in the `weblogic-cmp-rdbms-jar.xml` file.

When a bean with a relationship to another bean is removed, the container automatically removes the relationship.

Relationship Cardinality

WebLogic Server supports three types of relationship mappings that are managed by WebLogic container-managed persistence (CMP):

- One-to-one—A WebLogic Server one-to-one relationship also involves the physical mapping from a foreign key in one bean to the primary key in another bean. See "Groups" on page 5-36 for more information on primary keys.
- One-to-many—A WebLogic Server one-to-many relationship also involves the physical mapping from a foreign key in one bean to the primary key of another. However, in a one-to-many relationship, the foreign key is always contained in the role that occupies the "many" side of the relationship.
- Many-to-many—A WebLogic Server many-to-many relationship involves the physical mapping of a join table. Each row in the join table contains two foreign keys that map to the primary keys of the entities involved in the relationship.

Relationship Direction

Container-managed relationships can be either bidirectional or unidirectional.

- Unidirectional—can only navigate in one direction. These types of relationships are used with remote beans, and only unidirectional relationships can be remote. A remote bean is one whose abstract persistence schema is not defined in the same `EJB-jar` file as the bean with which it has a relationship. For example, if entity A and entity B are in a one-to-one, unidirectional relationship and the

direction is from entity A to entity B, than entity A is aware of entity B, but entity B is unaware of entity A. This type of relationship is implemented with a CMR-field on the entity bean from which navigation can take place and no related CMR-field on the target entity bean.

- Bidirectional Relationships—can be navigated in both directions. These types of container-managed relationships can exist only between beans whose abstract persistence schemas are defined in the same `EJB-jar` file and therefore managed by the same container. For example, if entity A and entity B are in a one-to-one bidirectional relationship, both are aware of each other.

Local Interfaces and Container-Managed Relationships

WebLogic Server provides support for local interfaces for session and entity beans. Local interfaces allow enterprise javabeans to work together within the same EJB container using different semantics and execution contexts. The EJBs are usually co-located within the same EJB container and execute within the same Java Virtual Machine (JVM). This way, they do not use the network to communicate and avoid the overhead of a Java Remote Method Invocation-Internet Inter-ORB Protocol (RMI-IIOP) connection.

EJB relationships with container-managed persistence are now based on the EJB's local interface. Any EJB that participates in a relationship must have a local interface. Local interface objects are lightweight persistent objects. They allow you to do more fine grade coding than do remote objects. Local interfaces also use pass-by-reference. The getter is in the local interface.

In earlier versions of WebLogic Server, you can base relationships on remote interfaces. However, CMP relationships that use remote interfaces should probably not be used in new code.

The EJB container makes the local home interface accessible to local clients through JNDI. To reference a local interface you need to have a local JNDI name. The objects that implement the entity beans' local home interface are called `EJBLocalHome` objects.

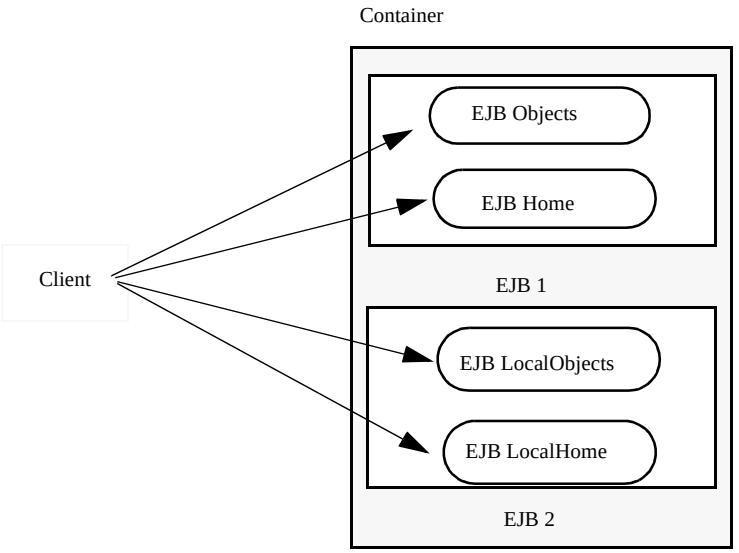
In pre-6.1 versions of WebLogic Server, `ejbSelect` methods were used to return remote interfaces. Now you can specify a `result-type-mapping` element in the `ejb-jar.xml` file that indicates whether the result returned by the query will be mapped to a local or remote object.

Using the Local Client

A local client of a session bean or entity bean can be another EJB, such as a session bean, entity bean, or message-driven bean. A local client can be a servlet as long as it is included as part of the same EAR file and as long as the EAR file is not remote. Clients of a local bean must be part of an EAR or a standalone JAR.

A local client accesses a session or entity bean through the bean's local interface and local home interfaces. The container provides classes that implement the bean's local and local home interfaces. The objects that implement these interfaces are local Java objects. The following diagram shows the container with a local client and local interfaces.

Figure 5-10 Local client and local interfaces



WebLogic Server provides support for both local and uni-directional remote relationships between EJBs. If the EJBs are on the same server and are part of the same JAR file, they can have local relationships. If the EJBs are not on the same server, the relationships must be remote. For a relationship between local beans, multiple column mappings are specified if the key implementing the relation is a compound key. For a

remote bean, only a single column-map is specified, since the primary key of the remote bean is opaque. No column-maps are specified if the role just specifies a group-name. No group-name is specified if the relationship is remote.

Changes to the Container for Local Interfaces

Changes made to the structure of the container to accommodate local interfaces include the following additions:

- EJB local home
- New model for handling exceptions that propagates the correct exception to the client.

Defining Container-Managed Relationships

Defining a CMR involves specifying the relationship and its cardinality and direction in `ejb-jar.xml`. You define database mapping details for the relationship and enable relationship caching in `weblogic-cmp-jar.xml`. These sections provide instructions:

- "Specifying Relationship in ejb-jar.xml" on page 5-31
- "Specifying Relationships in weblogic-cmp-jar.xml" on page 5-34

Specifying Relationship in ejb-jar.xml

Container-managed relationships are defined in the `ejb-relation` stanza of `ejb-jar.xml`. Figure 5-11 shows the `ejb-relation` stanza for a relationship between two entity EJBs: `TeacherEJB` and `StudentEJB`.

The `ejb-relation` stanza contains a `ejb-relationship-role` for each side of the relationship. The role stanzas specify each bean's view of the relationship.

Figure 5-11 One-to-Many, Bidirectional CMR in ejb-jar.xml

```
<ejb-relation>
   <ejb-relation-name>TeacherEJB-StudentEJB</ejb-relation-name>
      <ejb-relationship-role>
         <ejb-relationship-role-name>teacher-has-student
         </ejb-relationship-role-name>
         <multiplicity>One</multiplicity>
```

```
            <relationship-role-source>
               <ejb-name>TeacherEJB</ejb-name>
            </relationship-role-source>
            <cmr-field>
                <cmr-field-name>teacher</cmr-field-name>
            </cmr-field>
      </ejb-relationship-role>
      <ejb-relationship-role>
         <ejb-relationship-role-name>student-has-teacher
         </ejb-relationship-role-name>
         <multiplicity>Many</multiplicity>
         <relationship-role-source>
            <ejb-name>StudentEJB</ejb-name>
         </relationship-role-source>
         <cmr-field>
            <cmr-field-name>student</cmr-field-name>
            <cmr-field-type>java.util.Collection
         <cmr-field>
      </ejb-relationship-role>
```

Specifying Relationship Cardinality

The cardinality on each side of a relationship is indicated using the `<multiplicity>` element in its `ejb-relationship-role` stanza.

In Figure 5-11 the cardinality of the `TeacherEJB-StudentEJB` relationship is one-to-many—it is specified by setting `multiplicity` to `one` on the `TeacherEJB` side and `Many` on the `StudentEJB` side.

The cardinality of the CMR in Figure 5-12, is one-to-one—the `multiplicity` is set to `one` in both role stanza for the relationship

Figure 5-12 One-to-One, Unidirectional CMR in ejb-jar.xml

```
<ejb-relation>
   <ejb-relation-name>MentorEJB-StudentEJB</ejb-relation-name>
      <ejb-relationship-role>
         <ejb-relationship-role-name>mentor-has-student
         </ejb-relationship-role-name>
         <multiplicity>One</multiplicity>
         <relationship-role-source>
            <ejb-name>MentorEJB</ejb-name>
         </relationship-role-source>
         <cmr-field>
             <cmr-field-name>mentorID</cmr-field-name>
         </cmr-field>
   </ejb-relationship-role>
```

```
<ejb-relationship-role>
   <ejb-relationship-role-name>student-has-mentor
   </ejb-relationship-role-name>
   <multiplicity>One</multiplicity>
   <relationship-role-source>
      <ejb-name>StudentEJB</ejb-name>
   </relationship-role-source>
</ejb-relationship-role>
```

If a side of a relationship of a relationship has a `<multiplicity>` of `Many`, its `<cmr-field>` is a collection, and you must specify its `<cmr-field-type>` as `java.util.Collection`, as shown in the `StudentEJB` side of the relationship in Figure 5-11. It is not necessary to specify the `cmr-field-type` when the `cmr-field` is a single valued object.

Table 5-2 lists the contents of `cmr-field` for each bean in a relationship, based on the cardinality of the relationship.

Table 5-2 Cardinality and cmr-field-type

If relationship between EJB1 and EJB2 is...	EJB1's cmr-field contains ...	EJB2's cmr-field contains is a ...
one-to-one	single valued object	single valued object
one-to-many	single valued object	Collection
many-to-many	Collection	Collection

Specifying Relationship Directionality

The directionality of a CMR by configured by the inclusion (or exclusion) of a `cmr-field` in the `ejb-relationship-role` stanza for each side of the relationship.

A bidirectional CMR has a `cmr-field` element in the `ejb-relationship-role` stanza for both sides of the relationship, as shown in Figure 5-11.

A unidirectional relationship has a `cmr-field` in only one of the role stanzas for the relationship. The `ejb-relationship-role` for the starting EJB contains a `cmr-field`, but the role stanza for the target bean does not. Figure 5-12 specifies a unidirectional relationship from `MentorEJB` to `StudentEJB`— there is no `cmr-field` element in the `ejb-relationship-role` stanza for `StudentEJB`.

Specifying Relationships in weblogic-cmp-jar.xml

Each CMR defined in `ejb-jar.xml` must also be defined in a `weblogic-rdbms-relation` stanza in `weblogic-cmp-jar.xml`. `weblogic-rdbms-relation` identifies the relationship, and maps the database-level relationship between the participants in the relationship, for one or both sides of the relationship.

The `relation-name` in `weblogic-rdbms-relation` must be the same as the `ejb-relation-name` for the CMR in `ejb-jar.xml`.

One-to-One and One-to-Many Relationships

For one-to-one and one-to-many relationships, `column-map` is defined for only one side of the relationship.

For one-to-one relationships, the mapping is from a foreign key in one bean to the primary key of the other.

Figure 5-13 is the `weblogic-rdbms-relation` stanza for a the one-to-one relationship between `MentorEJB` and `StudentEJB`, whose `<ejb-relation>` is shown in Figure 5-12.

Figure 5-13 One-to-One CMR weblogic-cmp-jar.xml

```
<weblogic-rdbms-relation>
   <relation-name>MentorEJB-StudentEJB</relation-name>
   <weblogic-relationship-role>
       <relationship-role-name>
       mentor-has-student
       <column-map>
          <foreign-key-column>student</foreign-key-column>
          <key-column>StudentID/key-column>
       </column-map>
    </weblogic-relationship-role>
```

For one-to-many relationships, the mapping is also always from a foreign key in one bean to the primary key of another. In a one-to-many relationship, the foreign key is always associated with the bean that is on the many side of the relationship.

Figure 5-14 is the `weblogic-rdbms-relation` stanza for a the one-to-many relationship between `TeacherEJB` and `StudentEJB`, whose `<ejb-relation>` is shown in Figure 5-11.

Figure 5-14 weblogic-rdbms-relation for a One-to-Many CMR

```
<weblogic-rdbms-relation>
   <relation-name>TeacherEJB-StudentEJB</relation-name>
   <weblogic-relationship-role>
       <relationship-role-name>
       teacher-has-student
       </relationship-role-name>
       <column-map>
           <foreign-key-column>student</foreign-key-column>
           <key-column>StudentID/key-column>
        </column-map>
   </weblogic-relationship-role>
```

Many-to-Many Relationships

For many-to-many relationships, specify a `weblogic-relationship-role` stanza for each side of the relationship. The mapping involves a join table. Each row in the join table contains two foreign keys that map to the primary keys of the entities involved in the relationship. The direction of a relationship does not affect how you specify the database mapping for the relationship.

Figure 5-15 shows the `weblogic-rdbms-relation` stanza for the `friends` relationship between two employees.

The `FRIENDS` join table has two columns, `first-friend-id` and `second-friend-id`. Each column contains a foreign key that designates a particular employee who is a friend of another employee. The primary key column of the employee table is `id`.

Figure 5-15 weblogic-rdbms-relation for a Many-to-Many CMR

```
<weblogic-rdbms-relation>
  <relation-name>friends</relation-name>
  <table-name>FRIENDS</table-name>
  <weblogic-relationship-role>
    <relationship-role-name>first-friend
    </relationship-role-name>
      <column-map>
        <foreign-key-column>first-friend-id</foreign-key-column>
        <key-column>id</key-column>
     </column-map
  <weblogic-relationship-role>
    <weblogic-relationship-role>
      <relationship-role-name>second-friend</relationship-role-
      name>
```

```
      <column-map>
        <foreign-key-column>second-friend-id</foreign-key-column>
        <key-column>id</key-column>
      </column-map>
   </weblogic-relationship-role>
</weblogic-rdbms-relation>
```

Container-Managed Relationships and Caching

In versions of WebLogic Server up to and including WebLogic Server 6.1, loading a bean that participates in a container-managed relationship into the cache does not cause related instances to be loaded into cache.

Consider the following EJB code for `accountBean` and `addressBean`, which have a 1-to-1 relationship:

```
Account acct = acctHome.findByPrimaryKey("103243");
Address addr = acct.getAddress();
```

Execution of this code results in two database queries. The first line results in an SQL query to load `accountBean`. The second line results in another query to load `addressBean`.

Groups

In container-managed persistence, you use groups to specify certain persistent attributes of an entity bean. A field-group represents a subset of the cmp and CMR-fields of a bean. You can put related fields in a bean into groups that are faulted into memory together as a unit. You can associate a group with a query or relationship, so that when a bean is loaded as the result of executing a query or following a relationship, only the fields mentioned in the group are loaded.

A special group named “default” is used for queries and relationships that have no group specified. By default, the default group contains all of a bean's CMP-fields and any CMR-fields that add a foreign key to the persistent state of the bean.

A field can belong to multiple groups. In this case, the `getXXX()` method for the field will fault in the first group that contains the field.

Specifying Field Groups

Field groups are specified in the `weblogic-rdbms-cmp-jar.xml` file as follows:

```
<weblogic-rdbms-bean>
       <ejb-name>XXXBean</ejb-name>
       <field-group>
              <group-name>medical-data</group-name>
              <cmp-field>insurance</cmp-field>
              <cmr-field>doctors</cmr-fields>
       </field-group>
</weblogic-rdbms-bean>
```

You use field groups when you want to access a subset of fields.

Java Data Types for CMP Fields

The following table provides a list of the Java data types for CMP fields used in WebLogic Server and shows how they map to the Oracle extensions for the standard SQL data types.

Table 5-3 Java data types for CMP fields

Java Types for CMP Fields	Oracle Data Types
boolean	**SMALLINT**
byte	**SMALLINT**
char	**SMALLINT**
double	**NUMBER**
float	**NUMBER**
int	**INTEGER**
long	**NUMBER**
short	**SMALLINT**

Java Types for CMP Fields	Oracle Data Types
java.lang.String	**VARCHAR/VARCHAR2**
java.lang.Boolean	**SMALLINT**
java.lang.Byte	**SMALLINT**
java.lang.Character	**SMALLINT**
java.lang.Double	**NUMBER**
java.lang.Float	**NUMBER**
java.lang.Integer	**INTEGER**
java.lang.Long	**NUMBER**
java.lang.Short	**SMALLINT**
java.sql.Date	**DATE**
java.sql.Time	**DATE**
java.sql.Timestamp	**DATE**
java.math.BigDecimal	**NUMBER**
byte[]	**RAW, LONG RAW**
serializable	**RAW, LONG RAW**

Do not use the SQL CHAR data type for database columns that are mapped to CMP fields. This is especially important for fields that are part of the primary key, because padding blanks that are returned by the JDBC driver can cause equality comparisons to fail when they should not. Use the SQL VARCHAR data type instead of SQL CHAR.

A CMP field of type `byte[]` cannot be used as a primary key unless it is wrapped in a user-defined primary key class that provides meaningful `equals()` and `hashCode()` methods. This is because the `byte[]` class does not provide useful equals and hashCode.

6 Packaging EJBs for the WebLogic Server Container

The following sections describe how to package EJBs into a WebLogic Server container for deployment. They includes a description of the contents of a deployment package, including the source files, deployment descriptors, and the deployment mode.

Required Steps for Packaging EJBs

Packaging EJBs for deployment to WebLogic Server in an EJB container involves the following steps:

1. Review the EJB source file components.
2. Create the EJB deployment files.
3. Edit the EJB deployment descriptors.
4. Set the deployment mode.
5. Generate the EJB container classes.
6. Package the EJBs into a JAR or EAR file.
7. Load EJB classes into WebLogic Server.

Reviewing the EJB Source File Components

To implement entity and session beans, use the following components:

Component	Description
Bean Class	The *bean class* implements the bean's business and life cycle methods.
Remote Interface	The *remote interface* defines the beans's business logic that can be access from applications outside of the bean's EJB container.
Remote Home Interface	The *remote home interface* defines the bean's file cycle methods that can be accessed from applications outside of the bean's EJB container.
Local Interface	The *local interface* defines the bean's business methods that can be used by other beans that are co-located in the same EJB container.

Component	Description
Local Home Interface	The *local home interface* defines the bean's life cycle methods that can be used by other beans that are co-located in the same EJB container.
Primary Key	The *primary key class* provides a pointer into the database. Only entity beans need a primary key.

WebLogic Server EJB Deployment Files

Use the following WebLogic Server deployment files to specify the deployment descriptor elements for the EJB.

- `ejb-jar.xml`
- `weblogic-ejb-jar.xml`
- `weblogic-cmp-rdbms-jar.xml` (optional, for CMP only)

The deployment files become part of the EJB deployment when the bean is compiled. The XML deployment descriptor files should contain the minimum deployment descriptor settings for the EJB. Once the file exists, it can later be edited using the instructions in "Specifying and Editing the EJB Deployment Descriptors" on page 6-5. The deployment descriptor files must conform to the correct version of the Document Type Definition (DTD) for each file you use. All element and sub element (attribute) names for each of the EJB XML deployment descriptor files are described in the file's Document Type Definition (DTD) file. For a description of each file, see the following sections.

ejb-jar.xml

The `ejb-jar.xml` file contains the Sun Microsystem-specific EJB DTD. The deployment descriptors in this file describe the enterprise bean's structure and declares its internal dependences and the application assembly information, which describes

how the enterprise bean in the `ejb-jar` file is assembled into an application deployment unit. For a description of the elements in this file, see the JavaSoft specification.

weblogic-ejb-jar.xml

The `weblogic-ejb-jar.xml` file contains the WebLogic Server-specific EJB DTD that defines the caching, clustering, and performance behavior of EJBs. It also contains descriptors that map available WebLogic Server resources to EJBs. WebLogic Server resources include security role names and data sources such as JDBC pools, JMS connection factories, and other deployed EJBs. For a description of the elements in this file, see Chapter 10, "weblogic-ejb-jar.xml Document Type Definitions."

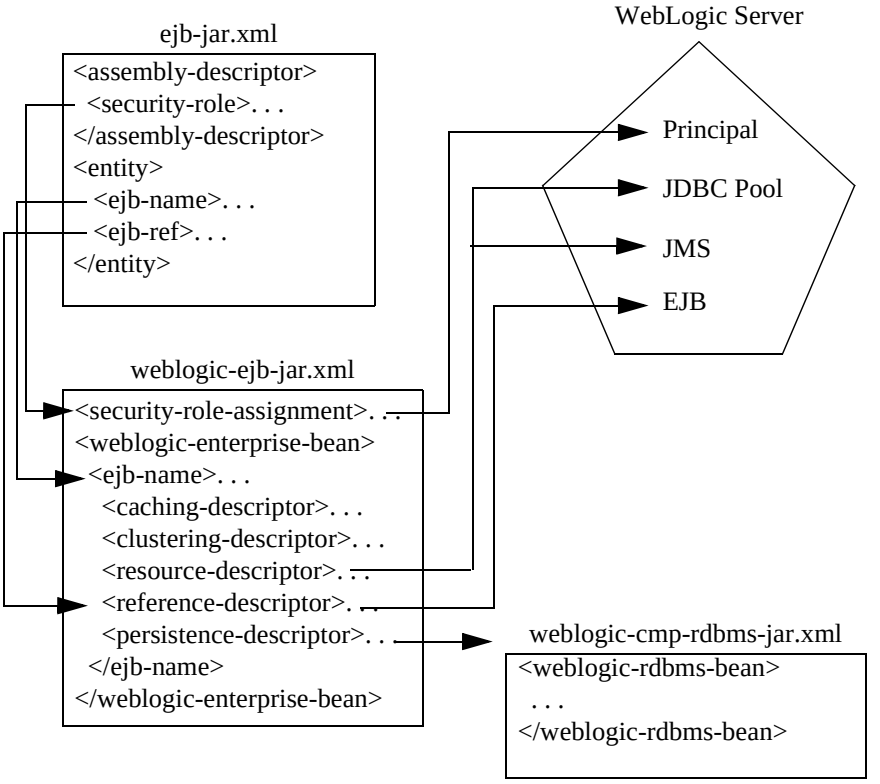
weblogic-cmp-rdbms.xml

The `weblogic-cmp-rdbms.xml` file contains the WebLogic Server-specific EJB DTD that defines container-managed persistence services. Use this file to specify how the container handles synchronizing the entity beans's instance fields with the data in the database. For a description of the elements in this file, see Chapter 11, "weblogic-cmp-rdbms- jar.xml Document Type Definitions."

