BEA WebLogic Server®

Programming WebLogic Enterprise JavaBeans

Release 8.1
Revised: April 28, 2006

Copyright

Copyright © 2004 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, Jolt, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Liquid Data for WebLogic, BEA Manager, BEA WebLogic Commerce Server, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic Personalization Server, BEA WebLogic Platform, BEA WebLogic Portal, BEA WebLogic Server, BEA WebLogic Workshop and How Business Becomes E-Business are trademarks of BEA Systems, Inc.

All other trademarks are the property of their respective companies.

Contents

About This Document

Audience xix
e-docs Web Site xix
How to Print the Document xix
Related Information xx
Contact Us! xx
Documentation Conventions xxi

1. Introduction and Roadmap

Document Scope and Audience 1-1
Guide to this Document 1-2
Related Documentation 1-3
Samples and Tutorials for the EJB Developer 1-3
Avitek Medical Records Application (MedRec) and Tutorials 1-3
EJB Examples in the WebLogic Server Distribution 1-4
Additional EJB Examples Available for Download 1-4
New and Changed EJB Features in This Release 1-4

2. Understanding Enterprise JavaBeans

How Do Applications Use EJBs? 2-1
Session EJBs Implement Business Logic 2-1
Stateless Session Beans 2-2
Stateful Session Beans 2-2

Entity EJBs Maintain Persistent Data 2-2
Message-Driven Beans Implement Loosely Coupled Business Logic 2-3
EJB Anatomy and Environment 2-3
EJB Components 2-4
The EJB Container 2-5
EJB Deployment Descriptors 2-5
Key Deployment Element Mappings 2-6
EJBs, Clients, and Application Objects 2-8
EJB Communications 2-10
WebLogic Server Value-Added EJB Features 2-10
Performance-Enhancing Features for WebLogic Server EJBs 2-10
Pooling Improves EJB Response Time 2-11
Caching Improves EJB Performance 2-11
Additional Caching Capabilities for CMP Entities 2-11
Field Groups for Efficient Queries (CMP Entities) 2-12
Configurable Write Behaviors 2-12
Operation Ordering and Batching (CMP Entities) 2-12
Optimized Database Updates (CMP Entities) 2-12
Read-Only Pattern and Read-Only Invalidation (CMP Entities) 2-13
CMP Beans Increase Developer Productivity 2-13
Automatic Primary Key Generation (CMP Entities) 2-13
Automatic Table Creation (CMP Entities) 2-13
Dynamic Queries (CMP Entities) 2-13
Reliability and Availability Features 2-13
Load Balancing Among Clustered EJBs Increases Scalability 2-14
Failover for Clustered EJBs Increases Reliability 2-14
Securing EJBs 2-16

3. Designing Enterprise Java Beans

Choosing the Right Bean Type 3-1
Session Bean Features 3-2
Stateful Session Beans. 3-2
Stateless Session Beans 3-2
Stateless Beans Offer Performance and Scalability Advantages 3-3
Entity Bean Features. 3-3
Key Features of Entity Beans 3-3
Read-Write versus Read-Only Entity Beans 3-4
Entity Bean Performance and Data Consistency Characteristics 3-5
Message-Driven Bean Features 3-6
Persistence Management Alternatives. 3-7
Use Container-Managed Persistence (CMP) for Productivity and Portability. 3-7
Use Bean-Managed Persistence (BMP) Only When Necessary 3-8
Transaction Design and Management Options 3-8
Understanding Transaction Demarcation Strategies and Performance 3-9
Demarcating Transactions at the Server Level is Most Efficient 3-9
Container-Managed Transactions Are Simpler to Develop and Perform Well . . 3-9
Bean-Level Transaction Management. 3-10
Client-Level Transaction Management is Costly 3-11
Transaction Isolation: A Performance vs. Data Consistency Choice. 3-11
Satisfying Application Requirements with WebLogic Server EJBs 3-12

4. Implementing Enterprise Java Beans

Overview of the EJB Development Process 4-2
Create a Source Directory 4-5
Create EJB Classes and Interfaces. 4-6
Using WebLogic Server Generic Bean Templates 4-6

Programming Client Access to EJBs 4-6
Programming Client to Obtain Initial Context 4-6
Programming Client to Look Up a Home Interface 4-7
Configuring EJBs to Send Requests to a URL 4-8
Specifying an HTTP Resource by URL 4-8
Specifying an HTTP Resource by Its JNDI Name 4-8
Accessing HTTP Resources from Bean Code 4-9
Programming Transaction 4-9
Programming Container-Managed Transactions 4-9
Programming Bean-Managed Transactions 4-10
Programming Transactions That Are Distributed Across EJBs 4-12
Compile Java Source 4-13
Generate Deployment Descriptors 4-13
Edit Deployment Descriptors 4-13
Security Elements 4-14
Resource Mapping Elements 4-15
Persistence Elements 4-16
Clustering Elements 4-17
Data Consistency Elements 4-19
Container-Managed Transactions Elements 4-19
Performance Elements 4-21
Generate EJB Wrapper Classes, and Stub and Skeleton Files 4-24
appc and Generated Class Name Collisions 4-25
Package 4-26
Packaging Considerations for EJBs with Clients in Other Applications 4-26
Deploy 4-26
Solving Problems During Development 4-27
Adding Line Numbers to Class Files 4-27

Monitoring Data 4-27
Creating Debug Messages 4-27
WebLogic Server Tools for Developing EJBs 4-27
WebLogic Workshop 4-28
Administration Console 4-28
javac 4-29
EJBGen 4-29
WebLogic Builder 4-30
DDInit 4-30
WebLogic Server Ant Utilities 4-30
weblogic.Deployer 4-31
appc 4-31
DDConverter 4-31
Comparison of EJB Tool Features 4-31

5. Session EJBs

Comparing Stateless and Stateful Session Beans 5-1
Pooling for Stateless Session EJBs 5-2
Caching and Passivating Stateful Session EJBs 5-4
Stateful Session EJB Creation 5-5
Stateful Session EJB Passivation 5-6
Controlling Passivation 5-6
Managing EJB Cache Size 5-7
Specifying the Persistent Store Directory for Passivated Beans 5-7
Configuring Concurrent Access to Stateful Session Beans 5-8
Design Decisions for Session Beans 5-8
Choosing Between Stateless and Stateful Beans 5-8
Choosing the Optimal Free Pool Setting for Stateless Session Beans 5-9

Implementing Session Beans 5-9
WebLogic-Specific Configurable Behaviors for Session Beans 5-9

6. Entity EJBs

Managing Entity Bean Pooling and Caching 6-1
Understanding Entity Pooling 6-3
Understanding Entity Caching 6-3
Understanding ejbLoad() and ejbStore() Behavior 6-4
Controlling the Behavior of ejbLoad() and ejbStore() 6-5
Disabling Cache Flushing 6-5
Configuring Application-Level Caching 6-6
Using Primary Keys 6-7
Specifying Primary Keys and Primary Key Classes 6-7
Guidelines for Primary Keys 6-8
Automatically Generating Primary Keys 6-9
Specifying Automatic Key Generation for Oracle 6-10
Generating Primary Keys with a Named Sequence Table 6-11
Declaring Primary Key Field Type 6-12
Support for Oracle SEQUENCE 6-12
Configuring Entity EJBs for Database Operations 6-12
Configuring Table Mapping 6-12
Automatic Table Creation (Development Only) 6-14
Delaying Database Inserts 6-16
Why Delay Database Inserts? 6-16
Configuring Delayed Database Inserts 6-17
Limiting Database Reads with cache-between-transactions (Long-Term Caching) 6-17
Updating the Database Before Transaction Ends 6-18
Dynamic Queries 6-18

Enabling Dynamic Queries 6-19
Executing Dynamic Queries 6-19
Enabling BLOB and CLOB Column Support for Oracle 6-19
Specifying a BLOB Column Using the Deployment Descriptor 6-20
Controlling Serialization for cmp-fields of Type byte[] Mapped to a Oracle Blob . 6-20
Specifying a CLOB Column Using the Deployment Descriptor 6-20
Optimized CLOB Column Insertion on Oracle 10g 6-20
Specifying Field Groups 6-21
Ordering and Batching Operations 6-22
Operation Ordering 6-22
Batch Operations Guidelines and Limitations 6-23
Using Container-Managed Relationships (CMRs) 6-23
CMR Requirements and Limitations 6-24
CMR Cardinality 6-24
CMR Direction 6-24
Removing CMRs 6-25
Defining Container-Managed Relationships (CMRs) 6-25
Specifying Relationship in ejb-jar.xml 6-25
Specifying Relationships in weblogic-cmp-jar.xml 6-28
About CMR Fields and CMR Field Accessor Methods 6-32
Using Cascade Delete for Entities in CMRs 6-32
Relationship Caching 6-33
Enabling Relationship Caching 6-34
Choosing a Concurrency Strategy 6-35
Exclusive Concurrency 6-36
Database Concurrency 6-36
Optimistic Concurrency 6-36

Check Data for Validity with Optimistic Concurrency 6-37
Optimistic Concurrency and Oracle Databases 6-38
Read Only Concurrency 6-39
Concurrency Strategy Trade-Offs 6-39
Configuring Concurrency Strategy 6-41
Preventing Deadlocks for Transactions That Use Exclusive Concurrency and Cascade Deletes 6-41
Using the Read-Mostly Pattern 6-42
Configuring Entity Beans for Read-Mostly Pattern 6-42
Invalidating Read-Only Entity EJBs Implicitly 6-43
CMP Entity Bean Descriptors Element by Feature 6-44
Container-Managed Relationship Elements 6-45
Primary Key Elements 6-46

7. Message-Driven EJBs

Message-Driven EJB Lifecycle and the Free Pool 7-1
MDBs and the Free Pool 7-2
MDBs and Concurrent Processing 7-3
MDBs and Messaging Models 7-3
Point-to-Point (Queue) Model: One Message Per Listener 7-4
Publish/Subscribe (Topic) Model 7-4
Exactly-Once Processing 7-5
MDB Deployment Options 7-6
Destination and MDBs: Collocated vs. Not Collocated 7-6
Collocated Destination/MDBs 7-6
Non-Collocated Destination/MDBs 7-7
JMS Distributed Destinations 7-9
Programming and Configuring MDBs: Main Steps 7-9

Required JMS Configuration 7-9
Create MDB Class and Configure Deployment Elements 7-10
Programming and Configuring MDBs: Details 7-12
Configuring Destination Type 7-13
Configuring Transaction Management Strategy for an MDB 7-13
Configuring MDBs for Destinations 7-14
Whether to Use Wrappers 7-15
How to Set provider-url 7-15
How to Set initial-context-factory 7-16
How to Set destination-jndi-name 7-16
How to Set connection-factory-jndi-name 7-16
Common Destination Scenarios: Illustrations and Key Element Settings 7-17
Configuring Durable Topic Subscriptions 7-21
Configuring a Durable Topic Subscription for a Non-Clustered Server 7-21
Configuring a Durable Topic Subscription for a Cluster 7-21
Configuring Message Handling Behaviors 7-22
Ensuring Message Receipt Order 7-22
Preventing and Handling Duplicate Messages 7-23
Redelivery and Exception Handling 7-24
Using the Message-Driven Bean Context 7-25
Deferring Message Processing Until Completion of Boot Up 7-25
Configuring a Security Identity for a Message-Driven Bean 7-25
Migration and Recovery for Clustered MDBs 7-27
Nutshell Summary: Deployment Elements for MDBs 7-28

8. Deployment Guidelines for Enterprise Java Beans

Before You Deploy an EJB 8-1
Understanding and Performing Deployment Tasks 8-2

Deployment Guidelines for EJBs 8-2
Deploy EJBs as Part of an Enterprise Application 8-2
Deploy EJBs That Call Each Other in the Same Application 8-3
Deploy Homogeneously to a Cluster 8-3
Deploying Pinned EJBs to a Cluster 8-3
Redeploying an EJB 8-4
Redeploying an Individual EJB Implementation Class 8-4
Understanding Warning Messages 8-5
Disabling EJB Deployment Warning Messages 8-5

A. weblogic-ejb-jar.xml Deployment Descriptor Reference

Document Type Definitions and DOCTYPE Header Information A-1
2.0 weblogic-ejb-jar.xml File Structure A-4
Changes to weblogic-ejb-jar.xml in WebLogic Server 8.1 . . . . . A-4
2.0 weblogic-ejb-jar.xml Elements A-5

B. weblogic-cmp-jar.xml Deployment Descriptor Reference

2.0 weblogic-cmp-jar.xml Deployment Descriptor File Structure B-1
Changes to weblogic-cmp-jar.xml in WebLogic Server 8.1 . . . . . B-2
2.0 weblogic-cmp-jar.xml Deployment Descriptor Elements B-3
Setting up Oracle for Cascade Delete B-16
Function B-28
Function B-29
Mapping a Bean on Foreign Key Side of a Relationship to Multiple Tables . . B-38
Mapping a Bean on Primary Key Side of a Relationship to Multiple Tables . . B-39

C. appc and ejbc Reference

appc C-1
Advantages of Using appc C-1

appc Syntax C-2
Designating Alternative Deployment Descriptors C-2
appc Options C-3
appc and EJBs C-4
ejbc C-5
Advantages of Using ejbc C-6
ejbc Syntax C-6
ejbc Arguments C-7
ejbc Options C-7

D. EJBGen Reference

Running EJBGen D-1
EJBGen Distribution and Examples D-2
Invoking EJBGen D-2
EJBGen Command Options D-2
Javadoc Warnings with Uncompiled Value Type Classes D-7
Using Key EJBGen Features D-8
Controlling the Files EJBGen Generates D-8
Using Property Files with EJBGen D-9
Using User Variables D-9
Using Predefined Variables D-10
Surround Attributes that Contain Spaces With Double Quotes D-11
EJBGen Supports Tag Inheritance D-11
EJBGen Supports Attribute Inheritance D-12
Examples of EJBGen Tag Usage D-13
A Sample Bean File Annotated with EJBGen Tags D-13
Mapping an Entity Bean to Several Tables with EJBGen D-15
Specifying Relationship Caching with EJBGen Tags D-15

Specifying Relationships with EJBGen Tags D-16
Upgrading Relationships to Use Local Interfaces D-16
EJBGen Tag Reference D-17

E. EJB Query Language (EJB-QL) and WebLogic Server

EJB QL Requirement for EJB 2.0 Beans E-1
Using the EJB 2.0 WebLogic QL Extension for EJB QL E-1
upper and lower Functions E-2
upper E-2
lower E-2
Using ORDERBY E-2
Using Subqueries E-3
Subquery Return Types E-4
Subqueries as Comparison Operands E-5
Correlated and Uncorrelated Subqueries E-7
Using Aggregate Functions E-9
Using Queries that Return ResultSets E-10
Using Oracle SELECT HINTS E-12
Properties-Based Methods of the Query Interface E-13
Migrating from WLQL to EJB QL E-14
Known Issue with Implied Cross Products E-14
EJB QL Error-Reporting E-15
Visual Indicator of Error in Query E-15
Multiple Errors Reported after a Single Compilation E-15

F. Important Information for EJB 1.1 Users

Writing for RDBMS Persistence for EJB 1.1 CMP F-1
Finder Signature F-2

finder-list Stanza F-2
finder-query Element F-3
Using WebLogic Query Language (WLQL) for EJB 1.1 CMP F-3
WLQL Syntax F-3
WLQL Operators F-4
WLQL Operands F-5
Examples of WLQL Expressions F-5
Using SQL for CMP 1.1 Finder Queries F-7
Tuned EJB 1.1 CMP Updates in WebLogic Server F-7
Using is-modified-method-name to Limit Calls to ejbStore() F-8
5.1 weblogic-ejb-jar.xml Deployment Descriptor File Structure F-9
5.1 weblogic-ejb-jar.xml Deployment Descriptor Elements F-9
caching-descriptor F-9
persistence-descriptor F-11
clustering-descriptor F-14
transaction-descriptor F-16
reference-descriptor F-16
transaction-isolation F-18
security-role-assignment F-20
1.1 weblogic-cmp-jar.xml Deployment Descriptor File Structure F-20
1.1 weblogic-cmp-jar.xml Deployment Descriptor Elements F-22
RDBMS Definition Elements F-22
EJB Field-Mapping Elements F-22
Finder Elements F-23

About This Document

This document describes how to design, implement and deploy Enterprise JavaBeans (EJBs) on WebLogic Server®.

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 resources may help you when using Enterprise JavaBeans with WebLogic Server.

- For more information about Sun Microsystem's Enterprise JavaBeans (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 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.

`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

Convention	Item
`{ }`	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.
`...`	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

Introduction and Roadmap

This section describes the contents and organization of this guide—*Programming WebLogic Enterprise Java Beans*.

- "Document Scope and Audience" on page 1-1
- "Guide to this Document" on page 1-2
- "Related Documentation" on page 1-3
- "Samples and Tutorials for the EJB Developer" on page 1-3
- "New and Changed EJB Features in This Release" on page 1-4

Document Scope and Audience

This document is a resource for software developers who develop applications that include WebLogic Server Enterprise Java Beans (EJBs). It also contains information that is useful for business analysts and system architects who are evaluating WebLogic Server or considering the use of WebLogic Server EJBs for a particular application.

The topics in this document are relevant during the design and development phases of a software project. The document also includes topics that are useful in solving application problems that are discovered during test and pre-production phases of a project.

This document does not address production phase administration, monitoring, or performance tuning. For links to WebLogic Server documentation and resources for these topics, see "Related Documentation" on page 1-3.

It is assumed that the reader is familiar with J2EE and EJB concepts. This document emphasizes the value-added features provided by WebLogic Server EJBs and key information about how to use WebLogic Server features and facilities to get an EJB application up and running.

Guide to this Document

- This chapter, Chapter 1, "Introduction and Roadmap," introduces the organization of this guide.
- Chapter 2, "Understanding Enterprise JavaBeans," reviews the different bean types, their capabilities, and how they work with other application objects and WebLogic Server.
- Chapter 3, "Designing Enterprise Java Beans," discusses design options for WebLogic Server Enterprise Java Beans (EJBs), factors to consider during the design process, and recommended design patterns.
- Chapter 4, "Implementing Enterprise Java Beans," describes the EJB implementation process, and provide guidance for how to get an EJB up and running in WebLogic Server.
- Chapter 5, "Session EJBs," describes how session beans work within the EJB container, and provides design and development guidelines that are specific to session beans.
- Chapter 6, "Entity EJBs," describes how entity beans work within the EJB container, and provides design and development guidelines that are specific to entity beans.
- Chapter 7, "Message-Driven EJBs," describes how message-driven beans work within the EJB container, and provides design and development guidelines that are specific to message-driven beans.
- Chapter 8, "Deployment Guidelines for Enterprise Java Beans," discusses EJB-specific deployment issues and procedures.
- Appendix A, "weblogic-ejb-jar.xml Deployment Descriptor Reference" describes the EJB 2.0 deployment descriptor elements in `weblogic-ejb-jar.xml` file, the WebLogic Server-specific deployment descriptor.
- Appendix B, "weblogic-cmp-jar.xml Deployment Descriptor Reference" describes the EJB 2.0 elements in `weblogic-cmp-jar.xml` file, the WebLogic Server-specific deployment descriptor for container-managed persistence features.
- Appendix C, "appc and ejbc Reference" is a complete reference for the WebLogic Server `appc` and `ejbc` tools for compiling, validating and generating EJB code.

- Appendix D, "EJBGen Reference" is a reference for the WebLogic Server EJBGen EJB 2.0 code generator.
- Appendix E, "EJB Query Language (EJB-QL) and WebLogic Server" discusses issues pertinent to using EJB QL with WebLogic Server.
- Appendix F, "Important Information for EJB 1.1 Users" contains important design and implementation information specific to EJB 1.1.

Related Documentation

This document contains EJB-specific design and development information.

For comprehensive guidelines for developing, deploying, and monitoring WebLogic Server applications, see the following documents:

- *Developing WebLogic Server Applications* is a guide to developing WebLogic Server applications.
- *Deploying WebLogic Server Applications* is the primary source of information about deploying WebLogic Server applications in development and production environments.
- *WebLogic Server Performance and Tuning* contains information on monitoring and improving the performance of WebLogic Server applications.

Samples and Tutorials for the EJB Developer

In addition to this document, BEA Systems provides a variety of code samples and tutorials for EJB developers. The examples and tutorials illustrate WebLogic Server EJBs in action, and provide practical instructions on how to perform key EJB development tasks.

BEA recommends that you run some or all of the EJB examples before developing your own EJBs.

Avitek Medical Records Application (MedRec) and Tutorials

MedRec is an end-to-end sample J2EE application shipped with WebLogic Server that simulates an independent, centralized medical record management system. The MedRec application provides a framework for patients, doctors, and administrators to manage patient data using a variety of different clients.

MedRec demonstrates WebLogic Server and J2EE features, and highlights BEA-recommended best practices. MedRec is included in the WebLogic Server distribution, and can be accessed

from the Start menu on Windows machines. For Linux and other platforms, you can start MedRec from the `WL_HOME\samples\domains\medrec` directory, where `WL_HOME` is the top-level installation directory for WebLogic Platform.

MedRec includes a service tier comprised primarily of EJBs that work together to process requests from web applications, web services, and workflow applications, and future client applications. The application includes message-driven, stateless session, stateful session, and entity EJBs.

As companion documentation to the MedRec application, BEA provides tutorials that provide step-by-step procedures for key development tasks, including EJB-specific tasks, such as:

- Using EJBGen to Generate EJB deployment descriptors
- Exposing a stateless Session EJB as a web service
- Securing EJB resources using the Administration Console

EJB Examples in the WebLogic Server Distribution

WebLogic Server 8.1 optionally installs API code examples in `WL_HOME\samples\server\examples\src\examples`, where `WL_HOME` is the top-level directory of your WebLogic Server installation. You can start the examples server, and obtain information about the samples and how to run them from the WebLogic Server 8.1 Start menu.

Additional EJB Examples Available for Download

Additional API examples for download at http://dev2dev.bea.com/code/certwls81.jsp. These examples are distributed as ZIP files that you can unzip into an existing WebLogic Server samples directory structure.

You build and run the downloadable examples in the same manner as you would an installed WebLogic Server example. See the download pages of individual examples for more information. at http://dev2dev.bea.com/code/certwls81.jsp.

New and Changed EJB Features in This Release

For information about new and changed features in WebLogic Server 8.1 see:

- "Changes to weblogic-ejb-jar.xml in WebLogic Server 8.1" on page A-4
- "Changes to weblogic-cmp-jar.xml in WebLogic Server 8.1" on page B-2

- These sections in *WebLogic Server 8.1 Release Notes*:
  - "WebLogic Server 8.1 Features and Changes" lists new, changed, and deprecated features.
  - "WebLogic Server 8.1 Known Issues" lists known problems by service pack, for all WebLogic Server APIs, including EJB.
  - "Resolved Problems for Service Pack 1"
  - "Resolved Problems for Service Pack 2"
  - "Resolved Problems for Service Pack 3"

CHAPTER 2

Understanding Enterprise JavaBeans

These sections briefly review the different Enterprise JavaBean (EJB) types and the functions they can serve in an application, and describe how they work with other application objects and WebLogic Server.

It is assumed the reader is familiar with Java programming and EJB 2.0 concepts and features.

- "How Do Applications Use EJBs?" on page 2-1
- "EJB Anatomy and Environment" on page 2-3
- "EJBs, Clients, and Application Objects" on page 2-8
- "Securing EJBs" on page 2-16
- "WebLogic Server Value-Added EJB Features" on page 2-10

How Do Applications Use EJBs?

These sections describe the purpose and capabilities of each bean type.

Session EJBs Implement Business Logic

Session beans implement business logic. A session bean instance serves one client at a time. There are two types of session beans: stateful and stateless.

Stateless Session Beans

A stateless session bean does not store session or client state information between invocations—the only state it might contain is not specific to a client, for instance, a cached database connection or a reference to another EJB. At most, a stateless session bean may store state for the duration of a method invocation. When a method completes, state information is not retained. Any instance of a stateless session bean can serve any client—any instance is equivalent. Stateless session beans can provide better performance than stateful session beans, because each stateless session bean instance can support multiple clients, albeit one at a time.

Example: An Internet application that allows visitors to click a "Contact Us" link and send an email could use a stateless session bean to generate the email, based on the to and from information gathered from the user by a JSP.

Stateful Session Beans

Stateful session beans maintain state information that reflects the interaction between the bean and a particular client across methods and transactions. A stateful session bean can manage interactions between a client and other enterprise beans, or manage a workflow.

Example: A company Web site that allows employees to view and update personal profile information could use a stateful session bean to call a variety of other beans to provide the services required by a user, after the user clicks "View my Data" on a page:

- Accept the login data from a JSP, and call another EJB whose job it is to validate the login data.
- Send confirmation of authorization to the JSP.
- Call a bean that accesses profile information for the authorized user.

Entity EJBs Maintain Persistent Data

An entity bean represents a set of persistent data, usually rows in a database, and provides methods for maintaining or reading that data. An entity bean is uniquely identified by a primary key, and can provide services to multiple clients simultaneously. Entity beans can participate in relationships with other entity beans. The relationships between entity beans are a function of the real-world entities that the entity beans model. An entity bean's fields and its relationships to other entity beans are defined in an object schema, which is specified in the bean's `ejb-jar.xml` deployment descriptor.

An entity bean can have other bean types, such as message-driven or session beans, as clients, or be directly accessed by Web components. The client uses the entity bean to access data in a persistent store. An entity bean encapsulates the mechanics of database access, isolating that complexity from its clients and de-coupling physical database details from business logic.

Example: The stateful session bean in the previous example, which orchestrates services for an employee accessing personal profile information on a company intranet, could use an entity bean for getting and updating the employee's profile.

Message-Driven Beans Implement Loosely Coupled Business Logic

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

Throughout its lifecycle, an MDB instance can process messages from multiple clients, although not simultaneously. It does not retain state for a specific client. All instances of a message-driven bean are equivalent—the EJB container can assign a message to any MDB instance. The container can pool these instances to allow streams of messages to be processed concurrently. It is acceptable or beneficial for customer orders to "stack up" before the associated supplier orders are issued.

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

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

EJB Anatomy and Environment

These sections briefly describe classes required for each bean type, the EJB run-time environment, and the deployment descriptor files that govern a bean's run-time behavior.

- "EJB Components" on page 2-4

- "The EJB Container" on page 2-5
- "EJB Deployment Descriptors" on page 2-5

EJB Components

The composition of a bean varies by bean type. Table 2-1 defines the classes that make up each type of EJB, and defines the purpose of the class type.

Table 2-1 Components of an EJB

EJB Component	Description	Stateless Session	Stateful Session	Entity	MDB
Remote interface	The remote interface exposes business logic to remote clients—clients running in a separate application from the EJB. It defines the business methods a remote client can call.	Yes	Yes	Yes	No
Local interface	The local interface exposes business logic to local clients—those running in the same application as the EJB. It defines the business methods a local client can call. **Note:** Not available for 1.1 EJBs.	Yes	Yes	Yes	No
Local home interface	The local home interface, also referred to as an EJB factory or life-cycle interface, provides methods that local clients—those running in the same application as the EJB—can use to create, remove, and in the case of an entity bean, find instances of the bean. The local home interface also has "home methods"—business logic that is not specific to a particular bean instance.	Yes	Yes	Yes	No
Remote home interface	The remote home interface, also referred to as an EJB factory, or life-cycle interface, provides methods that remote clients—those running in a separate application from the EJB—can use to create, remove, and find instances of the bean.	Yes	Yes	Yes	No

EJB Component	Description	Stateless Session	Stateful Session	Entity	MDB
Bean class	The bean class implements business logic.	Yes	Yes	Yes	Yes
Primary key class	Only entity beans have a primary key class. The primary key class maps to one or more fields in a database—identifying the persistent data to which the entity bean corresponds.	No	No	Yes	No

The EJB Container

An EJB container is a run-time container for beans that are deployed to an application server. The container is automatically created when the application server starts up, and serves as an interface between a bean and run-time services such as:

- Life-cycle management
- Code generation
- Persistence management
- Security
- Transaction management
- Locking and concurrency control

EJB Deployment Descriptors

The structure of a bean and its run-time behavior are defined in one or more XML deployment descriptor files. Programmers create deployment descriptors during the EJB packaging process, and the descriptors become a part of the EJB deployment when the bean is compiled.

WebLogic Server EJBs have three deployment descriptors:

- `ejb-jar.xml`—The standard J2EE deployment descriptor. All beans must be specified in an `ejb-jar.xml`. An `ejb-jar.xml` can specify multiple beans that will be deployed together.
- `weblogic-ejb-jar.xml`—WebLogic Server-specific deployment descriptor that contains elements related to WebLogic Server features such as clustering, caching, and transactions. This file is required if your beans take advantage of WebLogic Server-specific features.

Like `ejb-jar.xml`, `weblogic-ejb-jar.xml` can specify multiple beans that will be deployed together. For details, see Appendix A, "weblogic-ejb-jar.xml Deployment Descriptor Reference."

- `weblogic-cmp-jar.xml`—WebLogic Server-specific deployment descriptor that contains elements related to container-managed persistence for entity beans. Entity beans that use container-managed persistence must be specified in a `weblogic-cmp-jar.xml` file. For details, see Appendix B, "weblogic-cmp-jar.xml Deployment Descriptor Reference."

Key Deployment Element Mappings

As described in "EJB Deployment Descriptors" on page 2-5, a WebLogic Server EJB's runtime behavior can be controlled by elements in three different descriptor files: `ejb-jar.xml`, `weblogic-ejb-jar.xml`, and `weblogic-cmp-jar.xml`.

Table 2-2 lists the elements whose values should match in each descriptor file. The elements listed in the table are defined in "Bean and Resource References" on page 2-7 and "Security Roles" on page 2-8.

Table 2-2 Element Mapping Among Descriptor Files

Map this element	in this stanza of ejb-jar.xml	to the same element in this stanza of weblogic-ejb-jar.xml	and to this stanza in weblogic-cmp-jar.xml
`role-name`	`security-role`	`security-role-assignment`	N/A
`ejb-name`	`message-driven`, `entity`, or `session`	`weblogic-enterprise-bean`	`weblogic-rdbms-bean`

Map this element	in this stanza of ejb-jar.xml	to the same element in this stanza of weblogic-ejb-jar.xml	and to this stanza in weblogic-cmp-jar.xml
ejb-ref name	assembly-descriptor	ejb-reference-description if the referenced bean runs in a different container than the current bean. or ejb-local-reference-description if the referenced bean runs in the same container as the current bean.	N/A
res-ref-name	resource-ref	resource-description	N/A

Bean and Resource References

Each descriptor file contains elements that identify a bean, and the runtime factory resources it uses:

- ejb-name—the name used to identifier a bean in each deployment descriptor file, independent of name that application code uses to refer to the bean.
- ejb-ref-name—the name by which a bean in another JAR is referred to in the referencing bean's code.
- res-ref-name—the name by which a resource factory is referred to in the referencing bean's code

A given bean or resource factory is identified by the same value in each descriptor file that contains it. Table 2-2 lists the bean and resource references elements, and their location in each descriptor file.

For instance, for a container-managed persistence entity bean named LineItem, this line:

```
<ejb-name>LineItem</ejb-name>
```

would appear in the:

- assembly-descriptor stanza of ejb-jar.xml
- enterprise-bean stanza of weblogic-ejb-jar.xml
- weblogic-rdbms-bean stanza of weblogic-cmp-jar.xml

Security Roles

Security roles are defined in the `role-name` element `ejb-jar.xml` and `weblogic-ejb-jar.xml`.

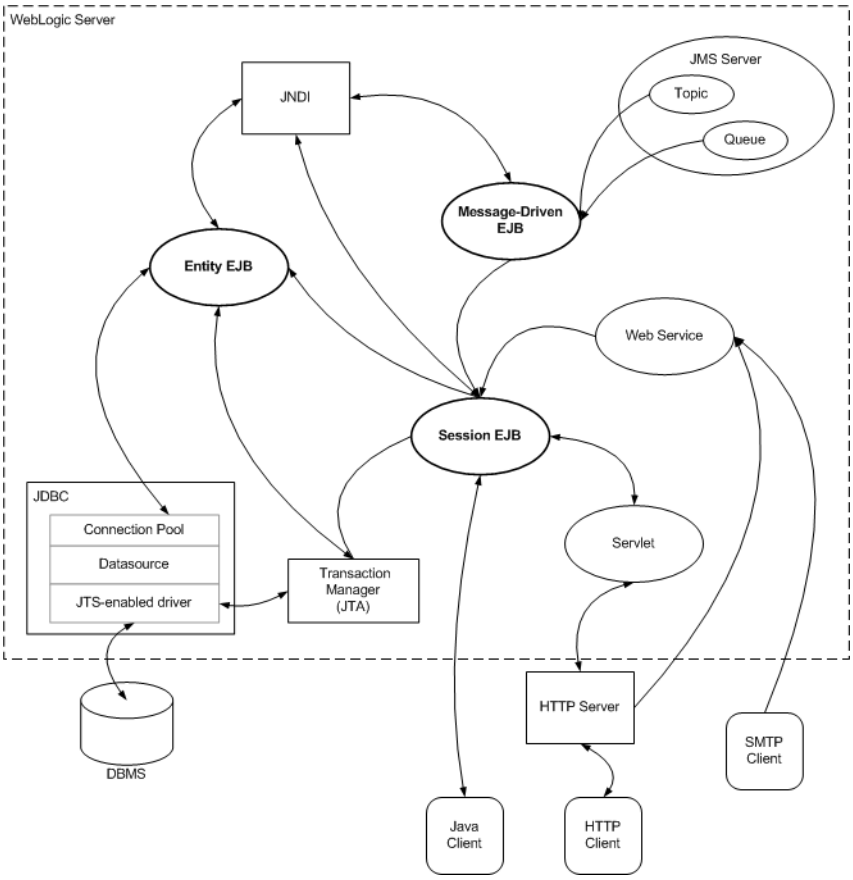
For information on:

- Programming security features for an EJB, see "Securing Enterprise JavaBeans (EJBs)" in *Programming WebLogic Security.*
- Creating or generating deployment descriptor files, see "Generate Deployment Descriptors" on page 4-13.
- Editing deployment descriptor files, see "Edit Deployment Descriptors" on page 4-13.
- Elements in `ejb-jar.xml`, see http://www.java.sun.com/dtd/ejb-jar_2_0.dtd.
- Elements in `weblogic-ejb-jar.xml`, see Appendix A, "weblogic-ejb-jar.xml Deployment Descriptor Reference".
- Elements in `weblogic-cmp-jar.xml`, see Appendix B, "weblogic-cmp-jar.xml Deployment Descriptor Reference".

EJBs, Clients, and Application Objects

Figure 2-1 illustrates how EJBs typically relate to other components of a WebLogic Server application and to clients.

Figure 2-1 EJBs and Other Application Components



An EJB can be accessed by server-side or client-side objects such as servlets, Java client applications, other EJBs, applets, and non-Java clients.