Relationships Among the Deployment Files

Descriptors in `weblogic-ejb-jar.xml` are linked to EJB names in `ejb-jar.xml`, to resource names in a running WebLogic Server, and to persistence type data defined in `weblogic-cmp-rdbms-jar.xml` (for entity EJBs using container-managed persistence). The following diagram shows the relationship among the deployment files and WebLogic Server.

Figure 6-1 The relationship among the components of the deployment files.



Specifying and Editing the EJB Deployment Descriptors

You specify or edit EJB deployment descriptors by any of the following methods:

- Using a text editor to manually editing the bean's deployment files. For instructions on manually editing the deployment files, see "Manually Editing EJB Deployment Descriptors" on page 6-6.

- Using the EJB Deployment Descriptor Editor in the WebLogic Server Administration Console to edit the bean's deployment files. For instructions on using the EJB Deployment Descriptor Editor, see "Using the EJB Deployment Descriptor Editor" on page 6-7.
- Using a WebLogic Server command line utility tool called `DDConverter` to convert EJB 1.1 deployment descriptors to EJB 2.0 XML. For instructions on using the `DDConverter` tool, see "DDConverter" on page 9-4.

Creating the Deployment Files

You create the basic XML deployment files for the EJB that conforms to the correct version of the Document Type Definition (DTD) for each file. You can use an existing EJB deployment file as a template or copy one from the EJB examples in your WebLogic Server distribution:

```
wlserver\samples\examples\ejb20
```

Manually Editing EJB Deployment Descriptors

To edit XML deployment descriptor elements manually:

1. Use an ASCII text editor that does not reformat the XML or insert additional characters that could invalidate the file.
2. Open the XML deployment descriptor file that you want to edit.
3. Type in your changes. Use the correct case for file and directory names, even if your operating system ignores the case.
4. To use the default value for an optional element, either omit the entire element definition or specify a blank value, as in:

```
<max-beans-in-cache></max-beans-in-cache>
```

Using the EJB Deployment Descriptor Editor

To edit the EJB deployment descriptors in the WebLogic Server Administration Console:

1. Start WebLogic Server.
2. Start the Administration Console and select `EJB` from the right pane.
3. In the left pane, choose the `Deployments` node under your server domain.
4. Expand the `Deployments` node and choose `EJB`.
5. From the expanded list of deployed EJBs, right-click on the bean to be edited.
6. Click `Edit EJB Descriptor...`
7. When the EJB Deployment Descriptor Editor is displayed, click the chosen EJB to expand the node.

   You should see the following items that represent the EJB deployment descriptor files:

   - **EJB Jar:** represents the `ejb-jar.xml` file deployment descriptors for this EJB.
   - **WebLogic EJB Jar:** represents the `weblogic-ejb-jar.xml` file deployment descriptors for this EJB.
   - **CMP:** represents the `weblogic-cmp-rdbms-jar.xml` file deployment descriptors for this EJB.
8. Expand the node for the deployment descriptors that you want to edit.

   The current settings for the deployment descriptor file that you selected appear in the left pane. When you right-click on an item in the list, a dialog window for that item appears in the right pane.
9. Clicking on the circles displays a dialog window in the right pane with various settings.

   You can change the settings in the dialog window to edit those deployment descriptors.
10. Clicking on the folders displays tables in the right pane where you can view your settings.

Here you can usually configure a new descriptor or customize your view of the existing settings. If an item in the table is underlined, you can click on it to display a dialog where you can change the settings.

11. By right-clicking on deployment descriptor items in the right pane, you can also delete descriptors.

Note: For more information on the EJB deployment descriptors, see either the online help in the Administration Console or Chapter 10, "weblogic-ejb-jar.xml Document Type Definitions," and Chapter 11, "weblogic-cmp-rdbms- jar.xml Document Type Definitions."

Setting WebLogic Server Deployment Mode

You deploy the enterprise archive file (EAR) or EJB to WebLogic Server by one of the following methods:

- Automatic mode, which automatically deploys the EJB or EAR to the applications directory on your server
- Production mode, which deploys the EJB or EAR as specified in the `config.xml` file.

Using the Automatic Mode for Deployment

The automatic mode deployment option is the default. This feature automatically polls the application directory of the active server during startup and while the server is running, to determine whether an EJB deployment has changed. If a deployment has changed, it is automatically deployed when the server is polled. Use the `applications` directory for EJBs or EARs that you want to deploy in development mode. Once deployed, these EJBs/EARs are automatically persisted to the `config.xml` file.

WebLogic Server also checks the contents of `applications` every ten seconds to determine whether an EJB deployment has changed. If a deployment has changed, it is automatically redeployed using the dynamic deployment feature.

Automatically Deploying the EJB Examples

The EJB examples shipped with WebLogic Server are automatically deployed in the `wlserver/config/applications` directory.

- The examples in the `wlserver/samples/examples/ejb` directory are shipped built and automatically deploy when the `examples` server is started.
- The examples in the `wlserver/samples/examples/ejb20` directory are shipped pre-built and need to be built before they can be deployed to the server.

Using the Production Mode for Deployment

The production mode deployment option turns off automatic deployment. When production deployment mode is enabled, applications specified in the `config.xml` file are deployed when the server is started.

To enable this mode, at the command line set the following to true:

```
-d production mode enabled true
```

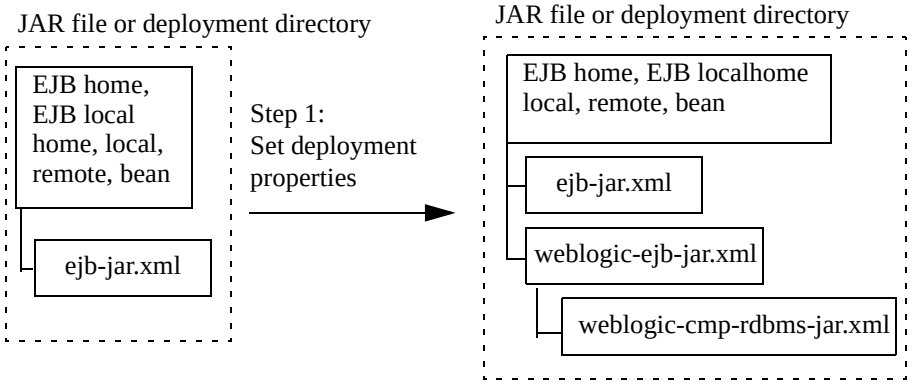
For more information on this production mode, see "Starting the WebLogic Administration Server from the Command Line" in `startstop.html` of the *Administration Guide*.

Packaging EJBs into a Deployment Directory

The deployment process begins with a JAR file or a deployment directory that contains the compiled EJB interfaces and implementation classes created by the EJB provider. Regardless of whether the compiled classes are stored in a JAR file or a deployment directory, they must reside in subdirectories that match their Java package structures.

The EJB provider should also supply an EJB compliant `ejb-jar.xml` file that describes the bundled EJB(s). The `ejb-jar.xml` file and any other required XML deployment file must reside in a top-level `META-INF` subdirectory of the JAR or deployment directory. The following diagram shows the first stage of packaging the the EJB and the deployment descriptor files into a deployment directory or JAR file.

Figure 6-2 Packaging the EJB classes and deployment descriptors into a deployment directory



As is, the basic JAR or deployment directory *cannot* be deployed to WebLogic Server. You must first create and configure the WebLogic-specific deployment descriptor elements in the `weblogic-ejb-jar.xml` file, and add that file to the deployment directory or `ejb.jar` file. For more information on creating the deployment descriptor files, see "WebLogic Server EJB Deployment Files" on page 6-3.

If you are deploying an entity EJB that uses container-managed persistence, you must also add the WebLogic -specific deployment descriptor elements for the bean's persistence type. For WebLogic Server container-managed persistence (CMP) services, the file is generally named `weblogic-cmp-rdbms-jar.xml`. You require a separate file for each bean that uses CMP. If you use a third-party persistence vendor, the file type as well as its contents may be different from `weblogic-cmp-rdbms-jar.xml`; refer to your persistence vendor's documentation for details.

If you do not have any of the deployment descriptor files needed for your EJB, you must manually create one. The best method is to copy an existing file and edit the settings to conform to the needs of your EJB. Use the instructions in "Specifying and Editing the EJB Deployment Descriptors" on page 6-5 to create the files.

ejb.jar file

You create the `ejb.jar` file with the Java Jar utility (javac). This utility bundles the EJB classes and deployment descriptors into a single Java ARchive (JAR) file that maintains the directory structure. The `ejb-jar` file is the unit that you deploy to WebLogic Server.

Compiling EJB Classes and Generating EJB Container Classes

For part of the process of building your deployment unit, you need to compile your EJB classes, add your deployment descriptors to the deployment unit, and generate the container classes used to access the deployment unit.

1. Compile the EJB classes using `javac` compiler from the command line.
2. Add the appropriate XML deployment descriptor files to the compiled unit using the guidelines in "WebLogic Server EJB Deployment Files" on page 6-3.
3. Generate the container classes that are used to access the bean using `ejbc`.

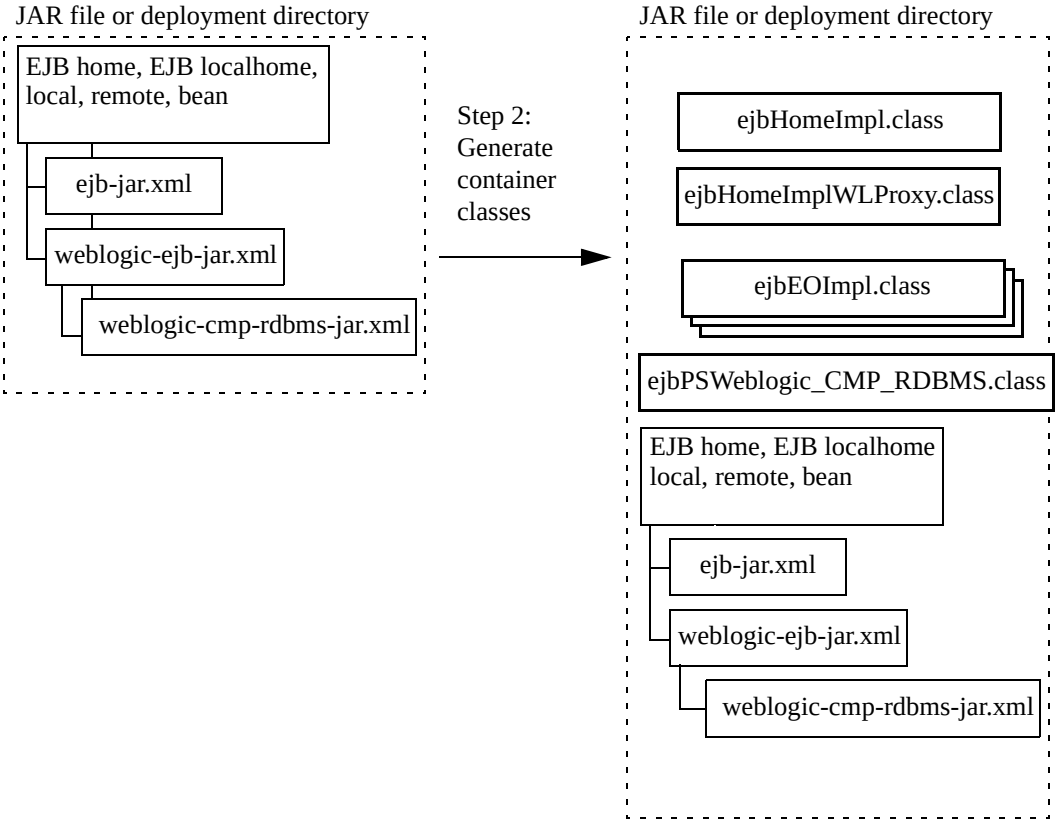
   Container classes include both the internal representation of the EJB that WebLogic Server uses, as well as implementation of the external interfaces (home, local, and/or remote) that clients use.

The `ejbc` compiler generates container classes according to the deployment descriptors you have specified in WebLogic-specific XML deployment descriptor files. For example, if you indicate that your EJBs will be used in a cluster, `ejbc` creates special cluster-aware classes that will be used for deployment.

You can also use `ejbc` directly from the command line by supplying the required options and arguments. See "ejbc" on page 9-1 for more information.

The following figure shows the container classes added to the deployment unit when the JAR file is generated

Figure 6-3 Generating the EJB container classes



Once you have generated the deployment unit, you can designate the file extension as either a JAR, EAR, or WAR archive.

Loading EJB Classes into WebLogic Server

Classloaders in Weblogic Server are hierarchical. When you start WebLogic Server, the Java system classloader is active and is the parent of all subsequent classloaders that WebLogic Server creates. When WebLogic Server deploys an application, it

automatically creates two new classloaders: one for EJBs and one for Web applications. The EJB classloader is a child of the Java system classloader and the Web application classloader is a child of the EJB classloader.

For more information on classloading, see "Classloader Overview" and "About Application Classloaders" in *Developing WebLogic Server Applications*.

Specifying an ejb-client.jar

WebLogic Server supports the use of `ejb-client.jar` files.

The `ejb-client.jar` contains the home and remote interfaces, the primary key class (as applicable), and the files they reference. WebLogic Server does not add files referenced in your classpath to `ejb-client.jar`. This enables WebLogic Server to add necessary custom classes to the `ejb-client.jar` without adding generic classes such as `java.lang.String`.

For example, the `ShoppingCart` remote interface might have a method that returns an `Item` class. Because this remote interface references this class, and it is located in the `ejb-jar` file, it will be included in the client jar.

You configure the creation of an `ejb-client.jar` file in the bean's `ejb-jar.xml` deployment descriptor file.When you compile the bean with `ejbc`, WebLogic Server creates the `ejb-client.jar`.

To specify an `ejb-client.jar`:

1. Compile the bean's Java classes into a directory, using the `javac` compiler from the command line.
2. Add the EJB XML deployment descriptor files to the compiled unit using the guidelines in "WebLogic Server EJB Deployment Files" on page 6-3.
3. Edit the `ejb-client-jar` deployment descriptor in the bean's `ejb-jar.xml` file, as follows, to specify support for `ejb-client.jar`:

   ```
   <ejb-client-jar>ShoppingCartClient.jar</ejb-client-jar>
   ```

4. Generate the container classes that are used to access the bean using `weblogic.ejbc` and create the `ejb-client.jar` using the following command:

```
$ java weblogic.ejbc <ShoppingCart.jar>
```

Container classes include both the internal representation of the EJB that WebLogic Server uses, as well as implementation of the external interfaces (home, local, and/or remote) that clients use.

External clients can include the `ejb-client.jar` in their classpath. Web applications would include the `ejb-client.jar` in their `/lib` directory.

Note: WebLogic Server classloading behavior varies, depending on whether or not the client is stand-alone. Stand-alone clients with access to the `ejb-client.jar` can load the necessary classes over the network. However, for security reasons, programmatic clients running in a server instance cannot load classes over the network.

Manifest Class-Path

Use the manifest file to specify that a JAR file can reference another JAR file. Standalone EJBs cannot use the Manifest Class-Path. It is only supported for components that are deployed within an EAR file. The clients should reference the `client.jar` in the classpath entry of the manifest file.

To use the manifest file to reference another JAR file:

1. Specify the name of the referenced JAR file in a Class-Path header in the referencing JAR file's Manifest file.

   The referenced JAR file is named using a URL relative to the URL of the referencing JAR file.

2. Name the manifest file META-INF/MANIFEST.MF in the JAR file

3. The Class-Path entry in the Manifest file is as follows:

   ```
   Class-Path: AAyy.jar BByy.jar CCyy.jar.
   ```

Note: The entry is a list of JAR files separated by spaces.

To place the home/remote interfaces for the EJB in the classpath of the calling component:

1. Use `ejbc` to compile the EJB into a JAR file.
2. Create a `client.jar` file. For instructions on using the `client.jar`, see "Specifying an ejb-client.jar" on page 6-13.
3. Place the `client.jar`, along with all the clients of the bean in an EAR.
4. Reference the EAR in the manifest file.

.

CHAPTER

7 Deploying EJBs to WebLogic Server

The following sections provides instructions for deploying EJBs to WebLogic Server at WebLogic Server startup or on a running WebLogic Server.

Roles and Responsibilities

The following sections are written primarily for:

- Deployers who configure EJBs to run in the WebLogic Server container
- Application assemblers who link multiple EJBs and EJB resources to create larger Web application systems

- EJB developers who create and configure new EJB JAR files

You can create, modify, and deploy EJBs in one or more instance of WebLogic Server. You can set up your EJB deployment, and map EJB references to actual resource factories, roles, and other EJBs available on a server by editing the XML deployment descriptor files.

Deploying EJBs at WebLogic Server Startup

To deploy EJBs automatically when WebLogic Server starts:

1. Follow the instructions in to ensure that your deployable EJB JAR file or deployment directory contains the required WebLogic Server XML deployment files.
2. Use a text editor or the EJB Deployment Descriptor Editor in the Administration Console to edit the XML deployment descriptor elements, as necessary.
3. Follow the instructions in "Compiling EJB Classes and Generating EJB Container Classes" on page 6-11 to compile implementation classes required for WebLogic Server.

   Compiling the container places the JAR file in the deployment directory specified in the deployment descriptors. If you want the EJB to automatically deploy when WebLogic Server starts, place the EJB be deployed to the following directory:

   `wlserver/config/mydomain/applications`

   If your EJB JAR file is located in a different directory, make sure that you copy it to this directory if you want to deploy it at startup.
4. Start WebLogic Server.

   When you boot WebLogic Server, it automatically attempts to deploy the specified EJB JAR file or deployment directory.
5. Launch the Administration Console.
6. In the right pane, click EJB Deployments.

   A list of the EJB deployments for the server displays in the right-hand pane.

Deploying EJBs in Different Applications

When you deploy EJBs with remote calls to each other different applications, you cannot use `call-by-reference` to invoke the EJBs. Instead, you use `pass-by-value`. Components that commonly interact with each other should be located in the same application where `call-by-reference` can be used. By default, EJB methods called from within the same server pass arguments by reference. This increases the performance of method invocation because parameters are not copied. `Pass-by-value` is always necessary when EJBs are called remotely (not from within the server).

Deploying EJBs on a Running WebLogic Server

Although placing the EJB JAR file or deployment directory in the `wlserver/config/mydomain/applications` directory allows the EJB to be immediately deployed, if you make a change to the deployed EJB, you must redeploy the EJB for the changes to take effect.

Automatic deployment is provided for situations where rebooting WebLogic Server is not feasible and is for development purposes only. Using automatic deployment only deploys the updated EJB to the Administration Server and does not deploy the EJB to any Managed Server on the domain. Using automatic deployment features, you can:

- Deploy a newly developed EJB to a running development system
- Remove a deployed EJB to restrict access to data
- Update a deployed EJB implementation class to fix a bug or test a new feature

Whether you deploy or update the EJB from the command line or the Administration Console, you use the automatic deployment features. The following sections describe automatic deployment concepts and procedures.

EJB Deployment Names

When you deploy an EJB JAR file or deployment directory, you specify a name for the deployment unit. This name is a shorthand reference to the EJB deployment that you can later use to undeploy or update the EJB.

When you deploy an EJB, WebLogic Server implicitly assigns a deployment name that matches the path and filename of the JAR file or deployment directory. You can use this assigned name to undeploy or update the bean after the server has started.

Note: The EJB deployment name remains active in WebLogic Server until the server is rebooted. Undeploying an EJB does not remove the associated deployment name, because you may later re-use that name to deploy the bean.

Deploying New EJBs into a Running Environment

To deploy an EJB JAR file or deployment directory that has not been deployed to WebLogic Server:

Use the command:

```
% java weblogic.deploy -port port_number -host host_name
       deploy password name source
```

where:

- *name* is the string you want to assign to this EJB deployment unit
- *source* is the full path and filename of the EJB JAR file you want to deploy, or the full path of the EJB deployment directory

For example:

```
% java weblogic.deploy -port 7001 -host localhost deploy
       weblogicpwd CMP_example
       c:\weblogic\myserver\unjarred\containerManaged\
```

Deploying Pinned EJBs - Special Step Required

There is a known issue with deploying or redeploying EJBs to a single server instance in a cluster—referred to as pinned deployment—if the `.jar` file contains contain uncompiled classes and interfaces.

During deployment, the uncompiled EJB is copied to each server instance in the cluster, but it is compiled only on the server instance to which it has been deployed. As a result, the server instances in the cluster to which the EJB was not targeted lack the classes generated during compilation that are necessary to invoke the EJB. When a client on another server instance tries to invoke the pinned EJB, it fails, and an `Assertion` error is thrown in the RMI layer.

If you are deploying or redeploying an EJB to a single server instance in a cluster, compile the EJB with `appc` or `ejbc` before deploying it, , to ensure that the generated classes copied to all server instances available to all nodes in the cluster.

Viewing Deployed EJBs

To view deployed EJBs:

- From the command line:

1. To list the EJBs that are deployed on a local WebLogic Server, enter the following:

```
% java weblogic.deploy list password
```

where *`password`* is the password for the WebLogic Server System account.

2. To list deployed EJBs on a remote server, specify the `port` and `host` options as follows:

```
% java weblogic.deploy -port port_number -host host_name
      list password
```

- From the WebLogic Server Administration Console:

1. Choose EJB from the Deployments node in the left pane of the Console.
2. View a list of deployed EJBs deployed on the server.

Undeploying Deployed EJBs

Undeploying an EJB effectively prohibits all clients from using the EJB. When you undeploy the EJB, the specified EJB's implementation class is immediately marked as unavailable in the server. WebLogic Server automatically removes the implementation class and propagates an `UndeploymentException` to all clients that were using the bean.

Undeployment does not automatically remove the specified EJB's public interface classes. Implementations of the home interface, remote interface, and any support classes referenced in the public interfaces, remain in the server until all references to those classes are released. At that point, the public classes may be removed due to normal Java garbage collection routines.

Similarly, undeploying an EJB does not remove the deployment name associated with the `ejb.jar` file or deployment directory. The deployment name remains in the server to allow for later updates of the EJB.

Undeploying EJBs

To undeploy a deployed EJB, use the following steps:

From the command line:

Reference the assigned deployment unit name, as in:

```
% java weblogic.deploy -port 7001 -host localhost undeploy
      weblogicpwd CMP_example
```

From the WebLogic Server Administration Console:

1. Choose EJB from the Deployments node in the left pane of the Console.
2. Click the EJB you want to undeploy from the list.
3. Choose the Configuration tab from the dialog in the right pane and uncheck the undeploy box.

Undeploying an EJB does not remove the EJB deployment name from WebLogic Server. The EJB remains undeployed for the duration of the server session, as long as you do not change it once it had been undeployed. You cannot re-use the deployment name with the `deploy` argument until you reboot the server. You can re-use the deployment name to `update` the deployment, as described in the following section.

Updating Deployed EJBs

When you update the contents of an `ejb.jar` file or deployment directory that has been deployed to WebLogic Server, those updates are not reflected in WebLogic Server until:

- You reboot the server (if the JAR or directory is to be automatically deployed), or
- You *update* the EJB deployment using the WebLogic ServerAdministration Console.

Updating an EJB deployment enables an EJB provider to make changes to a deployed EJB's implementation classes, recompile, and then "refresh" the implementation classes in a running server.

weblogic.deploy update and Targets

In WebLogic Server 6.1, updating an application on any single server instance to which it is targeted causes it to be updated on all servers to which is targeted. For instance, if an application is targeted to a cluster, and you update it on one of the clustered servers instances, the application will be updated on all members of cluster. Similarly, if the application is targeted to a cluster and to a standalone server instance, updating it on the standalone server instance will result in its update on the cluster, and vice versa.

If you want to be able to update an application or component selectively on a subset of the server instances to which it is targeted, deploy unique instances to of the application to different targets.

The Update Process

When you update the currently-loaded implementation, classes for the EJB are immediately marked as unavailable in the server, and the EJB's classloader and associated classes are removed. At the same time, a new EJB classloader is created, which loads and maintains the revised EJB implementation classes.

The entire EJB will be reloaded; you cannot redeploy part of the EJB JAR..

When clients next acquire a reference to the EJB, their EJB method calls use the updated EJB implementation classes.

Note: You can update only the EJB implementation classes, as described in "Loading EJB Classes into WebLogic Server" on page 6-12. You cannot update the EJB's public interfaces, or any support classes that are used by the public interfaces. If you make any changes to the EJB's public classes and attempt to update the EJB, WebLogic Server displays an incompatible class change error when a client next uses the EJB instance.

Updating the EJB

To update or redeploy the EJB implementation class, use the following steps:

From the command line:

Use the `update` argument and specify the active EJB deployment name:

```
% java weblogic.deploy -port 7001 -host localhost update
       weblogicpwd CMP_example
```

From the WebLogic Server Administration Console:

1. Choose EJB from the Deployments node in the left pane of the Console.
2. Click the EJB you want to update from the list.
3. Choose the Configuration tab from the dialog in the right pane and update the EJB by checking the deployed box.

You can update only the EJB implementation class, not the public interfaces or public support classes

Deploying Compiled EJB Files

To create compiled EJB 2.0 JAR or EAR files:

1. Compile your EJB classes and interfaces using `javac`.
2. Package the EJB classes and interfaces into a valid JAR or EAR file.
3. Use the `weblogic.ejbc` compiler on the JAR file to generate WebLogic Server container classes. For instructions on using ejbc, see "ejbc" on page 9-1.

To create compile EJBs from previous versions of WebLogic Server:

1. Run `weblogic.ejbc` against the `ejb.jar` file to generate EJB 2.0 container-classes.
2. Copy the compiled `ejb.jar` files into `wlserver/config/mydomain/applications`.

If you change the contents of a compiled `ejb.jar` file in `applications` (by repackaging, recompiling, or copying over the existing `ejb.jar`), WebLogic Server automatically attempts to redeploy the `ejb.jar` file using the automatic deployment feature.

Note: Because the automatic redeployment feature uses dynamic deployment, WebLogic Server can only redeploy an EJB's implementation classes. You cannot redeploy an EJB's public interfaces.

Deploying Uncompiled EJB Files

The WebLogic Server container also enables you to automatically deploy JAR files that contain uncompiled EJB classes and interfaces. An uncompiled EJB file has the same structure as a compiled file, with the following exceptions:

- You do not have to compile individual class files and interfaces.
- You do not have to use `weblogic.ejbc` on the packaged JAR file to generate WebLogic Server container classes.

The `.java` or `.class` files in the JAR file must still be packaged in subdirectories that match their Java package hierarchy. Also, as with all `ejb.jar` files, you must include the appropriate XML deployment files in a top-level `META-INF` directory.

After you package the uncompiled classes, simply copy the JAR into the `wlserver\config\mydomain\applications` directory. If necessary, WebLogic Server automatically runs `javac` (or a compiler you specify) to compile the `.java` files, and runs `weblogic.ejbc` to generate container classes. The compiled classes are copied into a new JAR file in `wlserver/config/mydomain/applications`, and deployed to the EJB container.

Should you ever modify an uncompiled ejbc `.jar` in the `applications` directory (either by repackaging or copying over the JAR file), WebLogic Server automatically recompiles and redeploys the JAR using the same steps.

Note: Because the automatic redeployment feature uses dynamic deployment, WebLogic Server can only redeploy an EJB's implementation classes. You cannot redeploy an EJB's public interfaces.

8 Configuring Security in EJBs

You can secure EJBs by restricting access to them. To restrict access to specified EJBs, apply security constraints to them.

Configuring Security Constraints

To figure security constraints, follow these steps:

1. Follow the directions in `weblogic\examples\ejb\basic\containerManaged\index.html` to set your environment.
2. Add the following to the bottom of the session stanza for the bean, after `<transaction-type>`:

```
<security-role-ref>
    <role-name>admin</role-name>
    <role-link<admin</role-link>
</security-role-ref>
```

3. Add the following to your `ejb-jar.xml` at the top of the `<assembly-descriptor>` stanza to specify which roles have access to your EJB methods:

```
<security-role>
<description></description>
    <role-name>admin</role-name>
</security-role>
```

```
<method-permission>
    <role-name>admin</role-name>
    <method>
           <ejb-name>containerManaged</ejb-name>
           <method-name>*</method-name>
    </method>
</method-permission>
```

4. Add the following to your `weblogic-ejb-jar.xml` at the end of the `weblogic-ejb-jar` stanza to map the role name to specific users and groups in your security realm:

```
<security-role-assignment>
  <role-name>admin</role-name>
         <principal-name>Accounting Managers</principal-name>
         <principal-name>HR Managers</principal-name>
         <principal-name>system</principal-name>
</security-role-assignment>
```

Note: Note that principals can be either users or groups in your security realm.

5. Use the build script to rebuild the bean.

Note: If something concerning EJBs was fixed in a service pack, you will need to add the service pack jar file to the front of the classpath in the build script in order to take advantage of the fix.

6. Modify the `Client.java` to use user and credential when programming the `InitialContext`.

7. Run the client by invoking this command:

```
java examples.ejb.basic.containerManaged.Client
"t3://WebLogicURL:Port" user password
```

Parameters are optional, but if any are supplied, they are interpreted in this order:

a. url - URL such as "t3://localhost:7001" of Server user

b. user - User name, default null

c. password - User password, default null accountID - String Account ID to test, default "10020"

9 WebLogic Server EJB Utilities

The following sections provide a complete reference to the utilities and support files supplied with WebLogic Server EJBs:

- ejbc (`weblogic.ejbc`)
- DDConverter (`weblogic.ejb.utils.DDConverter`)
- deploy (`weblogic.deploy`)

ejbc

Use the `weblogic.ejbc` command-line utility for generate and compiling EJB 2.0 and 1.1 container classes. If you compile JAR files for deployment into the EJB container, you must use `weblogic.ejbc` to generate the container classes.

The WebLogic Server command line utility, `weblogic.ejbc` does the following:

- Places the EJB classes, interfaces, and XML deployment descriptor files in a specified JAR file.
- Checks all EJB classes and interfaces for compliance with the EJB specification.
- Generates WebLogic Server container classes for the EJBs.
- Runs each EJB container class through the RMI compiler to create client-side dynamic proxies and server-side byte code.

If you specify an output JAR file, `ejbc` places all generated files into the JAR file.

By default, `ejbc` uses `javac` as a compiler. For faster performance, specify a different compiler (such as Symantec's `sj`) using the `-compiler` flag.

Note: You may encounter problems deploying EJBs if there is a mismatched version problem with `weblogic.ejbc`. When you start WebLogic Server it checks which version of `weblogic.ejbc` was used to compile the container classes. If the version of `weblogic.ejbc` used to compile the classes is different from the version you are currently running the EJB will not deploy. To avoid this problem, make sure that you do not put unnecessary classes in your class path.

ejbc Syntax

```
$ java weblogic.ejbc [options] <source jar file>
      <target directory or jar file>
```

Note: If you output to a JAR file, the output JAR name must be different from the input JAR name.

ejbc Arguments

The following table lists the `weblogic.ejbc` arguments:

Argument	Description
<source jar file>	Specifies the JAR file containing the compiled EJB classes, interfaces, and XML deployment files.
<target directory or jar file>	Specifies the destination JAR file or deployment directory in which `ejbc` places the output JAR. If you specify an output JAR file, `ejbc` places the original EJB classes, interfaces, and XML deployment files in the JAR, as well as the new container classes that `ejbc` generates.

ejbc Options

The following table lists the `weblogic.ejbc` command-line options:

Option	Description
`-help`	Prints a list of all options available for the compiler.
`-version`	Prints `ejbc` version information.
`-dispatchPolicy <queueName>`	Specifies a configured execute queue that the EJB should use for obtaining execute threads in WebLogic Server. See Using Execute Queues to Control Thread Usage for more information.
`-idl`	Generates CORBA Interface Definition Language for remote interfaces.
`-J`	Specifies the heap size for `weblogic.ejbc`. Use as follows: `java weblogic.ejbc -J-mx256m input.jar output.jar`
`-idlOverwrite`	Overwrites existing IDL files.
`-idlVerbose`	Displays verbose information while generating IDL.
`-idlDirectory <dir>`	Specifies the directory where `ejbc` creates IDL files. By default, `ejbc` uses the current directory.
`-keepgenerated`	Saves the intermediate Java files generated during compilation.
`-compiler <compiler name>`	Sets the compiler for `ejbc` to use.
`-normi`	Passed through to Symantec's java compiler, `sj`, to stop generation of RMI stubs. Otherwise `sj` creates its own RMI stubs, which are unnecessary for the EJB.
`-classpath <path>`	Sets a CLASSPATH used during compilation. This overrides the system or shell CLASSPATH.

ejbc Examples

The following example uses the `javac` compiler against an input JAR file in `c:\wlserver\samples\examples\ejb\basic\containerManaged\build`. The output JAR file is placed in `c:\wlserver\config\examples\applications`.

```
$ java weblogic.ejbc -compiler javac
c:\wlserver\samples\examples\ejb\basic\containerManaged\build\std
_ejb_basic_containerManaged.jar
c:\wlserver\config\examples\ejb_basic_containerManaged.jar
```

The following example checks a JAR file for compliance with the EJB 1.1 specification and generates WebLogic Server container classes, but does not generate RMI stubs:

```
$ java weblogic.ejbc -normi
c:\wlserver\samples\examples\ejb\basic\containerManaged\build\std
_ejb_basic_containerManaged.jar
```

DDConverter

The `DDConverter` is a command line utility that converts earlier versions EJB deployment descriptors into EJB deployment descriptors that conform to the WebLogic Server 6.x version. The WebLogic Server EJB container supports both the EJB 1.1 and EJB 2.0 specifications including the EJB 1.1 and EJB 2.0 document type definitions (DTD). Each WebLogic Server EJB deployment includes standard deployment descriptors in the following files:

- `ejb-jar.xml`

  This XML file contains the Sun Microsoft-specific EJB deployment descriptors.

- `weblogic-ejb-jar-.xml`

  This XML file contains the WebLogic-specific EJB deployment descriptors.

- `weblogic-cmp-rdbms-jar.xml`

  This XML file contains the WebLogic-specific container-managed persistence (CMP) deployment descriptors.

Conversion Options Available with DDConverter

The `DDConverter` command line utility includes the following conversion options:

- Converting beans from earlier versions of WebLogic Server (WLS).
- Converting CMP and non-CMP beans from earlier version of the EJB specification.

The following table lists the various conversion options for the DDconverter:

Conversion Options for the DDConverter Utility

WLS		EJB non-CMP		EJB CMP	
From	**To**	**From**	**To**	**From**	**To**
WLS 4.5	WLS 6.x	See **Note 1**		EJB CMP 1.0	EJB CMP 1.1 See **Note 2**
WLS 4.5	WLS 6.x	EJB 1.1	EJB 2.0	EJB CMP 1.0	EJB CMP 2.0
WLS 5.x	WLS 6.x	EJB 1.1	EJB 2.0	See **Note 3**	

Note 1: Converting non-CMP EJB 1.0 beans to non-CMP EJB 1.1 beans is not necessary because the EJB 1.1 non-CMP deployment descriptors are the same as the EJB 2.0 non-CMP deployment descriptors.

Note 2: Use the DDConverter command line option `-EJBVer` for converting EJB CMP 1.0 to EJB CMP 1.1. See "DDConverter Options" on page 9-7 for a description of this option.

Note 3: Even thought WLS 5.x CMP 1.1 beans and WLS 6.x CMP 1.1 beans are different, WLS 5.1 CMP 1.1 beans can run in WebLogic Server 6.x without any changes to the source code.

You should always recompile the beans after you use the DDConverter. We recommend that you use `weblogic.ejbc` and then deploy the new generated JAR file. Recompiling the bean makes sure that the code is compliant with the EJB Specifications and saves you time because you can skip the recompile process during server startup.

- When converting WLS 4.5 EJB 1.0 beans to WLS 6.x EJB 1.1 beans, the input to `DDConverter` is the WebLogic 4.5 deployment descriptor text. The output is a JAR file that only includes the WebLogic 6.x deployment descriptors. Run `weblogic-ejbc` to see if you need to make any additional changes to the source code following the steps in "Using DDConverter to Convert EJBs" on page 9-6. See the first row in the Conversion Options for the DDConverter Utility table.
- When converting WLS 4.5 EJB 1.1 beans to WLS 6.x EJB 2.0 beans, the input to `DDConverter` is the WebLogic Server 4.5 deployment descriptor text. The output is a JAR file that only includes the WebLogic 6.x deployment descriptors. Run `weblogic-ejbc` to see if you need to make any additional changes to the source code, follow the steps in "Using DDConverter to Convert EJBs" on page 9-6. See the second row in the Conversion Options for the DDConverter Utility table.
- You can deploy WLS 5.x EJB 1.1 beans to WLS 6.x without any making changes to the source code because WLS 6.x is backward compatible. WLS 6.x detects, recompiles, and then deploys beans from previous versions of WLS. However, we recommend that you use the DDConverter to upgrade the WLS 5.x EJB 1.1 beans to WLS 6.x EJB 2.0 beans.

  When converting WLS 5.x EJB 1.1 beans to WLS 6.x EJB 2.0 beans, the input to `DDConverter` is the WebLogic 5.1 JAR file. This file contains the deployment descriptor files and class files. The output goes to a JAR file that includes the WebLogic 6.0 deployment descriptor files and all necessary class files. See the third row in the Conversion Options for the DDConverter Utility table.

  You can convert non-CMP beans to EJB 2.0 beans with little or no changes to the source code. To do this, run weblogic.ejbc on the `output.jar` file and then deploy the generated JAR file. With CMP beans, you must make changes to the source code using the steps in "Using DDConverter to Convert EJBs" on page 9-6.

Using DDConverter to Convert EJBs

To convert earlier versions of EJBs for use in WebLogic Server:

1. Input the EJB's deployment descriptor file into the `DDConverter` using the command line format shown in "DDConverter Syntax" on page 9-7.

The output is a JAR file.

2. Extract the XML deployment descriptors from the JAR file.
3. Modify the source code according to the JavaSoft EJB Specification.
4. Compile the modified java file with the extracted XML deployment descriptors, using `weblogic.ejbc` to create a JAR file.
5. Deploy the JAR file.

DDConverter Syntax

```
$ java weblogic.ejb20.utils.DDConverter [options] file1 [file2...]
```

DDConverter Arguments

`DDConverter` takes the argument *`file1 [file2...],`* where file is one of the following:

- A text file containing EJB 1.0-compliant deployment descriptors.
- A JAR file containing EJB 1.1 compliant deployment descriptors.

`DDConverter` uses the `beanHomeName` property of EJBs in the text deployment descriptor to define new `ejb-name` elements in the resultant `ejb-jar.xml` file.

DDConverter Options

The following table lists the DDConverter command-line options:

Option	Description
`-d destDir`	Specifies the destination directory for the output of the JAR files. This is a required option.

`-c jar name`	Specifies a JAR file in which you combine all beans in the source files.
`-EJBVer` *`output EJB version`*	Specifies the output EJB version number, such as 2.0 or 1.1. The default is 2.0.
`-log log file`	Specifies a file into which the log information can be placed instead of the `ddconverter.log`.
`-verboseLog`	Specifies that extra information on the conversion be placed in the `ddconverter.log`.
`-help`	Prints a list of all options available for the `DDConverter` utility.

DDConverter Examples

The following example converts a WLS 5.x EJB 1.1 bean into a WLS 6.x EJB 2.0 bean.

The JAR file is created in the *`destDir`* subdirectory:

```
$ java weblogic.ejb20.utils.DDConverter -d destDir Employee.jar
```

Where the `Employee` bean is a WLS 5.x EJB 1.1 JAR file.

deploy

The `weblogic.deploy` command-line utility is used to deploy an EJB-compliant JAR file, the JAR's EJBs to a running instance of WebLogic Server.

deploy Syntax

```
$ java weblogic.deploy [options] [list|deploy|undeploy|update]
password {name} {source}
```

deploy Arguments

The following table lists the weblogic.deploy command line arguments:

Argument	Description
`list`	Lists all EJB deployment units in the specified WebLogic Server.
`deploy`	Deploys an EJB JAR to the specified server.
`delete`	Deletes an EJB deployment unit.
`undeploy`	Removes an existing EJB deployment unit from the specified server.
`update`	Redeploys an EJB deployment unit. **Note:** Updating an application or component on any single server instance to which it is targeted causes it to be updated on all servers to which is targeted. For instance, if an application is targeted to a cluster, and you update it on one of the clustered servers instances, the application will be updated on all members of cluster. Similarly, if the application is targeted to a cluster and to a standalone server instance, updating it on the standalone server instance will result in its update on the cluster, and vice versa.
`password`	Specifies the system password for the WebLogic Server.
{*`name`*}	Identifies the name of the EJB deployment unit. This name can be specified at deployment time, either with the `deploy` or `console` utilities.
{*`source`*}	Specifies the exact location of the EJB JAR file, or the path to the top level of an EJB deployment directory.

deploy Options

The following table lists the weblogic.deploy command line options:

Option	Description
`-help`	Prints a list of all options available for the `deploy` utility.
`-version`	Prints the version of the utility.
`-port <port>`	Specifies the port number of the WebLogic Server to use for deploying the JAR file. If you do not specify this option, the `deploy` utility attempts to connect using port number 7001.
`-host <host>`	Specifies the host name of the WebLogic Server to use for deploying the JAR file. If you do not specify this option, the `deploy` utility attempts to connect using host name `localhost`.
`-user`	Specifies the system username of the WebLogic Server to be used to deploy the JAR file. If you do not specify this option, `deploy` attempts to make a connection using the system username `system`. You use the `weblogic.system.user` property to define the system username.
`-debug`	Prints detailed debugging information during the deployment process.