Any client of an EJB, whether in the same or a different application, accesses it in a similar fashion. WebLogic Server automatically creates implementations of an EJB's home and business interfaces that can function remotely, unless the bean has only a local interface.

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

Most beans do not require a global JNDI name—specified in the `jndi-name` and `local-jndi-name` elements of `weblogic-ejb-jar.xml`. Most beans reference to each other using `ejb-links`, as described in "Using EJB Links" on page 4-7.

Because of network overhead, it is more efficient to access beans from a client on the same machine than from a remote client, and even more efficient if the client is in the same application.

See "Programming Client Access to EJBs" on page 4-6 for information on programming client access to an EJB.

EJB Communications

WebLogic Server EJBs use:

- T3—To communicate with local objects. T3 is a WebLogic-proprietary remote network protocol that implements the Remote Method Invocation (RMI) protocol.
- RMI—To communicate with remote objects. RMI enables an application to obtain a reference to an object located elsewhere in the network, and to invoke methods on that object as though it were co-located with the client on the same JVM locally in the client's virtual machine.

  An EJB with a remote interface is an RMI object. An EJB's remote interface extends `java.rmi.remote`. For more information on WebLogic RMI, see *Programming WebLogic RMI*.
- HTTP—An EJB can obtain an HTTP connection to a Web server external to the WebLogic Server environment by using the `java.net.URL` resource connection factory. For more information, see "Configuring EJBs to Send Requests to a URL" on page 4-8.

WebLogic Server Value-Added EJB Features

This section describes key WebLogic Server features that ease the process of EJB development, and enhance the performance, reliability, and availability of EJB applications.

Performance-Enhancing Features for WebLogic Server EJBs

WebLogic Server supports pooling, caching, and other features that improve the response time and performance of EJBs. In a production environment, these features can reduce the time it takes for a client to obtain an EJB instance and access and maintain persistent data.

Pooling Improves EJB Response Time

Weblogic Server maintains a *free pool* of ready-to-use EJB instances for stateless session beans, message-driven beans and entity beans. The EJB container creates a configurable number of bean instances at startup, so that a new instance does not have to be created for every request. When a client is done with an EJB instance, the container returns it to the pool for reuse. For more information see:

- "Understanding Entity Pooling" on page 6-3
- "Message-Driven EJB Lifecycle and the Free Pool" on page 7-1
- "Pooling for Stateless Session EJBs" on page 5-2

Caching Improves EJB Performance

Weblogic Server supports caching for stateful session beans and entity beans.

An inactive cached bean instance can be passivated—removed from the cache and written to disk—and restored to memory later as necessary. Passivating bean instances optimizes use of system resources.

You can configure the size of the cache, and related behaviors such as rules for removing beans from the cache. Caching is supported for entity EJBs, whether they use container-managed or bean-managed persistence.

For more information, see:

- "Understanding Entity Caching" on page 6-3
- "Caching and Passivating Stateful Session EJBs" on page 5-4

Additional caching features are available for EJBs that use container-managed persistence, as described in the following section.

Additional Caching Capabilities for CMP Entities

WebLogic Server provides these caching capabilities for entity beans that use container managed persistence:

- Relationship caching—Relationship caching improves the performance of entity beans by loading related beans into the cache and avoiding multiple queries by issuing a join query for the related bean. For more information, see "Relationship Caching" on page 6-33.

- Application-level caching—Application-level caching, also known as “combined caching,” allows multiple entity beans that are part of the same J2EE application to share a single runtime cache. For more information, see “Configuring Application-Level Caching” on page 6-6.
- Caching between transactions—Use caching between transactions or long term caching to enable the EJB container to cache an entity bean's persistent data between transactions. For more information, see “Limiting Database Reads with cache-between-transactions (Long-Term Caching)” on page 6-17.

Field Groups for Efficient Queries (CMP Entities)

A group specifies a set of persistent attributes of an entity bean. A field-group represents a subset of the container-managed persistence (CMP) and container-managed relation (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. For more information, see “Specifying Field Groups” on page 6-21.

Configurable Write Behaviors

You can configure the behavior of the `ejbLoad()` and `ejbStore()` methods to enhance performance, by avoiding unnecessary calls to `ejbStore()`. As appropriate, you can ensure that WebLogic Server calls `ejbStore()` after each method call, rather than at the conclusion of the transaction. For more information, see “Understanding ejbLoad() and ejbStore() Behavior” on page 6-4.

Operation Ordering and Batching (CMP Entities)

WebLogic Server allows you to batch and order multiple database operations so that they can be completed in a single database “round-trip”. This allows you to avoid the bottlenecks that might otherwise occur when multiple entity instances are affected by a single transaction. For more information, see “Ordering and Batching Operations” on page 6-22.

Optimized Database Updates (CMP Entities)

In WebLogic Server 8.1 SP01 and later, for CMP 2.0 entity beans, the `setXXX()` method does not write the values of unchanged primitive and immutable fields to the database. This optimization improves performance, especially in applications with a high volume of database updates.

Read-Only Pattern and Read-Only Invalidation (CMP Entities)

For 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. If you use the read-mostly pattern, you can use multicast invalidation to maintain data consistency—an efficient mechanism for invalidating read-only cached data after updates. Use of the `invalidate()` method after the transaction update has completed invalidates the local cache and results in a multicast message is sent to the other servers in the cluster to invalidate their cached copies. For more information, see "Using the Read-Mostly Pattern" on page 6-42.

CMP Beans Increase Developer Productivity

WebLogic Server provides a variety of value-added database access features for entity beans that use container-managed persistence:

Automatic Primary Key Generation (CMP Entities)

WebLogic Server supports multiple methods to automatically generate simple primary key for CMP entity EJBs, including use of an Oracle `SEQUENCE` (which can be automatically generated by WebLogic Server.) For more information, see "Automatically Generating Primary Keys" on page 6-9.

Automatic Table Creation (CMP Entities)

You can configure the EJB container to automatically change the underlying table schema as entity beans change, ensuring that tables always reflect the most recent object-relationship mapping. For more information, see "Automatic Table Creation (Development Only)" on page 6-14.

Dynamic Queries (CMP Entities)

WebLogic Server allow you to construct and execute EJB-QL queries programmatically in your application code. This allows you to create and execute new queries without having to update and deploy an EJB. It also reduces the size and complexity of the EJB deployment descriptors. For more information, see "Choosing a Concurrency Strategy" on page 6-35.

Reliability and Availability Features

WebLogic Server EJBs can be deployed to a cluster, allowing support for load balancing, and failover for remote clients of an EJB. EJBs must be deployed to all clustered servers.

A WebLogic Server cluster consists of multiple WebLogic Server server instances running simultaneously and working together to provide increased scalability and reliability. A cluster appears to clients as a single WebLogic Server instance. The server instances that constitute a cluster can run on the same machine, or be located on different machines.

Failover and load balancing for EJBs are handled by replica-aware stubs, which can locate instances of the object throughout the cluster. Replica-aware stubs are created for EJBs as a result of the object compilation process. EJBs can have two different replica-aware stubs: one for the `EJBHome` interface and one for the `EJBObject` interface. This allows some bean types to take advantage of load balancing and failover features at the home level when a client looks up an EJB object using the `EJBHome` stub and at the method level when a client makes method calls against the EJB using the EJBObject stub. Table 2-3, "Failover and Load Balancing for Clustered EJBs," on page 2-15 summarizes the load balancing and failover support (method level and home level) for each EJB type.

Load Balancing Among Clustered EJBs Increases Scalability

The bean stub contains the load balancing algorithm (or the call routing class) used to load balance method calls to the object. On each call, the stub can employ its load algorithm to choose which replica to call.

WebLogic Server clusters support multiple algorithms for load balancing clustered EJBs; the default is the round-robin method. For more information, see "Load Balancing in a Cluster" in *Using WebLogic Server Clusters*.

All bean types support load balancing at the home level. All bean types, except read-write entity beans, support load balancing at the method level.

Note: WebLogic Server does not always load-balance an object's method calls. In most cases, it is more efficient to use a replica that is collocated with the stub itself, rather than using a replica that resides on a remote server.

Failover for Clustered EJBs Increases Reliability

Failover for EJBs is accomplished using the `EJBHome` stub or, when supported, the `EJBObject` stub. When a client makes a call through a stub to a service that fails, the stub detects the failure and retries the call on another replica.

EJB failover requires that bean methods must be idempotent, and configured as such in `weblogic-ejb-jar.xml`. For more information see "Replication and Failover for EJBs and RMIs" in *Using WebLogic Server Clusters.*

Table 2-3 summarizes failover and load balancing features for clustered EJBs.

Table 2-3 Failover and Load Balancing for Clustered EJBs

EJB Type	Home Level Failover	Method Level Failover	Notes
Stateless Session	Supported	Supported	The `EJBObject` stub maintains the location of the EJB's primary and secondary states. Secondary server instances are selected using the same rules defined in "Using Replication Groups" in *Using WebLogic Clusters.* Stateless session EJB clustering behaviors are configured in `weblogic-ejb-jar.xml`. See "WebLogic-Specific Configurable Behaviors for Stateless Session EJBs".
Stateful session	Supported	Supported	Stateful session EJB clustering behaviors are configured in `weblogic-ejb-jar.xml`. See "WebLogic-Specific Configurable Behaviors for Session Beans" on page 5-9.
Read-Write Entity	Supported	Not supported	`EJBHome` stubs do not fail over in the event of a recoverable call failure. Failover is not supported during method execution, only after method completion, or if the method fails to connect to a server instance. A read-write bean's home obtains a local instance of the bean and returns a stub pinned to the local server instance. Entity clustering behaviors are configured in `weblogic-ejb-jar.xml`. See "Clustering Elements" on page 4-17.
Read-Only Entity	Supported	Supported	`EJBHome` stubs do not failover in the event of a recoverable call failure. Entity clustering behaviors are configured in `weblogic-ejb-jar.xml`. See "Clustering Elements" on page 4-17.
Message-Driven	WebLogic Java Messaging Service (JMS) supports clustering of JMS servers. To understand For more information, see "JMS and Clustering" in *Using WebLogic Server Clusters*.		

For more information, see "Replication and Failover for EJBs and RMIs" and "Load Balancing for EJBs and RMI Objects" *Using WebLogic Server Clusters*.

Securing EJBs

Using WebLogic Server security features, you control both access to your EJBs (authorization) and verification of an EJB's identity when it interacts with other application components and other EJBs (authentication).

WebLogic Server enables application developers to build security into their EJB applications using J2EE and WebLogic Server deployment descriptors, or allows system administrators to control security on EJB applications from the WebLogic Server Administration Console. The latter option frees developers from having to code security, and provides administrators with a centralized tool for defining security policies on: entire enterprise applications (EARs); an EJB JAR containing multiple EJBs; a particular EJB within that JAR; or a single method within that EJB.

For more information about security and EJBs:

- "Security Fundamentals" in *Introduction to WebLogic Security* has introductory information on authentication, authorization and other security topics.
- "Securing Enterprise JavaBeans (EJBs)" in *Programming WebLogic Security* provides instructions for configuring authentication and authorization for EJBs.
- *Securing WebLogic Resources* contains instructions for on configuring authentication and authorization for your EJBs using the Administration Console.

CHAPTER 3

Designing Enterprise Java Beans

These sections discuss design options for WebLogic Server Enterprise Java Beans (EJBs), bean behaviors to consider during the design process, and recommended design patterns.

- "Choosing the Right Bean Type" on page 3-1
- "Persistence Management Alternatives" on page 3-7
- "Transaction Design and Management Options" on page 3-8
- "Satisfying Application Requirements with WebLogic Server EJBs" on page 3-12

It is assumed the reader is familiar with Java programming, EJB 2.0, and the features described in "WebLogic Server Value-Added EJB Features" on page 2-10.

After finalizing your bean design, refer to programming and other implementation topics in Chapter 4, "Implementing Enterprise Java Beans."

Choosing the Right Bean Type

When you choose the bean type for a processing task, consider the different natures and capabilities of the various bean types.

Bean types vary in terms of the bean's relationship with its client. Some bean types stick with a client throughout a series of processes, serving as a sort of general contractor—acquiring and orchestrating services for the client. There are other bean types that act like subcontractors, they deliver the same single function to multiple client-oriented general contractor beans. A client-oriented bean keep track of client interactions and other information associated with the client process, throughout a client session—a capability referred to as *maintaining state*. Beans

that deliver commodity services to multiple client-oriented beans do not maintain state. For a broad discussion of client interaction and state management, see "J2EE and WebLogic Support for Service Types" in *Using WebLogic Server Clusters*.

The following sections describe the client interaction and state management characteristics of each bean type.

Note: For a basic overview of each bean type, including an example of a business process to which each bean type is well suited, see "How Do Applications Use EJBs?" on page 2-1.

Session Bean Features

A session bean represents a single client inside the server. To access an application that is deployed on the server, the client invokes the session bean's methods. The session bean performs work for its client, shielding the client from complexity by executing business tasks inside the server.

A session bean instance has a single client. Session beans are not persistent—when the client terminates, its session bean appears to terminate and is no longer associated with the client.

A session bean can be used as a *facade*, between Web applications and entity beans to contain complex interactions and reduce RMI calls. When a client application accesses a remote entity bean directly, each getter method is a remote call. A session facade bean can access the entity bean locally, collect the data in a structure, and return it by value. The two types of session beans, those that maintain state and those that do not, are described in the following sections.

Stateful Session Beans

Stateful session beans support conversational services with tightly-coupled clients. A stateful session bean accomplishes a task for a particular client. It maintains state for the duration of a client session. After session completion, state is not retained.

Stateful session beans are instantiated on a per client basis, and can multiply and consume resources rapidly.

Stateless Session Beans

Like a stateful session bean, a stateless session bean performs a task for a particular client. Unlike a stateful session bean, stateless session beans do not maintain client state. A stateless session bean may maintain state only for the duration of a method invocation. When the method is finished, the state is no longer retained.

Except during method invocation, all instances of a stateless bean are equivalent, allowing the EJB container to assign an instance to any client request. When a home interface creates a stateless bean, it returns a replica-aware stub that can route to any server where the bean is deployed. Because a stateless bean holds no state on behalf of the client, the stub is free to route any call to any server that hosts the bean.

Stateless Beans Offer Performance and Scalability Advantages

Because stateless session beans are never written to secondary storage, they typically offer better performance than stateful beans.

For applications that require large numbers of clients, stateless session beans offer better scalability than stateful session beans.

The system overhead of creating an instance is less for a stateless session bean than for a stateful session bean.

- You can simply obtain a stateless session bean instance from the free pool.
- A stateful session bean instance is instantiated upon client request, has to set the session context, and must be garbage-collected at the end of the session.

You can typically support more clients with a stateless session bean than a stateful session bean. A stateless session bean instance is only reserved for the duration of a single client request, while a stateful session bean is reserved for the duration of a session.

The number of stateless session bean instance required is usually roughly equivalent to the number of threads in the server execute queue—in the order of hundreds, while the number of stateful session bean instances required corresponds more closely to the number of clients of the application—which for some applications may be hundreds of thousands.

Entity Bean Features

An entity bean represents a business object in a persistent storage mechanism. Some examples of business objects are customers, orders, and products. In the J2EE SDK, the persistent storage mechanism is a relational database. Typically, each entity bean has an underlying table in a relational database, and each instance of the bean corresponds to a row in that table.

Key Features of Entity Beans

These are the key features of entity beans:

- Persistence—Entity bean persistence can be managed by the EJB container, or the bean itself. If a bean uses container-managed persistence, the EJB container automatically generates the necessary database access calls. The code that you write for the entity bean does not include these calls. With bean-managed persistence, you must write the database access code and include it in the bean.
- Shared Access—Throughout its lifecycle, an entity bean instance can support multiple clients, although not at the same time. Because the clients might want to change the same data, it is important that entity beans work within transactions. Typically, the EJB container provides transaction management. In this case, you specify the transaction attributes in the bean's `ejb-jar.xml` file that control how transactions are managed. You do not have to code the transaction boundaries in the bean—the container marks the boundaries for you. For information about transaction management, see "Features and Design Patterns" on page 3-12.
- Primary Key—Each entity bean has a unique object identifier. A customer entity bean, for example, might be identified by a customer number. The unique identifier, or primary key, enables the client to locate a particular entity bean. For more information, see "Using Container-Managed Relationships (CMRs)" on page 6-23.
- Relationships—Like a table in a relational database, an entity bean may be related to other entity beans. You implement relationships differently for entity beans with bean-managed persistence and for those with container-managed persistence. With bean-managed persistence, the code that you write implements the relationships. But with container-managed persistence, the EJB container takes care of the relationships for you. For this reason, relationships in entity beans with container-managed persistence are often referred to as container-managed relationships. For more information, see "Using Cascade Delete for Entities in CMRs" on page 6-32.

Read-Write versus Read-Only Entity Beans

WebLogic Server supports two types of entity beans: read-write and read-only.

Read-only beans perform better than read-write beans, because they reduce the number of times that data is read from the database.

Some applications require the use of read-write entity beans—the choice depends on frequency of updates and data consistency requirements. Table 3-1 provide key guidelines.

Table 3-1 Comparing Read-Write and Read-Only Entity Beans

Application Requirement	Choose read-write entity beans if	Choose read-only entity beans if
Frequency of updates	Your application data is updated often. **Example**: A real-time stock quote feed	Your application data is not updated often, or at all. **Example**: A swimwear catalogue
Data consistency	Your application requires transactionally consistent snapshots of data as it is updated. **Example**: An application that updates bank account balances.	You can tolerate some level of staleness in the data: it does not have to be completely up-to-date. **Example**: A news feed. Users want to see the story as soon as it is in the database, but it is not updated transactionally.

Entity Bean Performance and Data Consistency Characteristics

These sections describe approaches for choosing the entity bean implementation, based on your requirements for performance and data consistency.

Use Read-Only Beans to Improve Performance If Stale Data Is Tolerable

Read-only entity beans are recommended whenever stale data is tolerable—they are suitable for product catalogs and the majority of content within many applications. Primary key-based reads are performed against a local entity cache that is invalided on a timer basis.Other queries are made against the database. Read-only entity beans perform three to four times faster than transactional entities.

Note: A client can successfully call setter methods on a read-only entity bean; however, the data will never be moved into the persistent store.

Use Read-Write Beans for Higher Data Consistency

Read-write entity beans are recommended for applications that require high data consistency, for example, customer account maintenance. All reads and writes are performed against the database.

Note: For entity beans that use bean-managed persistence, or EJB 1.1 entity beans that use container-managed persistence, you can reduce the number of database writes by specifying the `isModified` method in `weblogic-ejb-jar.xml`.

Combine Read-Only and Read-Write Beans to Optimize Performance

For read-mostly applications, characterized by frequent reads, and occasional updates (for instance, a catalog)—a combination of read-only and read-write beans that extend the read-only beans is suitable. The read-only bean provides fast, weakly consistent reads, while the read-write bean provides strongly consistent writes.

Use Session Facades to Optimize Performance for Remote Entity Beans

To avoid the overhead imposed by remote calls, avoid accessing remote EJB entity beans from client or servlet code. Instead, use a session bean, referred to as a *facade*, to contain complex interactions and reduce calls from web applications to RMI objects. When a client application accesses a remote entity bean directly, each getter method is a remote call. A session facade bean can access the entity bean locally, collect the data in a structure, and return it by value.

Alternatively, there are no disadvantages to accessing a local entity bean instance directly from the web tier—it is preferable to do so than to use a facade.

Avoid the Use of Transfer Objects

Avoid the use of transfer objects, also referred to as value objects or helper classes. (A transfer object is a serializable class within an EJB that groups related attributes, forming a composite value, which is used as the return type of a remote business method.)

To optimize performance, accessing local entity instances is always preferable to the use of transfer objects.

Message-Driven Bean Features

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

These are the key features of message-driven beans:

- Clients do not access message-driven beans through interfaces. A message-driven bean has only a bean class.
- A message-driven bean's instances retain no data or conversational state for a specific client. All instances of a message-driven bean are equivalent, allowing the EJB container to

assign a message to any message-driven bean instance. The container can pool these instances to allow streams of messages to be processed concurrently.

- Throughout its lifecycle, a message-driven bean instance can process messages from multiple clients, although not concurrently.

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

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

For information about design alternatives for message-driven beans, see "MDBs and Messaging Models" on page 7-3.

Persistence Management Alternatives

Persistence management strategy determines how an entity bean's database access is performed.

Configure the persistence management strategy—either container-managed or bean-managed—for an entity bean in the `persistence-type` element in `ejb-jar.xml`.

Note: You can specify the use of a third-party persistence service for entity beans that use container-managed persistence in the `persistence-use` element in `weblogic-ejb-jar.xml`.

Use Container-Managed Persistence (CMP) for Productivity and Portability

A CMP bean relies upon the EJB container for all database interaction. The bean does not contain code that accesses the database. Instead, the EJB container generates the database access methods, based on information about the entity bean's persistent fields and relationships, in `weblogic-cmp-jar.xml`. For more information, see "weblogic-cmp-jar.xml Deployment Descriptor Reference".

CMP beans use EJB QL for database access. See Appendix E, "EJB Query Language (EJB-QL) and WebLogic Server".

Container-managed persistence offers these advantages:

- Reduced programming effort—You do not write methods to perform database access for a CMP bean. The EJB container generates the methods automatically.
- Increased portability—CMP increases bean portability in these ways:
  - De-coupling physical database details from business logic makes a bean logically independent of the associated database. If you implement a modified database design, or change to a different database server, you do not have to modify bean code.
  - You can redeploy the bean on a different J2EE application server without modifying or recompiling bean code.

Note: If you redeploy a bean that uses features that are not supported by the target application server, changes to the bean code might be necessary.

For more information on features supported by CMP entities, see "Entity EJBs" on page 6-1.

Use Bean-Managed Persistence (BMP) Only When Necessary

A bean that manages its own persistence must contain the methods that perform data access.

BMP is not encouraged—CMP offers many advantages over bean-managed persistence, as described in "Use Container-Managed Persistence (CMP) for Productivity and Portability" on page 3-7.

However, some application requirements cannot be satisfied by CMP beans. For instance, you must use BMP if:

- Your application must use an existing library of stored SQL procedures.
- The target database does not support JDBC.
- There is complex mapping between the bean and database tables. For instance, the bean maps multiple tables that do not share a primary key.

Transaction Design and Management Options

A transaction is a unit of work that changes application state—whether on disk, in memory or in a database—that, once started, is completed entirely, or not at all.

Understanding Transaction Demarcation Strategies and Performance

Transactions can be demarcated—started, and ended with a commit or rollback—by the EJB container, by bean code, or by client code.

Demarcating Transactions at the Server Level is Most Efficient

Transactions are costly application resources, especially database transactions, because they reserve a network connection for the duration of the transaction. In a multi-tiered architecture—with database, application server, and web layers—you optimize performance by reducing the network traffic "round trip." The best approach is to start and stop transactions at the application server level, in the EJB container.

Container-Managed Transactions Are Simpler to Develop and Perform Well

Container-managed transactions (CMTs) are supported by all bean types: session, entity, and message-driven. They provide good performance, and simplify development because the enterprise bean code does not include statements that begin and end the transaction.

Each method in a CMT bean can be associated with a single transaction, but does not have to be. In a container-managed transaction, the EJB container manages the transaction, including start, stop, commit, and rollback. Usually, the container starts the a transaction just before a bean method starts, and commits it just before the method exits.

For information about the elements related to transaction management in `ejb-jar.xml` and `weblogic-ejb-jar.xml`, see "Container-Managed Transactions Elements" on page 4-19.

Rollback

If a system exception is thrown during a transaction, the container will automatically roll back the transaction. You can also explicitly program rollbacks in your bean. For more information see Chapter 4, "Implementing Enterprise Java Beans."

Transaction Boundaries

You control how the EJB container manages the transaction boundaries when delegating a method invocation to an enterprise bean's business method for different invocation scenarios, with the `trans-attribute` element in `ejb-jar.xml`,

For example, if the client calling a transaction is itself running in a transaction, the `trans-attribute` element for the called transaction determines whether it will run in a new transaction or within the calling transaction.

Distributing Transactions Across Beans

A single database transaction can span multiple beans, on multiple servers instances. For information about implementing transactions that involve more than one bean, see "Programming Transactions That Are Distributed Across EJBs" on page 4-12.

Costly Option: Distributing Transactions Across Databases

Transactions that update multiple datastore must commit or roll back as a logical unit. The two-phase commit protocol is a method of coordinating a single transaction across multiple resource managers to ensure that updates are committed in all participating databases, or are fully rolled back out of all the databases.

Two-phase commit is resource-intensive. Avoid distributing transactions across databases.

Bean-Level Transaction Management

In a bean-managed transaction, the EJB code manages the transaction, including start, stop, commit, and rollback. Bean-managed transactions are supported by all session and message-driven beans; you cannot use bean-managed transactions with entity beans.

Notes: Bean-managed transactions cannot use container-provided transaction management features. Do not combine bean-managed and container-managed transactions in the same bean.

When to Use Bean-Managed Transactions

These are examples of requirements that may dictate the use of bean-managed transactions:

- You need to define multiple transactions with a single method call. With container-managed transactions, a method can only be associated with a single transaction. You can use a bean-managed transaction to define multiple transactions with a single method. However, consider avoiding the need for a bean-managed transaction by breaking the method in to multiple methods, each with its own container-managed transaction.
- You need to define a single transaction that spans multiple EJB method calls. For example, a stateful session EJB that uses one method to begin a transaction, and another method to commit or roll back a transaction.

Try to avoid this practice, 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.

Keep Bean-Managed Transaction Short

To simplify development, and improve reliability, keep bean-managed transactions reasonably short.

For information about implementing bean-managed transactions, see "Programming Bean-Managed Transactions" on page 4-10.

Client-Level Transaction Management is Costly

Client applications are subject to interruptions or unexpected terminations. If you start and stop a transaction at the client level, you risk:

- Consumption of network resources during waits for user actions, interruptions, until resumption of client activity or timeout.
- Consumption of processing resources and network resources to rollback the transaction after timeout or termination of the transaction.

Do not manage transactions in client applications unless there are overriding reasons to do so.

Transaction Isolation: A Performance vs. Data Consistency Choice

A transaction's isolation level is the degree to which it exposes updated but uncommitted data to other transactions. Allowing access to uncommitted data can improve performance, but increases the risk of incorrect data being supplied to other transactions.

Set the isolation level for bean-managed transactions in the bean's Java code. For instructions, see "Programming Bean-Managed Transactions" on page 4-10.

Set the isolation level for container-managed transactions in the `isolation-level` sub-element of the `transaction-isolation` element of `weblogic-ejb-jar.xml`. 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*.

Satisfying Application Requirements with WebLogic Server EJBs

WebLogic Server offers a variety of value-added features for enterprise beans that you can configure to meet the requirements of your application. They are described in "WebLogic Server Value-Added EJB Features" on page 2-10.

Table 3-2 cross references requirement types with topics that describe design strategies and WebLogic Server features you can use to satisfy your application requirements.

Table 3-2 Features and Design Patterns

When designing for	Consider these design patterns and WebLogic Server features
Availability and reliability	• "Failover for Clustered EJBs Increases Reliability" on page 2-14 • "Load Balancing Among Clustered EJBs Increases Scalability" on page 2-14
Scalability	• "Stateless Beans Offer Performance and Scalability Advantages" on page 3-3
Data Consistency	• "Use Container-Managed Persistence (CMP) for Productivity and Portability" on page 3-7 • "Use Read-Write Beans for Higher Data Consistency" on page 3-5. • "Transaction Isolation: A Performance vs. Data Consistency Choice" on page 3-11 • "Keep Bean-Managed Transaction Short" on page 3-11

When designing for	Consider these design patterns and WebLogic Server features
Developer and Administrator Productivity	• "Use Container-Managed Persistence (CMP) for Productivity and Portability" on page 3-7 • "Container-Managed Transactions Are Simpler to Develop and Perform Well" on page 3-9
Performance	Choosing bean types and design patterns: • "Combine Read-Only and Read-Write Beans to Optimize Performance" on page 3-6 • "Use Read-Only Beans to Improve Performance If Stale Data Is Tolerable" on page 3-5 • "Use Session Facades to Optimize Performance for Remote Entity Beans" on page 3-6 • "Avoid the Use of Transfer Objects" on page 3-6 • "Stateless Beans Offer Performance and Scalability Advantages" on page 3-3 Clustering features: • "Load Balancing Among Clustered EJBs Increases Scalability" on page 2-14 Pooling and caching: • "Performance-Enhancing Features for WebLogic Server EJBs" on page 2-10 Transaction management: • "Container-Managed Transactions Are Simpler to Develop and Perform Well" on page 3-9 • "Demarcating Transactions at the Server Level is Most Efficient" on page 3-9 • "Transaction Isolation: A Performance vs. Data Consistency Choice" on page 3-11 • "Costly Option: Distributing Transactions Across Databases" on page 3-10 • "Keep Bean-Managed Transaction Short" on page 3-11

CHAPTER 4

Implementing Enterprise Java Beans

The sections that follow describe the EJB implementation process, and provide guidance for how to get an EJB up and running in WebLogic Server.

It is assumed that you understand WebLogic Server's value-added EJB features, have selected a design pattern for your application, and have made key design decisions.

For a review of WebLogic Server EJB features, see "WebLogic Server Value-Added EJB Features" on page 2-10.

For discussion of design options for EJBs, factors to consider during the design process, and recommended design patterns see "Designing Enterprise Java Beans" on page 3-1.

- "Overview of the EJB Development Process" on page 4-2
- "Create a Source Directory" on page 4-5
- "Create EJB Classes and Interfaces" on page 4-6
- "Compile Java Source" on page 4-13
- "Generate Deployment Descriptors" on page 4-13
- "Edit Deployment Descriptors" on page 4-13
- "Generate EJB Wrapper Classes, and Stub and Skeleton Files" on page 4-24
- "Package" on page 4-26
- "Deploy" on page 4-26

- "Solving Problems During Development" on page 4-27
- "WebLogic Server Tools for Developing EJBs" on page 4-27

Overview of the EJB Development Process

This section is a brief overview of the EJB development process. It describes the key implementation tasks and associated results.

Figure 4-1 illustrates the process of developing an EJB. The steps in the process, and the results of each are described in Table 4-1. Subsequent sections detail each step in the process.

Figure 4-1 EJB Development Process Overview

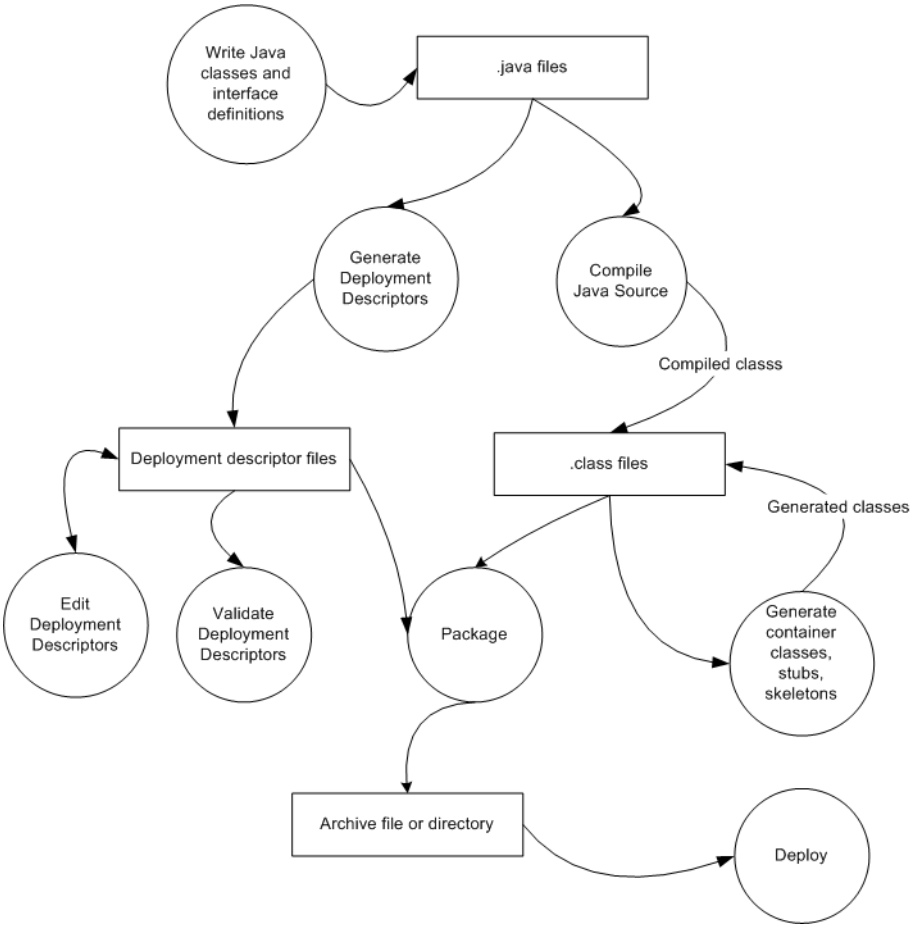


Table 4-1 EJB Development Tasks and Result

Step	Description	Result
1. Create a Source Directory	Create the directory structure for your source files, deployment descriptors, and files that are generated during the implementation process.	A directory structure on your local drive.
2. Create EJB Classes and Interfaces	Create the classes that make up your bean. Insert appropriate tags in your source code to enable automatic generation of deployment descriptor elements later in the implementation process.	`.java` file for each class
3. Compile Java Source	Compile source code.	`.class` file for each class
4. Generate Deployment Descriptors	Write or generate deployment descriptors that configure the runtime behavior and environment for the bean. If you used WebLogic Workshop for previous implementation tasks, generation of deployment descriptors is an automatic result of the compilation process.	`ejb-jar.xml`, and optionally: • `weblogic-ejb-jar.xml`, which contains elements that control WebLogic Server-specific features, and • `weblogic-cmp-jar.xml` if the bean is a container-managed persistence entity bean.
5. Edit Deployment Descriptors	You may need to edit deployment descriptors to ensure they correctly reflect all desired runtime behaviors for your bean. If your source was thoroughly tagged with markup that specifies the optional features the bean uses, and generated the deployment descriptors automatically, edits to your deployment descriptor should be minimal.	• `ejb-jar.xml`, • `weblogic-ejb-jar.xml`, which contains elements that control WebLogic Server-specific features, and • `weblogic-cmp-jar.xml` if the bean is a container-managed persistence entity bean.
6. Generate EJB Wrapper Classes, and Stub and Skeleton Files	Generate the container classes used to access the deployment unit, including classes for home and remote interfaces.	Generated classes are added to archive or directory.

Step	Description	Result
7. Package	Package compiled files, generated files, and deployment descriptors for deployment. If appropriate, you can leave your files unarchived in an exploded directory.	Archive, either a JAR or an EAR
8. Deploy	Target the archive or application directory to desired Managed Server, or a WebLogic Server cluster, in accordance with selected staging mode.	The deployment settings for the bean are written to `EJBComponent` element in `config.xml`.