CHAPTER

10 weblogic-ejb-jar.xml Document Type Definitions

The following sections describe the EJB 5.1 and EJB 6.0 deployment descriptor elements found in the `weblogic-ejb-jar.xml` file, the weblogic-specific XML document type definitions (DTD) file. Use these definitions to create the WebLogic-specific `weblogic-ejb-jar.xml` file that is part of your EJB deployment.

Note: Use the 6.0 deployment descriptors with the 6.x version of WebLogic Server.

EJB Deployment Descriptors

The EJB deployment descriptors contain structural and application assembly information for an enterprise bean. You specify this information by specifying values for the deployment descriptors in three EJB XML DTD files. These files are:

- `ejb-jar.xml`
- `weblogic-ejb-jar.xml`
- `weblogic-cmp-rdbms-jar.xml`

You package these three XML files with the EJB and other classes into a deployable EJB component, usually a JAR file, called `ejb.jar`.

The `ejb-jar.xml` file is based on the deployment descriptors found in Sun Microsystems's `ejb.jar.xml` file. The other two XML files are weblogic-specific files that are based on the deployment descriptors found in `weblogic-ejb-jar.xml` and `weblogic-cmp-rdbms-jar.xml`.

DOCTYPE Header Information

When you edit or create XML deployment files, it is critical to include the correct `DOCTYPE` header for the deployment file. In particular, using an incorrect `PUBLIC` element within the `DOCTYPE` header can result in parser errors that may be difficult to diagnose.

The correct text for the `PUBLIC` elements for the WebLogic Server-specific `weblogic-ejb-jar.xml` file are as follows.

XML File	PUBLIC Element String
`weblogic-ejb-jar.xml`	`'-//BEA Systems, Inc.//DTD WebLogic 6.0.0 EJB//EN' 'http://www.bea.com/servers/wls600/dtd/weblogic-ejb-jar.dtd'`

XML File	PUBLIC Element String
weblogic-ejb-jar.xml	'-//BEA Systems, Inc.//DTD WebLogic 5.1.0 EJB//EN' 'http://www.bea.com/servers/wls510/dtd/weblogic-ejb-jar.dtd'

The correct text for the PUBLIC elements for the Sun Microsystem-specific ejb-jar.xml file are as follows.

XML File	PUBLIC Element String
ejb-jar.xml	'-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN' '
ejb-jar.xml	'-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN' 'http://www.java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'

For example, the entire DOCTYPE header for a weblogic-ejb-jar.xml file is as follows:

```
<!DOCTYPE weblogic-ejb-jar PUBLIC
'-//BEA Systems, Inc.//DTD WebLogic 6.0.0 EJB//EN'
'http://www.bea.com/servers/wls600/dtd/weblogic-ejb-jar.dtd'>
```

XML files with incorrect header information may yield error messages similar to the following, when used with a utility that parses the XML (such as ejbc):

```
SAXException: This document may not have the identifier 'identifier_name'
```

identifier_name generally includes the invalid text from the PUBLIC element.

Document Type Definitions (DTDs) for Validation

The contents and arrangement of elements in your XML files must conform to the Document Type Definition (DTD) for each file you use. WebLogic Server ignores the DTDs embedded within the DOCTYPE header of XML deployment files, and instead

uses the DTD locations that were installed along with the server. However, the DOCTYPE header information must include a valid URL syntax in order to avoid parser errors.

Note: Most browsers do not display the contents of files having the .dtd extension. To view the DTD file contents in your browser, save the links as text files and view them with a text editor.

weblogic-ejb-jar.xml

The following links provide the new public DTD locations for the weblogic-ejb-jar.xml deployment files used with the WebLogic Server:

- For weblogic-ejb-jar.xml 6.0 DTD:

  http://www.bea.com/servers/wls600/dtd/weblogic-ejb-jar.dtd contains the DTD used for creating weblogic-ejb-jar.xml, which defines EJB properties used for deployment to WebLogic Server.

- For weblogic-ejb-jar.xml 5.1 DTD:

  weblogic-ejb-jar.dtd contains the DTD used for creating weblogic-ejb-jar.xml, which defines EJB properties used for deployment to WebLogic Server. This file is located at *http://www.bea.com/servers/wls510/dtd/weblogic-ejb-jar.dtd*

ejb-jar.xml

The following links provide the public DTD locations for the ejb-jar.xml deployment files used with WebLogic Server:

- For ejb-jar.xml 2.0 DTD:

  http://www.java.sun.com/dtd/ejb-jar_2_0.dtd contains the DTD for the standard ejb-jar.xml deployment file, required for all EJBs. This DTD is maintained as part of the JavaSoft EJB 2.0 specification; refer to the JavaSoft specification for information about the elements used in ejb-jar.dtd.

- For ejb-jar.xml 1.1 DTD:

  ejb-jar.dtd contains the DTD for the standard ejb-jar.xml deployment file, required for all EJBs. This DTD is maintained as part of the JavaSoft EJB 1.1 specification; refer to the JavaSoft specification for information about the elements used in ejb-jar.dtd.

Note: Refer to the appropriate JavaSoft EJB specification for a description of the `ejb-jar.xml` deployment descriptors.

Changed EJB Deployment Elements in WebLogic Server 6.1

These changes were made to `weblogic-ejb-jar.xml` in WebLogic Server 6.1:

- "cache-type" on page 10-11 was added.
- "ejb-local-reference-description" on page 10-22 was added.
- "invalidation-target" on page 10-36 was added.
- "jms-client-id" on page 10-39 was added.
- "jms-polling-interval-seconds" on page 10-40 was added.

6.0 weblogic-ejb-jar.xml Deployment Descriptor File Structure

The WebLogic Server 6.0 `weblogic-ejb-jar.xml` deployment descriptor file describes the elements that are unique to WebLogic Server. Although you can use both versions of the deployment descriptors in the EJB container, the WebLogic Server 6.0 version of `weblogic-ejb-jar.xml` is different from the version shipped with WebLogic Server Version 5.1.

The WebLogic Serve 6.0 `weblogic-ejb-jar.xml` includes elements for enabling stateful session EJB replication, configuring entity EJB locking behavior, and assigning JMS Queue and Topic names for message-driven beans.

The top level elements in the WebLogic Server 6.0 `weblogic-ejb-jar.xml` are as follows:

- description
- weblogic-version
- weblogic-enterprise-bean
  - ejb-name
  - entity-descriptor | stateless-session-descriptor | stateful-session-descriptor | message-driven-descriptor
  - transaction-descriptor
  - reference-descriptor
  - enable-call-by-reference
  - jndi-name
- security-role-assignment
- transaction-isolation

6.0 weblogic-ejb-jar.xml Deployment Descriptor Elements

allow-concurrent-calls

Range of values:	`true | false`
Default value:	`false`
Requirements:	Requires the server to throw a `RemoteException` when a stateful session bean instance is currently handling a method call and another (concurrent) method call arrives on the server.
Parent elements:	`weblogic-enterprise-bean` `stateful-session-descriptor`

Function

The `allow-concurrent-calls` element specifies whether a stateful session bean instance allows concurrent method calls. By default, `allows-concurrent-calls` is `false`. However, when this value is set to `true`, the EJB container blocks the concurrent method call and allows it to proceed when the previous call has completed.

Example

See "stateful-session-descriptor" on page 10-72.

cache-type

Range of values:	`NRU	LRU`
Default value:	`NRU`
Requirements:	
Parent elements:	`weblogic-enterprise-bean` `stateful-session-cache`

Function

The `cache-type` element specifies the order in which EJBs are removed from the cache. The values are:

- Least recently used (LRU)
- Not recently used (NRU)

Example

The following example shows the structure of the `cache-type` element.

```
<stateful-session-cache>
<cache-type>NRU</cache-type>
</stateful-session-cache>
```

connection-factory-jndi-name

Range of values:	`valid name`
Default value:	`weblogic.jms.MessageDrivenBeanConnectionFactory` in `config.xml`
Requirements:	Requires the server to throw a `RemoteException` when a stateful session bean instance is currently handling a method call and another (concurrent) method call arrives on the server.
Parent elements:	`weblogic-enterprise-bean` `message-driven-descriptor`

Function

The `connection-factory-jndi-name` element specifies the JNDI name of the JMS Connection Factory that the message-driven bean should look up to create its queues and topics. If this element is not specified, the default is the `weblogic.jms.MessageDrivenBeanConnectionFactory` in `config.xml`.

Example

```
<message-driven-descriptor>
   <connection-factory-jndi-name>weblogic.jms.MessageDrivenBean
    ConnectionFactory</connection-factory-jndi-name>
</message-driven-descriptor>
```

concurrency-strategy

Range of values:	`Exclusive | Database | ReadOnly`
Default value:	`Database`
Requirements:	Optional element. Valid only for entity EJBs.

Parent elements:	`weblogic-enterprise-bean,`
	`entity-descriptor,`
	`entity-cache`

Function

The `concurrency-strategy` element specifies how the container should manage concurrent access to an entity bean. Set this element to one of three values:

- `Exclusive` causes WebLogic Server to place an exclusive lock on cached entity EJB instances when the bean is associated with a transaction. Other requests for the EJB instance block until the transaction completes. This option was the default locking behavior for WebLogic Server versions 3.1 through 5.1.
- `Database` causes WebLogic Server to defer locking requests for an entity EJB to the underlying datastore. With the `Database` concurrency strategy, WebLogic Server does not cache the intermediate results of entity EJBs involved in a transaction. This is the current default option.
- `ReadOnly` designates an entity EJB that is never modified. WebLogic Server calls `ejbLoad()` for `ReadOnly` beans based on the `read-timeout-seconds` parameter.

See“Locking Services for Entity EJBs” on page 4-37 for more information on the `Exclusive` and `Database` locking behaviors. See“Read-Only Multicast Invalidation” on page 4-17 for more information about `read-only` entity EJBs.

Example

The following entry identifies the `AccountBean` class as a read-only entity EJB:

```
<weblogic-enterprise-bean>
       <ejb-name>AccountBean</ejb-name>
       <entity-descriptor>
              <entity-cache>

<concurrency-strategy>ReadOnly</concurrency-strategy>
              </entity-cache>
       </entity-descriptor>
</weblogic-enterprise-bean>
```

db-is-shared

Range of values:	`true | false`
Default value:	`true`
Requirements:	Optional element. Valid only for entity EJBs.

Parent elements:	`weblogic-enterprise-bean,`
	`entity-descriptor,`
	`persistence`

Function

The `db-is-shared` element applies only to entity beans. When it is set to `true`, WebLogic Server assumes that EJB data can be modified between transactions and reloads the data at the beginning of each transaction. When set to `false`, WebLogic Server assumes that it has exclusive access to the EJB data in the persistent store. See "Using db-is-shared to Limit Calls to ejbLoad()" on page 4-12 for more information.

Example

See "persistence" on page 10-53.

delay-updates-until-end-of-tx

Range of values:	`true | false`
Default value:	`true`
Requirements:	Optional element. Valid only for entity EJBs.
Parent elements:	`weblogic-enterprise-bean, entity-descriptor, persistence`

Function

Set the `delay-updates-until-end-of-tx` element to `true` (the default) to update the persistent store of all beans in a transaction at the completion of the transaction. This setting generally improves performance by avoiding unnecessary updates. However, it does not preserve the ordering of database updates within a database transaction.

If your datastore uses an isolation level of `TRANSACTION_READ_UNCOMMITTED`, you may want to allow other database users to view the intermediate results of in-progress transactions. In this case, set `delay-updates-until-end-of-tx` to `false` to update the bean's persistent store at the conclusion of each method invoke. See "ejbLoad() and ejbStore() Behavior for Entity EJBs" on page 4-12 for more information.

Note: Setting `delay-updates-until-end-of-tx` to false does not cause database updates to be "committed" to the database after each method invoke; they are only sent to the database. Updates are committed or rolled back in the database only at the conclusion of the transaction.

Example

The following example shows a `delay-updates-until-end-of-tx` stanza.

```
<entity-descriptor>
```

```
        <persistence>

<delay-updates-until-end-of-tx>false</delay-updates-until-end-of-
tx>

        </persistence>

</entity-descriptor>
```

description

Range of values:	n/a
Default value:	n/a
Requirements:	n/a

Parent elements:	`weblogic-ejb-jar`
	`transaction-isolation`
	`method`

Function

The `description` element is used to provide text that describes the parent element.

Example

The following examples specifies the `description` element.

destination-jndi-name

Range of values:	Valid JNDI name
Default value:	n/a
Requirements:	Required in `message-driven-descriptor`.
Parent elements:	`weblogic-enterprise-bean` `message-driven-descriptor`

Function

The `destination-jndi-name` element specifies the JNDI name used to associate a message-driven bean with an actual JMS Queue or Topic deployed in the in WebLogic Server JNDI tree.

Example

See "message-driven-descriptor" on page 10-46.

ejb-name

Range of values:	Name of an EJB defined in `ejb-jar.xml`
Default value:	n/a
Requirements:	Required element in method stanza. The name must conform to the lexical rules for an NMTOKEN.
Parent elements:	`weblogic-enterprise-bean` `method`

Function

`ejb-name` specifies the name of an EJB to which WebLogic Server applies isolation level properties. This name is assigned by the `ejb-jar` file's deployment descriptor. The name must be unique among the names of the enterprise beans in the same `ejb.jar` file. The enterprise bean code does not depend on the name; therefore the name can be changed during the application-assembly process without breaking the enterprise bean's function. There is no built-in relationship between the `ejb-name` in the deployment descriptor and the JNDI name that the deployer will assign to the enterprise bean's home.

Example

See "method" on page 10-47.

ejb-reference-description

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Optional element.
Parent elements:	`weblogic-enterprise-bean` `reference-descriptor`

Function

The `ejb-reference-description` element maps the JNDI name in the WebLogic Server of an EJB that is referenced by the bean in the `ejb-reference` element.

- `ejb-ref-name` specifies a resource reference name. This is the reference that the EJB provider places within the `ejb-jar.xml` deployment file.
- `jndi-name` specifies the JNDI name of an actual resource factory available in WebLogic Server.

Example

The `ejb-reference-description` stanza is shown here:

```
<ejb-reference-description>
       <ejb-ref-name>AdminBean</ejb-ref-name>
              <jndi-name>payroll.AdminBean</jndi-name>
       </ejb-reference-description>
```

ejb-ref-name

Range of values:	n/a
Default value:	n/a
Requirements:	Optional element.

Parent elements:	`weblogic-enterprise-bean`
	`reference-description`
	`ejb-reference-description`

Function

The `ejb-ref-name` element specifies a resource reference name. This element is the reference that the EJB provider places within the `ejb-jar.xml` deployment file.

Example

The `ejb-ref-name` stanza is shown here:

```
<reference-descriptor>
       <ejb-reference-description>
              <ejb-ref-name>AdminBean</ejb-ref-name>
              <jndi-name>payroll.AdminBean</jndi-name>
       </ejb-reference-description>
</reference-descriptor>
```

ejb-local-reference-description

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Optional element.
Parent elements:	`weblogic-enterprise-bean` `reference-descriptor`

Function

The `ejb-local-reference-description` element maps the JNDI name of an EJB in the WebLogic Server that is referenced by the bean in the `ejb-local ref` element.

Example

The following example shows the `ejb-local-reference-description` element.

```
<ejb-local-reference-description>
       <ejb-ref-name>AdminBean</ejb-ref-name>
       <jndi-name>payroll.AdminBean</jndi-name>
</ejb-local-reference-description>
```

enable-call-by-reference

Range of values:	`true | false`
Default value:	`true`
Requirements:	Optional element.

Parent elements:	`weblogic-enterprise-bean`
	`reference-descriptor`
	`ejb-reference-description`

Function

By default, EJB methods called from within the same EAR pass arguments by reference. This increases the performance of method invocation because parameters are not copied.

If you set `enable-call-by-reference` to `False`, parameters to the EJB methods are copied (pass-by-value) in accordance with the EJB 1.1 specification. Pass by value is always necessary when the EJB is called remotely (not from within the same application).

Example

The following example enables pass-by-value for EJB methods:

```
<weblogic-enterprise-bean>
       <ejb-name>AccountBean</ejb-name>
       ...
       <enable-call-by-reference>false</enable-call-by-reference>
</weblogic-enterprise-bean>
```

entity-cache

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	The `entity-cache` stanza is optional, and is valid only for entity EJBs.
Parent elements:	`weblogic-enterprise-bean`, `entity-descriptor`

Function

The `entity-cache` element defines the following options used to cache entity EJB instances within WebLogic Server:

- `max-beans-in-cache`
- `idle-timeout-seconds`
- `read-timeout-seconds`
- `concurrency-strategy`

See "EJB Life Cycle" on page 4-2 for a general discussion of the caching services available in WebLogic Server.

Example

```
<entity-descriptor>
       <entity-cache>
              <max-beans-in-cache>...</max-beans-in-cache>
              <idle-timeout-seconds>...</idle-timeout-seconds>
              <read-timeout-seconds>...<read-timeout-seconds>
              <concurrency-strategy>...</concurrency-strategy>
```

```
        </entity-cache>
        <lifecycle>...</lifecycle>
        <persistence>...</persistence>
        <entity-clustering>...</entity-clustering>
</entity-descriptor>
```

entity-clustering

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Optional element. Valid only for entity EJBs in a cluster.
Parent elements:	`weblogic-enterprise-bean`, `entity-descriptor`

Function

The `entity-clustering` element uses the following options to specify how an entity bean will be replicated in a WebLogic cluster:

- `home-is-clusterable`
- `home-load-algorithm`
- `home call-router-class-name`

Example

The following excerpt shows the structure of a `entity-clustering` stanza:

```
<entity-clustering>
```

```
        <home-is-clusterable>true</home-is-clusterable>
        <home-load-algorithm>random</home-load-algorithm>
<home-call-router-class-name>beanRouter</home-call-router-class-n
ame>
</entity-clustering>
```

entity-descriptor

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	One `entity-descriptor` stanza is required for each entity EJB in the *.jar*.
Parent elements:	`weblogic-enterprise-bean`

Function

The `entity-descriptor` element specifies the following deployment parameters that are applicable to an entity bean:

- `pool`
- `entity-cache`
- `lifecycle`
- `persistence`
- `entity-clustering`

Example

The following example shows the structure of the `entity-descriptor` stanza:

```
<entity-descriptor>
       <entity-cache>...</entity-cache>
       <lifecycle>...</lifecycle>
       <persistence>...</persistence>
       <entity-clustering>...</entity-clustering>
</entity-descriptor>
```

finders-load-bean

Range of values:	`true | false`
Default value:	`true`
Requirements:	Optional element. Valid only for CMP entity EJBs.
Parent elements:	`weblogic-enterprise-bean`, `entity-descriptor`, `persistence`

Function

The `finders-load-bean` element determines whether WebLogic Server loads the EJB into the cache after a call to a finder method returns a reference to the bean. If you set this element to `true`, WebLogic Server immediately loads the bean into the cache if a reference to a bean is returned by the finder. If you set this element to `false`, WebLogic Server does not load automatically load the bean into the cache until the first method invocation; this behavior is consistent with the EJB 1.1 specification.

Example

The following entry specifies that EJBs are loaded into the WebLogic Server cache automatically when a finder method returns a reference to the bean:

```
<entity-descriptor>
       <persistence>
              <finders-load-bean>true</finders-load-bean>
       </persistence>
</entity-descriptor>
```

home-call-router-class-name

Range of values:	Valid router class name
Default value:	null
Requirements:	Optional element. Valid only for entity EJBs, stateful session EJBs, and stateless session EJBs in a cluster.

Parent elements:	`weblogic-enterprise-bean,`
	`entity-descriptor,`
	`entity-clustering`
	`and`
	`weblogic-enterprise-bean`
	`stateful-session-descriptor`
	`stateful-session-clustering`

Function

`home-call-router-class-name` specifies the name of a custom class to use for routing bean method calls. This class must implement `weblogic.rmi.cluster.CallRouter()`. If specified, an instance of this class is

called before each method call. The router class has the opportunity to choose a server to route to based on the method parameters. The class returns either a server name or null, which indicates that the current load algorithm should select the server.

Example

See "entity-clustering" on page 10-25 and "stateful-session-clustering" on page 10-71.

home-is-clusterable

Range of values:	`true | false`
Default value:	`true`
Requirements:	Optional element. Valid for entity EJBs, stateless session EJBs, and stateful session EJBs in a cluster.

Parent elements:	`weblogic-enterprise-bean, entity-descriptor, entity-clustering`
	and
	`weblogic-enterprise-bean stateful-session-descriptor stateful-session-clustering`
	and
	`weblogic-enterprise-bean stateless-session-descriptor stateless-clustering`
	Note: This element is valid for stateless session EJBs as of WebLogic Server 6.1 SP03.

Function

Use `home-is-clusterable` to specify whether the home interface of an entity, stateless session, or stateful session bean is clustered.

When `home-is-clusterable` is `true` for an EJB deployed to a cluster, each server instance binds the bean's home interface to its cluster JNDI tree under the same name. When a client requests the bean's home from the cluster, the server instance that does the look-up returns a `EJBHome` stub that has a reference to the home on each server.

When the client issues a `create()` or `find()` call, the stub routes selects a server from the replica list in accordance with the load balancing algorithm, and routes the call to the home interface on that server. The selected home interface receives the call, and creates a bean instance on that server instance and executes the call, creating an instance of the bean.

Example

See "entity-clustering" on page 10-25.

home-load-algorithm

Range of values:	`round-robin | random | weight-based`
Default value:	Value of `weblogic.cluster.defaultLoadAlgorithm`
Requirements:	Optional element. Valid only for entity EJBs and stateful session EJBs in a cluster.

Parent elements:	`weblogic-enterprise-bean,`
	`entity-descriptor,`
	`entity-clustering`
	`and`
	`weblogic-enterprise-bean`
	`stateful-session-descriptor`
	`stateful-session-clustering`

Function

`home-load-algorithm` specifies the algorithm to use for load balancing between replicas of the EJB home. If this property is not defined, WebLogic Server uses the algorithm specified by the server property, `weblogic.cluster.defaultLoadAlgorithm`.

You can define `home-load-algorithm` as one of the following values:

- `round-robin`: Load balancing is performed in a sequential fashion among the servers hosting the bean.
- `random`: Replicas of the EJB home are deployed randomly among the servers hosting the bean.
- `weight-based`: Replicas of the EJB home are deployed on host servers according to the servers' current workload.

Example

See "entity-clustering" on page 10-25 and "stateful-session-clustering" on page 10-71.

idle-timeout-seconds

Range of values:	1 to *maxSeconds*, where *maxSeconds* is the maximum value of an int.
Default value:	600
Requirements:	Optional element
Parent elements:	weblogic-enterprise-bean, entity-descriptor, entity-cache and weblogic-enterprise-bean, stateful-session-descriptor, stateful-session-cache

Function

idle-timeout-seconds defines the maximum length of time a stateful session EJB should remain in cache. After this time has elapsed, WebLogic Server removes the bean instance if the number of beans in cache approaches the limit of max-beans-in-cache. The removed bean instances are passivated. See "EJB Life Cycle" on page 4-2 for more information.

Note: Although idle-timeout-seconds appears in the entity-cache stanza, WebLogic Server 6. 1 does not uses its value in managing the lifecycle of entity EJBs— idle-timeout-seconds has no effect on when entity beans are removed from cache.

Example

The following entry indicates that the stateful session EJB, AccountBean, should become eligible for removal if max-beans-in-cache is reached and the bean has been in cache for 20 minutes:

```
<weblogic-enterprise-bean>
```

```
        <ejb-name>AccountBean</ejb-name>
        <stateful-session-descriptor>
               <stateful_session-cache>
                      <max-beans-in-cache>200</max-beans-in-cache>
<idle-timeout-seconds>1200</idle-timeout-seconds>
               </stateful-session-cache>
        </stateful-session-descriptor>
</weblogic-enterprise-bean>
```

initial-beans-in-free-pool

Range of values:	`0` to *`maxBeans`*
Default value:	`0`
Requirements:	Optional element. Valid for stateless session, entity, and message-driven EJBs.
Parent elements:	`weblogic-enterprise-bean, stateless-session-descriptor, message-bean-descriptor, entity-descriptor pool`

Function

If you specify a value for `initial-beans-in-free-pool`, you set the initial size of the 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 EJB, because initial requests for the bean can be satisfied without generating a new instance.

Example

See "pool" on page 10-57.

initial-context-factory

Range of values:	`true | false`
Default value:	`weblogic.jndi.WLInitialContextFactory`
Requirements:	Requires the server to throw a `RemoteException` when a stateful session bean instance is currently handling a method call and another (concurrent) method call arrives on the server.
Parent elements:	`weblogic-enterprise-bean` `message-driven-descriptor`

Function

The `initial-context-factory` element specifies the initial contextFactory that the container will use to create its connection factories. If initial-context-factory is not specified, the default will be `weblogic.jndi.WLInitialContextFactory`.

Example

The following example specifies the `initial-context-factory` element.

```
<message-driven-descriptor>

<initial-context-factory>weblogic.jndi.WLInitialContextFactory
</initial-context-factory>

</message-driven-descriptor>
```

invalidation-target

Range of values:	
Default value:	
Requirements:	The target `ejb-name` must be a Read-Only entity EJB and this element can only be specified for an EJB 2.0 container-managed persistence entity EJB.
Parent elements:	`weblogic-enterprise-bean` `entity-descriptor`

Function

The `invalidation-target` element specifies a Read-Only entity EJB that should be invalidated when this container-managed persistence entity EJB has been modified.

Example

The following entry specifies that the EJB named `StockReaderEJB` should be invalidated when the EJB has been modified.

```
<invalidation-target>
       <ejb-name>StockReaderEJB</ejb-name>
</invalidation-target>
```

is-modified-method-name

Range of values:	Valid entity EJB method name
Default value:	None
Requirements:	Optional element. Valid only for entity EJBs.
Parent elements:	`weblogic-enterprise-bean`, `entity-descriptor`, `persistence`

Function

`is-modified-method-name` specifies a method that WebLogic Server calls when the EJB is stored. The specified method must return a `boolean` value. If no method is specified, WebLogic Server always assumes that the EJB has been modified and always saves it.

Providing a method and setting it as appropriate can improve performance for EJB 1.1-compliant beans, and for beans that use bean-managed persistence. However, any errors in the method's return value can cause data inconsistency problems. See"Using is-modified-method-name to Limit Calls to ejbStore() (EJB 1.1 Only)" on page 4-14 for more information.

Note: `isModified()` is no longer required for 2.0 CMP entity EJBs based on the EJB 2.0 specification However, it still applies to BMP and 1.1 CMP EJBs. When you deploy EJB 2.0 entity beans with container-managed persistence, WebLogic Server automatically detects which EJB fields have been modified, and writes only those fields to the underlying datastore.

Example

The following entry specifies that the EJB method named `semidivine` will notify WebLogic Server when the EJB has been modified:

```
<entity-descriptor>
       <persistence>

<is-modified-method-name>semidivine</is-modified-method-name>
       </persistence>
</entity-descriptor>
```

isolation-level

Range of values:	`Serializable | ReadCommitted | ReadUncommitted | RepeatableRead`
Default value:	n/a
Requirements:	Optional element.
Parent elements:	`weblogic-ejb-jar` `transaction-isolation`

Function

`isolation-level` specifies the isolation level for all of the EJB's database operations. The following are possible values for `isolation-level`:

- `TRANSACTION_READ_UNCOMMITTED`: The transaction can view uncommitted updates from other transactions.
- `TRANSACTION_READ_COMMITTED`: The transaction can view only committed updates from other transactions.
- `TRANSACTION_REPEATABLE_READ`: Once the transaction reads a subset of data, repeated reads of the same data return the same values, even if other transactions have subsequently modified the data.

- `TRANSACTION_SERIALIZABLE`: Simultaneously executing this transaction multiple times has the same effect as executing the transaction multiple times in a serial fashion.

Refer to your database documentation for more information on the implications and support for different isolation levels.

Example

See "transaction-isolation" on page 10-80.

jms-client-id

Range of values:	`N/A`
Default value:	ejb name for the ejb
Requirements:	Necessary for durable subscriptions to JMS topics.
Parent elements:	`weblogic-enterprise-bean`

Function

The `jms-client-id` element specifies the client ID associated with the message-driven bean. This ID is necessary for durable subscriptions to JMS topics.

The JMS specification allows JMS consumers to specify an associated ID. A message-driven bean with a durable subscription needs an associated client ID. If you use a separate connection factory, you can set the client ID on the connection factory. In this case, the message-driven bean uses this client ID.

If the associated client ID does not have a client ID or if you are using the default connection factory, the message-driven bean uses the `jms-client-id` value as its client ID.

Example

The following example specifies the use of the `jms-client-id` element.

```
<jms-client-id>MyClientID</jms-client-id>
```

jms-polling-interval-seconds

Range of values:	none
Default value:	10 seconds
Requirements:	.none
Parent elements:	`weblogic-enterprise-bean`

Function

The `jms-polling-interval-seconds` element determines the number of seconds between each attempt by WebLogic Server to reconnect to the JMS destination.

Each message-driven bean listens on an associated JMS destination. If the JMS destination is located on another WebLogic Server instance or a foreign JMS provider, the JMS destination may become unreachable. In this case, the EJB container automatically attempts to reconnect to the JMS server. Once the JMS server is running again the message-driven bean can again receive JMS messages.

Refer to your database documentation for more information on the implications and support for different isolation levels.

Example

The following example specifies the use of the `jms-polling-interval-seconds` element.

```
<jms-polling-interval-seconds>5</jms-polling-interval-seconds>
```

jndi-name

Range of values:	Valid JNDI name
Default value:	n/a
Requirements:	Required in `resource-description` and `ejb-reference-description`.

Parent elements:	`weblogic-enterprise-bean`
	and
	`weblogic-enterprise-bean`
	`reference-descriptor`
	`resource-description`
	and
	`weblogic-enterprise-bean`
	`reference-descriptor`
	`ejb-reference-description`

Function

`jndi-name` specifies the JNDI name of an actual EJB, resource, or reference available in WebLogic Server.

Example

See “resource-description” on page 10-65 and “ejb-reference-description” on page 10-20.

local-jndi-name

Range of values:	Valid JNDI name
Default value:	n/a
Requirements:	Required if the bean has a local home.
Parent elements:	`weblogic-enterprise-bean`

Function

The `local-jndi-name` element specifies a jndi-name for a bean’s local home. If a bean has both a remote and a local home, then it must have two JNDI names; one for each home.

Example

The following example shows the specifies the `local-jndi-name` element.

```
<local-jndi-name>weblogic.jndi.WLInitialContext
</local-jndi-name>
```

lifecycle

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	The `lifecycle` stanza is optional.

Parent elements:	`weblogic-enterprise-bean,`
	`  entity-descriptor`
	`and`
	`weblogic-enterprise-bean`
	`  stateful-session-descriptor`

Function

The `lifecycle` element defines options that affect the lifecycle of stateful and entity EJB instances within WebLogic Server. Currently, the `lifecycle` element includes only one element: `passivation-strategy`.

Example

The following example shows the specifies the `lifecycle` element.

```
<entity-descriptor>
       <lifecycle>
              <passivation-strategy>...</passivation-strategy>
       </lifecycle>
</entity-descriptor>
```

max-beans-in-cache

Range of values:	`1` to *`maxBeans`*
Default value:	`1000`
Requirements:	Optional element

Parent elements:	`weblogic-enterprise-bean,`
	`entity-descriptor,`
	`entity-cache`
	`and`
	`weblogic-enterprise-bean`
	`stateful-session-descriptor`
	`stateful-session-cache`

Function

The `max-beans-in-cache` element specifies the maximum number of objects of this class that are allowed in memory. When `max-bean-in-cache` is reached, WebLogic Server passivates some EJBs that have not been recently used by a client. `max-beans-in-cache` also affects when EJBs are removed from the WebLogic Server cache, as described in"Locking Services for Entity EJBs" on page 4-37.

Example

The following entry enables WebLogic Server to cache a maximum of 200 instances of the `AccountBean` class:

```
<weblogic-enterprise-bean>
        <ejb-name>AccountBean</ejb-name>
        <entity-descriptor>
                <entity-cache>
```

```
                        <max-beans-in-cache>200</max-beans-in-cache>
                    </entity-cache>
                </entity-descriptor>
            </weblogic-enterprise-bean>
```

max-beans-in-free-pool

Range of values:	`0` to *`maxBeans`*
Default value:	1000
Requirements:	Optional element.

Parent elements:	`weblogic-enterprise-bean, stateless-session-descriptor, pool`
	`weblogic-enterprise-bean, message-driven-descriptor, pool`
	`weblogic-enterprise-bean, entity-descriptor, pool`

Function

WebLogic Server maintains a free pool of EJBs for every stateless session, message-driven, and entity bean class `max-beans-in-free-pool` limits the size of the free pool. For more information, see “EJB Life Cycle” on page 4-2.

Example

See “pool” on page 10-57.

message-driven-descriptor

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	
Parent elements:	`weblogic-enterprise-bean`

Function

The `message-driven-descriptor` element associates a message-driven bean with a JMS destination in WebLogic Server. This element specifies the following deployment parameters:

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

Example

The following example shows the structure of the `message-driven-descriptor` stanza:

```
<message-driven-descriptor>
       <destination-jndi-name>...</destination-jndi-name>
</message-driven-descriptor>
```

method

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Optional element. You can specify more than one `method` stanza to configure multiple EJB methods.
Parent elements:	`weblogic-ejb-jar` `transaction-isolation`

Function

The `method` element defines a method or set of methods for an enterprise bean's home or remote interface.

Example

The `method` stanza can contain the elements shown here:

```
<method>
        <description>...</description>
        <ejb-name>...</ejb-name>
        <method-intf>...</method-intf>
        <method-name>...</method-name>
        <method-params>...</method-params>
</method>
```

method-intf

Range of values:	`Home | Remote | Local | Localhome`
Default value:	n/a
Requirements:	Optional element.

Parent elements:	`weblogic-ejb-jar`
	`transaction-isolation`
	`method`

Function

`method-intf` specifies the EJB interface to which WebLogic Server applies isolation level properties. Use this element to differentiate between methods having the same signature in the EJB's home, remote, and local interfaces.

Example

See "method" on page 10-47.

method-name

Range of values:	Name of an EJB defined in `ejb-jar.xml | *`
Default value:	n/a
Requirements:	Required element in method stanza.

Parent elements:	`weblogic-ejb-jar`
	`transaction-isolation`
	`method`

Function

`method-name` specifies the name of an individual EJB method to which WebLogic Server applies isolation level properties. Use the asterisk (*) to specify all methods in the EJB's home and remote interfaces.

If you specify a `method-name`, the method must be available in the specified `ejb-name`.

Example

See "method" on page 10-47.

method-param

Range of values:	Fully qualified Java type of a method parameter
Default value:	n/a
Requirements:	Required element in `method-params`.

Parent elements:	`weblogic-ejb-jar`
	`transaction-isolation`
	`method`
	`method-params`

Function

The `method-param` element specifies the fully qualified Java type name of a method parameter.

Example

See "method-params" on page 10-51.

method-params

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Optional stanza.

Parent elements:	`weblogic-ejb-jar`
	`transaction-isolation`
	`method`

Function

The `method-params` stanza contains one or more elements that define the Java type name of each of the method's parameters.

Example

The `method-params` stanza contains one or more `method-param` elements, as shown here:

```
<method-params>
        <method-param>java.lang.String</method-param>
        ...
</method-params>
```

passivation-strategy

Range of values:	`default | transaction`
Default value:	`default`
Requirements:	Optional element. Valid only for entity EJBs.
Parent elements:	`weblogic-enterprise-bean, entity-descriptor, lifecycle`

Function

The `passivation-strategy` element determines whether or not WebLogic Server maintains the intermediate state of entity EJBs in its cache. See "Locking Services for Entity EJBs" on page 4-37 for more information.

Example

The following entry reverts to WebLogic Server locking and caching behavior:

```
<entity-descriptor>
       <lifecycle>
              <passivation-strategy>default</passivation-strategy>
       </lifecycle>
</entity-descriptor>
```

persistence

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Required only for entity EJBs that use container-managed persistence services.
Parent elements:	`weblogic-enterprise-bean`, `entity-descriptor`

Function

The `persistence` element defines the following options that determine the persistence type, transaction commit behavior, and `ejbLoad()` and `ejbStore()` behavior for entity EJBs in WebLogic Server:

- `is-modified-method-name`
- `delay-updates-until-end-of-tx`
- `finders-load-bean`
- `persistence-type`
- `db-is-shared`
- `persistence-use`

Example

The following example specifies the `persistence` element.

```
<entity-descriptor>
       <persistence>
              <is-modified-method-name>...
              </is-modified-method-name>
              <delay-updates-until-end-of-tx>...
              </delay-updates-until-end-of-tx>
              <finders-load-bean>...</finders-load-bean>
```

```
                <persistence-type>...</persistence-type>
                <db-is-shared>...</db-is-shared>
                <persistence-use>...</persistence-use>
        </persistence>
</entity-descriptor>
```

persistence-type

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Required only for entity EJBs that use container-managed persistence services.
Parent elements:	`weblogic-enterprise-bean`, `entity-descriptor`, `persistence`

Function

Defines a persistence service that the entity EJB can use. You can define multiple `persistence-type` stanzas in `weblogic-ejb-jar.xml` for testing your EJB with multiple persistence services. At deployment, the bean uses the persistence service defined in `persistence-use`.

`persistence-type` includes these elements:

- `type-identifier`
- `type-version`
- `type-storage`

Example

```
<persistence-type>
        <type-identifier>WebLogic_CMP_RDBMS
```

```
        </type-identifier>
        <type-version>5.1.0</type-version>
        <type-storage>META-INF\weblogic-cmp-rdbms-jar.xml
        </type-storage>
</persistence-type>
```

persistence-use

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Required only for entity EJBs that use container-managed persistence services.

Parent elements:	`weblogic-enterprise-bean`
	`entity-descriptor`
	`persistence`

Function

Defines the persistence service used for the entity bean.

Example

```
<persistence-use>
       <type-identifier>WebLogic_CMP_RDBMS</type-identifier>
       <type-version>5.1.0</type-version>
</persistence-use>
```

persistent-store-dir

Range of values:	Fully qualified filesystem path
Default value:	n/a
Requirements:	Optional element.
Parent elements:	`weblogic-enterprise-bean` `stateful-session-descriptor`

Function

The `persistent-store-dir` element specifies a file system directory where WebLogic Server stores the state of passivated stateful session bean instances.

Example

See "stateful-session-descriptor" on page 10-72.

pool

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Optional element.
Parent elements:	`weblogic-enterprise-bean` `stateless-session-descriptor, message-bean-descriptor, entity-descriptor`

Function

The `pool` element configures the behavior of the WebLogic Server free pool for EJBs. You can configure:

- `max-beans-in-free-pool`
- `initial-beans-in-free-pool`

Example

```
<stateless-session-descriptor>
   pool>
     <max-beans-in-free-pool>500</max-beans-in-free-pool>
     <initial-beans-in-free-pool>250</initial-beans-in-free-pool>
   </pool>
</stateless-session-descriptor>
```

principal-name

Range of values:	valid WebLogic Server principal name
Default value:	n/a
Requirements:	At least one `principal-name` is required in the security-role-assignment stanza. You may define more than one `principal-name` for each role-name.
Parent elements:	`weblogic-enterprise-bean` `security-role-assignment`

Function

`principal-name` specifies the name of an actual WebLogic Server principal to apply to the specified `role-name`.

Example

See "security-role-assignment" on page 10-69.

provider-url

Range of values:	valid name
Default value:	n/a
Requirements:	Used in conjunction with `initial-context-factory` and `connection-factory-jndi-name`.
Parent elements:	`weblogic-enterprise-bean` `message-driven-descriptor`

Function

The `provider-url` element specifies the URL provider to be used by the `InitialContext`. Typically, this is the host:port and used in conjunction with `initial-context-factory` and `connection-factory-jndi-name`.

Example

The following example specifies the `provider-url` element.

```
<message-driven-descriptor>
<provider-url>WeblogicURL:Port</provider-url>
</message-driven-descriptor>
```

read-timeout-seconds

Range of values:	`0` to *`maxSeconds`*, where *`maxSeconds`* is the maximum value of an int.
Default value:	`600`
Requirements:	Optional element. Valid only for entity EJBs.
Parent elements:	`weblogic-enterprise-bean`, `entity-descriptor`, `entity-cache`

Function

The `read-timeout-seconds` element specifies the number of seconds between `ejbLoad()` calls on a `Read-Only` entity bean. A setting of 0 causes WebLogic Server to call `ejbLoad()` only when the bean is brought into the cache.

Example

The following entry causes WebLogic Server to call `ejbLoad()` for instances of the `AccountBean` class only when the instance is first brought into the cache:

```
<weblogic-enterprise-bean>
       <ejb-name>AccountBean</ejb-name>
       <entity-descriptor>
              <entity-cache>
                     <read-timeout-seconds>0</read-timeout-
                     seconds>
              </entity-cache>
       </entity-descriptor>
</weblogic-enterprise-bean>
```

reference-descriptor

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Optional element.
Parent elements:	`weblogic-enterprise-bean`

Function

The `reference-descriptor` element maps references in the `ejb-jar.xml` file to the JNDI names of actual resource factories and EJBs available in WebLogic Server.

Example

The `reference-descriptor` stanza contains one or more additional stanzas to define resource factory references and EJB references. The following shows the organization of these elements:

```
<reference-descriptor>
        <resource-description>
                ...
        </resource-description>
        <ejb-reference-description>
                ...
        </ejb-reference-description>
</reference-descriptor>
```

relationship-description

This element is no longer supported in WebLogic Server.

replication-type

Range of values:	`InMemory | None`
Default value:	`None`
Requirements:	Optional element. Valid only for stateful session EJBs in a cluster.

Parent elements:	`weblogic-enterprise-bean`
	`stateful-session-descriptor`
	`stateful-session-clustering`

Function

The `replication-type` element determines whether WebLogic Server replicates the state of stateful session EJBs across WebLogic Server instances in a cluster. If you select `InMemory`, the state of the EJB is replicated. If you select `None`, the state is not replicated.

See "In-Memory Replication for Stateful Session EJBs" on page 4-25 for more information.