Create a Source Directory

Create a source directory where you will assemble the EJB.

BEA recommends a s*plit development directory structure*, which segregates source and output files in parallel directory structures. For instructions on how to set up a split directory structure and package your EJB as an enterprise application archive (EAR), see "Introducing the Split Development Directory Structure" in *Developing WebLogic Server Applications*.

If you prefer to package and deploy your EJB in a JAR file, create a directory for your class files, and within that directory, a subdirectory named `META-INF` for deployment descriptor files.

Listing 4-1 Directory Structure for Packaging JAR

```
myEJB/
   META-INF/
      ejb-jar.xml
      weblogic-ejb-jar.xml
      weblogic-cmp-jar.xml
   foo.class
   fooHome.class
   fooBean.class
```

Create EJB Classes and Interfaces

The classes required depend on the type of EJB you are developing, as described in Table 2-1, "Components of an EJB," on page 2-4.

BEA offers productivity tools for developing class and interface files. The EJBGen command line utility automates the process of creating class and interface files, and also generates deployment descriptor files for the EJB. You can access EJBGen functionality using WebLogic Workshop, a development environment that EJBGen's capabilities with other development aids, and a provides a friendly user interface. For more information and instructions for using these tools see "EJBGen Reference" on page D-1 and *WebLogic Workshop Help*.

The sections that follow provide tips and guidelines for using WebLogic Server-specific EJB features.

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 {
```

Programming Client Access to EJBs

The following sections provide guidelines for programming client access to an EJB.

Programming Client to Obtain Initial Context

Local clients obtain initial context using the `getInitialContext` method, similar to the following excerpt.

Listing 4-2 Local Client Performing a Lookup

```
...
Context ctx = getInitialContexLt("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);

   return new InitialContext(h);
}
```

Remote clients obtain an `InitialContext` from the WebLogic Server `InitialContext` factory.

Programming Client to Look Up a Home Interface

A client can look up the entity bean's home interface in one of two ways:

- By following an EJB reference. This offers better performance, and is the preferred method. For instructions on using EJB references, see the following section, "Using EJB Links".
- Directly from the Java Naming and Directory Interface (JNDI). The container binds the entity bean's home interface in the global, server-side JNDI name space. For instructions see *Programming WebLogic JNDI*.

Using EJB Links

WebLogic Server fully supports EJB links as defined in the EJB 2.0 Specification. You can link an EJB reference that is declared in one application component to an enterprise bean that is declared in the same J2EE application.

In the `ejb-jar.xml` file, specify the link to the EJB using the `ejb-link` element of the `ejb-ref` element of the referencing application component. The value of `ejb-link` must match that the `ejb-name` in both `ejb-jar.xml` and `weblogic-ejb-jar.xml` of the target EJB. The target EJB can be in any EJB JAR file in the same J2EE application as the referencing application component.

Because ejb-names are not required to be unique across EJB JAR files, you may need to provide the qualified path for the link. Use the following syntax to provide the path name for the EJBs within the same J2EE application.

```
<ejb-link>../products/product.jar#ProductEJB</ejb-link>
```

This reference provides the path name of the EJB JAR file that contains the referenced EJB with the appended ejb-name of the target bean separated from the path by "#". The path name is relative to the referencing application component JAR file.

Configuring EJBs to Send Requests to a URL

To enable an EJB to open an HttpURLConnection to an external HTTP server using the java.net.URL resource manager connection factory type, specify the URL, or specify an object bound in the JNDI tree that maps to a URL, using the resource-ref element in ejb-jar.xml and the res-ref-name element in weblogic-ejb-jar.xml.

Specifying an HTTP Resource by URL

To specify the URL to which an EJB sends requests:

1. In ejb-jar.xml, specify the URL in the <jndi-name> element of the resource-ref stanza.
2. In weblogic-ejb-jar.xml, specify the URL in the <jndi-name> element of the resource-description stanza:

   ```
   <resource-description>
        <res-ref-name>url/MyURL</res-ref-name>
        <jndi-name>http://www.rediff.com/<;;/jndi-name>
   </resource-description>
   ```

   WebLogic Server creates a URL object with the jndi-name provided and binds the object to the java:comp/env.

Specifying an HTTP Resource by Its JNDI Name

To specify an object that is bound in JNDI and maps to a URL, instead of specifying a URL:

1. In ejb-jar.xml, specify the name by which the URL is bound in JNDI in the <jndi-name> element of the resource-ref stanza.
2. In weblogic-ejb-jar.xml, specify the name by which the URL is bound in JNDI in the <jndi-name> element of the resource-description stanza:

```
<resource-description>
     <res-ref-name>url/MyURL1</res-ref-name>
     <jndi-name>firstName</jndi-name>
</resource-description>
```

where `firstName` is the object bound to the JNDI tree that maps to the URL. This binding could be done in a startup class. When `jndi-name` is not a valid URL, WebLogic Server treats it as a object that maps to a URL and is already bound in the JNDI tree, and binds a `LinkRef` with that `jndi-name`.

Accessing HTTP Resources from Bean Code

Regardless of how you specified an HTTP resource—when by its URL or a JNDI name that maps to the URL—you can access it from EJB code in this way:

```
URL url = (URL) context.lookup("java:comp/env/url/MyURL");
connection = (HttpURLConnection)url.openConnection();
```

Programming Transaction

Transaction design decisions are discussed in "Features and Design Patterns" on page 3-12. The following sections contain guidelines for programming transactions.

For information using transactions with entity beans, see "Understanding ejbLoad() and ejbStore() Behavior" on page 6-4.

Programming Container-Managed Transactions

Container-managed transactions are simpler to program than bean-managed transactions, because they leave the job of demarcation—starting and stopping the transaction—to the EJB container.

You configure the desired transaction behaviors in `ejb-jar.xml` and `weblogic-ejb-jar.xml`. For related information see "Container-Managed Transactions Elements" on page 4-19.

Key programming guidelines for container-managed transaction include:

- Preserve transaction boundaries—Do not invoke methods that interfere with the transaction boundaries set by the container. Do not use:
  - The `commit`, `setAutoCommit`, and `rollback` methods of `java.sql.Connection`
  - The `getUserTransaction` method of `javax.ejb.EJBContext`
  - Any method of `javax.transaction.UserTransaction`

- Roll back transactions explicitly—To cause the container to roll back a container-managed transaction explicitly, invoke the `setRollbackOnly` method of the `EJBContext` interface. (If the bean throws a system exception, typically an `EJBException`, the rollback is automatic.)
- Avoid serialization problems—Many data stores provide limited support for detecting serialization problems, even for a single user connection. In such cases, even with `transaction-isolation` in `weblogic-ejb-jar.xml` set to `TransactionSerializable`, exceptions or rollbacks in the EJB client might occur if contention occurs between clients for the same rows. To avoid such exceptions, you can
  - Include code in your client application to catch SQL exceptions, and resolve them appropriately; for example, by restarting the transaction.
  - For Oracle databases, use the transaction isolation settings described in "isolation-level" on page A-44.

Programming Bean-Managed Transactions

This section contains programming considerations for bean-managed transactions. For a summary of the distinguishing features of bean-level transactions and a discussion of related design considerations, see "Bean-Level Transaction Management" on page 3-10.

- Demarcate transaction boundaries—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. To obtain the `UserTransaction` object, use this command:

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

  After obtaining the `UserTransaction` object, specify transaction boundaries with `tx.begin()`, `tx.commit()`, `tx.rollback()`.

  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 is reserved until the transaction commits or rolls back. To optimize performance and throughput, ensure that transactions complete quickly, so that the database connection can be released and made available to other client requests. See "Features and Design Patterns" on page 3-12 for more information.

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

- Setting transaction isolation level—For bean-managed transactions, you define isolation level in the bean code. Allowable isolation levels are defined in "isolation-level" on page A-44.

  Note: The Oracle-only isolation level values—`TRANSACTION_READ_COMMITTED_FOR_UPDATE` and `TRANSACTION_READ_COMMITTED_FOR_UPDATE_NO_WAIT` cannot be set for a bean-managed transaction.

  See Listing 4-3 for a code sample.

Listing 4-3 Setting Transaction Isolation Level in BMT

```
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();
```

- Avoid restricted methods—Do not invoke the `getRollbackOnly` and `setRollbackOnly` methods of the `EJBContext` interface in bean-managed transactions. These methods should be used only in container-managed transactions. For bean-managed transactions, invoke the `getStatus` and rollback methods of the `UserTransaction` interface.
- Use one connection per active transaction context—You can associate only a single database connection with an active transaction context.

Programming Transactions That Are Distributed Across EJBs

This section describes two approaches for distributing a transaction across multiple beans, which may reside on multiple server instances.

Calling multiple EJBs from a client's transaction context

The code fragment below is from a client application that obtains a `UserTransaction` object and uses it to begin and commit a transaction. The client invokes two EJBs within the context of the transaction.

```
import javax.transaction.*;
...
u = (UserTransaction)
jndiContext.lookup("javax.transaction.UserTransaction");
u.begin();
account1.withdraw(100);
account2.deposit(100);
u.commit();
...
```

The updates performed by the `account1` and `account2` beans occur within the context of a single `UserTransaction`. The EJBs commit or roll back together, as a logical unit, whether the beans reside on the same server instance, different server instances, or a WebLogic Server cluster.

All EJBs called from a single transaction context must both support the client transaction—each beans' `trans-attribute` element in `ejb-jar.xml` must be set to `Required`, `Supports`, or `Mandatory`.

Using an EJB "Wrapper" to Encapsulate a Cross-EJB Transaction

You can use a "wrapper" EJB that encapsulates a transaction. The client calls the wrapper EJB to perform an action such as a bank transfer, and the wrapper starts a new transaction and invokes 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 wrapper's `trans-attribute` element in `ejb-jar.xml` is set to `Required` or `RequiresNew`.

All EJBs invoked by the wrapper EJB must support the wrapper EJB's transaction context— their `trans-attribute` elements must be set to `Required`, `Supports` or `Mandatory`.

Compile Java Source

WebLogic Workshop is the recommended tool for compiling class files. For more information about WebLogic Workshop, see *WebLogic Workshop Help*. To see what other tools support the compilation process, see Table 4-11, "EJB Tools and Features," on page 4-32.

For information on the compilation process, see "Compiling Java Code" in *Developing WebLogic Server Applications*.

Generate Deployment Descriptors

If you have used WebLogic Workshop or EJBGen to annotate your source files with the tags for desired bean features and run-time behaviors, descriptor generation is an automatic process.

These tools automatically generate the necessary deployment descriptors from your class file, as well as home and remote interfaces, from a bean class file.

For more information see:

- *WebLogic Workshop Help*
- EJBGen Reference

Edit Deployment Descriptors

Elements in `ejb-jar.xml`, `weblogic-ejb-jar.xml`, and for container-managed persistence entity beans, `weblogic-cmp-jar.xml`, control the run-time characteristics of your application.

If you need to modify a descriptor element, you can edit the descriptor file with any plain text editor. However, to avoid introducing errors, use a tool designed for XML editing, such as EJBgen or WebLogic Builder. Descriptor elements that you can edit with the WebLogic Server Administration Console are listed in Table 4-10.

The following sections are a quick reference to WebLogic Server-specific deployment elements. Each section contains the elements related to a type of feature or behavior. The table in each section defines relevant elements terms of the behavior it controls, the bean type it relates to (if bean type-specific), the parent stanza in `weblogic-ejb-jar.xml` that contains the element, and the behavior you can expect if you do not explicitly specify the element in `weblogic-ejb-jar.xml`.

- "Security Elements" on page 4-14
- "Resource Mapping Elements" on page 4-15

- "Persistence Elements" on page 4-16
- "Clustering Elements" on page 4-17
- "Data Consistency Elements" on page 4-19
- "Container-Managed Transactions Elements" on page 4-19
- "Performance Elements" on page 4-21

For comprehensive documentation of the elements in each descriptor file, definitions, and sample usage, refer to:

- "Implementing Enterprise Java Beans" on page 4-1

  Note: In the sections that follow, click the element name in the "Element" column to view detailed documentation on the element in "weblogic-ejb-jar.xml Deployment Descriptor Reference"

- "weblogic-cmp-jar.xml Deployment Descriptor Reference" on page B-1
- Your Sun documentation for elements in `ejb-jar.xml`.

Security Elements

This table lists the elements in `weblogic-ejb-xml.jar` related to security.

Table 4-2 Security Elements in weblogic-ejb-jar.xml

Element	Description	Default
`security-role-assignment`	Maps security roles in `ejb-jar.xml` file to the names of security principals in WebLogic Server. Required if `ejb-jar.xml` defines application roles.	none
`security-permission`	Additional Java security permission that is granted to this EJB.	none
`run-as-principal-name`	Security principal name to use as the `run-as principal` for a bean that has specified a `security-identity run-as-role-name` in `ejb-jar.xml`.	none
`iiop-security-descriptor`	Security options for beans that use the RMI-IIOP protocol.	none

Resource Mapping Elements

This table lists the elements in `weblogic-ejb-xml.jar` that map the names of beans or resources used in source code to their JNDI names in the deployment environment.

Table 4-3 Resource Mapping Elements in weblogic-ejb-jar.xml

Element	Bean Type	Description	Default
`jndi-name`	All	JNDI name of a resource or reference available in WebLogic Server **Note:** Assigning a JNDI name to an bean is not recommended. Global JNDI names generate heavy multicast traffic during clustered server startup. See "Using EJB Links" on page 4-7 for the better practice.	none
`local-jndi-name`	All	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.	none
`connection-factory-jndi-name`	MDB	JNDI name of the JMS connection factory that the bean uses to create queues and topics.	`weblogic.jms.MessageDrivenBeanConnectionFactory`
`destination-jndi-name`	MDB	JNDI name that associates a message-driven bean with a queue or topic in the JNDI tree.	
`initial-context-factory`	MDB	Initial context factory that the EJB container uses to create connection factories.	`weblogic.jndi.WLInitialContextFactory`
`jms-client-id`	MDB	Client ID for the message-driven bean associated with a durable subscriber topic.	Value of `ejb-name`
`provider-url`	MDB	Specifies the URL provider to be used by the `InitialContext`.	`t3://localhost:7001`

Persistence Elements

This table lists elements in `weblogic-ejb-xml.jar` that specify how the state of a bean is persisted.

Table 4-4 Persistence Elements in weblogic-ejb-jar.xml

Element	Bean Type	Description	Default
`type-identifier`	Entity	Specifies EJB persistence type. WebLogic Server RDBMS-based persistence uses the identifier, `WebLogic_CMP_RDBMS`	
`type-storage`	Entity	Defines path, relative to the top level of the EJB's JAR deployment file or deployment directory, of the file that stores data for this persistence type. WebLogic Server RDBMS-based persistence generally uses an XML file named `weblogic-cmp-jar.xml` to store persistence data for a bean. This file is stored in the `META-INF` subdirectory of the JAR file.	
`type-version`	Entity	Version of the persistence type specified by `type-identifier`. For WebLogic 2.0 CMP persistence, use the value `2.0`. For WebLogic 1.1 CMP persistence, use the value 1.1.	
`delay-updates-until-end-of-tx`	Entity	If true, the EJB container attempts to delay writing updates to a bean's state to the database until the end of a transaction. However, the container still flushes updates to the database before executing an EJB finder or select query if the `include-updates` element (in the `weblogic-query stanza` of `weblogic-cmp-jar.xml)` for the query is true. Applicable to both container-managed persistence and bean-managed persistence beans.	True

Element	Bean Type	Description	Default
finders-load-bean	Entity	Causes beans returned by a `finder` or `ejbSelect` method to be loaded immediately into the cache before the method returns. **Note:** Applicable to container-managed persistence beans only.	True
persistent-store-dir	Stateful Session	Directory where state of passivated stateful session bean instances is stored.	
is-modified-method-name	Entity	The method called by the container to determine whether or not the bean has been modified and needs to have its changes written to the database. Applies to bean-managed persistence or EJB 1.1 container-managed persistence beans.	If not specified, bean state is persisted after each method completes.

Clustering Elements

This table lists the elements in `weblogic-ejb-jar.xml` related to clustering. These elements control failover and load balancing behaviors for clustered beans in a WebLogic Server cluster.

Table 4-5 Clustering Elements in weblogic-ejb-jar.xml

Element	Bean Type	Description	Default
home-call-router-class-name	Stateful Session Stateless Session Entity	Custom class to be used for routing home method calls. This class must implement `weblogic.rmi.extensions.CallRouter()`.	None
home-is-clusterable	Stateful Session Stateless Session Entity	Indicates whether the bean home can be clustered.	True

Element	Bean Type	Description	Default
home-load-algorithm	Stateful Session Stateless Session Entity	Algorithm to use for load-balancing among replicas of the bean home.	Value of weblogic.cluster.defaultLoad Algorithm
idempotent-methods		Idempotent methods for a clustered EJB. An idempotent method can be repeated with no negative side-effects. Methods of stateless session bean homes and read-only entity bean interfaces do not need to be explicitly identified—they are automatically set to be idempotent.	None
replication-type	Stateful Session	Indicates the replication used for stateful session beans in a cluster: in-memory or none.	none
stateless-bean-call-router-class-name	Stateless Session	Custom class to be used for routing bean method calls.	None
stateless-bean-is-clusterable	Stateless Session	Indicates that the bean is clusterable. Use only for session beans whose session-type in ejb-jar.xml is Stateless.	True
stateless-bean-load-algorithm	Stateless Session	Algorithm to use for load-balancing among replicas of the bean.	Value of the property weblogic.cluster.defaultLoad Algorithm
use-serverside-stubs	Stateless Session	Causes the bean home to use server-side stubs in the server context.	False

Data Consistency Elements

This table lists the elements in `weblogic-ejb-jar.xml` related to the consistency of the bean instance data and the database. These elements control behaviors such as how and when the database is updated to reflect the values in the bean instance is done.

Note: For elements related to container-managed persistence, see "Managing Entity Bean Pooling and Caching" on page 6-1.

Table 4-6 Data Consistency Elements in weblogic-ejb-jar.xml

Element	Bean Type	Description	Default
`concurrency-strategy`	Entity	How concurrent access to an entity bean is managed.	Database
`invalidation-target`	Entity	The read-only entity bean to invalidate when this container-managed persistence entity bean is modified. **Note:** Only applicable to EJB 2.0 CMP beans.	None
`delay-updates-until-end-of-tx`	Entity	If true, the EJB container attempts to delay writing updates to a bean's state to the database until the end of a transaction. However, the container still flushes updates to the database before executing an EJB finder or select query if the `include-updates` element (in the `weblogic-query stanza of weblogic-cmp-jar.xml)` for the query is true. Applicable to both container-managed persistence and bean-managed persistence beans.	True

Container-Managed Transactions Elements

Table 4-7 lists the elements in `ejb-xml.jar` related to container-managed transactions.

Table 4-7 Container-Managed Transaction Elements in ejb-jar.xml

Element	Description	Default
`transaction-type`	Allowable values are `Bean` or `Container`.	None, EJB 2.0 requires this attribute to be specified.
`trans-attribute`	Specifies how the container manages the transaction boundaries when delegating a method invocation to an enterprise bean's business method. Allowable value include: • `NotSupported` • `Supports` • `Required` • `RequiresNew` • `Mandatory` • `Never` **Note:** Because clients do not provide a transaction context for calls to an MDB, MDBs that use container-managed transactions must have `trans-attribute` of `Required`.	If not specified, the EJB container issues a warning, and uses `NotSupported` for MDBs and `Supports` for other types of EJBs.
`transaction-scope`	This optional element specifies whether an enterprise bean requires distributed transactions for its methods or whether the local transaction optimization may be used. Allowable values are `Local` and `Distributed`.	If not specified, the container assumes that distributed transactions must be used.

Table 4-8 lists the elements in `weblogic-ejb-xml.jar` related to container-managed transactions.

Table 4-8 Container-Managed Transaction Elements in weblogic-ejb-jar.xml

Element	Description	Default
transaction-isolation	The transaction isolation level used when method starts a transaction. The specified transaction level is not used if the method inherits an existing transaction.	The default of the underlying DBMS
trans-timeout-seconds	Maximum duration for a transaction.	None

Performance Elements

This table lists the elements in `weblogic-ejb-xml.jar` related to performance.

Table 4-9 Performance Elements in weblogic-ejb-jar.xmll

Element	Bean Type	Description	Default
allow-concurrent-calls	Stateful Session	Whether multiple clients can simultaneously access a bean without triggering a `Remote Exception`. The server throws a `RemoteException` when a stateful session bean instance is currently handling a method call and another (concurrent) method call arrives on the server.	False
cache-between-transactions	Entity	Causes the container to cache the persistent data of an entity bean between transactions.	False
cache-type	Stateful Session	Order in which stateful session beans are removed from the cache.	NRU (not recently used)
clients-on-same-server	All	Indicates that all clients of the bean are collocated with the bean on the same server instance. This element is only used if the EJB has a global JNDI name; setting it to `true` prevents the JNDI name from being replicated. A value of `true` can reduce cluster startup time in large clusters.	False

Element	Bean Type	Description	Default
delay-updates-until-end-of-tx	Entity	If `true`, the EJB container attempts to delay writing updates to a bean's state to the database until the end of a transaction. However, the container still flushes updates to the database before executing an EJB finder or select query if the include-updates element (in the `weblogic-query stanza` of `weblogic-cmp-jar.xml)` for the query is true. Applicable to both container-managed persistence and bean-managed persistence beans.	True
dispatch-policy	All	Specifies the thread pool used to handle requests to the bean.	None
enable-call-by-reference	All	Improves performance of method invocation for methods called within the same application, by allowing parameters to be passed by reference. **Note:** Method parameters are *always* passed by value when an EJB is called remotely.	False
entity-cache-name	Entity	The application-level entity cache, which can cache instances of multiple entity beans that are part of the same application. **Note:** Application level caches are declared in the `weblogic-application.xml`.	None
estimated-bean-size	Entity	Estimated average size, in bytes, of an entity bean instance.	None
finders-load-bean	Entity	Causes beans returned by a `finder` or `ejbSelect` method to be loaded immediately into the cache before the method returns. **Note:** Applicable to container-managed persistence beans only.	True
idle-timeout-seconds	Entity	Number of seconds of inactivity after which a bean is passivated. **Note:** This element is not currently used.	600

Element	Bean Type	Description	Default
idle-timeout-seconds	Stateful Session	Number of seconds of inactivity after which a bean is passivated.	600
initial-beans-in-free-pool	Entity Message-Driven Stateless Session	Number of instances of an EJB instantiated by the container at startup.	0
is-modified-method-name	Entity	The method that changes the state of bean. Specifying this method causes WebLogic server to persist the bean state when the method completes. **Note:** Applies to bean-managed persistence or EJB 1.1 container-managed persistence beans.	If not specified, bean state is persisted after each method completes.
jms-polling-interval-seconds	Message-driven	The number of seconds between attempts by the EJB container to reconnect to a JMS destination that has become unavailable.	10
max-beans-in-cache	Entity Stateful Session	Maximum number instances in the cache.	1000
max-beans-in-free-pool	Entity Stateless Session Message-Driven	Maximum number of instances in the free pool.	1000
read-timeout-seconds	Entity	The number of seconds between `ejbLoad` calls on a read-only entity bean. If `read-timeout-seconds` is 0, `ejbLoad` is only called when the bean is brought into the cache.	600
remote-client-timeout	All	Specifies the length of time that a remote RMI client will wait before it will timeout.	0

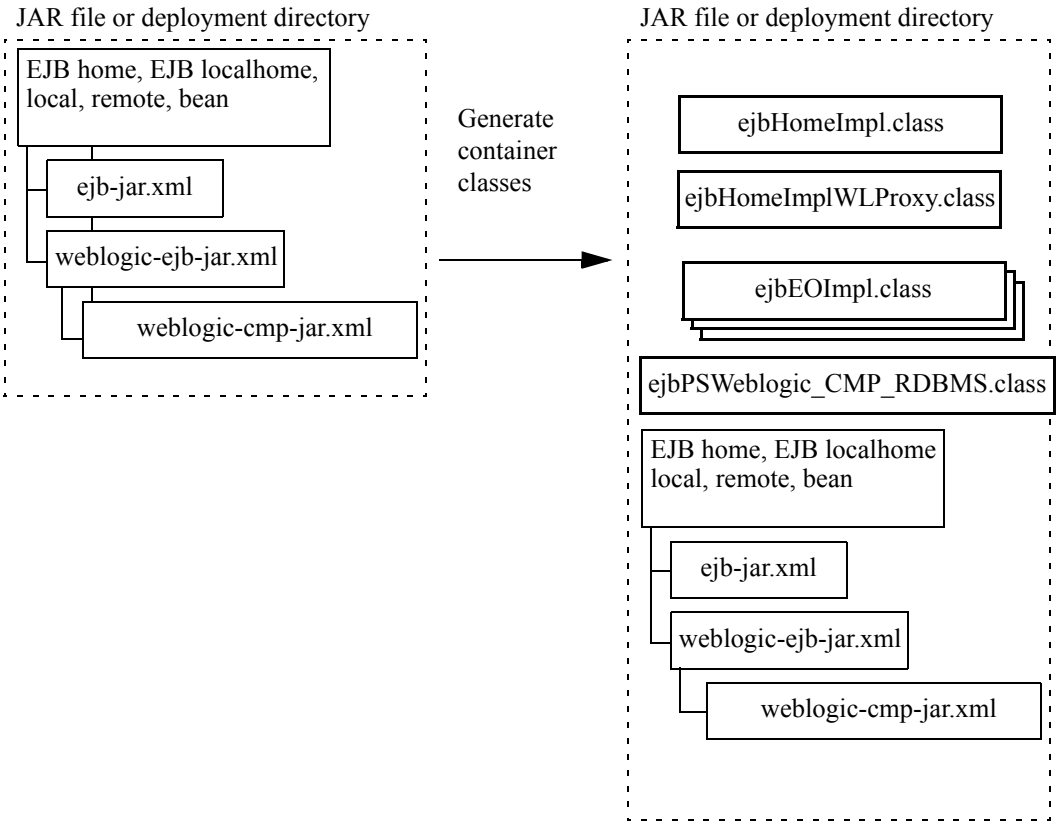
Generate EJB Wrapper Classes, and Stub and Skeleton Files

Container classes include the internal representation of the EJB that WebLogic Server uses and the implementation of the external interfaces (home, local, and/or remote) that clients use. You can use WebLogic Workshop or `appc` to generate container classes.

Container classes are generated in according to the descriptor elements in `weblogic-ejb-jar.xml`. For example, if you specify clustering elements, `appc` creates cluster-aware classes that will be used for deployment. You can use `appc` directly from the command line by supplying the required options and arguments. See "appc" on page C-1 for more information.

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

Figure 4-2 Generating EJB Container Classes



appc and Generated Class Name Collisions

Although infrequent, when you generate classes with `appc`, you may encounter a generated class name collision which could result in a `ClassCastException` and other undesirable behavior. This is because the names of the generated classes are based on three keys: the bean class name, the bean class package, and the `ejb-name` for the bean. This problem occurs when you use an EAR file that contains multiple JAR files and at least two of the JAR files contain an EJB with both the same bean class, package, or classname, and both of those EJBs have the same `ejb-name` in their respective JAR files. If you experience this problem, change the `ejb-name` of one of the beans to make it unique.

Because the `ejb-name` is one of the keys on which the file name is based and the `ejb-name` must be unique within a JAR file, this problem never occurs with two EJBs in the same JAR file. Also, because each EAR file has its own classloader, this problem never occurs with two EJBs in different EAR files.

Package

BEA recommends that you package EJBs as part of an enterprise application. For more information on recommending packaging practices and packaging alternatives, see "Creating WebLogic Server Applications" in *Developing WebLogic Server Applications*.

Packaging Considerations for EJBs with Clients in Other Applications

WebLogic Server supports the use of `ejb-client.jar` files for packaging the EJB classes that a programmatic client in a different application requires to access the EJB.

Specify the name of the client JAR in the `ejb-client-jar` element of the bean's `ejb-jar.xml` file. When you run the `appc` compiler, a JAR file with the classes required to access the EJB is generated.

Make the client JAR available to the remote client. For Web applications, put the `ejb-client.jar` in the `/lib` directory. For non-Web clients, include `ejb-client.jar` in the client's classpath.

Note: WebLogic Server classloading behavior varies, depending on whether 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.

Deploy

Deploying an EJB enables WebLogic Server to serve the components of an EJB to clients. You can deploy an EJB using one of several procedures, depending on your environment and whether or not your EJB is in production.

For general instructions on deploying WebLogic Server applications and modules, including EJBs, see *Deploying WebLogic Server Applications*. For EJB-specific deployment issues and procedures, see Chapter 8, "Deployment Guidelines for Enterprise Java Beans," in this book—*Programming WebLogic Enterprise JavaBeans*.

Solving Problems During Development

The following sections describe WebLogic Server features that are useful for checking out and debugging deployed EJBs.

Adding Line Numbers to Class Files

If you compile your EJBs with `appc`, you can use the `appc -lineNumbers` command option to add line numbers to generated class files to aid in debugging. For information, see "appc and ejbc Reference" on page C-1.

Monitoring Data

WebLogic Server collects a variety of data about the run-time operation of a deployed EJB. This data, which you can view in the Deployments node of the Administration Console, can be useful in determining if an EJB has completed desired processing steps. To access EJB run-time statistics, expand the Deployment node in the Administration Console, navigate to the JAR EAR that contains the bean, and select the Monitoring tab.

For information about the data available, see these pages in Administration Console Online Help:

- "Deployments-->EJB --> Monitoring --> Stateful Session EJBs"
- "Deployments-->EJB-->Monitoring-->Stateless EJBs"
- "Deployments-->EJB --> Monitoring--> Message-Driven EJBs"
- "Deployments-->EJB --> Monitoring --> Entity EJBs"

Creating Debug Messages

For instructions on how to create messages in your application to help you troubleshoot and solve bugs and problems, see "Writing Debug Messages" in *Using WebLogic Logging Services*.

WebLogic Server Tools for Developing EJBs

This section describes BEA tools that support the EJB development process. For a comparison of the features available in each tool, see Table 4-11, "EJB Tools and Features," on page 4-32.

WebLogic Workshop

WebLogic Workshop is a shared development environment for the BEA WebLogic Platform. Workshop it supports all aspects of EJB development, from code creation through deployment, and is the BEA-recommended tool for EJB development. Key features include:

- Graphical user interface
- Extensible control model for creating Java controls
- Versatile editor for handling a variety of J2EE file types
- Integrated debugger
- Integrated Ant build scripts
- Automatic insertion of EJBGen tags
- A full-function project environment for creating and maintaining EJBs; generating EJBs from database tables; and deploying applications that contain EJBs to WebLogic Server.

For information about WebLogic Workshop, see *WebLogic Workshop Help* at http://e-docs.bea.com/workshop/docs81/index.html.

Administration Console

Using the Descriptors tab in the Administration Console, you can view, modify, and persist to the descriptor file within the EJB a number of deployment descriptor elements. Descriptors are modified in the Administration Server copy of the EJB as well as in any deployed copies of the EJB (after deployment). When you modify descriptors using WebLogic Builder or a manual editing tool, changes are made to your (the user's) original copy of the EJB (prior to deployment).

However, updating these descriptor elements takes place dynamically at runtime without requiring that the EJB be redeployed. The descriptor element attributes contained in the Descriptors tab are limited to only those that may be dynamically changed at runtime, as summarized in Table 4-10.

Note: You cannot use the Administration Console to edit deployment descriptors for applications and modules that are deployed from archive files.

Table 4-10 Descriptor Elements Available from Administration Console

EJB Type	Editable Elements
Entity	• max-beans-in-cache • max-beans-in-free-pool • key-cache-size • trans-timeout-seconds • read-timeout-seconds (read-only entity beans only)
Message-Driven	• max-beans-in-free-pool • message-selector • jms-polling-interval-seconds • trans-timeout-seconds
Stateless	• max-beans-in-free-pool • trans-timeout-seconds
Stateful	• max-beans-in-free-pool • idle-timeout-seconds • trans-timeout-seconds

javac

The `javac` compiler provided with the Sun Java J2SE SDK provides java compilation capabilities. For information on `javac`, see http://java.sun.com/docs/.

EJBGen

EJBGen is an EJB 2.0 code generator. You can annotate your bean class file with javadoc tags and then use EJBGen to generate the remote and home interface classes and the deployment descriptor files for an EJB application, reducing to one the number of EJB files you need to edit and maintain.

BEA recommends that you use `EJBGen` to generate deployment descriptors; this is a BEA best practice which allows for easier and simpler maintenance of EJBs. When you use `EJBGen`, you have to write and annotate only one bean class file, which simplifies writing, debugging, and maintenance. If you use WebLogic Workshop as a development environment, WebLogic Workshop automatically inserts EJBGen tags for you.

For an example of an application that uses EJBGen, install the WebLogic 8.1 Server examples, and see `WL_HOME\samples\server\examples\src\examples\ejb20\ejbgen`. The application is called Bands.

For information on EJBGen, see Appendix D, "EJBGen Reference".

WebLogic Builder

WebLogic Builder is a visual environment for viewing deployment descriptor files, and modifying their contents without directly editing the XML.

WebLogic Builder supports these development tasks:

- Generate deployment descriptor files for a J2EE module
- Edit a module's deployment descriptor files
- View deployment descriptor files
- Compile and validate deployment descriptor files
- Deploy a module to a server

See *WebLogic Builder Online Help*.

DDInit

DDInit is a utility for generating deployment descriptors for WebLogic Server applications. DDInit uses information from the class files to create deployment descriptor files.

WebLogic Builder uses DDInit to generate deployment descriptors. For more information, see "WebLogic Builder" on page 4-30.

See "DDInit" in *WebLogic Server Command Reference*.

WebLogic Server Ant Utilities

WebLogic Server includes Ant utilities to create skeleton deployment descriptors.

The Ant task examines a directory containing an EJB and creates deployment descriptors based on the directory contents. Because the Ant utility does not have information about all 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 WebLogic Builder to edit the deployment descriptors and complete the configuration of your EJB.

For more information, see "wldeploy Ant Task" in *Deploying WebLogic Server Applications*.

weblogic.Deployer

The `weblogic.Deployer` command-line tool is a Java-based deployment tool that provides a command line interface to the WebLogic Server deployment API. This tool was developed for administrators and developers who need to initiate deployment from the command line, a shell script, or any automated environment other than Java.

See "weblogic.Deployer Utility" in *Deploying WebLogic Server Applications*.

appc

The `appc` compiler generates and compiles the classes needed to deploy EJBs and JSPs to WebLogic Server. It validates the deployment descriptors for compliance with the current specifications at both the individual module level and the application level. The application-level checks include checks between the application-level deployment descriptors and the individual modules as well as validation checks across the modules.

Note: appc replaces the deprecated `ejbc` utility. BEA recommends using `appc` instead `ejbc`.

See Appendix C, "appc and ejbc Reference"

DDConverter

The `DDConverter` is a command line tool that converts earlier versions EJB deployment descriptors into EJB deployment descriptors that conform to current version of WebLogic Server.

BEA recommends that you always convert descriptors when migrating applications to a new WebLogic Server release.

Comparison of EJB Tool Features

The following table lists BEA tools for EJB development, and the features provided by each.

Table 4-11 EJB Tools and Features

	Generate Interfaces and Home Interfaces	Compile Java Code	Generate Deployment Descriptors	View and Edit Deployment Descriptors	Deploy
WebLogic Workshop	X	X	X		X
appc		X			
javac		X			
EJBGen	X		X	X	
WebLogic Builder			X	X	X
DDinit			X		
Administration Console				X	X
Deployer					X
DDinit			X		
DDConverter			X		

CHAPTER 5

Session EJBs

This section describes how session beans work within the EJB container, and provides design and development guidelines that are specific to session beans. For a description of the overall bean development process, see "Implementing Enterprise Java Beans" on page 4-1.

It is assumed that the reader is familiar with Java programming and session bean features and capabilities. For an introduction to session bean features and how they are typically used in applications, see "Session EJBs Implement Business Logic" on page 2-1 and "Session Bean Features" on page 3-2.

The following sections describe the session bean life-cycle, design considerations, and instructions for key implementation tasks.

- "Comparing Stateless and Stateful Session Beans" on page 5-1
- "Pooling for Stateless Session EJBs" on page 5-2
- "Caching and Passivating Stateful Session EJBs" on page 5-4
- "Design Decisions for Session Beans" on page 5-8
- "Implementing Session Beans" on page 5-9

Comparing Stateless and Stateful Session Beans

This section looks at the key differences between stateless and stateful session beans.

Table 5-1 Comparing Stateless and Stateful Session Beans

Stateless Session Beans	Stateful Sessions Beans
Are pooled in memory, to save the overhead of creating a bean every time one is needed. WebLogic Server uses a bean instance when needed and puts it back in the pool when the work is complete. Stateless sessions beans provide faster performance than stateful beans.	Each client creates a new instance of a bean, and eventually removes it. Instances may be passivated to disk if the cache fills up. An application issues an `ejbRemove()` to remove the bean from the cache. Stateful sessions beans do not perform as well as stateless sessions beans.
Have no identity and no client association; they are anonymous.	Are bound to particular client instances.Each bean has an implicit identity. Each time a client interacts with a stateful session bean during a session, it is the same object.
Do not persist. The bean has no state between calls.	Persist. A stateful session bean's state is preserved for the duration of a session.

See "Choosing Between Stateless and Stateful Beans" on page 5-8 for a discussion of when to use which type of session bean.

Pooling for Stateless Session EJBs

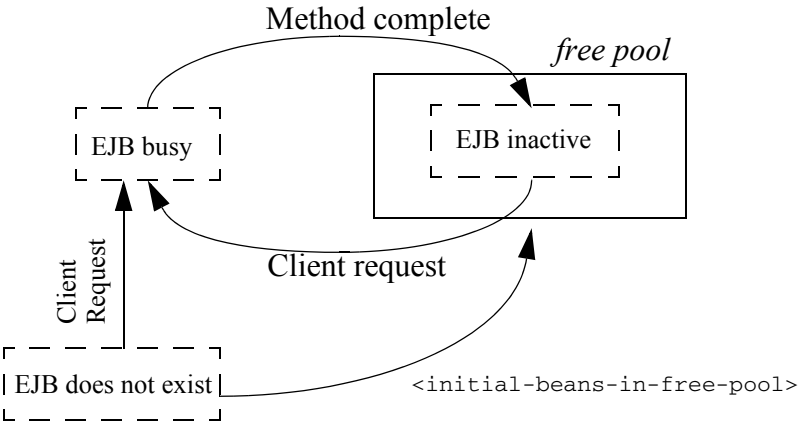
By default, no stateless session EJB instances exist in WebLogic Server at startup time. As individual beans are invoked, WebLogic Server initializes new instances of the EJB.

However, in a production environment, WebLogic Server can provide improved performance and throughput for stateless session EJBs by maintaining a free pool of unbound stateless session EJBs—instances that are not currently processing a method call. If an unbound instance is available to serve a request, response time improves, because the request does not have to wait for an instance to be created. The free pool improves performance by reusing objects and skipping container callbacks when it can.

Upon startup, WebLogic Server automatically creates and populates the free pool with the quantity of instances you specify in the bean's initial-beans-in-free-pool deployment element in the `weblogic-ejb-jar.xml` file. By default, `initial-beans-in-free-pool` is set to 0.

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 5-1 WebLogic Server Free Pool Showing Stateless Session EJB Life Cycle



If you configure a pool, WebLogic Server will service method calls with an EJB instance from the free pool, if one is available. 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` for a remote client or an `EJBException` for a local client.

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

When an application requests a bean instance from the free pool, there are three possible outcomes:

- An instance is available in the pool. WebLogic Server makes that instance available and your application proceeds with processing.

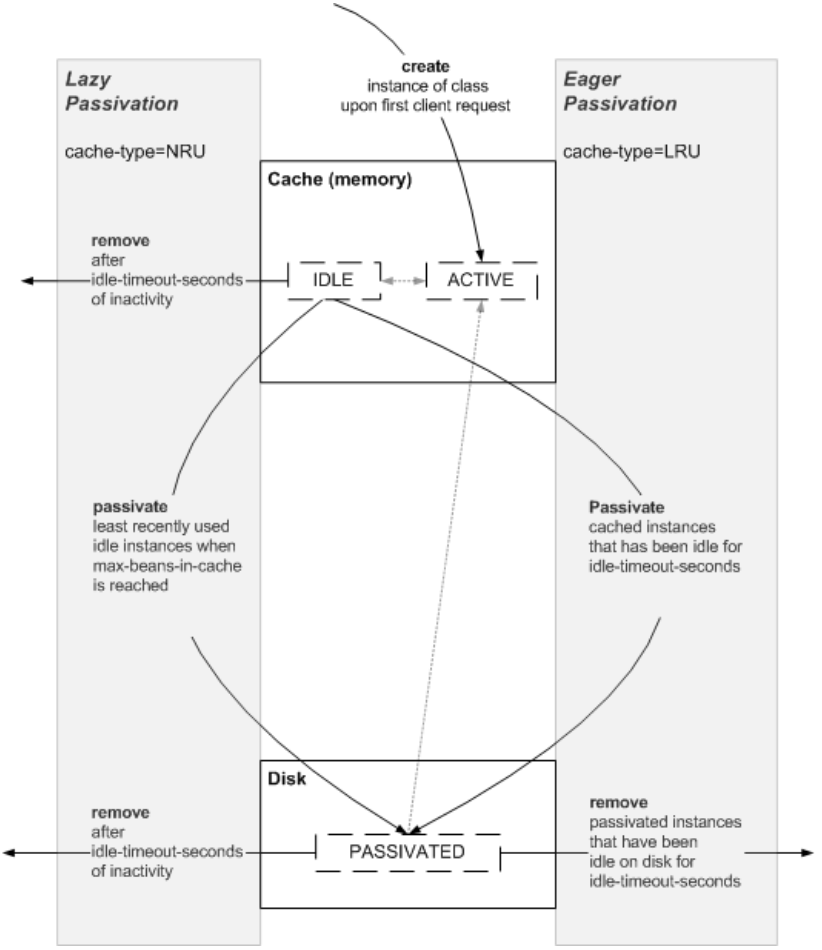
- No instance is available in the pool, but the number of instances in use is less then `max-beans-in-free-pool`. WebLogic Server allocates a new bean instance and gives it to you.
- No instances are available in the pool and the number of instances in use is already `max-beans-in-free-pool`. You application must wait until either your transaction times out or a bean instance that already exists in the pool becomes available.

Caching and Passivating Stateful Session EJBs

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 5-2 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 such that WebLogic Server can serialize its data and passivate the 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.

Controlling Passivation

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, or eager passivation.
- `NRU`—not recently used, or 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.

The container 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. Each server instance has its own directory for storing the state of passivated stateful session beans, 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 server instance directory, for example:

```
D:\releases\610\bea\user_domains\mydomain\myserver\pstore\
```

The path to the persistence store is:

`RootDirectory``\`*`ServerName`*`\`*`persistent-store-dir`*

where:

- *`RootDirectory`*—the directory where WebLogic Server runs, for example:

  ```
  D:\releases\810\bea\user_domains\mydomain
  ```

  `RootDirectory` can be specified at server startup with the `-Dweblogic.RootDirectory` property.

- *`ServerName`*—the name of the server instance.
- *`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, named with a hash code, for each passivated bean. For example, the subdirectory for a passivated bean in the example above might be:

```
D:\releases\810\bea\user_domains\mydomain\myserver\pstore\14t89gex0m2fr
```

Configuring 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.

Design Decisions for Session Beans

This section discusses some design decisions relevant to session beans.

Choosing Between Stateless and Stateful Beans

Stateless session beans are a good choice if your application does not need to maintain state for a particular client between business method calls. WebLogic Server is multi-threaded, servicing multiple clients simultaneously. With stateless session beans, the EJB container is free to use any available, pooled bean instance to service a client request, rather than reserving an instance for each client for the duration of a session. This results in greater resource utilization, scalability and throughput.

Stateless session beans are preferred for their light-weight implementation. They are a good choice if your application's beans perform autonomous, distinct tasks without bean-to-bean interaction.

Stateful session beans are a good choice if you need to preserve the bean's state for the duration of the session.

For examples of applications of stateless and stateful session beans, see "Stateless Session Beans" on page 2-2 and "Stateful Session Beans" on page 2-2.

Choosing the Optimal Free Pool Setting for Stateless Session Beans

When you choose values for `initial-beans-in-free-pool` and `max-beans-in-free-pool` you must weigh memory consumption against slowing down your application. If the number of stateless session bean instances is too high, the free pool contains inactive instances that consume memory. If the number is too low, a client may not obtain an instance when it needs it. This leads to client threads blocking until an instance frees up, slowing down the application.

Usually `max-beans-in-free-pool` should be equal to the number of worker threads in the server instance, so that when a thread tries to do work an instance is available.

Implementing Session Beans

Chapter 4, "Implementing Enterprise Java Beans," takes you through session bean implementation step-by-step. This section explains the details of configuring WebLogic-Server specific session bean behavior by setting bean-specific deployment descriptor elements.

WebLogic-Specific Configurable Behaviors for Session Beans

Table 5-2 summarizes the deployment descriptor elements you set to configure the behavior of a *stateless* session bean and how the bean behaves if you do not configure the element. All of the elements listed are sub-elements of the `stateless-session-descriptor` element in `weblogic-ejb-jar.xml`.

Table 5-3 summarizes the deployment descriptor elements you set to configure the behavior of a *stateful* session bean and how the bean behaves if you do not configure the element. All of the elements listed are sub-elements of the `stateful-session-descriptor` element in `weblogic-ejb-jar.xml`.

Table 5-2 WebLogic-Specific Features for Stateless Session EJBs

To control	Set the following weblogic-ejb-jar.xml element(s)	Default behavior
The number of inactive instances of a stateless session bean that exist in WebLogic Server when it is started. See "Pooling for Stateless Session EJBs" on page 5-2.	`initial-beans-in-free-pool`	WebLogic Server creates 0 beans in the free pool.
The maximum size of the free pool of inactive stateless session beans.	`max-beans-in-free-pool`	WebLogic Server limits the maximum number of beans in the free pool to 1000.

How WebLogic Server replicates stateless session EJB instances in a cluster.
See "Reliability and Availability Features" on page 2-13.
	`stateless-clustering`
	`home-is-clusterable`
	`home-load-algorithm`
	`home-call-router-class-name`
	`stateless-bean-is-clusterable`
	`stateless-bean-load-algorithm`
	`stateless-bean-call-router-class-name`
	`stateless-bean-methods-are-idempotent`
		The EJB can be deployed to multiple servers in a cluster.

Table 5-3 WebLogic-Specific Features for Stateful Session EJBs

Behavior	weblogic-ejb-jar.xml element	Default
Whether multiple clients can simultaneously access a bean without triggering a `Remote Exception`. See "Configuring Concurrent Access to Stateful Session Beans" on page 5-8.	`allow-concurrent-calls`	False—The server throws a `RemoteException` when a stateful session bean instance is currently handling a method call and another (concurrent) method call arrives on the server.
Whether the EJB container can remove a stateful session bean within a transaction context without provoking an error.	`allow-remove-during-transaction`	False—The server throws an exception when a stateful session bean is removed within a transaction context.
The number of stateful been instances that can exist in cache.	`max-beans-in-cache`	1000
The period of inactivity before a stateful session bean instance remains in cache (given that `max-beans-in-cache` has not been reached), and after passivation, remains on disk.	`idle-timeout-seconds`	600 seconds
The rules for removing a stateful session bean instance from the cache.	`cache-type`	NRU (not recently used)—For a description of this behavior, see "Lazy Passivation (NRU)" on page 5-6.
Where WebLogic Server stores the state of passivated stateful session bean instances.	`persistent-store-dir`	
To support method failover, specify the idempotent methods for a clustered EJB. An idempotent method can be repeated with no negative side-effects.	`idempotent-methods`	None
Custom class to be used for routing home method calls.	`home-call-router-class-name`	None

Behavior	weblogic-ejb-jar.xml element	Default
Indicates if the bean home can be clustered.	`home-is-clusterable`	True
Algorithm to use for load-balancing among replicas of the bean home.	`home-load-algorithm`	Algorithm specified by the property `weblogic.cluster.defaultLoadAlgorithm`
Indicates the replication used for stateful session beans in a cluster: in-memory or none.	`replication-type`	None

CHAPTER 6

Entity EJBs

The sections that follow describe WebLogic Server value-added features for programming and using entity beans in applications, and provide associated design and development guidelines.

It is assumed that the reader is familiar with Java programming and entity bean features and capabilities. For an introduction to entity beans and how they are typically used in applications, see "Entity EJBs Maintain Persistent Data" on page 2-2 and "Entity Bean Features" on page 3-3.

For a description of the overall bean development process, see "Implementing Enterprise Java Beans" on page 4-1.

- "Managing Entity Bean Pooling and Caching" on page 6-1
- "Using Primary Keys" on page 6-7
- "Configuring Entity EJBs for Database Operations" on page 6-12
- "Using Container-Managed Relationships (CMRs)" on page 6-23
- "Choosing a Concurrency Strategy" on page 6-35
- "CMP Entity Bean Descriptors Element by Feature" on page 6-44

Managing Entity Bean Pooling and Caching

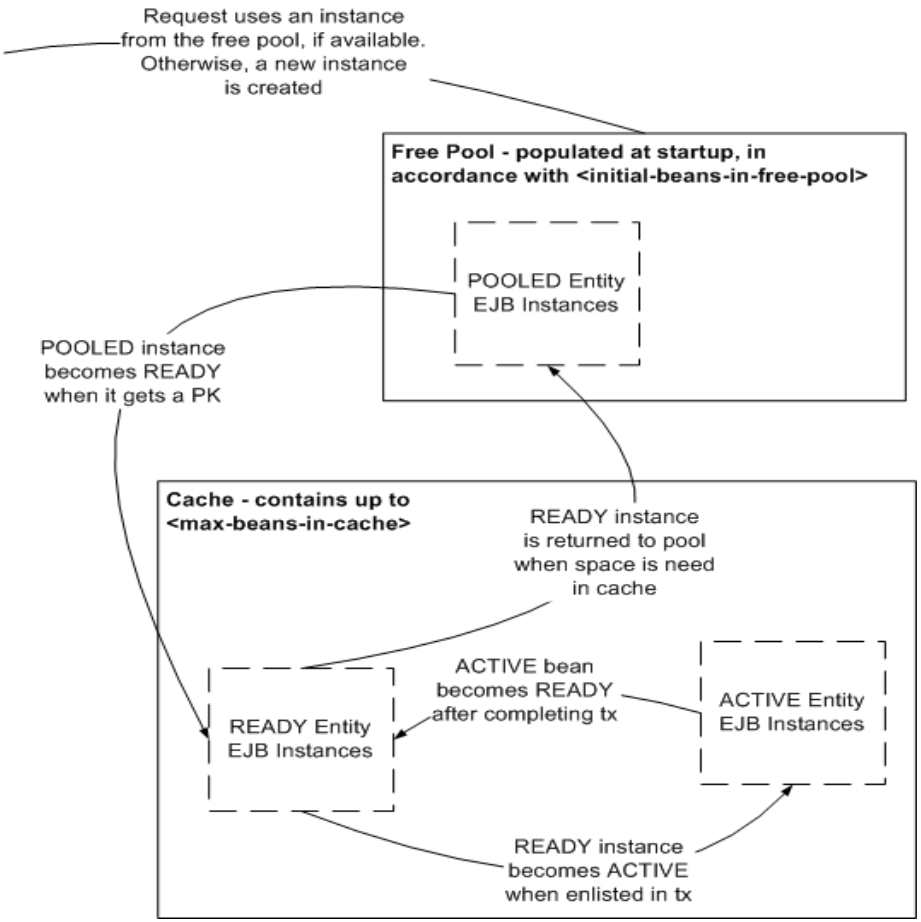
WebLogic Server provides these features to improve performance and throughput for entity EJBs:

- 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).

Figure 6-1 illustrates the lifecycle of an entity bean instance. The sections that follow describe and how the container populates and manages the free pool and the cache.

Figure 6-1 Entity Bean Lifecycle



Understanding Entity Pooling

If you specify a non-zero value for the `initial-beans-in-free-pool` element in `weblogic-ejb-jar.xml`, 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 be 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 `max-beans-in-free-pool` element, in `weblogic-ejb-jar.xml` which set to 1,000 by default.

Understanding Entity Caching

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 displays the count of READY and ACTIVE beans.

The effect of the `max-beans-in-cache` element, and the quantity of instances with the same primary key allowed in the cache vary by concurrency strategy. Table 6-1 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, and how many entity bean instances with the same primary key are allowed in the cache.

Table 6-1 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

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.

Understanding ejbLoad() and ejbStore() Behavior

This section describes how and when the persistent data for a CMP 2.0 entity bean is loaded to cache and written back to persistent storage.

- `findXXX()`—By default, calling a finder method on a CMP bean results in immediate load of the bean's persistent data to cache. This behavior is controlled by `finders-load-bean` element in the `persistence` stanza of `weblogic-ejb-jar.xml`.
- `ejbLoad()`—For CMP 2.0 entity beans, `ejbLoad()` causes a "lazy" load of a bean's persistent data to the entity cache when the next `getXXX()` for the bean's data is called. That is, when a transaction is initiated for a CMP 2.0 entity bean instance, WebLogic

Server reads the bean's data from the entity cache, rather than the database, unless `ejbLoad()` has been called since the bean was last loaded to cache.

By default, WebLogic Server calls `ejbLoad()` each time a new transaction is initiated for the entity bean.

Note: When used with CMP 1.1 entity beans and entity beans that use bean-managed persistence, `ejbLoad()` does not perform the a lazy load—for these bean types, the bean's persistent data is loaded to cache *during* the `ejbLoad()`

- `ejbStore()`—WebLogic Server writes the persistent fields of an entity EJB to the database using calls to `ejbStore()`.

  By default, WebLogic Server calls `ejbStore()` when the transaction commits.

Controlling the Behavior of ejbLoad() and ejbStore()

For applications in which multiple clients can currently access and modify a bean's underlying data, the default behavior of `ejbLoad()` and `ejbStore()` described in "Understanding ejbLoad() and ejbStore() Behavior" ensures database integrity by:

- Ensuring that each new transaction uses the latest version of the EJB's persistent data, and
- Updating the database upon transaction commitment.

However, depending on your requirements, you may prefer to call `ejbLoad()` and `ejbStore()` either more or less frequently.

For instance, you might want to limit calls that access the database for performance reasons. If you application does not allow multiple transactions to concurrently access the EJB—for example, if the bean uses `Exclusive` concurrency—loading the data at the beginning of each transaction is unnecessary. Given that no other clients or systems update the EJB's underlying data, the cached EJB data is always up-to-date, and calling `ejbLoad()` results in extra overhead. In such cases, you can safely reduce calls to `ejbLoad()`, as described in "Limiting Database Reads with cache-between-transactions (Long-Term Caching)".

Alternatively, you might want to deviate from the standard `ejbStore()` behavior, by calling it *before* a transaction commits, in order to access and use intermediate transaction results. For instructions, see "Updating the Database Before Transaction Ends" on page 6-18.

Disabling Cache Flushing

According to the EJB specification, updates made by a transaction must be reflected in the results of query-finders and `ejbSelect`s issued during the transaction. This requirement can slow

performance. If you prefer not to flush the cache before the query is executed, you can use change the value of the `include-updates` element in `weblogic-cmp-jar.xml` from its default value of `True` to `False`.

The decision to disable cache flushing depends on whether performance is more important than seeing the most current data. Setting `include-updates` to `False` provides the best performance but updates of the current transaction are not reflected in the query. If `include-updates` is `True` the container flushes all changes for the transactions to the database before executing the new query.

You can safely turn cache flushing off if your transactions do not re-query modified data—a common scenario—and get the best performance.

Note: In releases prior to WebLogic Server 8.1, the default for `include-updates` was `False`. For purposes of J2EE compliance, the default has been changed to `True` in WebLogic Server 8.1.

Configuring Application-Level Caching

Application-level caching—also known as "combined caching"—allows multiple entity beans that are part of the same J2EE enterprise application to share a single runtime cache. There are no restrictions on the number of different entity beans that may reference an individual cache.

Application-level caching offers the following advantages:

- Reduces the number of entity bean caches, and hence the effort to configure the cache.
- Better utilization of memory and heap space, because of reduced fragmentation. For example, if a particular EJB home experiences a burst of activity, it can make use of all memory available to the combined cache, while other EJBs that use the cache are paged out. If two EJBs use different caches, when one bean's cache becomes full, the container cannot page out EJBs in the other bean's cache, resulting in wasted memory
- Simplifies management; combined caching enables a system administrator to tune a single cache, instead of many caches.
- Provides better scalability

Application-level caching is *not* the best choice, however, for applications that experience high throughput. Because one thread of control exists per cache at a time, high throughput can create a bottleneck situation as tasks compete for control of the thread.

To configure an application-level cache:

1. Verify that the `weblogic-application.xml` file is located in the META-INF directory of the EAR file.
2. Define the application-level cache in the `entity-cache` stanza of `weblogic-application.xml`. For a definition of this stanza and the elements it contains, see "`entity-cache`" in *Developing WebLogic Server Applications.*
3. Reference the application-level cache in the `entity-cache-ref` element of the `entity-descriptor` stanza in `weblogic-ejb-jar.xml`.

   Note that:

   – `entity-cache-name` should be the name of the application-level cache, as specified in `weblogic-application.xml`.

   – The `concurrency-strategy` you specify for the bean must be compatible with the `caching-strategy` specified in `weblogic-application.xml`. A read-only entity can only use a Multiversion application-level cache. For more information, see "`caching-strategy`" in *Developing WebLogic Server Applications.*

The `weblogic-application.xml` deployment descriptor is documented in the "Application.xml Deployment Descriptor Elements" section of *Developing WebLogic Server Applications*.

Using Primary Keys

Every entity EJB must have a primary key that uniquely identifies an entity bean within its home. Each entity bean instance may define a different class for its primary key; multiple entity beans can use the same primary key class, as appropriate.

If two entity bean instances 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 bean instance's remote interface to determine the instance's identity within its home.

The instance identity associated with 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.

Specifying Primary Keys and Primary Key Classes

You can map a primary key to one or multiple fields:

- Mapping a Primary Key to a Single CMP Field

  In the entity bean class, you can have a primary key that maps to a single CMP field. CMP fields must be specified in both `ejb-jar.xml` and `weblogic-cmp-jar.xml`. In both descriptor files, CMP fields are specified in the `cmp-field` element. For simple primary keys, also specify the primary key in the `primkey-field` element in the `ejb-jar.xml`. In addition, specify the primary key field's class in the `prim-key-class` element in `ejb-jar.xml`.

- Wrapping One or More CMP Fields in a Primary Key Class

  You can define your own primary key class that maps to single or multiple CMP fields. The primary key class must be `public`, and have a `public` constructor with no parameters. Specify the name of the primary key class in the `prim-key-class` element in `ejb-jar.xml`. All fields in the primary key class must be `public`, and must have the same names as the corresponding `cmp-field`s in `ejb-jar.xml` and `weblogic-ejb-jar.xml`. For compound primary keys, which map to multiple CMP fields, do not specify `primkey-field` in `ejb-jar.xml`.

- Anonymous Primary Key Class

  If your entity EJB uses an anonymous primary key class, you must subclass the EJB and add a `cmp-field` of type `java.lang.Integer` to the subclass. Enable automatic primary key generation for the field so that the container fills in field values automatically, and map the field to a database column in the `weblogic-cmp-jar.xml` deployment descriptor.

  Finally, update the `ejb-jar.xml` file to specify the EJB subclass, rather than the original EJB class, and deploy the bean to WebLogic Server.

  Note: If you use the original EJB (instead of the subclass) with an anonymous primary key class, WebLogic Server displays the following error message during deployment:

  ```
  In EJB ejb_name, an 'Unknown Primary Key Class' ( <prim-key-class>
  == java.lang.Object ) MUST be specified at Deployment time (as
  something other than java.lang.Object).
  ```

Guidelines for Primary Keys

Follow these suggestions when using primary keys with WebLogic Server:

- Do not construct a new primary key class with an `ejbCreate`. Instead, allow the container to create the primary key class internally, as described in "Automatically Generating Primary Keys" on page 6-9.

- In an application that manages its own primary key values, for a simple primary key—one composed of a single atomic value such as a `String` or an `Integer`—make the primary

key class a container-managed field. Set the value of the primary key `cmp-field` using the `setXXX` method 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)`.

- If you are mapping a database column to a `cmp-field` and a `cmr-field` concurrently and the `cmp-field` is a primary key field, set the value when the `ejbCreate()` method is invoked by using the `setXXX` method for the `cmp-field`. In the case, the `cmr-field` is initialized automatically, and the `setXXX` method for the `cmr-field` cannot be used. Conversely, if the `cmp-field` is *not* a primary key field, the `cmp-field` is read-only. The column is updated using the `cmr-field`, and the `cmp-field` provides a read-only view of the foreign key.

Automatically Generating Primary Keys

WebLogic Server supports automatic primary key generation feature for CMP entity beans. This feature is supported for simple (non-compound) primary keys *only*.

WebLogic Server supports two methods of automatic primary key generation:

- Native database primary key generation—The database generates the primary key. To enable this feature, specify the database and a generator name in the `<automatic-key-generation>` stanza of `weblogic-cmp-jar.xml`. Based on the values you configure, the container generates code that obtains the primary key from the database. This feature is supported for Oracle and Microsoft SQL Server databases only. In addition, see the instructions in "Declaring Primary Key Field Type" on page 6-12.

- Primary keys generated from a SEQUENCE table.

Whichever method of generated primary keys you use, see the instructions in "Declaring Primary Key Field Type" on page 6-12.

Specifying Automatic Key Generation for Oracle

Generated primary key support for Oracle databases uses a SEQUENCE entity in the Oracle database to generate unique primary keys. The Oracle SEQUENCE is called when a new number is needed. Specify automatic key generation in the automatic-key-generation element in weblogic-cmp-jar.xml. Specify the name of the Oracle SEQUENCE in the generator-name element. If the Oracle SEQUENCE was created with a SEQUENCE INCREMENT, specify a key-cache-size. The value of key-cache-size must match the value of the Oracle SEQUENCE INCREMENT. If these two values are different, duplicate keys can result.

When using the Oracle SEQUENCE object for generating primary keys:

- Do not set the generator-type to USER_DESIGNATED_TABLE with Oracle. Doing so sets the TX ISOLATION LEVEL to SERIALIZABLE, which can cause the following exception:

  ```
  javax.ejb.EJBException: nested exception is: java.sql.SQLException:
  Automatic Key Generation Error: attempted to UPDATE or QUERY NAMED
  SEQUENCE TABLE NamedSequenceTable, but encountered SQLException
  java.sql.SQLException: ORA-08177: can't serialize access for this
  transaction.
  ```

  Instead, use the AutoKey option with Oracle.

- WebLogic Server 8.1 does not support Oracle's synonym feature for the SEQUENCE schema object. If you migrate an application that uses synonyms for SEQUENCEs from a previous version of WebLogic Server to WebLogic Server 8.1, the following errors result:

  ```
  [EJB:011066]During EJB deployment, error(s) were encountered while
  setting up The ORACLE SEQUENCE named 'XXX' with INCREMENT value '1'

  [EJB:011064]The ORACLE SEQUENCE named 'XXX' with INCREMENT '1' was not
  found in the database'
  ```

Specifying Automatic Key Generation 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 SQL Server table that contains an IDENTITY column, see Microsoft documentation.

Once the IDENTITY column is created in the table, specify automatic key generation in weblogic-cmp-jar.xml as shown below.

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

Generating Primary Keys with a Named Sequence Table

A sequence table is a database-neutral way to generate primary keys. The sequence table holds a monotonically increasing integer sequence value that is used as the primary key value in bean instances as they are created.

Create a table named `SEQUENCE` to hold 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)
```

To use this feature, make sure that the underlying database supports a transaction isolation level of `Serializable`. The `Serializable` value indicates that simultaneously executing a transaction multiple times has the same effect as executing the transaction multiple times in a serial fashion. This is important in a WebLogic Server cluster, in which multiple servers instances access the sequence table concurrently. See your database documentation to determine the isolation levels it supports.

Specify automatic key generation in the `weblogic-cmp-jar.xml` file, as shown below. In addition, see the instructions in "Declaring Primary Key Field Type" on page 6-12.

```
<automatic-key-generation>
     <generator-type>NamedSequenceTable</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 in the `generator-name` element.

Specify the size of the key cache— how many keys the container will fetch in a single DBMS call—in the `key-cache-size` element. BEA recommends a `key-cache-size` greater than one. This setting reduces the number of calls to the database to fetch the next key value.

BEA recommends 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.

Declaring Primary Key Field Type

For both native DBMS primary key generation, or key generation using a named sequence table, in the abstract `get` and `set` methods of the associated entity bean, declare the primary field type to be either:

- `java.lang.Integer`
- `java.lang.Long`

In `weblogic-cmp-jar.xml`, set the `key-cache-size` element to specify how many primary key values in the sequence should be fetched from the database at a time. For example, setting `key_cache_size` to 10 results in one database access for every 10 beans created, to update the sequence. The default value of `key_cache_size` is `1`. BEA recommends that you set `key_cache_size` to a value greater than one, to minimize database accesses and to improve performance.

Support for Oracle SEQUENCE

WebLogic Server can automatically create an Oracle `SEQUENCE`—a number generator that generates a unique integer each time it is called.

An Oracle `SEQUENCE` can use a specified "increment value", which is the value by which the integer is incremented on each subsequent generation. For example, if a `SEQUENCE` generates the integer 24 and the increment value is 10, then the next integer the `SEQUENCE` generates will be 34.

Configuring Entity EJBs for Database Operations

The following sections provide instructions for mapping entity EJBs to database tables and controlling database access behaviors.

Configuring Table Mapping

A CMP bean can be mapped to one or more database tables. When a CMP bean is mapped to multiple tables, each table contains a row that corresponds to a particular bean instance. So, each table to which a bean maps will have the same number of rows at any point in time, and contain the same set of homogeneous primary key values. Consequently, each table must have the same number of primary key columns, and corresponding primary key columns in different tables must have the same type, though they may have different names. Tables that map to the same bean must not have referential integrity constraints declared between their primary keys. If they do, removal of a bean instance can result in a runtime error.

You map the cmp-fields of a bean to the columns of a table using the `table-map` element in `weblogic-cmp-jar.xml`, specifying one `table-map` stanza for each database table to which the bean maps. Each `table-map` element maps the primary key column(s) of the table to the primary key field(s) of the bean. Non-primary key fields may only be mapped to a single table.

Listing 6-1 and Listing 6-2 contain `table-map` stanzas for a bean that maps to a single and a bean that maps to multiple tables, respectively.

Listing 6-1 Mapping a CMP Entity to One Database Table

```
<table-map>
     <table-name>TableName</table-name>
     <field-map>
          <cmp-field>name</cmp-field>
          <dbms-column>name_in_tablename</dbms-column>
     </field-map>
     <field-map>
          <cmp-field>street_address</cmp-field>
          <dbms-column>street_address_in_tablename
          </dbms_column>
     </field-map>
     <field-map>
          <cmp-field>phone</cmp-field>
          <dbms-column>phone_in_tablename</dbms-column>
     </field-map>
```

Listing 6-2 Mapping a CMP Entity to Two DBMS Tables

```
<table-map>
     <table-name>TableName_1</table-name>
          <field-map>
          <!--Note 'name'is the primary key field of this EJB -->
               <cmp-field>name</cmp-field>
               <dbms-column>name_in_tablename_1</dbms-column>
          </field-map>
          <field-map>
               <cmp-field>street_address</cmp-field>
```