Example

See "stateful-session-clustering" on page 10-71.

res-env-ref-name

Range of values:	A valid resource environment reference name from the `ejb-jar.xml` file
Default value:	n/a
Requirements:	n/a

Parent elements:	`weblogic-enterprise-bean`
	`reference-descriptor`
	`resource-env-description`

Function

The `res-env-ref-name` element specifies the name of a resource environment reference.

Example

See "resource-description" on page 10-65.

res-ref-name

Range of values:	A valid resource reference name from the `ejb-jar.xml` file
Default value:	n/a
Requirements:	Required element if the EJB specifies resource references in `ejb-jar.xml`

Parent elements:	`weblogic-enterprise-bean`
	`reference-descriptor`
	`resource-description`

Function

The `res-ref-name` element specifies the name of a `resourcefactory` reference. This is the reference that the EJB provider places within the `ejb-jar.xml` deployment file.

Example

See "resource-description" on page 10-65.

resource-description

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Optional element.
Parent elements:	`weblogic-enterprise-bean` `reference-descriptor`

Function

The `resource-description` element maps a resource reference defined in `ejb-jar.xml` to the JNDI name of an actual resource available in WebLogic Server.

Example

The `resource-description` stanza can contain additional elements as shown here:

```
<reference-descriptor>
       <resource-description>
              <res-ref-name>. . .</res-ref-name>
              <jndi-name>...</jndi-name>
       </resource-description>
       <ejb-reference-description>
              <ejb-ref-name>. . .</ejb-ref-name>
              <jndi-name>. . .</jndi-name>
       </ejb-reference-description>
</reference-descriptor>
```

resource-env-description

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Optional element.
Parent elements:	`weblogic-enterprise-bean` `reference-descriptor`

Function

The `resource-env-description` element maps a resource environment reference defined in `ejb-jar.xml` to the JNDI name of an actual resource available in WebLogic Server.

Example

The `resource-env-description` stanza can contain additional elements as shown here:

```
<reference-descriptor>
       <resource-env-description>
              <res-env-ref-name>. . .</res-env-ref-name>
              <jndi-name>...</jndi-name>
       <reference-env-description>
</reference-descriptor>
```

role-name

Range of values:	An EJB role name defined in `ejb-jar.xml`
Default value:	n/a
Requirements:	Required element in `security-role-assignment`.
Parent elements:	`weblogic-enterprise-bean` `security-role-assignment`

Function

The `role-name` element identifies an application role name that the EJB provider placed in the `ejb-jar.xml` deployment file. Subsequent `principal-name` elements in the stanza map WebLogic Server principals to the specified `role-name`.

Example

See "security-role-assignment" on page 10-69.

run-as-identity-principal

Note: This element has been deprecated. It is here for backward compatibility only.

Range of values:	Principal that will be used as the identity as defined in `ejb-jar.xml`
Default value:	n/a
Requirements:	Required element in `security-role-assignment`.

Parent elements: weblogic-enterprise-bean

Function

The `run-as-identity-principal` element specifies the principal to be used as the identity for beans that have a `security-identity.run-as-specified-identity` set in the `ejb-jar.xml`.

The principal named in this element must be one of the principals mapped to the `run-as-specified--identity` role.

Example

```
<weblogic-ejb-jar>
       <weblogic-enterprise-bean>
       <run-as-identity-principal>..</run-as-identity-principal>
       </weblogic-enterprise-bean>
</weblogic-ejb-jar>
```

security-role-assignment

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Required element if `ejb-jar.xml` defines application roles.
Parent elements:	`N/A`

Function

The `security-role-assignment` stanza maps application roles in the `ejb-jar.xml` file to the names of security principals available in WebLogic Server.

Example

The `security-role-assignment` stanza can contain one or more of the following elements:

```
<security-role-assignment>
        <role-name>PayrollAdmin</role-name>
        <principal-name>Tanya</principal-name>
        <principal-name>system</principal-name>
        ...
</security-role-assignment>
```

stateful-session-cache

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	The `stateful-session-cache` stanza is optional, and is valid only for stateful session EJBs.
Parent elements:	`weblogic-enterprise-bean`, `stateful-session-descriptor`

Function

The `stateful-session-cache` element defines the following options used to cache stateful session EJB instances within WebLogic Server.

- `max-beans-in-cache`
- `idle-timeout-seconds`
- `cache-type`

See"EJB Life Cycle" on page 4-2 for a general discussion of the caching services available in WebLogic Server.

Example

The following example shows how to specify the `stateful-session-cache` element

```
<stateful-session-cache>
       <max-beans-in-cache>...</max-beans-in-cache>
       <idle-timeout-seconds>...</idle-timeout-seconds>
       <cache-type>...<cache-type>
</stateful-session-cache>
```

stateful-session-clustering

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Optional element. Valid only for stateful session EJBs in a cluster.
Parent elements:	`weblogic-enterprise-bean`, `stateful-session-descriptor`

Function

The `stateful-session-clustering` stanza element specifies the following options that determine how WebLogic Server replicates stateful session EJB instances in a cluster:

- `home-is-clusterable`
- `home-load-algorithm`
- `home-call-router-class-name`
- `replication-type`

Example

The following excerpt shows the structure of a `entity-clustering` stanza:

```
<stateful-session-clustering>
        <home-is-clusterable>true</home-is-clusterable>
        <home-load-algorithm>random</home-load-algorithm>

<home-call-router-class-name>beanRouter</home-call-router-class-n
ame>
        <replication-type>InMemory</replication-type>
```

```
</stateful-session-clustering>
```

stateful-session-descriptor

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	One `stateful-session-descriptor` stanza is required for each stateful session EJB in the `.jar`.
Parent elements:	`weblogic-enterprise-bean`

Function

The `stateful-session-descriptor` element specifies the following deployment parameters that are applicable for stateful session EJBs in WebLogic Server:

- `stateful-session-cache`
- `lifecycle`
- `persistent-store-dir`
- `stateful-session-clustering`
- `allow-concurrent-calls`

Example

The following example shows the structure of the `stateful-session-descriptor` stanza:

```
<stateful-session-descriptor>
       <stateful-session-cache>...</stateful-session-cache>
       <lifecycle>...</lifecycle>
```

```
        <persistence>...</persistence>
        <allow-concurrent-calls>...</allow-concurrent-calls>
<persistent-store-dir>/weblogic/myserver</persistent-store-dir>
<stateful-session-clustering>...</stateful-session-clustering>
</stateful-session-descriptor>
```

stateless-bean-call-router-class-name

Range of values:	Valid router class name
Default value:	n/a
Requirements:	Optional element. Valid only for stateless session EJBs in a cluster.

Parent elements:	`weblogic-enterprise-bean`,
	`stateless-session-descriptor`
	`stateless-clustering`

Function

The `stateless-bean-call-router-class-name` element specifies the name of a custom class to use for routing bean method calls. This class must implement `weblogic.rmi.cluster.CallRouter()`. If specified, an instance of this class is called before each method call. The router class has the opportunity to choose a server to route to based on the method parameters. The class returns either a server name or null, which indicates that the current load algorithm should select the server.

Example

See "stateless-clustering" on page 10-77.

stateless-bean-is-clusterable

Range of values:	`true | false`
Default value:	`true`
Requirements:	Optional element. Valid only for stateless session EJBs in a cluster.

Parent elements:	`weblogic-enterprise-bean,`
	`stateless-session-descriptor`
	`stateless-clustering`

Function

Use `stateless-bean-is-clusterable` to specify whether a stateless session bean's `EJBObject` interface is clustered. Clustered `EJBObjects` support load balancing and failover.

If `stateless-bean-is-clusterable` is `true`, when a home interface of a clustered stateless session bean creates a bean instance, it returns a `EJBObject` stub to the client that lists all of the servers in the cluster. Given the stateless nature of the bean, any instance can service any client request

Example

See "stateless-clustering" on page 10-77.

stateless-bean-load-algorithm

Range of values:	`round-robin | random | weight-based`
Default value:	Value of `weblogic.cluster.defaultLoadAlgorithm`
Requirements:	Optional element. Valid only for stateless session EJBs in a cluster.

Parent elements:	`weblogic-enterprise-bean`,
	`stateless-session-descriptor`
	`stateless-clustering`

Function

`stateless-bean-load-algorithm` specifies the algorithm to use for load balancing between replicas of the EJB home. If this property is not defined, WebLogic Server uses the algorithm specified by the server property, `weblogic.cluster.defaultLoadAlgorithm`.

You can define `stateless-bean-load-algorithm` as one of the following values:

- `round-robin`: Load balancing is performed in a sequential fashion among the servers hosting the bean.
- `random`: Replicas of the EJB home are deployed randomly among the servers hosting the bean.
- `weight-based`: Replicas of the EJB home are deployed on host servers according to the servers' current workload.

Example

See "stateless-clustering" on page 10-77.

stateless-bean-methods-are-idempotent

Range of values:	`true | false`
Default value:	`false`
Requirements:	Optional element. Valid only for stateless session EJBs in a cluster.

Parent elements:	`weblogic-enterprise-bean,`
	`stateless-session-descriptor`
	`stateless-clustering`

Function

You can set this element to either `true` or `false`. Set `stateless-bean-methods-are-idempotent` to "true" only if the bean is written such that repeated calls to the same method with the same arguments has exactly the same effect as a single call. This allows the failover handler to retry a failed call without knowing whether the call actually completed on the failed server. Setting this property to `true` makes it possible for the bean stub to recover automatically from any failure as long as another server hosting the bean can be reached.

Example

See "stateless-clustering" on page 10-77.

stateless-clustering

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Optional element. Valid only for stateless session EJBs in a cluster.
Parent elements:	`weblogic-enterprise-bean`, `stateless-session-descriptor`

Function

The `stateless-clustering` element specifies the following options that determine how WebLogic Server replicates stateless session EJB instances in a cluster:

- `home-is-clusterable`
- `stateless-bean-is-clusterable`
- `stateless-bean-load-algorithm`
- `stateless-bean-call-router-class-name`
- `stateless-bean-methods-are-idempotent`

Example

The following excerpt shows the structure of a `stateless-clustering` stanza:

```
<stateless-clustering>

<stateless-bean-is-clusterable>true</stateless-bean-is-clusterable>

<stateless-bean-load-algorithm>random</stateless-bean-load-algorithm>
```

```
<stateless-bean-call-router-class-name>beanRouter</stateless-bean
-call-router-class-name>

<stateless-bean-methods-are-idempotent>true</stateless-bean-metho
ds-are-idempotent>

</stateless-clustering>
```

stateless-session-descriptor

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	One `stateless-session-descriptor` element is required for each stateless session EJB in the *.jar*.
Parent elements:	`weblogic-enterprise-bean`

Function

The `stateless-session-descriptor` element defines deployment parameters, such as caching, clustering, and persistence for stateless session EJBs in WebLogic Server.

Example

The following example shows the structure of the `stateless-session-descriptor` stanza:

```
<stateless-session-descriptor>

       <pool>...</pool>

       <stateless-clustering>...</stateless-clustering>
```

```
</stateless-session-descriptor>
```

transaction-descriptor

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Optional element.
Parent elements:	`weblogic-enterprise-bean`

Function

The `transaction-descriptor` element specifies options that define transaction behavior in WebLogic Server. Currently, this stanza includes only one element: `trans-timeout-seconds`.

Example

The following example shows the structure of the `transaction-descriptor` stanza:

```
<transaction-descriptor>
       <trans-timeout-seconds>20</trans-timeout-seconds>
<transaction-descriptor>
```

transaction-isolation

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Optional element.
Parent elements:	`weblogic-ejb-jar`

Function

The `transaction-isolation` element defines method-level transaction isolation settings for an EJB.

Example

The `transaction-isolation` stanza can contain the elements shown here:

```
<transaction-isolation>
       <isolation-level>Serializable</isolation-level>
       <method>
              <description>...</description>
              <ejb-name>...</ejb-name>
              <method-intf>...</method-intf>
              <method-name>...</method-name>
              <method-params>...</method-params>
       </method>
</transaction-isolation>
```

trans-timeout-seconds

Range of values:	`0` to *`max`*
Default value:	`30`
Requirements:	Optional element. Valid for any type of EJB.
Parent elements:	`weblogic-enterprise-bean`, `transaction-descriptor`

Function

The `trans-timeout-seconds` element specifies the maximum duration for an EJB's container-initiated transactions. If a transaction lasts longer than `trans-timeout-seconds`, WebLogic Server rolls back the transaction.

Example

See "transaction-descriptor" on page 10-79.

type-identifier

Range of values:	Valid string. `WebLogic_CMP_RDBMS` specifies WebLogic Server RDBMS-based persistence.
Default value:	n/a
Requirements:	Required only for entity EJBs that use container-managed persistence services.

Parent elements:	`weblogic-enterprise-bean,`
	`entity-descriptor,`
	`persistence`
	`persistence-type`
	`and`
	`weblogic-enterprise-bean,`
	`entity-descriptor,`
	`persistence`
	`persistence-use`

Function

The `type-identifier` element contains text that identifies an entity EJB persistence type. WebLogic Server RDBMS-based persistence uses the identifier, `WebLogic_CMP_RDBMS`. If you use a different persistence vendor, consult the vendor's documentation for information on the correct `type-identifier`.

Example

See "persistence-type" on page 10-54 for an example that shows the complete `persistence-type` definition for WebLogic Server RDBMS-based persistence.

type-storage

Range of values:	Valid string
Default value:	n/a
Requirements:	Required only for entity EJBs that use container-managed persistence services.

Parent elements:	`weblogic-enterprise-bean,`
	`entity-descriptor,`
	`persistence`
	`persistence-type`

Function

The `type-storage` element defines the full path of the file that stores data for this persistence type. The path must specify the file's location relative to the top level of the EJB's `.jar` deployment file or deployment directory.

WebLogic Server RDBMS-based persistence generally uses an XML file named `weblogic-cmp-rdbms-jar.xml` to store persistence data for a bean. This file is stored in the `META-INF` subdirectory of the `.jar` file.

Example

See "persistence-type" on page 10-54 for an example that shows the complete `persistence-type` definition for WebLogic Server RDBMS-based persistence.

type-version

Allowable values:	`5.1.0` for WebLogic persistence, EJB 1.1 `6.0` for WebLogic persistence, EJB 2.0.
Default value:	n/a
Requirements:	Required only for entity EJBs that use container-managed persistence services.

Parent elements:	`weblogic-enterprise-bean, entity-descriptor, persistence persistence-type`
	and
	`weblogic-enterprise-bean, entity-descriptor, persistence persistence-use`

Function

The `type-version` element identifies the version of the specified persistence type.

Note: If you use WebLogic Server RDBMS-based persistence, the specified version must *exactly* match the RDBMS persistence version for the WebLogic Server release. Specifying an incorrect version results in the error:

```
weblogic.ejb.persistence.PersistenceSetupException: Error
initializing the CMP Persistence Type for your bean: No installed
Persistence Type matches the signature of (identifier
'Weblogic_CMP_RDBMS', version 'version_number').
```

Example

See persistence-type for an example that shows the complete `persistence-type` definition for WebLogic Server RDBMS-based persistence.

weblogic-ejb-jar

Range of values:	N/A
Default value:	N/A
Requirements:	N/A
Parent elements:	N/A

Function

`weblogic-ejb-jar` is the root element of the weblogic component of the EJB deployment descriptor.

weblogic-enterprise-bean

Range of values:	
Default value:	
Requirements:	
Parent elements:	`weblogic-ejb-jar`

Function

The `weblogic-enterprise-bean` element contains the deployment information for a bean that is available in WebLogic Server.

5.1 weblogic-ejb-jar.xml Deployment Descriptor File Structure

The WebLogic Server 5.1 `weblogic-ejb-jar.xml` file defines the EJB document type definitions (DTD)s you use with EJB 1.1 beans. These deployment descriptor elements are WebLogic-specific. The top level elements in the WebLogic Server 5.1 `weblogic-ejb-jar.xml` are as follows:

- `description`
- `weblogic-version`
- `weblogic-enterprise-bean`
  - `ejb-name`
  - `caching-descriptor`
  - `persistence-descriptor`
  - `clustering-descriptor`
  - `transaction-descriptor`
  - `reference-descriptor`
  - `jndi-name`
  - `transaction-isolation`
- `security-role-assignment`

5.1 weblogic-ejb-jar.xml Deployment Descriptor Elements

The following sections describe WebLogic-Server 5.1 `weblogic-ejb-jar.xml` deployment descriptor elements.

caching-descriptor

The `caching-descriptor` stanza affects the number of EJBs in the WebLogic Server cache as well as the length of time before EJBs are passivated or pooled. The entire stanza, as well as each of its elements, is optional. WebLogic Server uses default values where no elements are defined.

The following is a sample `caching-descriptor` stanza that shows the caching elements described in this section:

```
<caching-descriptor>
   <max-beans-in-free-pool>500</max-beans-in-free-pool>
   <initial-beans-in-free-pool>50</initial-beans-in-free-pool>
   <max-beans-in-cache>1000</max-beans-in-cache>
   <idle-timeout-seconds>20</idle-timeout-seconds>
   <cache-strategy>Read-Write</cache-strategy>
   <read-timeout-seconds>0</read-timeout-seconds>
</caching-descriptor>
```

max-beans-in-free-pool

Note: This element is valid only for stateless session EJBs.

WebLogic Server maintains a free pool of EJBs for every bean class. This optional element defines the size of the pool. By default, `max-beans-in-free-pool` has no limit; the maximum number of beans in the free pool is limited only by the available memory. See "Stateful Session EJB Creation" on page 4-8 in "The WebLogic Server EJB Container and Supported Services" on page 4-1 for more information.

initial-beans-in-free-pool

Note: This element is valid only for stateless session EJBs.

If you specify a value for `initial-bean-in-free-pool`, WebLogic Server populates the free pool with the specified number of bean instances at startup. Populating the free pool in this way improves initial response time for the EJB, since initial requests for the bean can be satisfied without generating a new instance.

`initial-bean-in-free-pool` defaults to 0 if the element is not defined.

max-beans-in-cache

Note: This element is valid only for stateful session EJBs and entity EJBs.

This element specifies the maximum number of objects of this class that are allowed in memory. When `max-bean-in-cache` is reached, WebLogic Server passivates some EJBs that have not been recently used by a client. `max-beans-in-cache` also affects when EJBs are removed from the WebLogic Server cache, as described in "Stateful Session EJB Life Cycle" on page 4-7.

The default value of `max-beans-in-cache` is 100.

idle-timeout-seconds

`idle-timeout-seconds` defines the maximum length of time a stateful EJB should remain in the cache. After this time has elapsed, WebLogic Server may remove the bean instance if the number of beans in cache approaches the limit of `max-beans-in-cache`. See "EJB Life Cycle" on page 4-2 for more information.

`idle-timeout-seconds` defaults to 600 if you do not define the element.

cache-strategy

The `cache-strategy` element can be one of the following:

- `Read-Write`
- `Read-Only`

The default value is `Read-Write`. See "Setting Entity EJBs to Read-Only" on page 4-16 for more information.

read-timeout-seconds

The `read-timeout-seconds` element specifies the number of seconds between `ejbLoad()` calls on a `Read-Only` entity bean. By default, `read-timeout-seconds` is set to 600 seconds. If you set this value to 0, WebLogic Server calls `ejbLoad` only when the bean is brought into the cache.

persistence-descriptor

The `persistence-descriptor` stanza specifies persistence options for entity EJBs. The following shows all elements contained in the `persistence-descriptor` stanza:

```
<persistence-descriptor>
   <is-modified-method-name>. . .</is-modified-method-name>
   <delay-updates-until-end-of-tx>. .
.</delay-updates-until-end-of-tx>
   <persistence-type>
      <type-identifier>. . .</type-identifier>
      <type-version>. . .</type-version>
      <type-storage>. . .</type-storage>
   </persistence-type>
   <db-is-shared>. . .</db-is-shared>
   <stateful-session-persistent-store-dir>
      . . .
   </stateful-session-persistent-store-dir>
   <persistence-use>. . .</persistence-use>
</persistence-descriptor>
```

is-modified-method-name

`is-modified-method-name` specifies a method that WebLogic Server calls when the EJB is stored. The specified method must return a `boolean` value. If no method is specified, WebLogic Server always assumes that the EJB has been modified and always saves it.

Providing a method and setting it as appropriate can improve performance. However, any errors in the method's return value can cause data inconsistency problems. See "Using is-modified-method-name to Limit Calls to ejbStore() (EJB 1.1 Only)" on page 4-14 for more information.

delay-updates-until-end-of-tx

Set this property to `true` (the default), to update the persistent store of all beans in a transaction at the completion of the transaction. This generally improves performance by avoiding unnecessary updates. However, it does not preserve the ordering of database updates within a database transaction.

If your datastore uses an isolation level of `TRANSACTION_READ_UNCOMMITTED`, you may want to allow other database users to view the intermediate results of in-progress transactions. In this case, set `delay-updates-until-end-of-tx` to `false` to update the bean's persistent store at the conclusion of each method invoke. See "ejbLoad() and ejbStore() Behavior for Entity EJBs" on page 4-12 for more information.

Note: Setting `delay-updates-until-end-of-tx` to false does not cause database updates to be "committed" to the database after each method invoke; they are only sent to the database. Updates are committed or rolled back in the database only at the conclusion of the transaction.

persistence-type

A `persistence-type` defines a persistence service that can be used by an EJB. You can define multiple `persistence-type` entries in `weblogic-ejb-jar.xml` for testing with multiple persistence services. Only the persistence type defined in "persistence-use" on page 10-92 is used during deployment.

`persistence-type` includes several elements that define the properties of a service:

- `type-identifier` contains text that identifies the specified persistence type. For example, WebLogic Server RDBMS persistence uses the identifier, `WebLogic_CMP_RDBMS`.
- `type-version` identifies the version of the specified persistence type.

Note: The specified version must *exactly* match the RDBMS persistence version for the WebLogic Server release. Specifying an incorrect version results in the error:

```
weblogic.ejb.persistence.PersistenceSetupException: Error
initializing the CMP Persistence Type for your bean: No installed
Persistence Type matches the signature of (identifier
'Weblogic_CMP_RDBMS', version 'version_number').
```

- `type-storage` defines the full path of the file that stores data for this persistence type. The path must specify the file's location relative to the top level of the EJB's *.jar* deployment file or deployment directory.

  WebLogic Server RDBMS-based persistence generally uses an XML file named `weblogic-cmp-rdbms-jar.xml` to store persistence data for a bean. This file is stored in the `META-INF` subdirectory of the *.jar* file.

The following shows an example `persistence-type` stanza with values appropriate for WebLogic Server RDBMS persistence:

```
<persistence-type>

    <type-identifier>WebLogic_CMP_RDBMS</type-identifier>

    <type-version>5.1.0</type-version>

    <type-storage>META-INF\weblogic-cmp-rdbms-jar.xml</type-stora
ge>

</persistence-type>
```

db-is-shared

The `db-is-shared` element applies only to entity beans. When set to `true` (the default value), WebLogic Server assumes that EJB data could be modified between transactions and reloads data at the beginning of each transaction. When set to `false`, WebLogic Server assumes that it has exclusive access to the EJB data in the persistent store. See "Using db-is-shared to Limit Calls to ejbLoad()" on page 4-12 for more information.

stateful-session-persistent-store-dir

`stateful-session-persistent-store-dir` specifies the file system directory where WebLogic Server stores the state of passivated stateful session bean instances.

persistence-use

The `persistence-use` property is similar to `persistence-type`, but it defines the persistence service actually used during deployment. `persistence-use` uses the `type-identifier` and `type-version` elements defined in a `persistence-type` to identify the service.

For example, to actually deploy an EJB using the WebLogic Server RDBMS-based persistence service defined in `persistence-type`, the `persistence-use` stanza would resemble:

```
<persistence-use>
   <type-identifier>WebLogic_CMP_RDBMS</type-identifier>
   <type-version>5.1.0</type-version>
</persistence-use>
```

clustering-descriptor

The `clustering-descriptor` stanza defines the replication properties and behavior for EJBs deployed in a WebLogic Server cluster. The `clustering-descriptor` stanza and each of its elements are optional, and are not applicable to single-server systems.

The following shows all elements contained in the `clustering-descriptor` stanza:

```
<clustering-descriptor>
   <home-is-clusterable>...</home-is-clusterable>
   <home-load-algorithm>...</home-load-algorithm>
   <home-call-router-class-name>.. .</home-call-router-class-name>
   <stateless-bean-is-clusterable>...</stateless-bean-is-
   clusterable>
   <stateless-bean-load-algorithm>...
   </stateless-bean-load-algorithm>
```

```
   <stateless-bean-call-router-class-name>. .
.</stateless-bean-call-router-class-name>

   <stateless-bean-methods-are-idempotent>. .
.</stateless-bean-methods-are-idempotent>

</clustering-descriptor>
```

home-is-clusterable

You can set this element to either `true` or `false`. When `home-is-clusterable` is true, the EJB can be deployed from multiple WebLogic Servers in a cluster. Calls to the home stub are load-balanced between the servers on which this bean is deployed, and if a server hosting the bean is unreachable, the call automatically fails over to another server hosting the bean.

home-load-algorithm

`home-load-algorithm` specifies the algorithm to use for load balancing between replicas of the EJB home. If this property is not defined, WebLogic Server uses the algorithm specified by the server property, `weblogic.cluster.defaultLoadAlgorithm`.

You can define `home-load-algorithm` as one of the following values:

- `round-robin`: Load balancing is performed in a sequential fashion among the servers hosting the bean.
- `random`: Replicas of the EJB home are deployed randomly among the servers hosting the bean.
- `weight-based`: Replicas of the EJB home are deployed on host servers according to the servers' current workload.

home-call-router-class-name

`home-call-router-class-name` specifies the custom class to use for routing bean method calls. This class must implement `weblogic.rmi.cluster.CallRouter`. If specified, an instance of this class is called before each method call. The router class has the opportunity to choose a server to route to based on the method parameters. The class returns either a server name or null, which indicates that the current load algorithm should select the server.

stateless-bean-is-clusterable

Use `stateless-bean-is-clusterable` to specify whether a stateless session bean's `EJBObject` interface is clustered. Clustered `EJBObject`s support load balancing and failover.

If `stateless-bean-is-clusterable` is `true`, when a home interface of a clustered stateless session bean creates a bean instance, it returns a `EJBObject` stub to the client that lists all of the servers in the cluster. Given the stateless nature of the bean, any instance can service any client request.

stateless-bean-load-algorithm

Use `stateless-bean-is-clusterable` to specify whether a stateless session bean's `EJBObject` interface is clustered. Clustered `EJBObject`s support load balancing and failover.

If `stateless-bean-is-clusterable` is `true`, when a home interface of a clustered stateless session bean creates a bean instance, it returns a `EJBObject` stub to the client that lists all of the servers in the cluster. Given the stateless nature of the bean, any instance can service any client request.

stateless-bean-call-router-class-name

This property is similar to `home-call-router-class-name`, but it is applicable only to stateless session EJBs.

stateless-bean-methods-are-idempotent

You can set this element to either `true` or `false`. Set `stateless-bean-methods-are-idempotent` to `true` only if the bean is written such that repeated calls to the same method with the same arguments has exactly the same effect as a single call. This allows the failover handler to retry a failed call without knowing whether the call actually completed on the failed server. Setting this property to `true` makes it possible for the bean stub to automatically recover from any failure as long as another server hosting the bean can be reached.

Note: This property is applicable only to stateless session EJBs.

transaction-descriptor

The `transaction-descriptor` stanza contains elements that define transaction behavior in WebLogic Server. Currently, this stanza includes only one element:

```
<transaction-descriptor>
   <trans-timeout-seconds>20</trans-timeout-seconds>
<transaction-descriptor>
```

trans-timeout-seconds

The `trans-timeout-seconds` element specifies the maximum duration for the EJB's container-initiated transactions. If a transaction lasts longer than `trans-timeout-seconds`, WebLogic Server rolls back the transaction.

If you specify no value for `trans-timeout-seconds`, container-initiated transactions timeout after five minutes, by default.

reference-descriptor

The `reference-descriptor` stanza maps references in the `ejb-jar.xml` file to the JNDI names of actual resource factories and EJBs available in WebLogic Server.

The `reference-descriptor` stanza contains one or more additional stanzas to define resource factory references and EJB references. The following shows the organization of these elements:

```
<reference-descriptor>
   <resource-description>
      <res-ref-name>. . .</res-ref-name>
      <jndi-name>. . .</jndi-name>
   </resource-description>
   <ejb-reference-description>
      <ejb-ref-name>. . .</ejb-ref-name>
```

```
        <jndi-name>. . .</jndi-name>
      </ejb-reference-description>
    </reference-descriptor>
```

resource-description

The following elements define an individual `resource-description`:

- `res-ref-name` specifies a resource reference name. This is the reference that the EJB provider places within the `ejb-jar.xml` deployment file.
- `jndi-name` specifies the JNDI name of an actual resource factory available in WebLogic Server.

ejb-reference-description

The following elements define an individual `ejb-reference-description`:

- `ejb-ref-name` specifies an EJB reference name. This is the reference that the EJB provider places within the `ejb-jar.xml` deployment file.
- `jndi-name` specifies the JNDI name of an actual EJB available in WebLogic Server.

enable-call-by-reference

By default, EJB methods called from within the same EAR pass arguments by reference. This increases the performance of method invocation since parameters are not copied.

If you set `enable-call-by-reference` to `false`, parameters to EJB methods are copied (pass by value) in accordance with the EJB 1.1 specification. Pass by value is always necessary when the EJB is called remotely (not from within the same application).

jndi-name

The `jndi-name` element specifies a jndi-name for a bean, resource, or reference.

transaction-isolation

The `transaction-isolation` stanza specifies the transaction isolation level for EJB methods. The stanza consists of one or more `isolation-level` elements that apply to a range of EJB methods. For example:

```
<transaction-isolation>
        <isolation-level>Serializable</isolation-level>
        <method>
                <description>...</description>
                <ejb-name>...</ejb-name>
                <method-intf>...</method-intf>
                <method-name>...</method-name>
                <method-params>...</method-params>
        </method>
</transaction-isolation>
```

The following sections describe each element in `transaction-isolation`.

isolation-level

`isolation-level` defines a valid transaction isolation level to apply to specific EJB methods. The following are possible values for `isolation-level`:

- `TRANSACTION_READ_UNCOMMITTED`: The transaction can view uncommitted updates from other transactions.
- `TRANSACTION_READ_COMMITTED`: The transaction can view only committed updates from other transactions.

- TRANSACTION_REPEATABLE_READ: Once the transaction reads a subset of data, repeated reads of the same data return the same values, even if other transactions have subsequently modified the data.
- TRANSACTION_SERIALIZABLE: Simultaneously executing this transaction multiple times has the same effect as executing the transaction multiple times in a serial fashion.

Refer to your database documentation for more information on the implications and support for different isolation levels.

method

The method stanza defines the EJB methods to which an isolation level applies. method defines a range of methods using the following elements:

- description is an optional element that describes the method.
- ejb-name identifies the EJB to which WebLogic Server applies isolation level properties.
- method-intf is an optional element that specifies the EJB interface to which WebLogic Server applies isolation level properties, if the method has the same signature in multiple interfaces. The value of this element must be "Home", "Remote", "Local", or "Localhome". If you do not specify method-intf, you can apply an isolation to methods in both interfaces.
- method-name specifies either the name of an EJB method or an asterisk (*) to designate all EJB methods.
- method-params is an optional stanza that lists the Java types of each of the method's parameters. The type of each parameter must be listed in order, using individual method-param elements within the method-params stanza.

For example, the following method stanza designates all methods in the "AccountBean" EJB:

```
<method>
        <ejb-name>AccountBean</ejb-name>
        <method-name>*</method-name>
</method>
```

The following stanza designates all methods in the remote interface of "AccountBean:"

```
<method>
        <ejb-name>AccountBean</ejb-name>
        <method-intf>Remote</method-intf>
        <method-name>*</method-name>
</method>
```

security-role-assignment

The `security-role-assignment` stanza maps application roles in the `ejb-jar.xml` file to the names of security principals available in WebLogic Server.-

`security-role-assignment` can contain one or more pairs of the following elements:

- `role-name` is the application role name that the EJB provider placed in the `ejb-jar.xml` deployment file.
- `principal-name` specifies the name of an actual WebLogic Server principal.

CHAPTER

11 weblogic-cmp-rdbms-jar.xml Document Type Definitions

The chapter describes both the EJB 5.1 and EJB 6.0 deployment descriptor elements found in the `weblogic-cmp-rdbms-jar.xml` file, the weblogic-specific XML document type definitions (DTD) file. Use these definitions to create the WebLogic-specific `weblogic-cmp-rdbms-jar.xml` file that is part of your EJB deployment.

The following sections provide a complete reference of both versions of the WebLogic-specific XML including the DOCTYPE header information. Use these deployment descriptor elements to specify container-managed-persistence (CMP).

EJB Deployment Descriptors

The EJB deployment descriptors provide structural and application assembly information for an enterprise bean. You specify this information by specifying values for the deployment descriptors in three EJB XML DTD files. These files are:

- `ejb-jar.xml`
- `weblogic-ejb-jar.xml`
- `weblogic-cmp-rdbms-jar.xml`

You package these three XML files with the EJB and other classes into a deployable EJB component, usually a JAR file, called `ejb.jar`.

The `ejb-jar.xml` file is based on the deployment descriptors found in Sun Microsystems's `ejb.jar.xml` file. The other two XML files are weblogic-specific files that are based on the deployment descriptors found in `weblogic-ejb-jar.xml` and `weblogic-cmp-rdbms-jar.xml`.

DOCTYPE Header Information

When editing or creating XML deployment files, it is critical to include the correct `DOCTYPE` header for each deployment file. In particular, using an incorrect `PUBLIC` element within the `DOCTYPE` header can result in parser errors that may be difficult to diagnose. The correct text for the `PUBLIC` element for each XML deployment file is as follows.

The correct text for the `PUBLIC` element for the WebLogic Server-specific `weblogic-cmp-rdbms-jar.xml` files are as follows.

XML File	PUBLIC Element String
`weblogic-cmp-rdbms -jar.xml`	`'-// BEA Systems, Inc.//DTD WebLogic 6.0.0 EJB RDBMS20 Persistence//EN' 'http://www.bea.com/servers/wls600/dtd/weblogic-rdbms 20-persistence-600.dtd'`

XML File	PUBLIC Element String
weblogicmp-rdbms -jar.xml	'-//BEA Systems, Inc.//DTD WebLogic 5.1.0 EJB RDBMS Persistence//EN' http://www.bea.com/servers/wls510/dtd/weblogic-rdbms-persistence.dtd

The correct text for the PUBLIC elements for the Sun Microsystem-specific ejb-jar files are as follows.

XML File	PUBLIC Element String
ejb-jar.xml	'-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN' '
ejb-jar.xml	'-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN' 'http://www.java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'

For example, the entire DOCTYPE header for a weblogic-cmp-rdbms-jar.xml file is as follows:

```
<!DOCTYPE weblogic-cmp-rdbms-jar PUBLIC
'-//BEA Systems, Inc.//DTD WebLogic 6.0.0 EJB RDBMS20
Persistence//EN'
'http://www.bea.com/servers/wls600/dtd/weblogic-rdbms20-persisten
ce-600.dtd '>
```

XML files with incorrect header information may yield error messages similar to the following, when used with a utility that parses the XML (such as ejbc):

```
SAXException: This document may not have the identifier 'identifier_name'
```

identifier_name generally includes the invalid text from the PUBLIC element.

Document Type Definitions (DTDs) for Validation

The contents and arrangement of elements in your XML files must conform to the Document Type Definition (DTD) for each file you use. WebLogic Server utilities ignore the DTDs embedded within the DOCTYPE header of XML deployment files, and

instead use the DTD locations that were installed along with the server. However, the DOCTYPE header information must include a valid URL syntax in order to avoid parser errors.

Note: Most browsers do not display the contents of files having the `.dtd` extension. To view the DTD file contents in your browser, save the links as text files and view them with a text editor.

weblogic-cmp-rdbms-jar.xml

The following links provide the public DTD locations for the `weblogic-cmp-rdbms-jar.xml` deployment files used with WebLogic Server:

- For `weblogic-cmp-rdbms-jar.xml` 6.0 DTD:

  `http://www.bea.com/servers/wls600/dtd/weblogic-rdbms20-persistence-600.dtd` contains the DTD that defines container-managed persistence properties for entity EJBs. This DTD is changed from WebLogic Server Version 5.1, and you must still include a `weblogic-cmp-rdbms-jar.xml` file for entity EJBs using WebLogic Server RDBMS-based persistence.

  Use the existing DTD file located at:

  `http://www.bea.com/servers/wls600/dtd/weblogic-rdbms-persistence-600.dtd`

- For `weblogic-cmp-rdbms-jar.xml` 5.1 DTD:

  `weblogic-rdbms-persistence.dtd` contains the DTD that defines container-managed persistence properties for entity EJBs. This DTD is used to create the `weblogic-rdbms-persistence.xml` file for using WebLogic Server persistence services. Third-party persistence vendors may also create XML deployment files that conform to this DTD. This file is located at *http://www.bea.com/servers/wls510/dtd/weblogic-rdbms-persistence.dtd*

ejb-jar.xml

The following links provide the public DTD locations for the `ejb-jar.xml` deployment files used with WebLogic Server:

- For `ejb-jar.xml` 2.0 DTD:

`http://www.java.sun.com/dtd/ejb-jar_2_0.dtd` contains the DTD for the standard `ejb-jar.xml` deployment file, required for all EJBs. This DTD is maintained as part of the JavaSoft EJB 2.0 specification; refer to the JavaSoft specification for information about the elements used in `ejb-jar.dtd`.

- For `ejb-jar.xml` 1.1 DTD:

  `ejb-jar.dtd` contains the DTD for the standard `ejb-jar.xml` deployment file, required for all EJBs. This DTD is maintained as part of the JavaSoft EJB 1.1 specification; refer to the JavaSoft specification for information about the elements used in `ejb-jar.dtd`.

Note: Refer to the appropriate JavaSoft EJB specification for a description of the `ejb-jar.xml` deployment descriptors.

6.0 weblogic-cmp-rdbms-jar.xml Deployment Descriptor File Structure

`weblogic-cmp-rdbms-jar.xml` defines deployment descriptors for a entity EJBs that uses WebLogic Server RDBMS-based persistence services. The EJB 2.0 container uses a version of `weblogic-cmp-rdbms-jar.xml` that is different from the one shipped with WebLogic Server Version 5.1. See Locking Services for Entity EJBs for more information.

You can continue to use the earlier `weblogic-cmp-rdbms-jar.xml` DTD for EJB 1.1 beans that you will deploy on the WebLogic Server Version 6.0. However, if you want to use any of the new CMP 2.0 features, you must use the new DTD described below.

The top-level element of the WebLogic Server 6.0 `weblogic-cmp-rdbms-jar.xml` consists of a `weblogic-rdbms-jar` stanza:

```
description

weblogic-version

weblogic-rdbms-jar

       weblogic-rdbms-bean
              ejb-name
```

```
            data-source-name
            table-name
            field-name
            field-map
            field-group
            weblogic-query
            delay-database-insert-until
            automatic-key-generation

      weblogic-rdbms-relation
            relation-name
      table-name
      weblogic-relationship-role
```

6.0 weblogic-cmp-rdbms-jar.xml Deployment Descriptor Elements

automatic-key-generation

Range of values:	n/a
Default value:	n/a
Requirements:	Optional.
Parent elements:	`weblogic-rdbms-bean`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `automatic-key-generation` element specifies the use of the Sequence/Key Generation feature.

Example

The `XML` stanza can contain the elements shown here:

```
<automatic-key-generation>
       <generator-type>ORACLE</generator-type>
       <generator-name>test_sequence</generator-name>
       <key-cache-size>10</key-cache-size>
</automatic-key-generation>


<automatic-key-generation>
       <generator-type>SQL-SERVER</generator-type>
</automatic-key-generation>


<automatic-key-generation>
       <generator-type>NAMED_SEQUENCE_TABLE</generator-type>
       <generator-name>MY_SEQUENCE_TABLE_NAME</generator-name>
```

```
        <key-cache-size>100</key-cache-size>
</automatic-key-generation>
```

cmp-field

Range of values:	Valid name
Default value:	n/a
Requirements:	Field is case sensitive and must match the name of the field in the bean and must also have a `cmp-field` entry in the `ejb-jar.xml`.

Parent elements:	`weblogic-rdbms-bean`
	`field-map`
	`weblogic-rdbms-relation`
	`field-group`

Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

This name specifies the mapped field in the bean instance which should be populated with information from the database.

Example

See “field-map” on page 11-21.

cmr-field

Range of values:	Valid name
Default value:	n/a
Requirements:	The field referenced in `cmr-field` must have a matching `cmr-field` entry in the `ejb-jar.xml`.
Parent elements:	`weblogic-rdbms-relation` `field-group`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `cmr-field` element specifies the name of a cmr-field.

Example

The `XML` stanza can contain the elements shown here:

```
<weblogic-rdbms-jar>
       <weblogic-rdbms-relation>
              <field-group>employee</field-group>
            <cmp-field>employee stock purchases</cmp-field>
            <cmr-field>stock options</cmr-field>
     </weblogic-rdbms-relation>
</weblogic-rdbms-jar>
```

column-map

Range of values:	n/a.
Default value:	n/a
Requirements:	The `key-column` element is not specified, if the `foreign-key-column` refers to a remote bean.
Parent elements:	`weblogic-rdbms-bean` `weblogic-relationship-role`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

This element represents the mapping of a foreign key column in one table in the database to a corresponding primary key. The two columns may or may not be in the same table. The tables to which the column belong are implicit from the context in which the `column-map` element appears in the deployment descriptor.

Example

```
<column-map
   <foreign-key-column>account-id</foreign-key-column>
   <key-column>id</key-column>
</column-map>
```

create-default-dbms-tables

Range of values:	True | False.
Default value:	False
Requirements:	Use this element only for convenience during the development and prototyping phases. This is because the Table Schema in the DBMS CREATE statement used will be the container's best approximation of the definition. A production environment most likely, will require a more precise schema definition.
Parent elements:	`weblogic-rdbms-jar`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `create-default-dbms-table` element turns on or off a feature that automatically creates a default table based on the descriptions in the deployment files and the bean class. When set to False, this feature is turned off and table will not automatically be generated. When set to `True`, this feature is turned on and the table is automatically created. If `TABLE CREATION` fails, a `Table Not Found` error is thrown and the table must be created by hand.