```
                <dbms-column>street_address_in_tablename_1</dbms-column>
            </field-map>
    </table-map>
    <table-map>
            <table-name>TableName_2</table-name>
            <field-map>
                <!--Note 'name'is the primary key field of this EJB -->
                <cmp-field>name</cmp-field>
                <dbms-column>name_in_tablename_2</dbms-column>
                </field-map>
                <field-map>
                    <cmp-field>phone</cmp-field>
                    <dbms-column>phone_in_tablename_2</dbms-column>
                </field-map>
    </table-map>
```

Automatic Table Creation (Development Only)

To make iterative development easier, the WebLogic Server EJB container can be configured to automatically change the underlying table schema as entity beans change, ensuring that tables always reflect the most recent object relationship mapping.

Note: This feature is disabled when a server instance is running in production mode, as a production environment may require the use of more precise table schema definitions, for To ensure that the container only changes tables it created, container-created tables include an extra column, called `wls_temp`.

The syntax of table creation statements (DDL) varies from database to database, so table creation may fail on databases that are not fully supported. If this occurs, create the tables manually.

Table 6-2 Controlling Automatic Table Creation Behavior with <create-default-dbms-tables>.

Setting <create-default-dbms-tables> to this value	Results in this behavior:
`Disabled`	The EJB container takes no action when underlying table schema changes. This is the default value.
`CreateOnly`	For each CMP bean in the `JAR`, if there is no table in the database for the bean, the container attempts to create the table based on information found in the deployment descriptor, including join tables, sequence tables, and Oracle sequences. If table creation fails, a `Table Not Found` error is thrown, and the user must create the table manually.
`DropAndCreate`	For each CMP bean in the `JAR`: • if table columns have not changed, no action is taken, and the existing data is preserved. • if the columns have changed, then the container drops and recreates the table and all table data is lost.
`DropAndCreate Always`	For each CMP bean listed in the `JAR`, the container drops and creates the table whether or not columns have changed.
`AlterOrCreate`	For each CMP bean in the `JAR`: • if the table exists, the container attempts to alter the table schema using the `ALTER TABLE` SQL command and the container saves the data. • if the table does not exist, the container creates the table during deployment. **Note:** Do not use `AlterOrCreate` if a new column is specified as a primary key or a column with null values is specified as the new primary key column. Due to database limitations, use `DropAndCreate` instead.

Note: Sequence tables, join tables, and Oracle SEQUENCEs are supported.

Enable this feature using the `create-default-dbms-tables` element in `weblogic-cmp-jar.xml`. The behavior of this feature varies according to the value of the element, as described in the following table. The EJB container actions described in the table occur during deployment.

Delaying Database Inserts

Because of timing issues that may occur when creating a CMP bean, WebLogic Server enables you to specify at what point during the bean creation process the bean is inserted into the database.

Why Delay Database Inserts?

You cannot set a `cmr-field` until the primary key value for the bean—which is set when `ejbPostCreate` is called—is available. Hence, you cannot set a `cmr-field` until `ejbPostCreate`. This factor in combination with other conditions can lead to these problems:

- Problem 1: Non-null foreign key constraint

  If a database row is inserted after `EJBCreate` (that is, before `ejbPostCreate`), then the foreign key is given a null value. This is because foreign key columns are set when `cmr-field`s are set.

  Solution: Set `delay-insert-until` in `weblogic-cmp-jar.xml` to `ejbCreate`, which causes the insert to be done immediately after the `ejbCreate`, and set the relationship during `ejbPostCreate`.

- Problem 2: Creating beans during `ejbPostCreate`

  When a related bean is created, the database insert for that bean happens before the create call finishes. If the database row for the related bean contains a foreign key that refers to the current bean and the foreign key has a referential integrity constraint defined, the insert will fail if the current bean's database row has not been inserted yet.

  Solution: Set `delay-insert-until` to `ejbCreate` so that the row for the current bean exists during the `ejbPostCreate`.

  Note: In a one-to-one relationship, if the parent bean's primary key is embedded in the child bean's CMR field, when the EJB container creates the beans, it will not check if the parent bean has children, for performance reasons. To avoid a `duplicationKeyException` database exception, you must set the foreign key constraint on the child table in the database.

- Problem 3: Both Problem 1 and Problem 2 exist

  Solution: Do not create related beans during `ejbPostCreate`. Create the current bean and after the creation is finished, create the related beans and set up relationships.

  BEA recommends that applications always do this. Applications should never create related beans during creation of another bean.

Configuring Delayed Database Inserts

You can delay database inserts until the end of the `EJBCreate` method or `EJBPostCreate` method, using the `delay-database-insert-until` element in `weblogic-cmp-jar.xml`. To batch, order, and perform updates at the end of the transaction, set both `enable-batch-operations` and `order-database-operations` in `weblogic-cmp-jar.xml` to "`y`".

If you choose to delay database updates for a transaction that updates related beans, you must define the relationship between the beans in the `weblogic-rdbms-relation` of `weblogic-cmp-jar.xml`. Otherwise, database constraint errors may result when the EJB container attempts to perform the updates.

Limiting Database Reads with cache-between-transactions (Long-Term Caching)

As described in "Understanding ejbLoad() and ejbStore() Behavior", by default, WebLogic Server calls `ejbLoad()` each time a transaction is initiated for an entity bean.

If you wish, you can configure WebLogic Server to call `ejbLoad()` only when a client first references the bean or when a transaction is rolled back. This behavior is referred to as *long-term caching*. You enable long-term caching by setting the `cache-between-transactions` element in `weblogic-ejb-jar.xml` to `true`.

Long-term caching is allowed only if the `concurrency-strategy` for a bean is `Exclusive`, `ReadOnly`, or `Optimistic`. When long-term caching is configured for a:

- `ReadOnly` bean, WebLogic Server ignores the value of the `cache-between-transactions`. WebLogic Server always performs long-term caching of read-only data, regardless of the value of `cache-between-transactions`.
- `Exclusive` bean, the EJB container must have exclusive update access to the underlying data: the data must not be updated by another application outside of the EJB container.

  Note: If a bean with `Exclusive` concurrency is deployed in a cluster long-term caching is automatically disabled because any server instance in the cluster may update the bean data. This would make caching between transactions impossible.

- `Optimistic` bean, the EJB container reuses cached data for each transaction after the client first references the bean, but ensures that updates are transactionally consistent by checking for optimistic conflicts at the end of each transaction.

Note: In a cluster, when a bean with Optimistic concurrency is updated, notifications are broadcast to other cluster members to prevent optimistic conflicts.

Table 6-3 lists the allowable values for the `cache-between-transactions` element by entity bean type and concurrency strategy.

Table 6-3 Permitted cache-between-transactions values, by Concurrency Strategy and Entity Type

Concurrency Strategy	BMP Entity	CMP 2.0 Bean	CMP 1.1 Entity
`Database`	`False`	`False`	`False`
`Exclusive`	`True` or `False`	`True` or `False`	`True` or `False`
`Optimistic`	Not applicable. `Optimistic` concurrency is not available for BMP beans.	`True` or `False`	Not applicable. `Optimistic` concurrency is not available for CMP 1.1 beans.

Updating the Database Before Transaction Ends

As described in "Understanding ejbLoad() and ejbStore() Behavior", by default, WebLogic Server calls `ejbStore()` only when the transaction commits.

To make intermediate results of uncommitted transactions available to other database users, set `delay-updates-until-end-of-tx` in the `persistence` element of `weblogic-ejb-jar.xml` to `False`—this causes WebLogic Server to call `ejbStore()` after each method call.

Note: While setting `delay-updates-until-end-of-tx` to `false` results in database updates after each method call, the updates are not committed until the end of the transaction.

Dynamic Queries

Dynamic queries allow you to construct and execute EJB-QL queries programmatically in your application code.

Using dynamic queries provides the following benefits:

- The ability to create and execute new queries without having to update and deploy an EJB.
- The ability to reduce the size of the EJB's deployment descriptor file. This is because finder queries can be dynamically created instead of statically defined in the deployment descriptors.

Enabling Dynamic Queries

To enable dynamic queries:

1. Set the `enable-dynamic-queries` element in the EJB's `weblogic-ejb-jar.xml` to `True`:

   ```
   <enable-dynamic-queries>True</enable-dynamic-queries>
   ```

2. Set standard method permissions to control access to dynamic queries by specifying the `method-permission` element in the `ejb-jar.xml` deployment descriptor file.

   Setting `method-permission` for the `createQuery()` method of the `weblogic.ejb.QueryHome` interface controls access to the `weblogic.ejb.Query` object necessary to executes the dynamic queries.

   If you specify `method-permission` for the `createQuery()` method, the `method-permission` settings apply to the execute and find methods of the `Query` class.

Executing Dynamic Queries

The following code sample demonstrates how to execute a dynamic query.

```
InitialContext ic=new InitialContext();
FooHome fh=(FooHome)ic.lookup("fooHome");
QueryHome qh=(QueryHome)fh;
String ejbql="SELECT OBJECT(e)FROM EmployeeBean e WHERE e.name='rob'"
Query query=qh.createQuery();
query.setMaxElements(10)
Collection results=query.find(ejbql);
```

Enabling BLOB and CLOB Column Support for Oracle

WebLogic Server supports Oracle Binary Large Object (BLOB) and Character Large Object (CLOB) DBMS columns for CMP entity beans.

WebLogic Server maps BLOBs to a `cmp-field` that has a byte array or serializable type. 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-jar.xml` file.
3. Create the BLOB or CLOB in the Oracle database.

Specifying a BLOB Column Using the Deployment Descriptor

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

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

Controlling Serialization for cmp-fields of Type byte[] Mapped to a Oracle Blob

Prior to WebLogic Server 8.1 SP02, by default, `cmp-fields` of type byte[] mapped to an OracleBlob were serialized. In WebLogic Server 8.1 SP02, by default, they are not.

To cause WebLogic Server to serialize cmp-fields of type byte[] mapped to an OracleBlob, set the `serialize-byte-array-to-oracle-blob` compatibility flag, which was introduced in WebLogic Server 8.1 SP02.

By default, the value of `serialize-byte-array-to-oracle-blob` is `false`—the container will persist the byte[] directly and not serialize it.

Specifying a CLOB Column Using the Deployment Descriptor

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

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

Optimized CLOB Column Insertion on Oracle 10g

The Oracle 9i and 10g drivers have different requirements for successful insertion of CLOB column values into database rows. The Oracle 9i driver requires that a database row is locked before a CLOB value can be inserted into it. As a result, on Oracle 9i, WebLogic Server does the following to insert a row that contains a CLOB column value into a table:

1. Inserts a row with all values other than the CLOB column into the table.

2. Issues a `SELECT FOR UPDATE` statement on the row created in step 1, above.
3. Inserts the CLOB value into the row.

While these steps are necessary for successful insertion of a row that contains a CLOB column value on Oracle 9i, the steps cause an unnecessary performance hit on Oracle 10g. The Oracle 10g driver features improved handling of CLOBS and does not require a lock on a row before a CLOB column value can be inserted into it. On Oracle 10g, WebLogic Server uses a single INSERT statement to insert a row with a CLOB column value into a table, which results in increased performance of CMP EJBs.

To make use of this WebLogic Server optimization for Oracle 10g, you do not need to configure anything additional. Simply specify Oracle as your database and WebLogic Server checks to see if your Oracle version is Oracle 9i or Oracle 10g. If WebLogic Server identifies your database as Oracle 10g, rows containing CLOB values are inserted into tables in single `INSERT` statements. If WebLogic Server identifies your database as Oracle 9i, rows containing CLOB values are inserted into tables in three steps as previously described.

For more information, see Handling CLOBs - Made easy with Oracle JDBC 10g and Oracle product documentation.

Specifying Field Groups

A field group represents a subset of a bean's container-managed persistence (CMP) and container-managed relationship (CMR) fields. To improve performance, 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 bean's table.

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

You specify field groups in the `field-group` element of `weblogic-cmp-jar.xml` file, as shown in Listing 6-3.

Listing 6-3 Specifying Field Groups

```
<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>
```

Ordering and Batching Operations

Multiple instances of the same container-managed persistence (CMP) entity bean are often changed in a single transaction. If the EJB container issues SQL for every CMP entity bean instance, updates can become a performance bottleneck.

The EJB batch operations features solves this problem by updating multiple entries in a database table in one batch of SQL statements. This can dramatically improve performance by doing multiple database inserts, deletes, or updates for CMP beans in one database round-trip.

To permit batch database inserts, updates or deletes, set the `enable-batch-operations` element in the `weblogic-cmp-jar.xml` file to `True`.

Operation Ordering

Database operation ordering prevents constraint errors by taking into account database dependencies, and ordering inserts, updates and deletes accordingly.

Enabling database ordering causes the EJB container to do two things:

- Delay all database operations—insert, update, and delete—to commit time
- Order database operations at commit time

For example, assume a Customer A, who is related to Salesman A. If Salesman A is deleted, and Customer A is assigned to Salesman B, the container would order the operations in this fashion:

1. Update Customer A to refer to Salesman B.
2. Remove Salesman A.

This ensures that Customer A never refers to a salesman that does not exist, which would cause a database referential integrity error.

To enable the EJB container to correctly order database operations for related beans, you must specify the relationship between the beans, in the `weblogic-rdbms-relation` of `weblogic-cmp-jar.xml`. Otherwise, database constraint errors may result when the EJB container attempts to perform the updates.

Batch Operations Guidelines and Limitations

When using batch operations, you must set the boundary for the transaction, as batch operations only apply to the inserts, updates or deletes between `transaction begin` and `transaction commit`.

Batch operations only work with drivers that support the `addBatch()` and `executeBatch()` methods. If the EJB container detects unsupported drivers, it reports that batch operations are not supported and disables batch operations.

There are several limitations on using batch operations:

- You cannot use the "Automatically Generating Primary Keys" on page 6-9 feature with batch operations if the `generator-type` is set to `sybase` or `sqlServer`. If you have enabled automatic primary key generation with either of these types, WebLogic Server automatically disables batch operations and issues a warning.
- The total number of entries created in a single batch operation cannot exceed the `max-beans-in-cache` setting, which is specified in the `weblogic-ejb-jar.xml` file.
- If you set the `dbms-column-type` element in `weblogic-cmp-jar.xml` to either `OracleBlob` or `OracleClob`, batch operation automatically turns off because you will not save much time if a Blob or Clob column exist in the database table. In this case, WebLogic Server performs one insert per bean, which is the default behavior.

Using Container-Managed Relationships (CMRs)

Container-managed relationships (CMRs) are relationships that you define between two entity EJBs, analogous to the relationships between the tables in a database. If you define a CMR between two EJBs that are involved in the same processing task, your application can benefit from these features:

- Related beans can be cached together, reducing the number of queries necessary to accomplish a processing task.

- Batched database operations can be ordered correctly at the end of a transaction, avoiding data consistency problems.
- Related beans can be deleted automatically, using the cascade delete feature.

The sections that follow describe the features and limitations of WebLogic Server CMRs. For instruction on configuring CMRs, see "Defining Container-Managed Relationships (CMRs)" on page 6-25.

CMR Requirements and Limitations

You can define a relationship between two WebLogic Server entity beans that will be packaged in the same JAR and whose data persist in the same database. Entities that participate in the same relationship must map to the same datasource. WebLogic Servers does not support relationships between entity beans that are mapped to different datasources. The abstract schema for each bean that participates in a container-managed relationship must be defined in the same `ejb-jar.xml` file.

Note: EJB 2.1 states that if an entity bean does not have a local interface, the only CMR in which it can participate is a unidirectional one, from itself to another entity bean.

However, WebLogic Server allows an entity bean with only a remote interface to:

- Participate in CMRs that are bidirectional, or
- Be the target of a unidirectional CMR with another entity.

Because this feature is not specified in EJB 2.1, entity beans that have only remote interfaces, and either participate in bidirectional relationships or are the target of a unidirectional relationship, may not be portable to other application servers.

CMR Cardinality

An entity bean can have a one-to-one, one-to-many, or many-to-many relationship with another entity bean.

CMR Direction

Any CMR, whether one-to-one, one-to-many, or many-to-many, can be either unidirectional or bidirectional. The direction of a CMR determines whether the bean on one side of the relationship can be accessed by the bean on the other side.

Unidirectional CMRs can be navigated in one direction only—the "dependent" bean" is unaware of the other bean in the relationship. CMR-related features such as cascade deletes can only be applied to the dependent bean. For example, if cascade deletes have been configured for a unidirectional CMR from to `EJB1` to `EJB2`, deleting `EJB1` will cause deletion of `EJB2`, but deleting `EJB2` will not cause deletion of `EJB1`.

Note: For the cascade delete feature, the cardinality of the relationship is a factor—cascade deletes are not supported from the many side of a relationship, even if the relationship is bidirectional.

Bidirectional relationships can be navigated in both directions—each bean in the relationship is aware of the other. CMR-related features are supported in both directions. For example, if cascade deletes have been configured for a bidirectional CMR between `EJB1` to `EJB2`, deleting either bean in the CMR will cause deletion of the other bean.

Removing CMRs

When a bean instance that participates in a relationship is removed, the container automatically removes the relationship. For instance, given a relationship between an employee and a department, if the employee is removed, the container removes the relationship between the employee and the department as well.

Defining Container-Managed Relationships (CMRs)

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 6-25
- "Specifying Relationships in weblogic-cmp-jar.xml" on page 6-28

Specifying Relationship in ejb-jar.xml

Container-managed relationships are defined in the `ejb-relation` stanza of `ejb-jar.xml`. Listing 6-4 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.

Listing 6-4 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 Listing 6-5, 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 Listing 6-5, is one-to-one—the `multiplicity` is set to `one` in both role stanza for the relationship

Listing 6-5 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 Listing 6-4. It is not necessary to specify the `cmr-field-type` when the `cmr-field` is a single valued object.

Table 6-4 lists the contents of `cmr-field` for each bean in a relationship, based on the cardinality of the relationship.

Table 6-4 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 6-4.

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 6-5 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 contains the `relationship-role-map` stanza, which 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, `relationship-role-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 6-6 is the `weblogic-rdbms-relation` stanza for a the one-to-one relationship between `MentorEJB` and `StudentEJB`, whose `<ejb-relation>` is shown in Figure 6-5.

Listing 6-6 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
```

```
        </relationship-role-name>
           <relationship-role-map>
             <column-map>
                <foreign-key-column>student</foreign-key-column>
                <key-column>StudentID/key-column>
             </column-map>
           <relationship-role-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.

Listing 6-7 is the `weblogic-rdbms-relation` stanza for a the one-to-many relationship between `TeacherEJB` and `StudentEJB`, whose `<ejb-relation>` is shown in Figure 6-4.

Listing 6-7 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>
          <relationship-role-map>
            <column-map>
               <foreign-key-column>student</foreign-key-column>
               <key-column>StudentID/key-column>
            </column-map>
          <relationship-role-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.

Listing 6-8 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`. The example assumes that the employee bean is mapped to a single table. If employee bean is mapped to multiple tables, then the table containing the primary key column must be specified in the `relation-role-map`. For an example, see "Specifying CMRs for EJBs that Map to Multiple Tables" on page 6-31.

Listing 6-8 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>
    <relationship-role-map>
      <column-map>
        <foreign-key-column>first-friend-id</foreign-key-column>
        <key-column>id</key-column>
     </column-map
    </relationship-role-map>
  <weblogic-relationship-role>
    <weblogic-relationship-role>
      <relationship-role-name>second-friend</relationship-role-
      name>
    <relationship-role-map>
      <column-map>
        <foreign-key-column>second-friend-id</foreign-key-column>
        <key-column>id</key-column>
      </column-map>
    </relationship-role-map>
   </weblogic-relationship-role>
</weblogic-rdbms-relation>
```

Specifying CMRs for EJBs that Map to Multiple Tables

A CMP bean that is involved in a relationship may be mapped to multiple DBMS tables.

- If the bean on the foreign-key side of a one-to-one or one-to-many relationship is mapped to multiple tables then the name of the table containing the foreign-key columns must be specified using the `foreign-key-table` element.
- Conversely, if the bean on the primary-key side of a one-to-one or one-to-many relationship or a bean participating in a many-to-many relationship is mapped to multiple tables then the name of the table containing the primary-key must be specified using the `primary-key-table` element.

If neither of the beans in a relationship is mapped to multiple tables, then the `foreign-key-table` and `primary-key-table` elements may be omitted since the tables being used are implicit.

Listing 6-9 contains a `relationship-role-map` for a CMR in which the bean on the foreign-key side of a one-to-one relationship, `Fk_Bean`, is mapped to two tables: `Fk_BeanTable_1` and `Fk_BeanTable_2`.

The foreign key columns for the relationship, `Fk_column_1` and `Fk_column_2`, are located in `Fk_BeanTable_2`. The bean on the primary key side, `Pk_Bean`, is mapped to a single table with primary-key columns `Pk_table_pkColumn_1` and `Pk_table_pkColumn_2`.

The table that contains the foreign-key columns is specified by the `<foreign-key-table>` element.

Listing 6-9 One-to-One CMR, One Bean Maps to Multiple Tables

```
<relationship-role-map
   <foreign-key-table>Fk_BeanTable_2</foreign-key-table>
   <column-map>
      <foreign-key-column>Fk_column_1</foreign-key-column>
      <key-column>Pk_table_pkColumn_1</key-column>
   </column-map>
   <column-map>
      <foreign-key-column>Fk_column_2</foreign-key-column>
      <key-column>Pk_table_pkColumn_2</key-column>
   </column-map>
</relationship-role-map>
```

About CMR Fields and CMR Field Accessor Methods

CMR fields are not stored as attributes of an implementation class—they are accessible in the bean through CMR field accessor methods that you write. CMR field accessor methods:

- Must begin with `get...()` and `set...()` and the text following `get.../set...` must match the name of the relation field as it is declared in the `ejb-jar.xml`.
- Must exist in the implementation class for every CMR field.
- Must be declared abstract

To allow remote clients to use CMR field accessor methods, put the getter and/or setter method signatures in the remote interface. However, CMR fields of a Collection datatype cannot be exposed in the remote interface. A CMR field that is a Collection can only be accessed by local methods.

Using Cascade Delete for Entities in CMRs

When a *cascade delete* is performed, the deletion of a bean instance that participates in a relationship with another bean instance causes the container to also automatically delete all of the dependent bean instances. This feature is an efficient way to maintain data integrity.

For example, if the `Employee` bean has a one-to-many relationship to the `Employee_Projects` bean, deleting an `Employee` bean instance causes instances of the `Employee_Projects` bean to also be deleted.

You can specify cascade delete for one-to-one and one-to-many relationships; many-to-many relationships are not supported.

WebLogic Server supports two methods of cascade delete:

- J2EE Cascade Delete—This method does not require that the underlying database have built-in support for cascade deletes. Configure the behavior using the `cascade-delete` element in `ejb-jar.xml`, as shown in the following example:

```
<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: The `cascade-delete` element can only be specified for a `ejb-relationship-role` element if the other `ejb-relationship-role` in the relationship has `multiplicity` of `one`.

- Database-Level Cascade Delete (Oracle only)—This method allows an application to take advantage of a database's built-in support for cascade delete, and possibly improve performance. When `db-cascade-delete` is enabled, the underlying database must be set up for cascade deletes. For instructions and examples of setting up database cascade delete in WebLogic and Oracle, "db-cascade-delete" on page B-16.

In a high volume transaction environment, transactions that use exclusive concurrency can encounter deadlocks when a transaction that performs a cascade delete needs access to the same entity bean as a transaction that does not perform a cascade delete. For information on how to avoid such deadlocks, see "Preventing Deadlocks for Transactions That Use Exclusive Concurrency and Cascade Deletes" on page 6-41.

Relationship Caching

Relationship caching—also known as "pre-fetching" or "eager relationship caching'—improves the performance of entity beans by loading related beans into the cache and preventing multiple queries by issuing a join query for the related bean. If a set of beans are accessed as part of the same unit of work, then your application should load them into cache at the same time.

Suppose your application contains entity beans with the following relationships:

`customerBean`	has a `one-to-many` relationship with	`accountBean`
`accountBean`	has a `one-to-one` relationship with	`addressBean`
`customerBean`	has a `one-to-one` relationship with	`phoneBean`

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();
```

Without relationship caching, an SQL query is issued by the first line of code to load the `accountBean` and another SQL query is issued by the second line of code to load the `addressBean`; this results in two queries to the database.

With relationship caching, a single query is issued to load both `accountBean` and `addressBean` by the first line of code, which should result in better performance. So, if you know that a related bean will be accessed after executing a particular finder method, it is a good idea to let the finder method know via the relationship caching feature.

Note: The relationship caching feature has the following limitations:

- Relationship caching is supported for `one-to-one`, `one-to-many`, and `many-to-one` relationships. It is not supported for `many-to-many` relationships.
- When using `weblogic-ql`, this feature only works with finder methods that return references to either `EJBObject` or `EJBLocalObject` beans.

If you enable relationship caching for a finder or a select method, the result of the query will always be a distinct set even if the `distinct` keyword is not specified. This is due to a technical limitation that does not allow the EJB container to distinguish duplicates in the underlying JDBC result set.

With relationship caching enabled, changes to the relationship are automatically reflected in cache. For instance, if a instance is added on the "many" side of a one-to-many relationship, that change is reflected in the cached relationship—a subsequent query to the bean on the "one" side of the relationship causes the relationship to be refreshed in cache, so that the new instance is available to the query.

Enabling Relationship Caching

To enable relationship caching:

1. Specify the `caching-name` element in the `weblogic-query` stanza of the `weblogic-cmp-jar.xml` file.

   If a `caching-name` element is specified in the `weblogic-query` stanza, when the finder query is executed, WebLogic Server loads data for related beans as specified by the `relationship-caching` element that the `caching-name` element specifies.

2. Make sure that the `finders-load-bean` element (in `weblogic-ejb-jar.xml`) for the bean that contains the finder is not set to `False` or relationship caching will not be enabled. The default value of `finder-load-bean` is `True`.

3. Specify the `database-type` element in `weblogic-cmp-jar.xml` file. Relationship caching uses outer joins for queries and outer join syntax can vary by database type.

Because relationship caching uses join queries, the number of `caching-element` stanzas in the `relationship-caching` element can increase duplicates in the result set. Specify one one-to-many relationships per `caching-element` to avoid duplicates in the result set.

Specify the `relationship-caching` stanza in `weblogic-cmp-jar.xml`, as shown in this example:

```
<relationship-caching>
       <caching-name>cacheMoreBeans</caching-name>
       <caching-element>
              <cmr-field>accounts<</cmr-field>
              <group-name>acct_group</group-name>
              <caching-element>
                     <cmr-field>address</cmr-field>
                     <group-name>addr_group</group-name>
              </caching-element>
       </caching-element>

       <caching-element>
              <cmr-field>phone</cmr-field>
              <group-name>phone_group</group-name>
       </caching-element>
</relationship-caching>
```

The `accounts` and `phone` fields are cmr-fields in `customerBean`; the `address` field is a cmr-field in the `accountBean`; and `addr_group` and `phone_group` are groups in `addressBean` and `phoneBean`.

Using nested `caching-element` elements enables the bean to load more than one level of related beans. In the above sample, `addressBean` is the second level related bean because it is nested in the `accountBean`. Currently, there is no limitation on the number of `caching-element`s that you can specify. However, setting too many `caching-element` levels could have an impact on the performance of the current transaction.

Choosing a Concurrency Strategy

An entity bean's concurrency strategy specifies how the EJB container should manage concurrent access to the bean; it determines how and when WebLogic Server synchronizes its cached copy of the entity with the database.

Concurrency strategies supported by WebLogic Server are described in the following sections.

Exclusive Concurrency

With exclusive concurrency, the container places an exclusive lock on cached EJB instances when the bean is associated with a transaction. Other requests for the EJB instance are blocked until the transaction completes. Exclusive locks are local to the server instance, so this strategy is most appropriate for a single server architecture. When used in a cluster, exclusive concurrency serializes all requests to a bean instance within a server instance, but concurrency between servers in the cluster that access the same bean concurrently is governed by the database.

Multiple clients can use the exclusive concurrency option to access entity 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.

Database Concurrency

With database concurrency, concurrency control for an EJB for each transaction is deferred to the underlying datastore. WebLogic Server allocates a separate entity bean instance to each transaction and allows concurrency control to be handled by the database. This is the default option.

With the `Database` mechanism, the EJB container continues to cache instances of entity bean instances. However, the container does not cache the intermediate state of a bean instance between transactions. Instead, WebLogic Server issues a SQL `SELECT` for each instance at the beginning of a transaction to obtain the latest EJB data. A SQL `UPDATE` is issued if there are changes.

The request to commit data is subsequently passed along to the database. The database, therefore, handles all concurrency control and deadlock detection for the EJB's data.

Optimistic Concurrency

As with the `Database` concurrency strategy, `Optimistic` concurrency gives each transaction its own bean instance. The `Optimistic` concurrency strategy does not hold any locks in the EJB container or the database while the transaction is in process.

Note: For databases that do read-locking (non-Oracle databases) optimistic beans read data in a separate, local transaction. The local transaction commits a soon as the read completes. This strategy avoids read locks and can allow for better scalability when transactions do not update the same data concurrently.

Check Data for Validity with Optimistic Concurrency

You can configure the EJB container to validate an Optimistic bean's transaction data before committing the transaction, to verify that no data read or updated by the transaction has bean changed by another transaction. If it detects changed data, the EJB container rolls back the transaction.

Note: The EJB container does not validate Blob or Clob fields in a bean with `Optimistic` concurrency. The workaround is to use version or timestamp checking.

Configure validity checking for a bean with `Optimistic` concurrency using the `verify-columns` and `verify-rows` elements in the `table-map` stanza for the bean in `weblogic-cmp-jar.xml` file.

The `verify-columns` element specifies how columns in a table are checked for validity when you use the optimistic concurrency strategy.

1. Set the value of the `verify-columns` element to:
   - `Read`—To check all columns in the table that have been read during the transaction. This includes both rows that are simply read and rows that are read and then updated or deleted by the transaction.
   - `Modified`—To check only the columns that have been updated or deleted by the current transaction.
   - `Version`—To check that a version column exists in the table and that this column is used to implement optimistic concurrency.
   - A version column must be created with an initial value of 0, and must increment by 1 whenever the row is modified.
   - `Timestamp`—To check that a timestamp column exists in the table and that this column is used to implement optimistic concurrency. Timestamp-based optimistic concurrency requires a 1 second granularity for the database column.

   The EJB container manages the version and timestamp columns and ensures that these columns are kept up to date.

   Note: The version or timestamp column is *not* updated if the transaction did not modify any regular CMP or CMR fields—if the only data changed during the transaction was the value of the version or timestamp column (as a result of transaction initiation) the column used for optimistic checking will not be updated at the end of the transaction.

2. Set the `verify-rows` element to:

- Read—To check all rows in the table that have been read during the transaction. This includes both rows that are simply read and rows that are read and then updated or deleted by the transaction. Checking all rows entails additional overhead because it generally increases the amount of optimistic checking that must be performed by the EJB container. With the Read option, committed transactions read a set of rows that are guaranteed not to be modified by another transaction until after the transaction commits. This results in a high level of consistency which is very close to the ANSI definition of SERIALIZABLE consistency.

  Note: If verify-rows is set to Read then the verify-columns element may not have a value of Modified since this combination would result in only checking the modified rows.

- Modified—To check only the rows that have been updated or deleted by the current transaction. This setting ensures that two transactions will not update the same row concurrently, resulting in a lost update, but it allows reads and updates of different transactions to be interleaved. This results in a level of consistency that falls between the ANSI READ_COMMITTED and REPEATABLE_READ levels of consistency.

3. If verify-columns is set to Version or Timestamp, specify the version or timestamp column using the optimistic-column in the table-map stanza in the weblogic-cmp-jar.xml file. Mapping this column to a cmp-field is optional.

   The optimistic-column element identifies a database column that contains a version or timestamp value used to implement optimistic concurrency. This element is case maintaining, though not all databases are case sensitive. The value of this element is ignored unless verify-columns is set to Version or Timestamp.

If the EJB is mapped to multiple tables, optimistic checking is only performed on the tables that are updated during the transaction.

Optimistic Concurrency and Oracle Databases

For Oracle databases, if you set verify-columns to Modified for an entity EJB with a CMP non-key field type java.util.Date and implementation type Oracle DATE, WebLogic Server throws an optimistic concurrency violation exception when a simple update is made to the non-key DATE field—even though only one user is updating the record.

This problem occurs because of a mismatch in date value precision between the Oracle DATE column and the java.util.Date type. The java.util.Date type is in milliseconds, and the Oracle DATE column is not. There are two ways to avoid this error:

- Set the Oracle database column type to Timestamp, a higher precision type introduced in Oracle9i.

- Include logic in your application to zero out the milliseconds of a `java.util.Date` value. To accomplish this, prepare a date field for an entity bean `java.util.Date` field in this way:

```
Calendar cal = Calendar.getInstance();
cal.set(Calendar.MILLISECOND, 0); // clears millisecond
Date myDate = cal.getTime();
```

Read Only Concurrency

Used to signify that a bean is read-only. The container activates a new bean instance for each transaction so that requests proceed in parallel. WebLogic Server calls `ejbLoad()` for read-only beans based on the `read-timeout-seconds` parameter.

Prior to version WebLogic Server 8.1 SP02, EJBs that used ReadOnly concurrency were allowed to use create and remove operations. In WebLogic Server 8.1 SP02, create and remove operations are not allowed for EJBs that used ReadOnly concurrency, because this practice is discouraged, and to allow generation of more efficient code for read-only beans.

To support the pre-SP02 behavior and allow read-only beans to use create and remove operations, set the `allow-readonly-create-and-remove` element in `weblogic-cmp-jar.xml`.

Concurrency Strategy Trade-Offs

Concurrency strategies present a trade-off between performance and freshness of data. You should choose a concurrency strategy based on which of these factors weighs more heavily in your application. The trade-offs are summarized in Table 6-5.

Table 6-5 Concurrency Strategy Trade-offs

Concurrency Strategy	Benefits	Risks and Limitations	When to Choose It
`Database`	Deferring concurrency control to the database improves throughput, compared to exclusive concurrency, for concurrent access to data and provides deadlock detection.	Risk of deadlock, as each transaction must obtain an update lock from the database. Mitigate deadlock risk by setting `use-select-for-update` in `weblogic-cmp-jar.xml`. This causes the database to take out an exclusive lock when the read is done, avoiding the deadlock that can occur when a read lock is upgraded to an exclusive lock. Makes the bean more dependent on the database's lock policies, which might reduce the bean's portability.	If the database concurrency control is sufficient for your application and you do not require additional features provided by the container. **Note:** Use the `transaction-isolation` element in combination with Database concurrency to achieve the desired concurrency behavior.
`Optimistic`	Provides highest level of concurrent access, as it holds no locks in the EJB container or database during a transaction.	Multiple transaction can access the same application data at the same time.	If multiple transactions are unlikely to attempt to modify the same application data at the same time.

Concurrency Strategy	Benefits	Risks and Limitations	When to Choose It
`Exclusive`	Serializes access to EJB data in a single server (non-clustered environment) for a high level of consistency. Avoids deadlocks due to lock upgrades, and prevents unnecessary calls to `ejbLoad()` to refresh the bean instance's persistent fields.	Performance degradation can result. Once a client has locked an EJB instance, other clients are blocked from the EJB's data, even those who require only read-access. **Note:** Using Entity Beans with parameter settings concurrency-strategy=Exclusive and use-select-for-update=true together may cause a temporary deadlock. Avoid using these parameter settings in conjunction or set the parameter cache-between-transaction=true.	To provides backwards compatibility for applications that rely on this strategy. For applications in which a high level of concurrency is essential, and more important than performance.
`Read Only`	N/A	N/A	N/A

Configuring Concurrency Strategy

Specify the concurrency mechanism for a bean by setting the `concurrency-strategy` element in the `entity-cache` stanza in `weblogic-ejb-jar.xml`. Because `concurrency-strategy` is defined at the bean level, different beans in the same application can use different concurrency strategies, as appropriate.

If you do not specify a `concurrency-strategy`, WebLogic Server uses `Database` concurrency by default.

Preventing Deadlocks for Transactions That Use Exclusive Concurrency and Cascade Deletes

In situations of high throughput, transactions that use an exclusive concurrency strategy can encounter deadlocks if a transaction that performs a cascade delete needs access to the same entity bean as a transaction that does not perform a cascade delete.

You can prevent such deadlocks with the lock-order element of `weblogic-cmp-jar.xml` deployment descriptor file. `lock-order` defines the order in which beans are locked during a cascade delete of related beans. Its value is an integer value. The bean with the lowest `lock-order` value is processed first, the bean with the next lowest `lock-order` value is processed next, and so on.

The locking order specified should be the same as the locking order used by other transactions in the application.