Example

The following example specifies the `create-default-dbms-tables` element.

```
<create-default-dbms-tables>True</create-default-dbms-tables>
```

data-source-name

Range of values:	Valid name of the data source used for all data base connectivity for this bean.
Default value:	n/a
Requirements:	Must be defined as a standard WebLogic Server JDBC data source for database connectivity. See Programming WebLogic JDBC for more information.
Parent elements:	weblogic-rdbms-bean
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `data-source-name` that specifies the JDBC data source name to be used for all database connectivity for this bean.

Example

See “table-name” on page 11-37.

db-cascade-delete

Range of values:	
Default value:	n/a
Requirements:	Only supported for Oracle database. Can only be specified for one-to-one or one-to-many relationships.
Parent elements:	`weblogic-rdbms-bean` `weblogic-relationship-role`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `db-cascade-delete` element specifies whether the database cascade feature is turned on. If this element is not specified, WebLogic Server assumes that database cascade delete is not specified.

Example

See “Cascade Delete Method” on page 5-16.

dbms-column

Range of values:	Valid name
Default value:	n/a
Requirements:	`dbms-column` is case maintaining, although not all database are case sensitive.
Parent elements:	`weblogic-rdbms-bean` `field-map`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The name of the database column to which the field should be mapped.

Example

See “field-map” on page 11-21.

dbms-column-type

Range of values:	Valid name
Default value:	n/a
Requirements:	`Available for use with Oracle database only.`
Parent elements:	`weblogic-rdbms-bean` `field-map`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The dbms-column-type element maps the current field to a Blob or Clob in an Oracle database or a LongString in a Sybase database. This element can be one of the following:

- OracleBlob
- OracleCLob
- LongString

Example

```
<field-map>
       <cmp-field>photo</cmp-field>
       <dbms-column>PICTURE</dbms-column>
       <dbms_column-type>OracleBlob</dbms-column-type>
</field-map>
```

delay-database-insert-until

Range of values:	
Default value:	`ejbPostCreate`
Requirements:	Database insert is delayed until after `ejbPostCreate` when a `cmr-field` is mapped to a `foreign-key column` that does not allow null values. In this case, the `cmr-field` must be set to a non-null value in `ejbPostCreate` before the bean is inserted into the database. The `cmr-fields` may not be set during `ejbCreate`, before the primary key of the bean is known.
Parent elements:	`weblogic-rdbms-bean`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `delay-database-insert-until` element specifies the precise time when a new bean that uses RDBMS CMP is inserted into the database.

It is advisable to delay the database insert until after the `ejbPostCreate` method modifies the persistent fields of the bean. This can yield better performance by avoiding an unnecessary store operation.

For maximum flexibility, you should avoid creating related beans in your `ejbPostCreate` method. This may make delaying the database insert impossible if database constraints prevent related beans from referring to a bean that has not yet been created.

Example

The following example specifies the `delay-database-insert-until` element.

```
<delay-database-insert-until>ejbPostCreate</delay-database-insert
-until>
```

ejb-name

Range of values:	Valid name of an EJB.
Default value:	n/a
Requirements:	Must match the `ejb-name` of the cmp entity bean defined in the `ejb-jar.xml`.
Parent elements:	`weblogic-rdbms-bean`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The name that specifies an EJB as defined in the ejb-cmp-rdbms.xml. This name must match the `ejb-name` of a cmp entity bean contained in the `ejb-jar.xml`.

Example

See “table-name” on page 11-37.

enable-tuned-updates

Note: This deployment descriptor applies to EJB 1.1 only.

Range of values:	`True/False`

Default value:	`True`
Requirements:	
Parent elements:	`weblogic-rdbms-bean`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `enable-tuned-updates` element specifies that when `ejbStore` is called that the EJB container automatically determine which container-managed fields have been modified and then writes only those fields back to the database.

Example

The following examples shows how to specify the `enable-tuned-updates` element.

```
<enable-tuned-updates>True</enable-tuned-updates>
```

field-group

Range of values:	Valid name
Default value:	A special group named `default` is used for finders and relationships that have no group specified.
Requirements:	The default group contains all of a bean's cmp-fields, but none of its cmr-fields.
Parent elements:	`weblogic-rdbms-relation`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `field-group` element represents a subset of the cmp and cmr-fields of a bean. Related fields in a bean can be put into groups that are faulted into memory together as a unit. A group can be associated with a finder or relationship, so that when a bean is loaded as the result of executing a finder or following a relationship, only the fields specified in the group are loaded.

A field may belong to multiple groups. In this case, the `getXXX` method for the field faults in the first group that contains the field.

Example

The `XML` stanza can contain the elements shown here:

```
<weblogic-rdbms-bean>
       <ejb-name>XXXBean</ejb-name>
       <field-group>
              <group-name>medical-data</group-name>
              <cmp-field>insurance</cmp-field>
              <cmr-field>doctors</cmr-fields>
       </field-group>
</weblogic-rdbms-bean>
```

field-map

Range of values:	Valid name
Default value:	n/a
Requirements:	Field mapped to the column in the database must correspond to a cmp field in the bean.
Parent elements:	`weblogic-rdbms-bean`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The name of the mapped field for a particular column in a database that corresponds to a cmp field in the bean instance.

Example

The `XML` stanza can contain the elements shown here:

```
<weblogic-rdbms-jar>
  <weblogic-rdbms-bean>

      <field-map>
          <cmp-field>accountId</cmp-field>
              <dbms-column>id</dbms-column>
      </field-map>

      <field-map>
          <cmp-field>balance</cmp-field>
              <dbms-column>bal</dbms-column>
      </field-map>

     <field-map>
          <cmp-field>accountType</cmp-field>
              <dbms-column>type</dbms-column>
      </field-map>
```

```
  </weblogic-rdbms-bean>
</weblogic-rdbms-jar>
```

foreign-key-column

Range of values:	Valid name
Default value:	n/a
Requirements:	Must correspond to a column of a foreign key.
Parent elements:	`weblogic-rdbms-bean` `column-map`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `foreign-key-column` element represents a column of a foreign key in the database.

Example

See “column-map” on page 11-11.

generator-name

Range of values:	n/a
Default value:	n/a
Requirements:	Optional.
Parent elements:	`weblogic-rdbms-bean` `automatic-key-generation`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `generator-name` element is used to specify the name of the generator.

For example;

- If the `generator-type` element is `ORACLE`, then the `generator-name` element would be the name of the `ORACLE_SEQUENCE` to be used.
- If the `generator-type` element is `NAMED_SEQUENCE_TABLE`, then the `generator-name` element would be the name of the `SEQUENCE_TABLE` to be used.

Example

See “automatic-key-generation” on page 11-8.

generator-type

Range of values:	n/a
Default value:	n/a
Requirements:	Optional
Parent elements:	`weblogic-rdbms-bean` `automatic-key-generation`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The generator-type element specifies the key generation method to use. The options include:

- `ORACLE`
- `SQL_SERVER`
- `NAMED_SEQUENCE_TABLE`

Example

See “automatic-key-generation” on page 11-8.

group-name

Range of values:	Valid name
Default value:	n/a
Requirements:	n/a

Parent elements:	`weblogic-rdbms-relation`
	`field-group`
	`weblogic-rdbms-bean`
	`finder`
	`finder-query`

Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `group-name` element specifies the name of a field group.

Example

The `XML` stanza can contain the elements shown here:

```
<weblogic-rdbms-jar>
      <weblogic-rdbms-relation>
          <field-group>employee</field-group>
              <cmp-field>employee stock purchases</cmp-field>
              <cmr-field>stock options</cmr-field>
                  <group-name>financial data</group-name>
      </weblogic-rdbms-relation>
</weblogic-rdbms-jar>
```

include-updates

Range of values:	`True | False`
Default value:	`False`
Requirements:	The default value, which is `False`, provides the best performance.
Parent elements:	weblogic-rdbms-bean weblogic-query
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The include-updates element specifies that updates made during the current transaction must be reflected in the result of a query. If this element is set to `True`, the container will flush all changes made by the current transaction to disk before executing the query.

Example

The `XML` stanza can contain the elements shown here:

```
<include-updates>False</include_updates>
```

key-cache-size

Range of values:	n/a
Default value:	1
Requirements:	Optional
Parent elements:	`weblogic-rdbms-bean` `automatic-key-generation`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The key-cache-size element specifies the optional size of the primary key cache available in the automatic primary key generation feature.

Example

See “automatic-key-generation” on page 11-8.

key-column

Range of values:	Valid name
Default value:	n/a
Requirements:	Must correspond to a column of a primary key.
Parent elements:	`weblogic-rdbms-bean` `column-map`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `key-column` element represents a column of a primary key in the database.

Example

See “column-map” on page 11-11.

max-elements

Range of values:	n/a
Default value:	n/a
Requirements:	n/a
Parent elements:	weblogic-rdbms-bean weblogic-query
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

`max-elements` specifies the maximum number of elements that should be returned by a multi-valued query. This element is similar to the `maxRows` feature in JDBC.

Example

The `XML` stanza can contain the elements shown here:

```
<max-elements>100</max-elements>

      <!ELEMENT max-element (PCDATA)>
```

method-name

Range of values:	n/a
Default value:	n/a
Requirements:	The '*' character may not be used as a wildcard.
Parent elements:	`weblogic-rdbms-bean` `query-method`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `method-name` element specifies the name of a finder or `ejbSelect` method.

Example

See "weblogic-query" on page 11-39.

method-param

Range of values:	Valid name
Default value:	n/a
Requirements:	n/a
Parent elements:	`weblogic-rdbms-bean` `method-params`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `method-param` element contains the fully qualified Java type name of a method parameter.

Example

The `XML` stanza can contain the elements shown here:

```
<method-param>java.lang.String</method-param>
```

method-params

Range of values:	list of valid names
Default value:	n/a
Requirements:	n/a
Parent elements:	`weblogic-rdbms-bean` `query-method`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `method-params` element contains an ordered list of the fully-qualified Java type names of the method parameters.

Example

See “weblogic-query” on page 11-39.

query-method

Range of values:	n/a
Default value:	n/a
Requirements:	n/a
Parent elements:	`weblogic-rdbms-bean`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `query-method` element specifies the method that is associated with a `weblogic-query`. It also uses the same format as the `ejb-jar.xml` descriptor.

Example

See “weblogic-query” on page 11-39.

relation-name

Range of values:	Valid name
Default value:	n/a
Requirements:	Must match the `ejb-relation-name` of an ejb-relation in the associated `ejb-jar.xml` descriptor file.
Parent elements:	`weblogic-rdbms-relation`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `relation-name` element specifies the name of a relation.

Example

The `XML` stanza can contain the elements shown here:

```
<weblogic-rdbms-jar>
        <weblogic-rdbms-relation>
                <relation-name>stocks-holders</relation-name>
                        <table-name>stocks</table-name>
        </weblogic-rdbms-relation>
</weblogic-rdbms-jar>
```

relationship-role-name

Range of values:	Valid name
Default value:	n/a
Requirements:	The name must match the `ejb-relationship-role-name` of an `ejb-relationship-role` in the associated `ejb-jar.xml` descriptor file.
Parent elements:	`weblogic-rdbms-relation` `weblogic-relationship-role`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `relationship-role-name` element specifies the name of a relationship role.

Example

The `XML` stanza can contain the elements shown here:

```
<weblogic-rdbms-jar>

     <weblogic-rdbms-relation>

       <weblogic-relationship-role>stockholder</weblogic-
relationship-role>

            <relationship-role-name>stockholders</relationship-
role-name>

       </weblogic-rdbms-relation>

</weblogic-rdbms-jar>
```

sql-select-distinct

Range of values:	`True | False`
Default value:	`False`
Requirements:	The Oracle database does not allow you to use a SELECT DISTINCT in conjunction with a FOR UPDATE clause. Therefore, you cannot use the `sql-select-distinct` element if any bean in the calling chain has a method with a `transaction-isolation` element set to the `isolation-level` sub element with a value of `TRANSACTION_READ_COMMITED_FOR_UPDATE` You specify the `transaction-isolation` element in the `weblogic-ejb-jar.xml` file.
Parent elements:	`weblogic-query`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `sql-select-distinct` element controls whether the generated SQL SELECT statement will contain a a DISTINCT qualifer. Using the DISTINCT qualifer caused the database to return unique rows.

Example

The `XML` example contains the element shown here:

```
<sql-select-distinct>True</sql-select-distinct>
```

table-name

Range of values:	Valid, fully qualified SQL name of the source table in the database.
Default value:	n/a
Requirements:	`table-name` must be set in all cases.

Parent elements:	`weblogic-rdbms-bean`
	
	`weblogic-rdbms-bean`
	`  weblogic-rdbms-relation`

Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The fully qualified SQL name of the table. The user defined for the `data-source` for this bean must have read and write privileges for this table, but does not necessarily need schema modification privileges.

Example

```
<table-name>Accounts</table-name>
```

weblogic-ql

Range of values:	n/a
Default value:	n/a
Requirements:	n/a
Parent elements:	weblogic-rdbms-bean weblogic-query
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `weblogic-ql` element specifies a query that contains a WebLogic specific extension to the ejb-ql language. You should specify queries that only use standard EJB-QL language features in the `ejb-jar.xml` deployment descriptor.

Example

See “weblogic-query” on page 11-39.

weblogic-query

Range of values:	n/a
Default value:	n/a
Requirements:	n/a
Parent elements:	`weblogic-rdbms-bean`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `weblogic-query` element allows you to associate WebLogic specific attributes with a query, as necessary. For example, `weblogic-query` can be used to specify a query that contains a WebLogic specific extension to EJB-QL. Queries that do not take advantage of WebLogic extensions to EJB-QL should be specified in the `ejb-jar.xml` deployment descriptor.

Also, the `weblogic-query` element is used to associate a `field-group` with the query if the query retrieves an entity bean that should be pre-loaded into the cache by the query.

Example

The `XML` stanza can contain the elements shown here:

```
<weblogic-query>
        <query-method>
          <method-name>findBigAccounts</method-name>
          <method-params>
                              <method-param>double</method-param>
```

```
            </method-params>
          <query-method>
                                  <weblogic-ql>WHERE BALANCE>10000
ORDERBY NAME</weblogic-ql>
               </weblogic-query>
```

weblogic-rdbms-relation

Parent elements: `weblogic-rdbms-jar`

Function

The `weblogic-rdbms-relation` element represents a single relationship that is managed by the WebLogic CMP persistence type. deployment descriptor. WebLogic Server supports the following three relationship mappings:

- For one-to-one relationships, the mapping is from a foreign key in one bean to the primary key of the other bean.
- For one-to-many relationships, the mapping is also from a foreign key in one bean to the primary key of another bean.
- For many-to-many relationships, the mapping involves a join table. Each row in the join table contains two foreign keys that map to he primary keys of the entities involved in the relationship.

For more information on container managed relationships, see “Container-Managed Relationships” on page 5-27.

Example

```
weblogic-rdbms-relation
   relationship-role-name
   group-name
```

```
column-map
db-cascade-delete
```

weblogic-relationship-role

Range of values:	Valid name
Default value:	n/a
Requirements:	The mapping of a role to a table is specified in the associated `weblogic-rdbms-bean` and `ejb-relation` elements.
Parent elements:	`weblogic-rdbms-relation`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `weblogic-relationship-role` element is used to express a mapping from a foreign key to a primary key. Only one mapping is specified for one-to-one relationships when the relationship is local. However, with a many-to-many relationship, you must specify two mappings

Multiple column mappings are specified for a single role, it the key is complex. No `column-map` is specified if the role is just specifying a group-name.

Example

The `XML` stanza can contain the elements shown here:

```
<weblogic-rdbms-jar>
   <weblogic-rdbms-relation>
      <relation-name>stocks-holders</relation-name>
      <table-name>stocks</table-name>
      <weblogic-relationship-role>stockholder
      </weblogic-relationship-role>
```

```
   </weblogic-rdbms-relation>
</weblogic-rdbms-jar>
```

5.1 weblogic-cmp-rdbms-jar.xml Deployment Descriptor File Structure

`weblogic-cmp-rdbms-jar.xml` defines deployment elements for a single entity EJB that uses WebLogic Server RDBMS-based persistence services. See "Locking Services for Entity EJBs" on page 4-37 for more information.

The top-level element of the WebLogic Server 5.1 `weblogic-cmp-rdbms-jar.xml` consists of a `weblogic-enterprise-bean` stanza:

```
description

weblogic-version

<weblogic-enterprise-bean>
   <pool-name>finance_pool</pool-name>
   <schema-name>FINANCE_APP</schema-name>
   <table-name>ACCOUNT</table-name>
   <attribute-map>
      <object-link>
         <bean-field>accountID</bean-field>
         <dbms-column>ACCOUNT_NUMBER</dbms-column>
      </object-link>
      <object-link>
         <bean-field>balance</bean-field>
         <dbms-column>BALANCE</dbms-column>
      </object-link>
   </attribute-map>
```

```
    <finder-list>
       <finder>
          <method-name>findBigAccounts</method-name>
          <method-params>
             <<method-param>double</method-param>
          </method-params>
       <finder-query><![CDATA[(> balance $0)]]></finder-query>
       <finder-expression>. . .</finder-expression>
       </finder>
    </finder-list>
</weblogic-enterprise-bean>
```

5.1 weblogic-cmp-rdbms-jar.xml Deployment Descriptor Elements

RDBMS Definition Elements

This section describes the RDBMS definition elements.

pool-name

`pool-name` specifies name of the WebLogic Server connection pool to use for this EJB's database connectivity. See Using connection pools for more information.

schema-name

`schema-name` specifies the schema where the source table is located in the database. This element is required only if you want to use a schema that is not the default schema for the user defined in the EJB's connection pool.

Note: This field is case sensitive, although many SQL implementations ignore case.

table-name

`table-name` specifies the source table in the database. This element is required in all cases.

Note: The user defined in the EJB's connection pool must have read and write privileges to the specified table, though not necessarily schema modification privileges. This field is case sensitive, although many SQL implementations ignore case.

EJB Field-Mapping Elements

This section describes the EJB field-mapping elements.

attribute-map

The `attribute-map` stanza links a single field in the EJB instance to a particular column in the database table. The `attribute-map` must have exactly one entry for each field of an EJB that uses WebLogic Server RDBMS-based persistence.

object-link

Each `attribute-map` entry consists of an `object-link` stanza, which represents a link between a column in the database and a field in the EJB instance.

bean-field

`bean-field` specifies the field in the EJB instance that should be populated from the database. This element is case sensitive and must precisely match the name of the field in the bean instance.

The field referenced in this tag must also have a `cmp-field` element defined in the `ejb-jar.xml` file for the bean.

dbms-column

`dbms-column` specifies the database column to which the EJB field is mapped. This tag is case sensitive, although many databases ignore the case.

Note: WebLogic Server does not support quoted RDBMS keywords as entries to `dbms-column`. For example, you cannot create an attribute map for column names such as “create” or “select” if those names are reserved in the underlying datastore.

Finder Elements

This section describes the finder elements.

finder-list

The `finder-list` stanza defines the set of all finders that are generated to locate sets of beans. See “Writing for RDBMS Persistence for EJB 1.1 CMP” on page 5-4 for more information.

`finder-list` must contain exactly one entry for each finder method defined in the home interface, except for `findByPrimarykey`. If an entry is not provided for `findByPrimaryKey`, one is generated at compilation time.

Note: If you do provide an entry for `findByPrimaryKey`, WebLogic Server uses that entry without validating it for correctness. In most cases, you should omit an entry for `findByPrimaryKey` and accept the default, generated method.

finder

The `finder` stanza describes a finder method defined in the home interface. The elements contained in the `finder` stanza enable WebLogic Server to identify which method in the home interface is being described, and to perform required database operations.

method-name

`method-name` defines the name of the finder method in the home interface. This tag must contain the exact name of the method.

method-params

The `method-params` stanza defines the list of parameters to the finder method being specified in method-name.

Note: WebLogic Server compares this list against the parameter types for the finder method in the EJB's home interface; the order and type for the parameter list must exactly match the order and type defined in the home interface.

method-param

`method-param` defines the fully-qualified name for the parameter's type. The type name is evaluated into a `java.lang.Class` object, and the resultant object must precisely match the respective parameter in the EJB's finder method.

You can specify primitive parameters using their primitive names (such as "double" or "int"). If you use a non-primitive data type in a `method-param` element, you must specify a fully qualified name. For example, use `java.sql.Timestamp` rather than `Timestamp`. If you do not use a qualified name, `ejbc` generates an error message when you compile the deployment unit.

finder-query

`finder-query` specifies the WebLogic Query Language (WLQL) string that is used to retrieve values from the database for this finder. See "Using WebLogic Query Language (WLQL) for EJB 1.1 CMP" on page 5-6 for more information.

Note: Always define the text of the `finder-query` value using the XML `CDATA` attribute. Using `CDATA` ensures that any special characters in the WLQL string do not cause errors when the finder is compiled.

finder-expression

`finder-expression` specifies a Java language expression to use as a variable in the database query for this finder.

Note: Future versions of the WebLogic Server EJB container will use the EJB QL query language (as required by the EJB 2.0 specification). EJB QL does not provide support for embedded Java expressions. Therefore, to ensure easier upgrades to future EJB containers, create entity EJB finders *without* embedding Java expressions in WLQL.