`lock-order` causes a cascade delete to lock bean instances in the same order as their other transactions. If the normal lock order is BeanA, then BeanB, specify this `lock-order`, and cascade delete will use it.

Using the Read-Mostly Pattern

For persistent data that is only occasionally updated, you can implement a "read-mostly pattern" in WebLogic Server by mapping a read-only and a read-write entity bean to the same data. You use the read-only entity bean for reads and the read-write entity bean for writes.

The read-only entity EJB loads bean data at intervals specified by the `read-timeout-seconds` element in the `entity-cache` (or `entity-cache-ref`) stanza for the bean in `weblogic-ejb-jar.xml`. To ensure that the read-only bean always returns current data, the read-only bean must be must invalidated when the read-write bean changes the entity bean data. You can configure WebLogic Server to automatically invalidate the read-only bean, or explicitly invalidate it in bean code, as described in "Invalidating Read-Only Entity EJBs Implicitly" and "Invalidating Read-Only Entity EJBs Explicitly" respectively.

In a WebLogic Server cluster, the read-mostly pattern gives clients of the read-only EJB the performance advantage of reading from cache, while clients of read-write EJB enjoy true transactional behavior—the `read-write` EJB's cached state always matches the persistent data in the database.

Configuring Entity Beans for Read-Mostly Pattern

These practices will reduce the likelihood of data consistency problems with the read-mostly pattern.

- Configuring the read-only beans' `read-timeout-seconds` element in `weblogic-ejb-jar.xml`:
  - To the same value in all beans that can be updated in the same transaction.
  - To the shortest period that yields acceptable performance levels.

- Design read-write beans:
  - To update the minimum data set necessary; avoid implementing beans that write numerous, unchanged fields to the datastore at each `ejbStore()`.
  - To update their data on a timely basis; do not use a read-write bean in lengthy transactions that might run longer than the `read-timeout-seconds` setting for its read-only counterparts.
  - To invalidate the read-only counterpart of a read-write bean when the read-write bean updates bean data.

If you are running EJB 2.0, you can approximate the read-mostly pattern using a single bean that uses optimistic concurrency. An optimistic bean acts like a read-only beans when performing a read—it reads from the cache and can return stale data. However, when an optimistic bean performs a write, the container ensures that the data being updated has not changed—providing the same level of consistency for writes as a bean that uses Database concurrency. See "Choosing a Concurrency Strategy" on page 6-35.

Invalidating Read-Only Entity EJBs Implicitly

The `invalidation-target` element in the `entity-descriptor` stanza in `weblogic-ejb-jar.xml` identifies a read-only entity EJB that should be invalidated when a CMP entity bean has been modified.

`invalidation-target` may only be specified for an EJB 2.0 CMP entity bean. The target `ejb-name` must be a read-only entity EJB.

Invalidating Read-Only Entity EJBs Explicitly

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

Listing 6-10 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 code shows how to cast the home to `CachingHome` and then call the method:

Listing 6-11 Casting the Home and Calling 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();
```

The `invalidate()` method causes the read-only entity beans to be invalidated in the local server instance. If the server instance is a member of a cluster, it multicasts a message to the other clustered servers to invalidate their cached copies of the bean. Upon the next `getXXX()` to an invalidated read-only entity bean, the container loads the current version of the bean's persistent data to the entity cache from the database, as described in "Understanding ejbLoad() and ejbStore() Behavior" on page 6-4.

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

CMP Entity Bean Descriptors Element by Feature

The following sections are a quick reference to WebLogic Server-specific deployment for CMP entity beans. Each section contains the elements related to a particular the type of feature or behavior. The table in each section defines relevant elements terms of: the behavior it controls, the parent stanza in `weblogic-cmp-jar.xml` that contains the element, and the behavior you can expect if you do not explicitly specify the element in `weblogic-cmp-jar.xml`.

- "Container-Managed Relationship Elements" on page 6-45

- "Primary Key Elements" on page 6-46

Container-Managed Relationship Elements

Table 6-6 Container-managed Relationship Elements in weblogic-cmp-jar.xml

Element	Description
`relation-name`	Name of the relationship. **Note:** If a `ejb-relation-name` for the relationship is specified in `ejb-jar.xml`, `relation-name` must contain the same value as `ejb-relation-name`.
`relationship-role-name`	The name of the relationship role. (A relationship has two roles—one for each side of the relationship). For a 1-1 or 1-m relationship, specify only the role on the `foreign-key` side. For an examples, see "Defining a One-to-One Relationship" on page B-51 and "Defining a One-to-Many Relationship" on page B-52. For a m:m relationship, specify the roles on both sides of the relationship. roles for both sides of the relationship are specified. For an example, see "Defining a Many-to-Many Relationship" on page B-53. **Note:** The value of `<relationship-role-name>` should match the name of an `ejb-relationship-role` in `ejb-jar.xml`.
`foreign-key-column`	Specifies the target side of the key column mapping for the relationship—the foreign key column.
`key-column`	Specifies the initiating side of the key column mapping for the relationship—the primary key column.

Primary Key Elements

Table 6-7 Primary key elements in weblogic-cmp-jar.xml

Element	Description	Default
generator-type	Identifies the facility used to generate primary keys. Values include `Oracle`, SQLServer, or SQLServer2000, `NamedSequenceTable`.	
generator-name	Defines the Oracle `SEQUENCE`, or a the name of a SEQUENCE table used.	
key-cache-size	Specifies the size of the key cache.	
create-default-dbms-tables	Determines behavior related to if and how the EJB container will create database tables.	Disabled

CHAPTER 7

Message-Driven EJBs

The following sections describe the message-driven bean (MDB) lifecycle, design considerations, and instructions for key implementation tasks. For a description of the overall EJB development process, see "Implementing Enterprise Java Beans" on page 4-1.

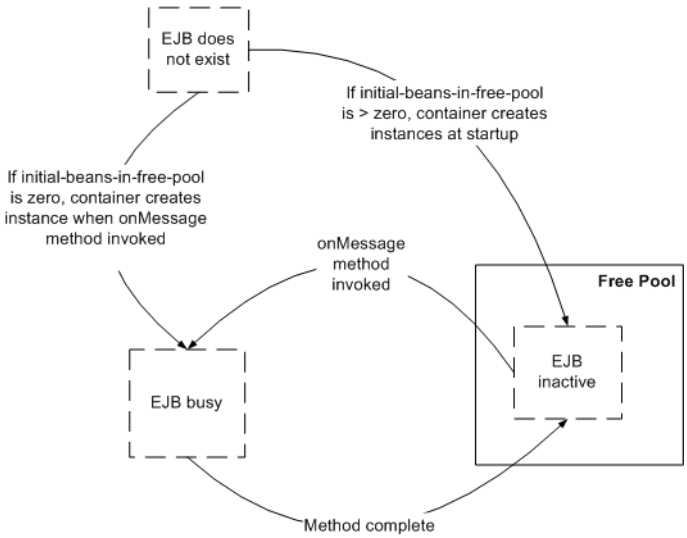
- "Message-Driven EJB Lifecycle and the Free Pool" on page 7-1
- "MDBs and Messaging Models" on page 7-3
- "MDB Deployment Options" on page 7-6
- "Programming and Configuring MDBs: Main Steps" on page 7-9
- "Programming and Configuring MDBs: Details" on page 7-12
- "Migration and Recovery for Clustered MDBs" on page 7-27
- "Nutshell Summary: Deployment Elements for MDBs" on page 7-28

For an introduction to MDBs and how they are typically used in applications, see "Message-Driven Beans Implement Loosely Coupled Business Logic" on page 2-3 and "Message-Driven Bean Features" on page 3-6. For information about WebLogic JMS, see *Programming WebLogic JMS*.

Message-Driven EJB Lifecycle and the Free Pool

This section describes the phases in the lifecycle of a message-driven bean instance and how you can configure an MDB to control the lifecycle. Figure 7-1 illustrates the key events and states in the MDB lifecycle.

Figure 7-1 MDB Life Cycle



MDBs and the Free Pool

WebLogic Server maintains a *free pool* where MDB instances that are not currently servicing requests reside. For best response time and throughput, you can configure WebLogic Server to populate the pool at startup by specifying the desired quantity in the bean's `initial-beans-in-free-pool` element in `weblogic-ejb-jar.xml`. By default, `initial-beans-in-free-pool` is 0.

The number of MDB instances in the free pool is limited by the value of the `max-beans-in-free-pool` element in `weblogic-ejb-jar.xml` (by default, 1,000), the number of threads in the MDB's execute queue, and available memory. The effect that thread pool size has on the maximum number of MDB instances in the free pool varies, based on whether the MDB uses the default execute queue, or is configured to use another execute queue. For more information, see "dispatch-policy" on page A-20.

An MDB processes messages from a JMS destination. When a message is received, the EJB container calls the MDB's `onMessage()` method, which contains the business logic the EJB performs. When an MDB's `onMessage` method is invoked:

- If a bean instance is available in the free pool, WebLogic Server uses that instance. This results in the best response time, as the request is served without waiting for a new instance to be created.

- If no bean instance is located in the free pool, WebLogic Server initializes one by calling:
  - The MDB's constructor method
  - The MDB's `ejbCreate()` method
  - The MDB'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 7-25.
- If all instances of an MDB are active, and `max-beans-in-free-pool` has been reached, JMS `onMessage()` requests will be blocked until an active MDB completes a method call. Your application waits until your transaction times out or until a bean instance that already exists in the pool becomes available.

After an MDB instance's `onMessage()` method returns, the request is complete, and the instance it is placed in the free pool for re-use, assuming that doing so does not cause `max-beans-in-free-pool` to be exceeded.

MDBs and Concurrent Processing

MDBs support concurrent processing for both topics and queues. For more information about topics and queues, see "MDBs and Messaging Models" on page 7-3.

The default setting for the `max-beans-in-free-pool` element in `weblogic-ejb-jar.xml`—1,000—provides the most parallelism. The only reason to change this setting would be to limit the number of parallel consumers.

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

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

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

MDBs and Messaging Models

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

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

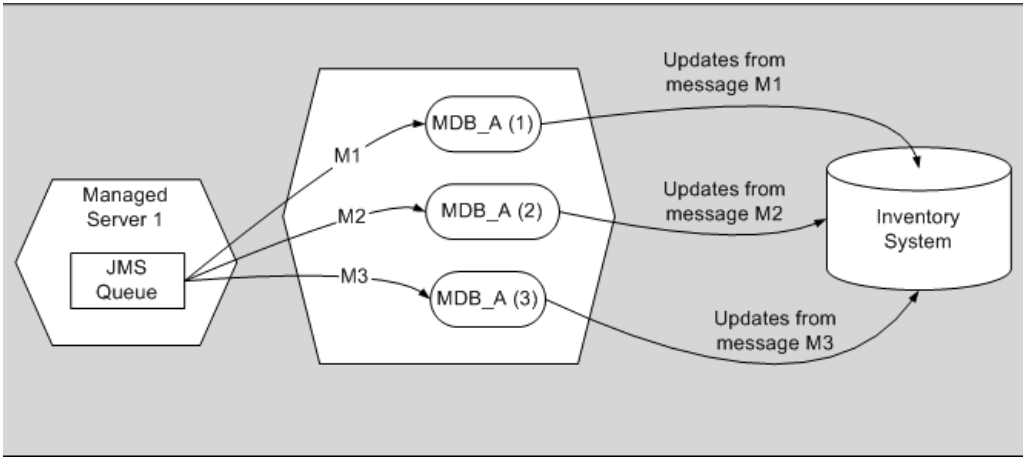
Point-to-Point (Queue) Model: One Message Per Listener

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

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

Figure 7-2 illustrates a point-to-point application. Each message is processed by single instance of MDB_A. Message "M1" is processed by MDB_A(1), "M2" is processed by MDB_A(2), and "M3" is processed by MDB_A(3).

Figure 7-2 Point-to-Point Model



Publish/Subscribe (Topic) Model

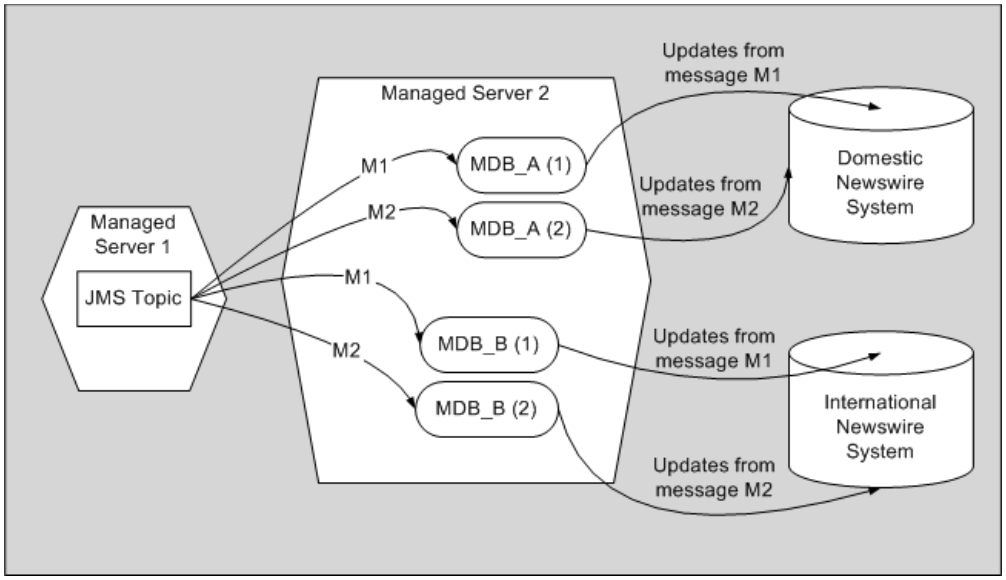
In the publish/subscribe model, a JMS topic publishes all messages to all subscribed listeners. If an MDB listener goes down, that MDB will miss the message, unless the topic is a *durable subscription* topic.

For more information on durable subscriptions and for configuration instructions, see "Setting Up Durable Subscriptions" in *Programming WebLogic JMS* and "Configuring Durable Topic Subscriptions" on page 7-21.

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

Figure 7-3 illustrates a publish/subscribe application. In contrast to a point-to-point application, in a publish/subscribe model, a message is processed by one instance of *each* of the subscribing MDBs. Message "M1" is processed by an instance of MDB_A and an instance of MDB_B. Similarly, "M2" is processed by an instance of each of the subscribing MDBs.

Figure 7-3 Publish/Subscribe Model



Exactly-Once Processing

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

- If an application fails, a transaction rolls back, or the hosting server instance fail during or after the `onMessage()` method completes but before the message is acknowledged or committed, the message will be redelivered and processed again.

- Non-persistent messages are also redelivered in the case of failure, except for the case where the message's host JMS server shuts down or crashes, in which case the messages are destroyed.

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

MDB Deployment Options

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

Destination and MDBs: Collocated vs. Not Collocated

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

Collocated Destination/MDBs

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

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

Figure 7-4 and Figure 7-5 illustrate architectures in which the MDB application is deployed to the server instance that hosts the associated JMS destination. These architectures vary in that the first has a *distributed destination* and the second does not. In an application that uses distributed destinations, the MDB is deployed to each server instance that hosts a member of the distributed destination set. For more information about distributed destinations, see "JMS Distributed Destinations" on page 7-9. As illustrated in Figure 7-4 the message "M1" is delivered to an instance of MDB_A on each server instance where a distributed destination and MDB_A are deployed.

Figure 7-4 Collocated Destination/MDBs, Distributed Destination

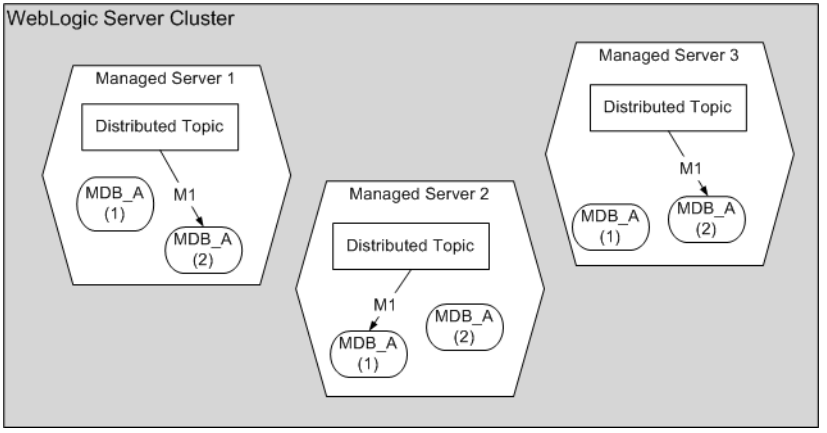
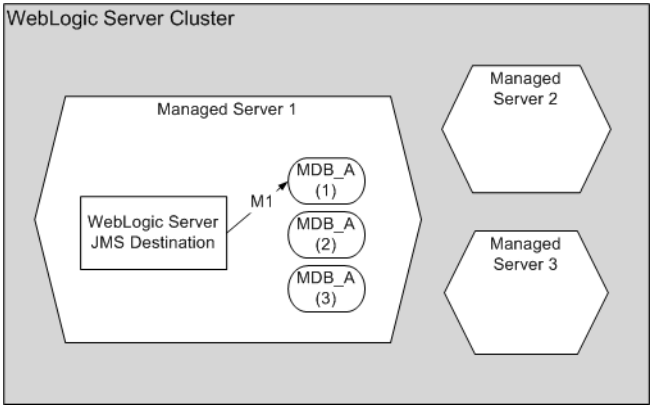


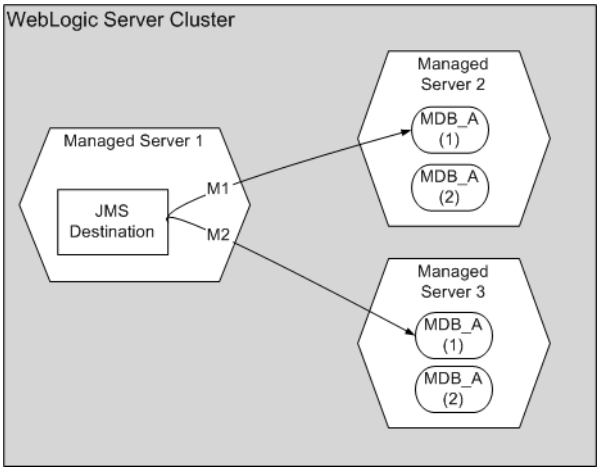
Figure 7-5 Collocated Destination/MDBs, Non-Distributed Destination



Non-Collocated Destination/MDBs

Figure 7-6 illustrates an architecture in which an MDB runs on a separate server instance than the JMS Destination to which the MDB listens.

Figure 7-6 Non-Collocated Application, Non-Distributed Destination



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

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

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

JMS Distributed Destinations

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

If you configure a distributed destination, WebLogic JMS distributes the messaging load across available members of the distributed destination. If a member of the destination becomes unavailable due to a server failure, message traffic is re-directed to the other available physical destinations in the distributed destination set.

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

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

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

For an example, see Figure 7-4. For additional information about distributed destinations, see "Using Distributed Destinations" in *Programming WebLogic JMS*.

Programming and Configuring MDBs: Main Steps

This section provides step-by-step instructions for implementing an MDB.

Note: For a summary of key deployment elements for MDBs, see "Nutshell Summary: Deployment Elements for MDBs" on page 7-28.

Required JMS Configuration

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

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

  The default WebLogic MDB connection factory is XA-capable. For information about the default connection factories, see "Using the Default Connection Factories" in *Programming WebLogic JMS*. For instructions to configure a custom WebLogic JMS connection factory, see "Configuring a JMS Connection Factory" in *Administration Console Online Help*.

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

- A JMS destination

  For instructions on configuring WebLogic JMS destinations, see "JMS Destination" in *Administration Console Online Help*.

Note: If your JMS provider is a remote WebLogic Server JMS provider or a foreign JMS provider, and you use the wrapper approach recommended in "Whether to Use Wrappers" on page 7-15, in addition to configuring the non-local JMS components, you must also configure a Foreign Connection Factory and Foreign JMS Destination in your local JNDI tree.

Create MDB Class and Configure Deployment Elements

Follow these steps to implement a message-driven bean:

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

   The MDB class must define the following methods:

   – One `ejbCreate()` method that the container uses to create an instance of the message-driven bean in the free pool.

   – One `onMessage()` method that is called by the EJB container when a message is received. This method contains the business logic that processes the message.

   – One `setMessageDrivenContext{}` method that provides information to the bean instance about its environment (certain deployment descriptor values); the MDB also uses the context to access container services. See "Using the Message-Driven Bean Context" on page 7-25.

   – One `ejbRemove()` method that removes the message-driven bean instance from the free pool.

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

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

The key behaviors to configure are:

- Transaction management strategy—The MDB's transaction management strategy, in the `transaction-type` element. For instructions, see "Configuring Transaction Management Strategy for an MDB" on page 7-13.
- Destination type—The type of destination to which the MDB listens. For more information, see "Configuring Destination Type" on page 7-13.

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

```
<weblogic-ejb-jar>
   <weblogic-enterprise-bean>
      <ejb-name>exampleMessageDrivenA</ejb-name>
      <message-driven-descriptor>
         <pool>
           <max-beans-in-free-pool>...</max-beans-in-free-pool>
           <initial-beans-in-free-pool>...</initial-beans-in-free-pool>
         </pool>
         <destination-jndi-name>...</destination-jndi-name>
         <initial-context-factory>...</initial-context-factory>
         <provider-url> </provider-url>
```

```
            <connection-factory-jndi-name>..<connection-factory-jndi-name>
            <jms-client-id>....   </jms-client-id>
         </message-driven-descriptor>
         <jndi-name>someid</jndi-name>
         <dispatch-policy>custom_execute_queue</dispatch-policy>
      </weblogic-enterprise-bean>
   </weblogic-ejb-jar>
```

The key elements to configure are:

- The elements that specify how to access the destination. For instructions, see "Configuring MDBs for Destinations" on page 7-14.
- The elements related to the free pool. For more information, see "Message-Driven EJB Lifecycle and the Free Pool" on page 7-1.

4. Set `start-mdbs-with-application` to `false` in the `weblogic-application.xml`. This ensures that the MDB does not start message processing until WebLogic Server is completely booted up. For more information, see "Deferring Message Processing Until Completion of Boot Up" on page 7-25.
5. Compile and generate the MDB class using the instructions in "Compile Java Source" on page 4-13.
6. Deploy the bean on WebLogic Server using the instructions in the section "Quickstart Guide to Deploying Modules" in *Deploying WebLogic Server Applications*.

   If WebLogic Server cannot find an MDB's JMS destination during deployment, deployment succeeds, but WebLogic Server prints a message saying the destination was not found. The MDB bean then periodically tries to connect to its JMS queue until it succeeds. For more information, see "Migration and Recovery for Clustered MDBs" on page 7-27.

Programming and Configuring MDBs: Details

The topics in this section supplement the instructions in "Programming and Configuring MDBs: Main Steps" on page 7-9.

Note: For a summary of key deployment elements for MDBs, see "Nutshell Summary: Deployment Elements for MDBs" on page 7-28.

- "Configuring Destination Type" on page 7-13
- "Configuring MDBs for Destinations" on page 7-14
- "Configuring Durable Topic Subscriptions" on page 7-21

- "Configuring Message Handling Behaviors" on page 7-22
- "Using the Message-Driven Bean Context" on page 7-25
- "Configuring a Security Identity for a Message-Driven Bean" on page 7-25

Configuring Destination Type

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

To specify a topic, set `destination-type` to `javax.jms.Topic`

To specify a queue, set `destination-type` to `javax.jms.Queue`. If the destination is a topic, specify `subscription-durability` as either `Durable` or `NonDurable`. For more information, see "Configuring Durable Topic Subscriptions" on page 7-21.

Configuring Transaction Management Strategy for an MDB

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

To configure container-level transaction management:

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

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

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

To configure bean-level transaction management:

- Set the `transaction-type` element in the `message-driven` stanza in `ejb-jar.xml` file to `Bean`.

– Set the `acknowledge-mode` element to specify the desired JMS acknowledgment semantics, either `AUTO_ACKNOWLEDGE` (the default) or `DUPS_OK_ACKNOWLEDGE`.

  For more information, see "Session Object" in *Programming WebLogic JMS*.

Configuring MDBs for Destinations

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

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

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

Destinations that run under a non-BEA JMS provider are referred to as *foreign*. Foreign JMS providers have their own JNDI provider and foreign JMS objects do not share the same context with a WebLogic Server MDB—unless the foreign JMS objects are configured with wrappers to appear in the MDB's JNDI context. This approach is discussed in "Whether to Use Wrappers" on page 7-15.

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

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

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

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

- "Whether to Use Wrappers" on page 7-15
- "How to Set provider-url" on page 7-15
- "How to Set initial-context-factory" on page 7-16
- "How to Set destination-jndi-name" on page 7-16
- "How to Set connection-factory-jndi-name" on page 7-16

To configure the elements in `message-driven-descriptor` for specific scenarios, see "Common Destination Scenarios: Illustrations and Key Element Settings" on page 7-17.

Whether to Use Wrappers

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

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

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

How to Set provider-url

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

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

How to Set initial-context-factory

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

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

How to Set destination-jndi-name

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

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

How to Set connection-factory-jndi-name

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

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

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

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

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

Common Destination Scenarios: Illustrations and Key Element Settings

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

- "A. Destination on a Local WebLogic JMS Server" on page 7-18
- "B. Destination On a Remote WebLogic JMS Server—No Wrappers" on page 7-18
- "C. Destination on Foreign JMS Server—No Wrappers" on page 7-19
- "D. Destination on a Remote WebLogic Server or Foreign JMS Server—With Wrappers" on page 7-19

Table 7-1, "Common Configuration Scenarios," on page 7-20 summarizes how to configure the elements in the `message-driven-descriptor` stanza of `weblogic-ejb-jar.xml` for each scenario.

Figure 7-7 A. Destination on a Local WebLogic JMS Server

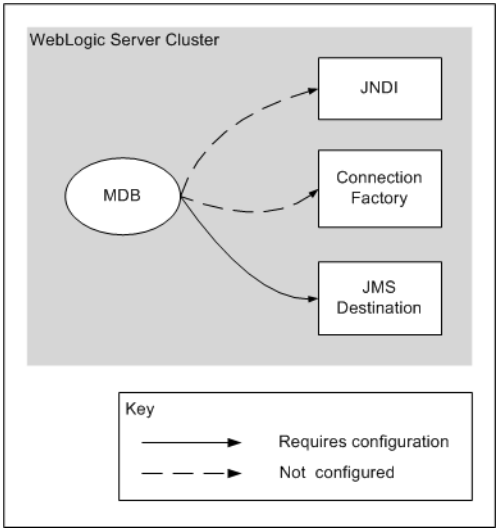


Figure 7-8 B. Destination On a Remote WebLogic JMS Server—No Wrappers

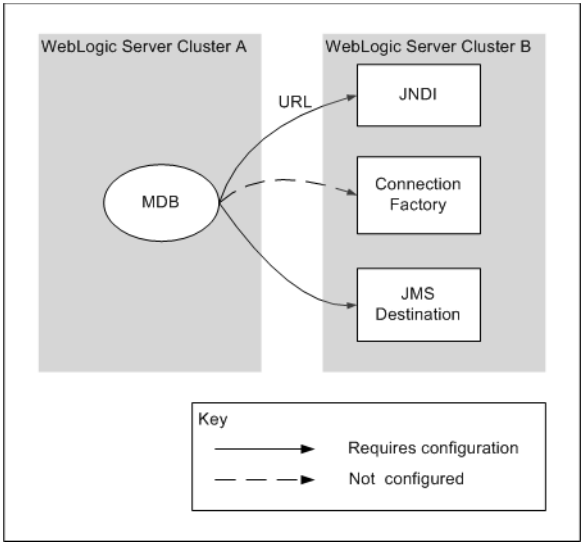


Figure 7-9 C. Destination on Foreign JMS Server—No Wrappers

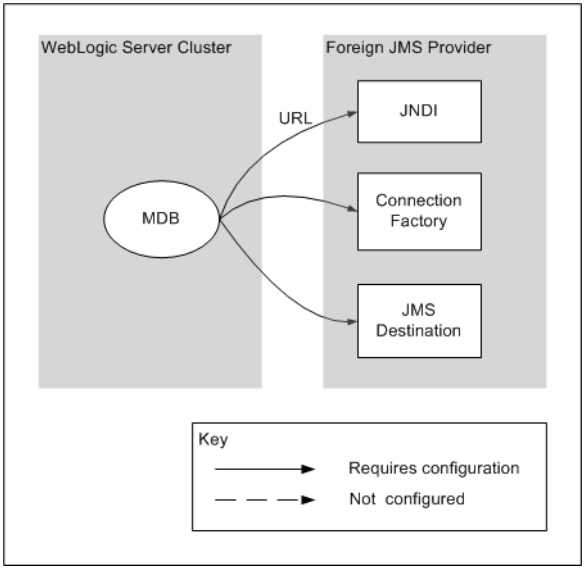


Figure 7-10 D. Destination on a Remote WebLogic Server or Foreign JMS Server—With Wrappers

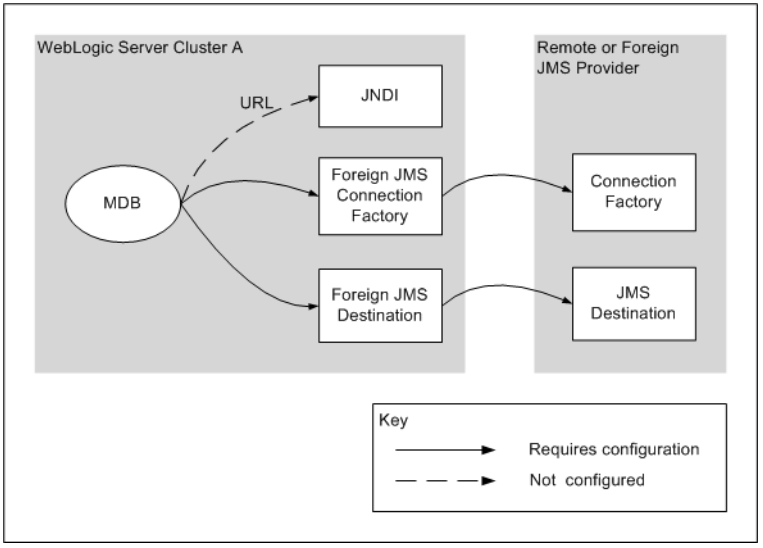


Table 7-1 Common Configuration Scenarios

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

Configuring Durable Topic Subscriptions

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

Configuring a Durable Topic Subscription for a Non-Clustered Server

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

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

   If you configure your own connection factory to suit specific application requirements, as described in "Configuring a JMS Connection Factory" in *Administration Console Online Help*, you can define the `ClientID` for the MDB when you configure the connection factory.

   If you set up your connection factory and do not assign it a `ClientID`, or if you use a default connection factory, as described in "Using a Default Connection Factory" in *Administration Console Online Help*, the MDB uses the value of `jms-client-id` in `weblogic-ejb-jar.xml` as its client ID. If `jms-client-id` is not specified, the default value is the `ejb-name` for the MDB.

Configuring a Durable Topic Subscription for a Cluster

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

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

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

Durable subscribers cannot subscribe directly to a WebLogic Server distributed destination topic, but must instead subscribe to the JNDI name of the destination topic physical members. There are two ways to accomplish this:

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

Configuring Message Handling Behaviors

These topics provide guidelines for behaviors related to message delivery:

- "Ensuring Message Receipt Order" on page 7-22
- "Preventing and Handling Duplicate Messages" on page 7-23
- "Redelivery and Exception Handling" on page 7-24

Ensuring Message Receipt Order

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

- Set `max-beans-in-free-pool` to 1 for the MDB. This ensures that the MDB is the sole consumer of the message.
- If your MDBs are deployed on a cluster, deploy them to a single node in the cluster, as illustrated in Figure 7-5.

To ensure message ordering in the event of transaction rollback and recovery, configure a custom connection factory with `MessagesMaximum` set to 1, and ensure that no redelivery delay is

configured. For more information see "Ordered Redelivery of Messages" in *Programming WebLogic JMS*.

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

Preventing and Handling Duplicate Messages

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

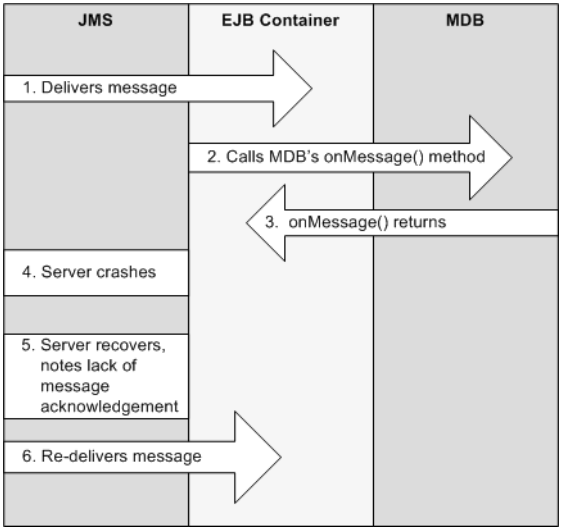
Your MDB design should allow for the likelihood of duplicate messages. Duplicate messages can be undesirable in certain cases. For example, if an MDB's `onMessage()` method includes code to debit a bank account, receiving and processing that message twice would result in the account being debited twice.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

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

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

Even if an MDB's `onMessage()` method completes successfully, the MDB can still receive duplicate messages if the server crashes between the time `OnMessage()` completes and the time the container acknowledges message delivery. Figure 7-11 illustrates this scenario.

Figure 7-11 Server Crash Between Completion of onMessage() and Container Delivery Acknowledgement



Redelivery and Exception Handling

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

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

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

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

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

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

Using the Message-Driven Bean Context

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

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

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

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

Deferring Message Processing Until Completion of Boot Up

By default, an MDB starts processing messages as soon as it is deployed, even if its target WebLogic Server instance has not completed booting. This can cause an MDB application to access uninitialized services or applications during the boot process, and to fail. To avoid this problem, set `start-mdbs-with-application` to `false` in `weblogic-application.xml`.

Setting `start-mdbs-with-application` to `false` forces MDBs to defer starting until after the server instance opens its listen port, near the end of the server boot up process.

Configuring a Security Identity for a Message-Driven Bean

When a message-driven bean (MDB) receives messages from a JMS queue or topic, the EJB container uses a Credential Mapping provider and a credential map to obtain the security

identity—username and password—to use when establishing the JMS connection. This credential mapping occurs only once, when the MDB is started.

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

To configure a security identity for an MDB:

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

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

3. In the `weblogic-ejb-jar.xml` deployment descriptor, map the `run-as` identity to the user defined in the previous step, as shown below:

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

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

4. If the JMS provider is not WebLogic JMS, configure the credential mapper:

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

   To configure the credential mapper:

   a. In the left pane of the WebLogic Server Administration Console, expand Deployments to show the types of WebLogic resources that can be deployed.

   b. Expand the Applications folder or the EJB folder to navigate to the application containing the MDB for which you want to create a credential map, right-click on it, and choose the Define Policies and Roles for Individual Beans... option.

      The Define Policies and Roles for ... page lists the beans in the selected archive.

   c. Click the [Define Credential Mappers] link for the MDB for which you want to create a credential map.

The Credential Mappings page is displayed.

d. Click the Configure a new Credential Mapping link.

   The Create a new Credential Mapping... page is displayed.

e. Enter the WebLogic Server user name for the user you created in step 1, and the user name of the remote provider, and click Apply.

   The Credential Mapping Page displays the new credential map. The map associates the WebLogic Server user name with the remote provider user name, which is shown as a hyperlink.

f. Click the Remote User hyperlink in the Credential Map table.

   The *realm-->user* page displays the remote provider user name, and fields for entering and confirming the remote password.

g. Enter the password for the user in the Remote Password and Confirm Remote Password fields, and click Apply to save your changes.

Migration and Recovery for Clustered MDBs

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

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

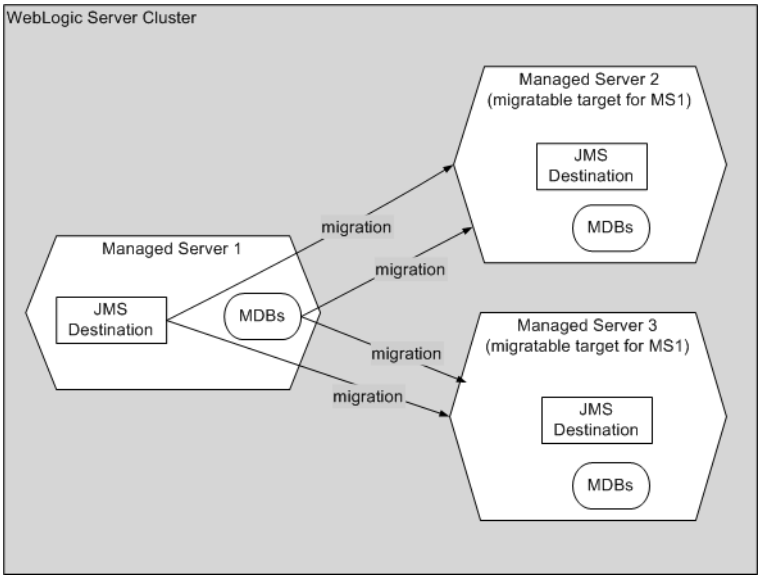
After a MDB application migrates to another server, it reconnects to the migrated JMS destination and begins to receive messages from the JMS destination again.

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

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

In Figure 7-12, Managed Servers 1, 2 and 3 are in the same cluster. Managed Server 2 and Managed Server 3 are configured as Managed Server 1's migratable targets. In the event that Managed Server 1 fails, the JMS destination and MDB migrate to the next available Managed Server. If a Managed Server on the target list is unavailable, the destination and MDBs migrate to the next available Managed Server on the target list. For instance if Managed Server 2 is unavailable with Managed Server 1 fails, the JMS destination and MDB application migrate to Managed Server 3.

Figure 7-12 Migration of JMS Destination



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

Nutshell Summary: Deployment Elements for MDBs

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

Table 7-2 summarizes the deployment elements in the `message-driven-descriptor` stanza of `weblogic-ejb-jar.xml`.

Table 7-3 lists key deployment elements for MDBs in the `ejb` stanza of `weblogic-application.xml`.

Table 7-4 lists key J2EE deployment elements for MDBs that you configure in the `message-driven` stanza of `ejb-jar.xml`.

Table 7-2 weblogic-ejb-jar.xml Deployment Elements for MDBs

Element	Description	Default
`connection-factory-jndi-name`	The JNDI name of the JMS ConnectionFactory that the message-driven EJB should look up to create its queues and topics. See "How to Set connection-factory-jndi-name" on page 7-16.	`weblogic.jms.MessageDrivenBeanConnectionFactory`
`destination-jndi-name`	The JNDI name used to associate a MDB with an actual JMS Queue or Topic deployed in the WebLogic Server JNDI tree. See "How to Set destination-jndi-name" on page 7-16.	N/A
`dispatch-policy`	This optional element allows you to specify a particular execute queue for the bean. See "MDBs and the Free Pool" on page 7-2.	
`initial-beans-in-free-pool`	The number of inactive instances of an MDB that exist in WebLogic Server when it is started. For more information, see "MDBs and the Free Pool" on page 7-2.	0
`initial-context-factory`	The initial `contextFactory` that the EJB container uses to create its connection factories. See "How to Set initial-context-factory" on page 7-16.	`weblogic.jndi.WLInitialContextFactory`
`jms-client-id`	The client id for the a message-driven bean associated with a durable subscriber topic. See "Defining the Client ID" in *Programming WebLogic JMS*.	ejb-name
`jms-polling-interval-seconds`	The number of seconds between attempts by the EJB container to reconnect to a JMS destination that has become unavailable. See "Migration and Recovery for Clustered MDBs" on page 7-27	The EJB container attempts reconnection every 10 seconds

Element	Description	Default
max-beans-in-free-pool	The maximum size of the free pool of inactive MDBs. See "MDBs and the Free Pool" on page 7-2	1000
provider-url	The URL provider to be used by the `InitialContext`. Typically, this is the host:port. See "How to Set provider-url" on page 7-15.	t3://localhost:7001
trans-timeout-seconds	The maximum duration for an EJB's container-initiated transactions, after which the transaction is rolled back. See "Configuring Transaction Management Strategy for an MDB" on page 7-13.	30 seconds

Table 7-3 weblogic-application.xml Elements for MDBs

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

Table 7-4 Key J2EE Deployment Elements for MDBs

Element	Description	Allowable Values
`acknowledge-mode`	Specifies JMS message acknowledgment semantics for the `onMessage` method of a message-driven bean that uses bean managed transaction demarcation.	• AUTO_ACKNOWLEDGE • DUPS_OK_ACKNOWLEDGE
`destination-type`	Specifies the type of the JMS destination—the Java interface expected to be implemented by the destination.	• javax.jms.Queue • javax.jms.Topic
`subscription-durability`	Specifies whether a JMS topic subscription is intended to be durable or nondurable.	• Durable • NonDurable
`transaction-type`	Specifies an enterprise bean's transaction management type. **Note:** If transaction-type is set to Container, trans-attribute must be set to Required.	• Bean • Container
`trans-attribute`	Specifies how the container must manage the transaction boundaries when delegating a method invocation to an enterprise bean's business method. Set to `Required` for container-managed transactions. For more information, see "Configuring Transaction Management Strategy for an MDB" on page 7-13.	• NotSupported • Support • Required • RequiresNew • Mandatory • Never

CHAPTER 8

Deployment Guidelines for Enterprise Java Beans

The following sections contain EJB-specific deployment guidelines. For deployment topics that are common to all deployable application units, you will see cross-references to topics in *Deploying WebLogic Server Applications,* a comprehensive guide to deploying WebLogic Server applications and modules.

- "Before You Deploy an EJB" on page 8-1
- "Understanding and Performing Deployment Tasks" on page 8-2
- "Deployment Guidelines for EJBs" on page 8-2

Before You Deploy an EJB

Before starting the deployment process you should have:

- Functional, tested bean code, in an exploded directory format or packaged in an archive file—a JAR for a stand-alone EJB or an EAR if the EJB is part of an enterprise application—along with the deployment descriptors. For production environments, BEA recommends that you package your application as an EAR.

  For an overview of the steps required to create and package an EJB, see "Overview of the EJB Development Process" on page 4-2.

- Configured the required deployment descriptors—`ejb-jar.xml` and `weblogic-ejb-jar.xml`, and, for entity EJBs that use container-managed persistence, `weblogic-cmp-jar.xml`.

To create EJB deployment descriptors, see "Generate Deployment Descriptors" on page 4-13 and "Edit Deployment Descriptors" on page 4-13.

Understanding and Performing Deployment Tasks

Table 8-1 is a guide to WebLogic Server documentation topics that help you make decisions about deployment strategies and provide instructions for performing deployment tasks. For EJB-specific deployment topics, see "Deployment Guidelines for EJBs" on page 8-2.

Table 8-1 Deployment Tasks and Topics

If You Want To....	See This Topic
Deploy in a development environment	"Deployment Topics for Developers" in *Deploying WebLogic Server Applications*.
Select a deployment tool	"Deployment Tools" in *Deploying WebLogic Server Applications*.
Determine appropriate packaging for a deployment	"Deployment Files" in *Deploying WebLogic Server Applications*.
Organizing EJB components in a split directory structure.	"EJBs" in *Developing WebLogic Server Applications.*
Select staging mode	"Staging Modes" in *Deploying WebLogic Server Applications*.
Perform specific deployment tasks	"Performing Common Deployment Tasks" in *Deploying WebLogic Server Applications*.

Deployment Guidelines for EJBs

The following sections provide guidelines for deploying EJBs.

Deploy EJBs as Part of an Enterprise Application

BEA recommends that you package and deploy your stand-alone EJB applications as part of an Enterprise application. An Enterprise application is a J2EE deployment unit that bundles together Web applications, EJBs, and Resource Adapters into a single deployable unit.

This is a BEA best practice, which allows for easier application migration, additions, and changes. Also, packaging your applications as part of an Enterprise application allows you to take

advantage of the split development directory structure, which provides a number of benefits over the traditional single directory structure. See "Introducing the Split Development Directory Structure" in *Developing WebLogic Server Applications.*

Deploy EJBs That Call Each Other in the Same Application

When an EJB in one application calls an EJB in another application, WebLogic Server passes method arguments by value, due to classloading requirements. When EJBs are in the same application, WebLogic Server can pass method arguments by reference; this improves the performance of method invocation because parameters are not copied.

For best performance, package components that call each other in the same application, and set `enable-call-by-reference` in `weblogic-ejb-jar.xml` to `True`. (By default, `enable-call-by-reference` is `False`).

Deploy Homogeneously to a Cluster

If your EJBs will run on a WebLogic Server cluster, BEA recommends that you deploy them homogeneously—to each Managed Server in the cluster. Alternatively, you can deploy an EJB to only to a single server in the cluster (that is, "pin" a module to a servers). This type of deployment is less common, and should be used only in special circumstances where pinned services are required. For more information, "Understanding Cluster Configuration and Application Deployment" in *Using WebLogic Server Clusters*.

Deploying Pinned EJBs to a Cluster

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 are copied to all server instances available to all nodes in the cluster.

For more information on pinned deployments, see "Deploying to a Single Server Instance (Pinned Deployment)" in *Using WebLogic Server Clusters*.

Redeploying an EJB

When you make changes to a deployed EJB's classes, you must redeploy the EJB. If you use automatic deployment, deployment occurs automatically when you restart WebLogic Server. Otherwise, you must explicitly redeploy the EJB.

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

When you redeploy, the classes currently loaded 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 classes.

When clients next acquire a reference to the EJB, their EJB method calls use the changed EJB classes.

You can redeploy an EJB that is standalone or part of an application, using the `weblogic.Deployer` tool or the Administration Console. For instructions, see "Redeploying or Stopping a Deployment Unit" in *Deploying WebLogic Server Applications*.

Redeploying an Individual EJB Implementation Class

During iterative development of an EJB application, you make many modifications to the EJB implementation class file, typically redeploying an EJB module multiple times during its development.

Prior to WebLogic Server 8.1, you could redeploy EJB implementation classes at the EJB module level, including all implementation and utility classes. Starting in WebLogic Server 8.1, you can redeploy an individual implementation class. For information about this feature, see "Individual EJB Classloader for Implementation Classes" in *Developing WebLogic Server Applications*.

To enable this feature, set the `enable-bean-class-redeploy` element to `True` in `weblogic-ejb-jar.xml`. This feature is not recommended for use in production environments. For considerations and limitations related to deploying an individual implementation class, feature, see "enable-bean-class-redeploy" on page A-24.

Understanding Warning Messages

To get information about a particular warning, use the `weblogic.GetMessage` tool. For example:

```
java weblogic.GetMessage -detail -id BEA-010202.
```

Disabling EJB Deployment Warning Messages

You can disable certain WebLogic Server warning messages that occur during deployment. You may find this useful if the messages provide information of which you are already aware.

For example, if the methods in your EJB makes calls by reference rather than by value, WebLogic Server generates this warning during deployment: `"Call-by-reference not enabled."`

You can use the `disable-warning` element in `weblogic-ejb-jar.xml` to disable certain messages. For a list of messages you can disable, and instructions for disabling the messages, see "disable-warning" on page A-19

APPENDIX A

weblogic-ejb-jar.xml Deployment Descriptor Reference

The following sections describe the EJB 2.0 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.

For information on the EJB 1.1 deployment descriptor elements, see Appendix F, "Important Information for EJB 1.1 Users".

- "Document Type Definitions and DOCTYPE Header Information" on page A-1
- "2.0 weblogic-ejb-jar.xml File Structure" on page A-4
- "Changes to weblogic-ejb-jar.xml in WebLogic Server 8.1" on page A-4
- "2.0 weblogic-ejb-jar.xml Elements" on page A-5

Document Type Definitions and DOCTYPE Header Information

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.

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 header refers to the location and version of the Document Type Definition (DTD) file for the deployment descriptor. Although this header references an external URL at java.sun.com, WebLogic Server contains its own copy of the DTD file, so your host server need not have access to the Internet. However, you must still include this `<!DOCTYPE...>` element in your `weblogic-ejb-jar.xml` and `weblogic-cmp-jar.xml` files, and have them reference the external URL because the version of the DTD contained in this element is used to identify the version of this deployment descriptor.

XML files with incorrect header information may yield error messages similar to the following, when used with a tool that parses the XML (such as `appc`):

```
SAXException: This document may not have the identifier
`identifier_name'
```

where `identifier_name` generally includes the invalid text from the `PUBLIC` element.

The correct text for the `PUBLIC` elements for the WebLogic-Server-specific `weblogic-ejb-jar.xml` file is listed, by WebLogic Server version, in Table A-1.

Table A-1 PUBLIC Elements of weblogic-ejb-jar.xml

WebLogic Server Version	XML File	PUBLIC Element String
8.1.x	`weblogic-ejb-jar.xml`	`'-//BEA Systems, Inc.//DTD WebLogic 8.1.0 EJB//EN' 'http://www.bea.com/servers/wls810/dtd/weblogic-ejb-jar.dtd'`
7.0.x	`weblogic-ejb-jar.xml`	`'-//BEA Systems, Inc.//DTD WebLogic 7.0.0 EJB//EN' 'http://www.bea.com/servers/wls700/dtd/weblogic-ejb-jar.dtd'`
6.1.x and 6.0.x	`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'`
5.1.0	`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 WebLogic-Server-specific weblogic-cmp-jar.xml file is listed, by WebLogic Server version, in Table A-2.

Table A-2 PUBLIC Elements of weblogic-cmp-jar.xml

WebLogic Server Version	XML File	PUBLIC Element String
8.1.x	weblogic-cmp-jar.xml	'-// BEA Systems, Inc.//DTD WebLogic 8.1.0 EJB RDBMS Persistence//EN' 'http://www.bea.com/servers/wls810/dtd/weblogic-rdbms20-persistence-810.dtd'
7.0.x	weblogic-cmp-jar.xml	'-// BEA Systems, Inc.//DTD WebLogic 7.0.0 EJB RDBMS Persistence//EN' 'http://www.bea.com/servers/wls700/dtd/weblogic-rdbms20-persistence-700.dtd'
6.1.x and 6.0.x	weblogic-cmp-jar.xml	'-// BEA Systems, Inc.//DTD WebLogic 6.0.0 EJB RDBMS Persistence//EN' 'http://www.bea.com/servers/wls600/dtd/weblogic-rdbms20-persistence-600.dtd'

See Appendix B, "weblogic-cmp-jar.xml Deployment Descriptor Reference," for more information on the weblogic-cmp-jar.xml file.

The correct text for the PUBLIC elements for the Sun-Microsystems-specific ejb-jar.xml file is listed, by Enterprise JavaBeans version, in Table A-3.

Table A-3 PUBLIC Elements of ejb-jar.xml

EJB Version	XML File	PUBLIC Element String
2.0	ejb-jar.xml	'-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN' 'http://java.sun.com/dtd/ejb-jar_2_0.dtd'
1.1	ejb-jar.xml	'-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN' 'http://www.java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'

2.0 weblogic-ejb-jar.xml File Structure

The WebLogic Server `weblogic-ejb-jar.xml` deployment descriptor file describes the elements that are unique to WebLogic Server.

The top level elements in the WebLogic Server 8.1 `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`
  - `clients-on-same-server`
  - `jndi-name`
- `security-role-assignment`
- `transaction-isolation`

Changes to weblogic-ejb-jar.xml in WebLogic Server 8.1

These changes were made to `weblogic-ejb-jar.xml` in WebLogic Server 8.1:

- "disable-warning" on page A-19 was added in Weblogic Server 8.1.
- "enable-bean-class-redeploy" on page A-24 was added in Weblogic Server 8.1.
- "enable-call-by-reference" on page A-25 was changed to have a default value of `False` in Weblogic Server 8.1.
- "passivate-as-principal-name" on page A-55 was added in Weblogic Server 8.1 SP01.
- "run-as-identity-principal" on page A-67 was deprecated in Weblogic Server 8.1.
- "run-as-role-assignment" on page A-69 was added in Weblogic Server 8.1.

2.0 weblogic-ejb-jar.xml Elements

This list of the elements in `weblogic-ejb-jar.xml` includes all elements that were supported in any service pack of WebLogic Server 8.1. The previous section, "Changes to weblogic-ejb-jar.xml in WebLogic Server 8.1" lists elements that were new, changed, or deprecated in Weblogic Server 8.1, or a subsequent service pack.

- "allow-concurrent-calls" on page A-9
- "allow-remove-during-transaction" on page A-10
- "cache-between-transactions" on page A-10
- "cache-type" on page A-11
- "client-authentication" on page A-12
- "client-cert-authentication" on page A-12
- "clients-on-same-server" on page A-13
- "concurrency-strategy" on page A-14
- "confidentiality" on page A-15
- "connection-factory-jndi-name" on page A-16
- "create-as-principal-name" on page A-16
- "delay-updates-until-end-of-tx" on page A-17
- "description" on page A-18
- "destination-jndi-name" on page A-19

- “disable-warning” on page A-19
- “dispatch-policy” on page A-20
- “ejb-local-reference-description” on page A-21
- “ejb-name” on page A-22
- “ejb-reference-description” on page A-23
- “ejb-ref-name” on page A-23
- “enable-bean-class-redeploy” on page A-24
- “enable-call-by-reference” on page A-25
- “enable-dynamic-queries” on page A-26
- “entity-cache” on page A-26
- “entity-cache-name” on page A-27
- “entity-cache-ref” on page A-28
- “entity-clustering” on page A-29
- “entity-descriptor” on page A-30
- “estimated-bean-size” on page A-31
- “externally-defined” on page A-32
- “finders-load-bean” on page A-32
- “global-role” on page A-33
- “home-call-router-class-name” on page A-34
- “home-is-clusterable” on page A-35
- “home-load-algorithm” on page A-36
- “idempotent-methods” on page A-37
- “identity-assertion” on page A-38
- “idle-timeout-seconds” on page A-39
- “iiop-security-descriptor” on page A-40

- “initial-beans-in-free-pool” on page A-41
- “initial-context-factory” on page A-41
- “integrity” on page A-42
- “invalidation-target” on page A-42
- “is-modified-method-name” on page A-43
- “isolation-level” on page A-44
- “jms-client-id” on page A-45
- “jms-polling-interval-seconds” on page A-46
- “jndi-name” on page A-47
- “local-jndi-name” on page A-48
- “max-beans-in-cache” on page A-48
- “max-beans-in-free-pool” on page A-49
- “message-driven-descriptor” on page A-50
- “method” on page A-51
- “method-intf” on page A-52
- “method-name” on page A-53
- “method-param” on page A-54
- “method-params” on page A-55
- “passivate-as-principal-name” on page A-55
- “persistence” on page A-56
- “persistence-use” on page A-57
- “persistent-store-dir” on page A-58
- “pool” on page A-58
- “principal-name” on page A-59
- “provider-url” on page A-60

- “read-timeout-seconds” on page A-60
- “reference-descriptor” on page A-61
- “remote-client-timeout” on page A-62
- “remove-as-principal-name” on page A-63
- “replication-type” on page A-64
- “res-env-ref-name” on page A-64
- “res-ref-name” on page A-65
- “resource-description” on page A-65
- “resource-env-description” on page A-66
- “role-name” on page A-67
- “run-as-identity-principal” on page A-67
- “run-as-principal-name” on page A-68
- “run-as-role-assignment” on page A-69
- “security-permission” on page A-72
- “security-permission-spec” on page A-72
- “security-role-assignment” on page A-73
- “session-timeout-seconds” on page A-74
- “stateful-session-cache” on page A-75
- “stateful-session-clustering” on page A-76
- “stateful-session-descriptor” on page A-77
- “stateless-bean-call-router-class-name” on page A-77
- “stateless-bean-is-clusterable” on page A-78
- “stateless-bean-load-algorithm” on page A-79
- “stateless-bean-methods-are-idempotent” on page A-80
- “stateless-clustering” on page A-80

- "stateless-session-descriptor" on page A-81
- "transaction-descriptor" on page A-82
- "transaction-isolation" on page A-82
- "transport-requirements" on page A-83
- "trans-timeout-seconds" on page A-84
- "type-identifier" on page A-84
- "type-storage" on page A-85
- "type-version" on page A-86
- "weblogic-ejb-jar" on page A-87
- "weblogic-enterprise-bean" on page A-88

allow-concurrent-calls

Range of values:	`True | False`
Default value:	`False`
Parent elements:	`weblogic-enterprise-bean` `stateful-session-descriptor`

Function

Specifies whether a stateful session bean instance allows concurrent method calls. By default, `allows-concurrent-calls` is `False`, in accordance with the EJB specification, and WebLogic Server will throw a `RemoteException` when a stateful session bean instance is currently handling a method call and another (concurrent) method call arrives on the server.

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 A-77.

allow-remove-during-transaction

Range of values:	`True` | `False`
Default value:	`False`
Parent elements:	`weblogic-enterprise-bean` `stateful-session-descriptor`

Function

Specifies that the `remove` method on a stateful session bean can be invoked within a transaction context.

Note: Stateful session beans implementing the `Synchronization` interface should not use this tag and then call `remove` before the transaction ends. If this is done the EJB container will not invoke the synchronization callbacks.

Example

See "stateful-session-descriptor" on page A-77.

cache-between-transactions

Range of values:	`True` | `False`
Default value:	`False`
Parent elements:	`weblogic-enterprise-bean`, `entity-descriptor`, `entity-cache` or `entity cache-ref`

Function

Formerly the `db-is-shared` element, specifies whether the EJB container will cache the persistent data of an entity bean across (between) transactions.

Specify `True` to enable the EJB container to perform long term caching of the data. Specify `False` to enable the EJB container to perform short term caching of the data.

A Read-Only bean ignores the value of the `cache-between-transactions` element because WebLogic Server always performs long term caching of Read-Only data.

See "Limiting Database Reads with cache-between-transactions (Long-Term Caching)" on page 6-17 for more information.

Example

See "persistence" on page A-56.

cache-type

Range of values:	`NRU | LRU`
Default value:	`NRU`
Parent elements:	`weblogic-enterprise-bean` `stateful-session-cache`

Function

Specifies the order in which EJBs are removed from the cache. The values are:

- Least recently used (LRU)
- Not recently used (NRU)

  The minimum cache size for NRU is 8. If max-beans-in-cache is less than 3, WebLogic Server uses a value of 8 for `max-beans-in-cache`.

Example

```
<stateful-session-cache>
     <cache-type>NRU</cache-type>
</stateful-session-cache>
```

client-authentication

Range of values:	`none | supported | required`
Default value:	
Parent elements:	`weblogic-enterprise-bean` `iiop-security-descriptor`

Function

Specifies whether the EJB supports or requires client authentication.

Example

See "iiop-security-descriptor" on page A-40.

client-cert-authentication

Range of values:	`none | supported | required`
Default value:	

Parent elements:	`weblogic-enterprise-bean`
	`iiop-security-descriptor`
	`transport-requirements`

Function

Specifies whether the EJB supports or requires client certificate authentication at the transport level.

Example

See "transport-requirements" on page A-83.

clients-on-same-server

Range of values:	`True | False`
Default value:	`False`
Parent elements:	`weblogic-enterprise-bean`

Function

Determines whether WebLogic Server sends JNDI announcements for this EJB when it is deployed. When this attribute is `False` (the default), a WebLogic Server cluster automatically updates its JNDI tree to indicate the location of this EJB on a particular server. This ensures that all clients can access the EJB, even if the client is not collocated on the same server.

You can set `clients-on-same-server` to `True` when you know that all clients that will access this EJB will do so from the same server on which the bean is deployed. In this case, a WebLogic Server cluster does not send JNDI announcements for this EJB when it is deployed. Because JNDI updates in a cluster utilize multicast traffic, setting `clients-on-same-server` to `True` can reduce the startup time for very large clusters.

See "Optimization for Collocated Objects" in *Using WebLogic Server Clusters* for more information on collocated EJBs.

Example

```
<weblogic-enterprise-bean>
       <ejb-name>AccountBean</ejb-name>
       ...
       <clients-on-same-server>True</clients-on-same-server>
</weblogic-enterprise-bean>
```

concurrency-strategy

Range of values:	`Exclusive` | `Database` | `ReadOnly` | `Optimistic`
Default value:	`Database`
Parent elements:	`weblogic-enterprise-bean`, `entity-descriptor`, `entity-cache` or `entity-cache-ref`

Function

Specifies how the container should manage concurrent access to an entity bean. Set this element to one of four 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 are blocked 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 allocates a separate entity bean instance and allows locking and caching to be handled by the database. This is the default option.
- `ReadOnly` is used for read-only entity beans. Activates a new instance for each transaction so that requests proceed in parallel. WebLogic Server calls `ejbLoad()` for `ReadOnly` beans are based on the `read-timeout-seconds` parameter.
- `Optimistic` holds no locks in the EJB container or database during a transaction. The EJB container verifies that none of the data updated by a transaction has changed before committing the transaction. If any updated data changed, the EJB container rolls back the transaction.

See "Choosing a Concurrency Strategy" on page 6-35 for more information on the `Exclusive` and `Database` locking behaviors. See "Read-Write versus Read-Only Entity Beans" on page 3-4 for more information about `read-only` entity EJBs.

Example

```
<weblogic-enterprise-bean>
       <ejb-name>AccountBean</ejb-name>
```

```
        <entity-descriptor>
               <entity-cache>
                      <concurrency-strategy>ReadOnly</concurrency-strategy>
               </entity-cache>
        </entity-descriptor>
</weblogic-enterprise-bean>
```

confidentiality

Range of values:	`none` | `supported` | `required`
Default value:	

Parent elements:	`weblogic-enterprise-bean`
	`iiop-security-descriptor`
	`transport-requirements`

Function

Specifies the transport confidentiality requirements for the EJB. Using the `confidentiality` element ensures that the data is sent between the client and server in such a way as to prevent other entities from observing the contents.

Example

See "transport-requirements" on page A-83.

connection-factory-jndi-name

Range of values:	Valid JNDI name.
Default value:	If not specified, the default is `weblogic.jms.MessageDrivenBeanConnectionFactory`, which must have been declared in the `JMSConnectionFactory` stanza in `config.xml`
Parent elements:	`weblogic-enterprise-bean` `message-driven-descriptor`

Function

Specifies the JNDI name of the JMS Connection Factory that a message-driven EJB looks up to create its queues and topics. See "Configuring MDBs for Destinations" on page 7-14 and "How to Set connection-factory-jndi-name" on page 7-16.

Example

```
<message-driven-descriptor>
     <connection-factory-jndi-name>
     java:comp/env/jms/MyConnectionFactory
     </connection-factory-jndi-name>
</message-driven-descriptor>
```

create-as-principal-name

Range of values:	Valid principal name.
Default value:	n/a
Parent elements:	`weblogic-enterprise-bean`

Function

Introduced in WebLogic Server 8.1 SP01, specifies the principal to be used in situations where `ejbCreate` would otherwise run with an anonymous principal. Under such conditions, the choice of which principal to run as is governed by the following rule:

if `create-as-principal-name` is set
then use that principal
else
    If a `run-as` role has been specified for the bean in `ejb-jar.xml`
    then use a principal according to the rules for setting the `run-as-role-assignment`
    else
        run `ejbCreate` as an anonymous principal.

The `create-as-principal-name` element only needs to be specified if operations within `ejbCreate` require more permissions than the anonymous principal would have.

This element effects the `ejbCreate` methods of stateless session beans and message-driven beans.

See also "remove-as-principal-name" on page A-63, "passivate-as-principal-name" on page A-55, and "principal-name" on page A-59.

delay-updates-until-end-of-tx

Range of values:	`True | False`
Default value:	`True`

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 `TransactionReadCommittedUncommitted`, you may want to allow other database users to view the intermediate results of in-progress transactions. In this case, set `delay-updates-to-end-of-tx` to `False` to update the bean's persistent store at the conclusion of each method invoke. See "Understanding ejbLoad() and ejbStore() Behavior" on page 6-4 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

```
<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

Parent elements:	`weblogic-ejb-jar`
	and
	`weblogic-ejb-jar`
	`transaction-isolation`
	`method`
	and
	`weblogic-ejb-jar`
	`idempotent-methods`
	`method`

Function

Describes the parent element.

Example

```
<dscription>Contains a description of parent element</description>
```

destination-jndi-name

Range of values:	Valid JNDI name.
Default value:	n/a
Parent elements:	`weblogic-enterprise-bean` `message-driven-descriptor`

Function

Specifies the JNDI name used to associate a message-driven bean with an actual JMS Queue or Topic deployed in the WebLogic Server JNDI tree.

Example

See "message-driven-descriptor" on page A-50.

disable-warning

Range of values:	BEA-010001 | BEA-010054 | BEA-010200 | BEA-010202
Default value:	n/a
Parent elements:	`weblogic-ejb-jar`

Function

Specifies that WebLogic Server should disable the warning message whose ID is specified. Set this element to one of four values:

- `BEA-010001`—Disables this warning message: "EJB class loaded from system classpath during deployment."
- `BEA-010054`—Disables this warning message: "EJB class loaded from system classpath during compilation."
- `BEA-010200`—Disables this warning message: "EJB impl class contains a public static field, method or class."

- `BEA-010202`—Disables this warning message: “Call-by-reference not enabled.”

Example

To disable the warning message: “Call-by-reference not enabled”, set `<disable-warning>`, as shown below.

```
<disable-warning>BEA-010202</disable-warning>
```

dispatch-policy

Range of values:	Valid execute queue name
Default value:	n/a
Parent elements:	`weblogic-enterprise-bean`

Function

Designates which server execute thread pool the EJB should run in. Dispatch polices are supported for all types of beans, including entity, session and message-driven.

If no `dispatch-policy` is specified, or the specified `dispatch-policy` refers to a nonexistent server execute thread pool, then the server's default execute thread pool is used instead.

WebLogic Server ignores `dispatch-policy` if the host server instance does not have an execute thread queue bearing a corresponding name.

If a message-driven bean (MDB) is driven by a foreign (non-WebLogic) destination source, WebLogic Server might ignore `dispatch-policy`, as the MDB may instead run in the foreign provider's threading mechanism. For example, for the IBM WebSphere MQSeries messaging software, `dispatch-policy` is not honored for non-transactional queues; instead the application code runs in an MQSeries thread. For MQSeries transactional queues, and both non-transactional and transactional topics, `dispatch-policy` is honored.

The maximum number of concurrently running MDB instances is designated by a combination of max-beans-in-free-pool and `dispatch-policy` values.

- For the default thread pool the algorithm is:

```
maxConcurrentMDBs = Min(max-beans-free-pool,
default-thread-pool-size/2+1).
```

MDBs that run in the default thread pool limit their concurrency to half the thread pool size plus one to prevent deadlocks with other services and applications that share the default thread pool.

- For all other thread pools, the algorithm is

  `maxConcurrentMDBs = Min(max-beans-free-pool, thread-pool-size).`

Example

```
<dispatch-policy>queue_name</dispatch-policy>
```

ejb-local-reference-description

Range of values:	n/a
Default value:	n/a
Parent elements:	`weblogic-enterprise-bean` `reference-descriptor`

Function

Maps the JNDI name of an EJB in the WebLogic Server instance that is referenced by the bean in the `ejb-local-ref` element.

Example

```
<ejb-local-reference-description>
       <ejb-ref-name>AdminBean</ejb-ref-name>
       <jndi-name>payroll.AdminBean</jndi-name>
</ejb-local-reference-description>
```

ejb-name

Range of values:	Name, which conform to the lexical rules for an NMTOKEN, of an EJB that is defined in `ejb-jar.xml`
Default value:	n/a

Parent elements:	`weblogic-enterprise-bean`
	and
	`weblogic-enterprise-bean`
	`method`

Function

Specifies an enterprise bean's name, using the same name for the bean that is specified in `ejb-jar.xml`. 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 architected relationship between the `ejb-name` in the deployment descriptor and the JNDI name that the Deployer will assign to the enterprise bean's home.

Note: Not recommended in `weblogic-enterprise-bean`. For more information, see "Using EJB Links" on page 4-7.

Example

See "method" on page A-51.

ejb-reference-description

Range of values:	n/a
Default value:	n/a
Parent elements:	`weblogic-enterprise-bean` `reference-descriptor`

Function

Maps the JNDI name of an EJB in WebLogic Server to the name by which it is specified in the `ejb-ref-name` element in `ejb-jar.xml`.

Example

```
<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:	Valid `ejb-ref-name` specified in the associated `ejb-jar.xml` file.
Default value:	n/a

Parent elements:	`weblogic-enterprise-bean`
	`reference-description`
	`ejb-reference-description`

Function

Specifies a resource reference name. This element is the reference that the EJB provider places within the `ejb-jar.xml` deployment file.

Example

```
<reference-descriptor>
       <ejb-reference-description>
              <ejb-ref-name>AdminBean</ejb-ref-name>
              <jndi-name>payroll.AdminBean</jndi-name>
       </ejb-reference-description>
</reference-descriptor>
```

enable-bean-class-redeploy

Range of values:	`True | False`
Default value:	`False`
Parent elements:	`weblogic-enterprise-jar`

Function

By default, the EJB implementation class is loaded in the same classloader as the rest of the EJB classes. When the `enable-bean-class-redeploy` element is enabled, the implementation class, along with its super classes, gets loaded in a child classloader of the EJB module classloader. This allows the EJB implementation class to be redeployed without redeploying the entire EJB module.

There are some potential problems with loading the bean class in a child classloader. First, the bean class will no longer be visible to any classes loaded in the parent classloader, so those classes cannot refer to the bean class or errors will occur. Also, the bean class will not be able to invoke any package protected methods on any classes loaded in a different classloader. So, if your bean class invokes a helper class in the same package, the helper class methods must be declared public or `IllegalAccessErrors` will result.

Note: Two-phase deployment must be used for this feature to be enabled. WebLogic Server ignores the `enable-bean-class-redeploy` setting will be ignored if two-phase deployment is not used. For information on two-phase deployment, see "Two-Phase Deployment Protocol" in *Deploying WebLogic Server Applications*.

Example

The following XML stanza enables redeployment of an individual bean class:

```
<enable-bean-class-redeploy>True</enable-bean-class-redeploy>
```

enable-call-by-reference

Range of values:	`True | False`
Default value:	`False`
Parent elements:	`weblogic-enterprise-bean`

Function

When `enable-call-by-reference` is `False`, parameters to the EJB methods are copied—or passed by value—regardless of whether the EJB is called remotely or from with the same EAR.

When `enable-call-by-reference` is `True`, EJB methods called from within the same EAR file or standalone JAR file will pass arguments by reference. This improves the performance of method invocation since parameters are not copied.

Note: Method parameters are *always* passed by value when an EJB is called remotely.

Example

```
<weblogic-enterprise-bean>
     <entity-descriptor>
          <ejb-name>AccountBean</ejb-name>
          ...
          <enable-call-by-reference>False</enable-call-by-reference>
     </entity-descriptor>
</weblogic-enterprise-bean>
```

enable-dynamic-queries

Range of values:	`True | False`
Default value:	`True`
Parent elements:	`weblogic-enterprise-bean` `entity-descriptor`

Function

Set to `True` to enable dynamic queries. Dynamic queries are only available for use with EJB 2.0 CMP beans.

Example

```
<enable-dynamic-queries>True</enable-dynamic-queries>
```

entity-cache

Range of values:	n/a
Default value:	n/a
Parent elements:	`weblogic-enterprise-bean` `entity-descriptor`

Function

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 "Understanding Entity Caching" on page 6-3 for more information.

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>
       <persistence>...</persistence>
       <entity-clustering>...</entity-clustering>
</entity-descriptor>
```

entity-cache-name

Range of values:	Name assigned to an application level entity cache in the `weblogic-application.xml` file.
Default value:	n/a

Parent elements:	`weblogic-enterprise-bean`
	`entity-descriptor`
	`entity-cache-ref`

Function

Refers to an application level entity cache that the entity bean uses. An application level cache is a cache that may be shared by multiple entity beans in the same application. The value you specify for `entity-cache-name` must match the name assigned to an application level entity cache in the `weblogic-application.xml` file

For more information about the `weblogic-application.xml` file, see "Enterprise Application Deployment Descriptor Elements" in *Developing WebLogic Server Applications*.

Example

See "entity-cache-ref" on page A-28.

entity-cache-ref

Range of values:	n/a
Default value:	n/a
Parent elements:	`weblogic-enterprise-bean`, `entity-descriptor`

Function

Refers to an application level entity cache which can cache instances of multiple entity beans that are part of the same application. Application level entity caches are declared in the `weblogic-application.xml` file.

Use "concurrency-strategy" on page A-14 to define the type of concurrency you want the bean to use. The `concurrency-strategy` must be compatible with the application level cache's caching strategy. For example, an `Exclusive` cache only supports beans with a `concurrency-strategy` of `Exclusive`. A `MultiVersion` cache supports the `Database`, `ReadOnly`, and `Optimistic` concurrency strategies.

Example

```
<entity-cache-ref>
       <entity-cache-name>AllEntityCache</entity-cache-name>
       <read-timeout-seconds>600</read-timeout-seconds>
       <cache-between-transactions>true</cache-between-transactions>
       <concurrency-strategy>ReadOnly</concurrency-strategy>
       <estimated-bean-size>20</estimated-bean-size>
</entity-cache-ref>
```

entity-clustering

Range of values:	n/a
Default value:	n/a
Parent elements:	`weblogic-enterprise-bean` `entity-descriptor`

Function

Specifies how an entity bean will be replicated in a WebLogic cluster:

- `home-is-clusterable`
- `home-load-algorithm`
- `home call-router-class-name`
- `use-serverside-stubs`

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-name>
       <use-servside-stubs>True</use-servside-stubs>
</entity-clustering>
```

entity-descriptor

Range of values:	n/a
Default value:	n/a
Parent elements:	`weblogic-enterprise-bean`

Function

Specifies the following deployment parameters that are applicable to an entity bean:

- `pool`
- `entity-cache` or `entity-cache-ref`
- `persistence`
- `entity-clustering`
- `invalidation-target`
- `enable-dynamic-queries`

Example

```
<entity-descriptor>
       <entity-cache>...</entity-cache>
       <persistence>...</persistence>
       <entity-clustering>...</entity-clustering>
       <invalidation-target>...</invalidation-target>
       <enable-dynamic-queries>...</enable-dynamic-queries>
</entity-descriptor>
```

estimated-bean-size

Range of values:	n/a
Default value:	n/a
Parent elements:	`weblogic-enterprise-bean` `entity-descriptor`

Function

Specifies the estimated average size of the instances of an entity bean in bytes. This is the average number of byte of memory that is consumed by each instance.

Use the `estimated-bean-size` element when the application level cache you use to cache beans is also specified in terms of bytes and megabytes.

Although you may not know the exact number of bytes consumed by the entity bean instances, specifying a size allows you to give some relative weight to the beans that share a cache at one time.

For example, suppose bean A and bean B share a cache, called AB-cache, that has a size of 1000 bytes and the size of A is 10 bytes and the size of B is 20 bytes, then the cache can hold at most 100 instances of A and 50 instances of B. If 100 instance s of A are cached, this implies that 0 instances of B are cached.

Example

See “entity-cache-ref” on page A-28.

externally-defined

Range of values:	`True` | `False`
Default value:	
Parent elements:	`weblogic-ejb-jar` `security-role-assignment`

Function

Indicates that a particular security role is defined externally in a security realm, outside of the deployment descriptor. Because the security role and its principal-name mapping is defined elsewhere, principal-names are not to be specified in the deployment descriptor. This tag is used as an indicative placeholder instead of a set of `principal-name` elements.

finders-load-bean

Range of values:	`True` | `False`
Default value:	`True`

Parent elements:	`weblogic-enterprise-bean`
	`entity-descriptor`
	`persistence`

Function

Valid only for CMP entity EJBs. 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 automatically load the bean into the cache until the first method invocation; this behavior is consistent with the EJB 1.1 specification.

Example

```
<entity-descriptor>
       <persistence>
```

```
            <finders-load-bean>True</finders-load-bean>
        </persistence>
</entity-descriptor>
```

global-role

Range of values:	`True | False`
Default value:	`True`
Parent elements:	`weblogic-ejb-jar` `security-role-assignment`

Function

The `global-role` element is deprecated and will be removed in a future version of WebLogic. Use the `externally-defined` element instead.

The `global-role` element indicates that a particular security role is defined "globally" in a security realm. Because the security role and its principal-name mapping is defined elsewhere, principal-names are not to be specified in the deployment descriptor. This tag is used as an indicative placeholder instead of a set of `principal-name` elements.

home-call-router-class-name

Range of values:	Valid name of a custom class.
Default value:	n/a

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-session-clustering`

Function

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 A-29 and "stateful-session-clustering" on page A-76.

home-is-clusterable

Range of values:	`True | False`
Default value:	`True`

Parent elements:	`weblogic-enterprise-bean, entity-descriptor, entity-clustering`
	`and`
	`weblogic-enterprise-bean stateful-session-descriptor stateful-session-clustering`
	`and`
	`weblogic-enterprise-bean stateful-session-descriptor stateless-clustering`

Function

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.

Example

See "entity-clustering" on page A-29.

home-load-algorithm

Range of values:	`round-robin` | `random` | `weight-based` | `RoundRobinAffinity` | `RandomAffinity` | `WeightBasedAffinity`
Default value:	Value of `weblogic.cluster.defaultLoadAlgorithm`

Parent elements:	`weblogic-enterprise-bean,`
	`  entity-descriptor,`
	`  entity-clustering`
	and
	`weblogic-enterprise-bean`
	`  stateful-session-descriptor`
	`  stateful-session-clustering`
	and
	`weblogic-enterprise-bean,`
	`  entity-descriptor,`
	`  entity-clustering`

Function

Specifies the algorithm to use for load balancing between replicas of the EJB home in a cluster. If this element is not defined, WebLogic Server uses the algorithm specified by the server element, `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.
- `round-robin-affinity`—server affinity governs connections between external Java clients and server instances; round robin load balancing is used for connections between server instances.

- `weight-based-affinity`—server affinity governs connections between external Java clients and server instances; weight-based load balancing is used for connections between server instances.
- `random-affinity`—server affinity governs connections between external Java clients and server instances; random load balancing is used for connections between server instances.

For more information, see "Load Balancing for EJBs and RMI Objects" in *Using WebLogic Server Clusters*.

Example

See "entity-clustering" on page A-29 and "stateful-session-clustering" on page A-76.

idempotent-methods

Range of values:	n/a
Default value:	n/a
Parent elements:	`weblogic-ejb-jar`

Function

Defines list of methods of a clustered EJB which are written in such a way 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 compiled on the failed server. When you enable idempotent-methods for a method, the EJB stub can automatically recover from any failure as long as it can reach another server hosting the EJB.

Clustering must be enabled for the EJB. To enable clustering, see "entity-clustering" on page A-29, "stateful-session-clustering" on page A-76, and "stateless-clustering" on page A-80.

The methods on stateless session bean homes and read-only entity beans are automatically set to be idempotent. It is not necessary to explicitly specify them as idempotent.

Example

```
<idempotent-method>
       <method>
              <description>...</description>
```

```
            <ejb-name>...</ejb-name>
            <method-intf>...</method-intf>
            <method-name>...</method-name>
            <method-params>...</method-params>
        </method>
</idempotent-method>
```

identity-assertion

Range of values:	`none	supported	required`
Default value:	
Parent elements:	`weblogic-enterprise-bean` `iiop-security-descriptor`

Function

Specifies whether the EJB supports or requires identity assertion.

Example

See "iiop-security-descriptor" on page A-40.

idle-timeout-seconds

Range of values:	1 to *maxSeconds*, where *maxSeconds* is the maximum value of an int.
Default value:	600
Parent elements:	weblogic-enterprise-bean, entity-descriptor, entity-cache and weblogic-enterprise-bean, stateful-session-descriptor, stateful-session-cache

Function

Defines the maximum length of time a stateful session EJB should remain in the 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 "Caching and Passivating Stateful Session EJBs" on page 5-4 for more information.

Note: Although idle-timeout-seconds appears in the entity-cache stanza, WebLogic Server 8.1 SP1 and SP2 do not use its value in managing the lifecycle of entity EJBs—in those service packs, 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>
```

iiop-security-descriptor

Range of values:	n/a
Default value:	n/a
Parent elements:	`weblogic-enterprise-bean`

Function

Specifies security configuration parameters at the bean level. These parameters determine the IIOP security information contained in the IOR.

Example

```
<iiop-security-descriptor>
        <transport-requirements>...</transport-requirements>
        <client-authentication>supported<client-authentication>
        <identity-assertion>supported</identity-assertion>
</iiop-security-descriptor>
```

initial-beans-in-free-pool

Range of values:	0 to *maxBeans*
Default value:	0
Parent elements:	weblogic-enterprise-bean, stateless-session-descriptor or message-bean-descriptor or 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 A-58.

initial-context-factory

Range of values:	Valid name of an initial context factory.
Default value:	weblogic.jndi.WLInitialContextFactory
Parent elements:	weblogic-enterprise-bean message-driven-descriptor

Function

Specifies the initial context factory used by the JMS provider to create initial context. See "Configuring MDBs for Destinations" on page 7-14 and "How to Set initial-context-factory" on page 7-16.

Example

```
<message-driven-descriptor>
    <initial-context-factory>fiorano.jms.rtl.FioranoInitialContextFactory
    </initial-context-factory>
</message-driven-descriptor>
```

integrity

Range of values:	`none` | `supported` | `required`
Default value:	

Parent elements:	`weblogic-enterprise-bean`
	`iiop-security-descriptor`
	`transport-requirements`

Function

Specifies the transport integrity requirements for the EJB. Using the `integrity` element ensures that the data is sent between the client and server in such a way that it cannot be changed in transit.

Example

See "transport-requirements" on page A-83.

invalidation-target

Range of values:	n/a
Default value:	n/a
Parent elements:	`weblogic-enterprise-bean` `entity-descriptor`

Function

Specifies a Read-Only entity EJB that should be invalidated when this container-managed persistence entity EJB has been modified.

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.

Example

```
<invalidation-target>
       <ejb-name>StockReaderEJB</ejb-name>
</invalidation-target>
```

is-modified-method-name

Range of values:	Valid entity EJB method name
Default value:	n/a
Parent elements:	`weblogic-enterprise-bean`, `entity-descriptor`, `persistence`

Function

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.

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

```
<entity-descriptor>
       <persistence>
       <is-modified-method-name>semidivine</is-modified-method-name>
```

```
        </persistence>
    </entity-descriptor>
```

isolation-level

Range of values:	`TransactionSerializable | TransactionReadCommitted | TransactionReadUncommitted | TransactionRepeatableRead | TransactionReadCommittedForUpdate | TransactionReadCommittedForUpdateNoWait`
Default value:	Default value of the underlying database.
Parent elements:	`weblogic-ejb-jar` `transaction-isolation`

Function

Defines method-level transaction isolation settings for an EJB. Allowable values include:

- `TRANSACTION_SERIALIZABLE`—Simultaneously executing this transaction multiple times has the same effect as executing the transaction multiple times in a serial fashion.
- `TRANSACTION_READ_COMMITTED`—The transaction can view only committed updates from other transactions
- `TRANSACTION_READ_UNCOMMITTED`—The transaction can view uncommitted 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.

These addition values are supported only for Oracle databases, and only for container-managed persistence (CMP) EJBs:

- `TRANSACTION_READ_COMMITTED_FOR_UPDATE`— Supported only for Oracle databases, for container-managed persistence (CMP) EJBs only. This value sets the isolation level to `TRANSACTION_READ_COMMITTED`, and for the duration of the transaction, all SQL `SELECT` statements executed in any method are executed with `FOR UPDATE` appended to them. This causes the secluded rows to be locked for update. If Oracle cannot lock the rows affected by the query immediately, then it waits until the rows are free. This condition remains in effect until the transaction does a `COMMIT` or `ROLLBACK`

This isolation level can be used to avoid the error:

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

which can occur when using the `TRANSACTION_SERIALIZABLE` isolation level with Oracle databases.

- `TRANSACTION_READ_COMMITTED_FOR_UPDATE_NO_WAIT`—Supported only for Oracle databases, for container-managed persistence (CMP) EJBs only.

  This value sets the isolation level to `TRANSACTION_READ_COMMITTED`, and for the duration of the transaction, all SQL `SELECT` statements executed in any method are executed with FOR UPDATE NO WAIT appended to them. This causes the selected rows to be locked for update.

  In contrast to the `TRANSACTION_READ_COMMITTED_FOR_UPDATE` setting, `TRANSACTION_READ_COMMITTED_FOR_UPDATE_NO_WAIT` causes the Oracle DBMS to `NOT WAIT` if the required locks cannot be acquired immediately—the affected `SELECT` query will fail and an exception will be thrown by the Container.

Refer to your database documentation for more information support for different isolation levels.

Example

See "transaction-isolation" on page A-82.

jms-client-id

Range of values:	n/a
Default value:	`ejb-name` for the EJB
Parent elements:	`message-driven-descriptor`

Function

Specifies a client ID for the MDB when it connects to a JMS destination. Required for durable subscriptions to JMS topics.

If you specify the connection factory that the MDB uses in `connection-factory-jndi-name`, the client ID can be defined in the `ClientID` element of the associated `JMSConnectionFactory` stanza in `config.xml`.

If `JMSConnectionFactory` in `config.xml` does not specify a `ClientID`, or if you use the default connection factory, (you do not specify `connection-factory-jndi-name`) the message-driven bean uses the `jms-client-id` value as its client id.

Example

```
<jms-client-id>MyClientID</jms-client-id>
```

jms-polling-interval-seconds

Range of values:	n/a
Default value:	10 seconds
Parent elements:	`message-driven-descriptor`

Function

Specifies the number of seconds between each attempt 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, then 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 up again, the message-driven bean can again receive messages.

Example

```
<jms-polling-interval-seconds>5</jms-polling-interval-seconds>
```

jndi-name

Range of values:	Valid JNDI name
Default value:	n/a

Parent elements:	`weblogic-enterprise-bean`
	`and`
	`weblogic-enterprise-bean`
	`  reference-descriptor`
	`  resource-description`
	`and`
	`weblogic-enterprise-bean`
	`  reference-descriptor`
	`  ejb-reference-description`
	`and`
	`weblogic-enterprise-bean`
	`  reference-descriptor`
	`  ejb-local-reference-description`

Function

Specifies the JNDI name of an actual EJB, resource, or reference available in WebLogic Server.

Assigning a JNDI name to a bean is not recommended. Global JNDI names generate heavy multicast traffic during clustered server startup. See "Using EJB Links" on page 4-7 for the better practice.

Example

See "resource-description" on page A-65 and "ejb-reference-description" on page A-23.

local-jndi-name

Range of values:	Valid JNDI name
Default value:	n/a
Parent elements:	`weblogic-enterprise-bean`

Function

JNDI name for a bean's local home. If a bean has both a remote and a local home, then it may be assigned two JNDI names; one for each home.

Example

```
<local-jndi-name>weblogic.jndi.WLInitialContext
</local-jndi-name>
```

max-beans-in-cache

Range of values:	`1` to *`maxBeans`*
Default value:	`1000`

Parent elements:	`weblogic-enterprise-bean,`
	`entity-descriptor,`
	`entity-cache`
	`and`
	`weblogic-enterprise-bean`
	`stateful-session-descriptor`
	`stateful-session-cache`

Function

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 recently been used by a client. `max-beans-in-cache` also affects when EJBs are removed from the WebLogic Server cache, as described in "Caching and Passivating Stateful Session EJBs" on page 5-4.

Note: Evaluate the amount of memory available with the amount you require as, assigning a very high value to the max-beans-in-cache parameter may result in an OutOfMemoryError. The amount of memory cache required can be calculated as the number of beans in the cache times the size of the EJB.

Example

```
<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
Parent elements:	weblogic-enterprise-bean, stateless-session-descriptor, pool and weblogic-enterprise-bean message-driven-descriptor pool and weblogic-enterprise-bean entity-descriptor pool

Function

WebLogic Server maintains a free pool of EJBs for every entity bean, stateless session bean and message-driven bean class. The `max-beans-in-free-pool` element defines the size of this pool.

Example

See "pool" on page A-58.

message-driven-descriptor

Range of values:	n/a
Default value:	n/a
Parent elements:	`weblogic-enterprise-bean`

Function

Associates a message-driven bean with a JMS destination in WebLogic Server.

Example

The following example shows the structure of the `message-driven-descriptor` stanza:

```
<message-driven-descriptor>
   <pool>...</pool>
   <destination-jndi-name>...</destination-jndi-name>
   <initial-context-factory>...</initial-context-factory>
   <provider-url>...</provider-url>
   <connection-factory-jndi-name>...<connection-factory-jndi-name>
   <jms-polling-interval-seconds>...</jms-polling-interval-seconds>
   <jms-client-id>...</jms-client-id>
</message-driven-descriptor>
```

method

Range of values:	n/a
Default value:	n/a

Parent elements:	`weblogic-ejb-jar`
	`transaction-isolation`
	and
	`weblogic-ejb-jar`
	`idempotent-methods`

Function

Defines a method or set of methods for an enterprise bean's home or remote interface.

Example

```
<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

Parent elements:	`weblogic-ejb-jar`
	`transaction-isolation`
	`method`
	and
	`weblogic-ejb-jar`
	`idempotent-methods`
	`method`

Function

Specifies the EJB interface to which WebLogic Server applies isolation level properties, if the method has the same signature in multiple interfaces.

Example

See "method" on page A-51.

method-name

Range of values:	Name of an EJB defined in `ejb-jar.xml` | `*`
Default value:	n/a

Parent elements:	`weblogic-ejb-jar`
	`transaction-isolation`
	`method`
	and
	`weblogic-ejb-jar`
	`idempotent-methods`
	`method`

Function

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 A-51.

method-param

Range of values:	Fully-qualified Java type name of a method parameter.
Default value:	n/a

Parent elements:	`weblogic-ejb-jar`
	`transaction-isolation`
	`method`
	`method-params`
	and
	`weblogic-ejb-jar`
	`idempotent-methods`
	`method`
	`method-params`

Function

Specifies the fully qualified Java type name of a method parameter.

Example

See "method-params" on page A-55.

method-params

Range of values:	n/a
Default value:	n/a

Parent elements:	`weblogic-ejb-jar`
	`transaction-isolation`
	`method`
	and
	`weblogic-ejb-jar`
	`idempotent-methods`
	`method`

Function

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>
```

passivate-as-principal-name

Range of values:	Valid WebLogic Server principal.
Default value:	n/a
Parent elements:	`weblogic-enterprise-bean`

Function

The `passivate-as-principal-name` element, introduced in WebLogic Server 8.1 SP01, specifies the principal to be used in situations where `ejbPassivate` would otherwise run with

an anonymous principal. Under such conditions, the choice of which principal to run as is governed by the following rule:

If `passivate-as-principal-name` is set
then use that principal
else
    if a `run-as` role has been specified for the bean in `ejb-jar.xml`
    then use a principal according to the rules for setting the `run-as-role-assignment`
    else
        run `ejbPassivate` as an anonymous principal.

The `passivate-as-principal-name` element only needs to be specified if operations within `ejbPassivate` require more permissions than the anonymous principal would have.

This element affects the `ejbPassivate` methods of stateless session beans when passivation occurs due to a cache timeout.

See also "remove-as-principal-name" on page A-63, "create-as-principal-name" on page A-16, and "principal-name" on page A-59.

persistence

Range of values:	n/a
Default value:	n/a
Parent elements:	`weblogic-enterprise-bean`, `entity-descriptor`

Function

Required only for entity EJBs that use container-managed persistence services. 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-use`

Example

```
<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-beand>...</finders-load-bean>
      <persistence-use>...</persistence-use>
   </persistence>
</entity-descriptor>
```

persistence-use

Range of values:	n/a
Default value:	n/a
Parent elements:	`weblogic-enterprise-bean`, `entity-descriptor`, `persistence`

Function

Required only for entity EJBs that use container-managed persistence services. The `persistence-use` element stores an identifier of the persistence type to be used for this particular bean.

Example

```
<persistence-use>
       <type-identifier>WebLogic_CMP_RDBMS</type-identifier>
       <type-version>5.1.0</type-version>
       <type-storage>META-INF/weblogic-cmp-jar.xml</type-storage>
</persistence-use>
```

persistent-store-dir

Range of values:	Valid file system directory.
Default value:	`pstore`
Parent elements:	`weblogic-enterprise-bean` `stateful-session-descriptor`

Function

Specifies a file system directory where WebLogic Server stores the state of passivated stateful session bean instances. For more information, see "Specifying the Persistent Store Directory for Passivated Beans" on page 5-7.

Example

```
<stateful-session-descriptor>
   <stateful-session-cache>...</stateful-session-cache>
   <allow-concurrent-calls>...</allow-concurrent-calls>
   <persistent-store-dir>MyPersistenceDir</persistent-store-dir>
   <stateful-session-clustering>...</stateful-session-clustering>
   <allow-remove-during-transaction>
</stateful-session-descriptor>
```

pool

Range of values:	n/a
Default value:	n/a
Parent elements:	`weblogic-enterprise-bean` `stateless-session-descriptor` or `message-bean-descriptor` or `entity-descriptor`

Function

Configures the behavior of the WebLogic Server free pool for entity EJBs, stateless session EJBs, and message-driven EJBs. The options are:

- 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 (XML stanza)
Parent elements:	weblogic-ejb-jar security-role-assignment

Function

Specifies the name of an actual WebLogic Server principal to apply to the specified role-name. 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.

Example

See "security-role-assignment" on page A-73.

provider-url

Range of values:	Valid URL.
Default value:	`t3://localhost:7001`
Parent elements:	`weblogic-enterprise-bean` `message-driven-descriptor`

Function

Specifies the URL to be used by the `InitialContext`. See "Configuring MDBs for Destinations" on page 7-14 and "How to Set provider-url" on page 7-15.

Example

```
<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`

Parent elements:	`weblogic-enterprise-bean`
	`entity-descriptor`
	`entity-cache`
	or
	`weblogic-enterprise-bean`
	`entity-descriptor`
	`entity-cache-ref`

Function

Specifies the number of seconds between `ejbLoad()` calls on a `Read-Only` entity bean. A value 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
Default value:	`n/a`
Parent elements:	`weblogic-enterprise-bean`

Function

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>
       <resource-env-description>
              ...
       </resource-env-description>
```

```
        <ejb-reference-description>
                ...
        </ejb-reference-description>
        <ejb-local-reference-description>
                ...
        </ejb-local-reference-description>
</reference-descriptor>
```

remote-client-timeout

Range of values:	0 to *maxSeconds*, where *maxSeconds* is the maximum value of an int.
Default value:	0
Parent elements:	weblogic-enterprise-bean

Function

Specifies the length of time that a remote RMI client will wait before it will timeout. See Using the RMI Timeout in *Programming WebLogic RMI*.

Example

The following entry causes a remote RMI client to timeout after waiting 5 seconds.

```
<weblogic-enterprise-bean>
        <ejb-name>AccountBean</ejb-name>
        ...
        <remote-client-timeout>5</remote-client-timeout>
</weblogic-enterprise-bean>
```

remove-as-principal-name

Range of values:	n/a
Default value:	n/a
Parent elements:	`weblogic-enterprise-bean`

Function

This parameter only needs to be specified if operations within `ejbRemove` need more permissions than the anonymous principal would have.

The `remove-as-principal-name` element, introduced in WebLogic Server 8.1 SP1, specifies the principal to be used in situations where `ejbRemove` would otherwise run with an anonymous principal. Under such conditions, the choice of which principal to run as is governed by the following rule:

If `remove-as-principal-name` is set
then use that principal
else
    if a `run-as` role has been specified for the bean in `ejb-jar.xml`
    then use a principal according to the rules for setting the `run-as-role-assignment`
    else
        run `ejbRemove` as an anonymous principal

The `remove-as-principal-name` element only needs to be specified if operations within `ejbRemove` require more permissions than the anonymous principal would have.

This element effects the `ejbRemove` methods of stateless session beans and message-drive beans.

See also "passivate-as-principal-name" on page A-55, "create-as-principal-name" on page A-16, and "principal-name" on page A-59.

replication-type

Range of values:	`InMemory | None`
Default value:	`None`

Parent elements:	`weblogic-enterprise-bean`
	`stateful-session-descriptor`
	`stateful-session-clustering`

Function

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.

Example

See "stateful-session-clustering" on page A-76.

res-env-ref-name

Range of values:	A valid resource environment reference name from the `ejb-jar.xml` file.
Default value:	n/a

Parent elements:	`weblogic-enterprise-bean`
	`reference-descriptor`
	`resource-env-description`

Function

Specifies the name of a resource environment reference.

Example

See "resource-description" on page A-65.

res-ref-name

Range of values:	A valid resource reference name from the `ejb-jar.xml` file.
Default value:	n/a

Parent elements:	`weblogic-enterprise-bean`
	`reference-descriptor`
	`resource-description`

Function

Specifies the name of a `resourcefactory` reference. This is the reference that the EJB provider places within the `ejb-jar.xml` deployment file. Required element if the EJB specifies resource references in `ejb-jar.xml`

Example

See "resource-description" on page A-65.

resource-description

Range of values:	n/a
Default value:	n/a
Parent elements:	`weblogic-enterprise-bean` `reference-descriptor`

Function

Maps a resource reference defined in `ejb-jar.xml` to the JNDI name of an actual resource available in WebLogic Server.

Example

```
<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
Default value:	n/a
Parent elements:	`weblogic-enterprise-bean` `reference-descriptor`

Function

Maps a resource environment reference defined in `ejb-jar.xml` to the JNDI name of an actual resource available in WebLogic Server.

Example

```
<reference-descriptor>
       <resource-env-description>
            <res-env-ref-name>. . .</res-env-ref-name>
            <jndi-name>...</jndi-name>
       <reference-env-description>
</reference-descriptor>
```

When `jndi-name` is not a valid URL, WebLogic Server treats it as a object that maps to a URL and is already bound in the JNDI tree, and binds a `LinkRef` with that `jndi-name`.

role-name

Range of values:	Valid application role name.
Default value:	n/a
Parent elements:	`weblogic-enterprise-bean` `security-role-assignment`

Function

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 A-73.

run-as-identity-principal

Range of values:	Valid security principal name.
Default value:	n/a
Parent elements:	`weblogic-enterprise-bean`

Function

Note: The `run-as-identity-principal` element is deprecated in this release of WebLogic Server. Use `run-as-principal-name` instead.

The `run-as-identity-principal` element specifies which security principal name is to be used as the run-as principal for a bean that has specified a security identity `run-as-role-name` its `ejb-jar.xml` deployment descriptor.

For an explanation of how the mapping of run-as role-names to run-as-identity-principals or run-as-principal-names occurs, see the comments for the run-as-role-assignment element.

Example

```
<run-as-identity-principal>
     Fred
</run-as-identity-principal>
```

run-as-principal-name

Range of values:	Valid principal.
Default value:	n/a
Parent elements:	`weblogic-enterprise-bean`

Function

Specifies which security principal name is to be used as the run-as principal for a bean that has specified a `security-identity` run-as `role-name` in its `ejb-jar.xml` deployment descriptor.

For an explanation of how the mapping of run-as `role-names` to `run-as-principal-names` occurs, see the comments for the `run-as-role-assignment` element.

Example

```
<run-as-principal-name>
     Fred
</run-as-principal-name>
```

run-as-role-assignment

Range of values:	n/a
Default value:	n/a
Parent elements:	`weblogic-enterprise-bean` `security-role-assignment`

Function

Maps a given security-identity run-as `role-name` specified in the `ejb-jar.xml` deployment descriptor file to a `run-as-principal-name`.

The value of the `run-as-principal-name` for a given role-name that is specified here is scoped to all beans in the `ejb-jar.xml` deployment descriptor; it applies to all beans that specify that role-name as their security-identity `run-as-role-name`.

The `run-as-principal-name` value specified here can be overridden at the individual bean level by specifying a run-as-principal-name under that bean's `weblogic-enterprise-bean` element.

Note: For a given bean, if there is no `run-as-principal-name` specified in either a `run-as-role-assignment` or in a bean specific `run-as-principal-name` tag, then the EJB container chooses the first `principal-name` of a security user in the `weblogic-enterprise-bean` `security-role-assignment` for the `role-name` and uses that `principal-name` as the `run-as-principal-name`.

Example

Suppose that in the `ejb-jar.xml` deployment descriptor file:

- Beans 'A_EJB_with_runAs_role_X' and 'B_EJB_with_runAs_role_X'

  specify a security-identity run-as role-name 'runAs_role_X'.

- Bean 'C_EJB_with_runAs_role_Y'

  specifies a security-identity run-as role-name 'runAs_role_Y'.

Consider the following excerpts from the corresponding `weblogic-ejb-jar.xml` deployment descriptor file:

```
<weblogic-ejb-jar>
     <weblogic-enterprise-bean>
          <ejb-name>
               A_EJB_with_runAs_role_x
          </ejb-name>
     </weblogic-enterprise-bean>

     <weblogic-enterprise-bean>
          <ejb-name>
               B_EJB_with_runAs_role_X
          </ejb-name>
          <run-as-principal-name>
               Joe
          </run-as-principal-name>
     </weblogic-enterprise-bean>

     <weblogic-enterprise-bean>
          <ejb-name>
               C_EJB_with_runAs_role_Y
          </ejb-name>
     </weblogic-enterprise-bean>

     <security-role-assignment>
          <role-name>
               runAs_role_Y
          </role-name>
          <principal-name>
               first_principal_of_role_Y
          </principal-name>
          <principal-name>
               second_principal_of_role_Y
          </principal-name>
     </security-role-assignment>

     <run-as-role-assignment>
          <role-name>
               runAs_role_X
          </role-name>
          <run-as-principal-name>
               Fred
```

```
            </run-as-principal-name>
        </run-as-role-assignment>
</weblogic-ejb-jar>
```

Each of the beans chooses a different principal name to use as its `run-as-principal-name`:

A_EJB_with_runAs_role_X

This bean's run-as role-name is 'runAs_role_X'. The jar scoped `<run-as-role-assignment>` mapping will be used to look up the name of the principal to use.

The `<run-as-role-assignment>` mapping specifies that for `<role-name>` 'runAs_role_X' we are to use `<run-as-principal-name>` 'Fred'.

“Fred” is the principal name that will be used.

B_EJB_with_runAs_role_X

This bean's run-as role-name is also 'runAs_role_X'. This bean will not use the jar scoped `<run-as-role-assignment>` to look up the name of the principal to use because that value is overridden by this bean's `<weblogic-enterprise-bean>` `<run-as-principal-name>` value 'Joe'.

“Joe” is the principal name that will be used.

C_EJB_with_runAs_role_Y

This bean's run-as role-name is 'runAs_role_Y'. There is no explicit mapping of 'runAs_role_Y' to a run-as principal name; that is, there is no jar-scoped `<run-as-role-assignment>` for 'runAs_role_Y' nor is there a bean scoped `<run-as-principal-name>` specified in this bean's `weblogic-enterprise-bean`.

To determine the principal name to use, the `<security-role-assignment>` for `<role-name>` “runAs_role_Y” is examined. The first `<principal-name>` corresponding to a User (i.e. not a Group) is chosen.

“first_principal_of_role_Y” is the principal name that will be used.

security-permission

Range of values:	n/a
Default value:	n/a
Parent elements:	`weblogic-ejb-jar`

Function

Specifies a security permission that is associated with a J2EE Sandbox.

For more information, see Sun's implementation of the security permission specification:

http://java.sun.com/j2se/1.3/docs/guide/security/PolicyFiles.html#FileSyntax

Example

```
<security-permission>
     <description>Optional explanation goes here</description>
          <security-permission-spec>
          ...
     </security-permission-spec>
</security-permission>
```

security-permission-spec

Range of values:	n/a
Default value:	n/a
Parent elements:	`security-permission`

Function

Specifies a single security permission based on the Security policy file syntax.

For more information, see Sun's implementation of the security permission specification:

http://java.sun.com/j2se/1.3/docs/guide/security/PolicyFiles.html#FileSyntax

Example

To grant the "read" permission to "java.vm.version," and prevent it from being overwritten:

1. Set the `security-permission-spec` as shown below:

```
<security-permission>
   <description>Optional explanation goes here</description>
   <security-permission-spec> grant { permission
   java.util.PropertyPermission "java.vm.version", "read"; };
   </security-permission-spec>
</security-permission>
```

2. Modify the `startWeblogic` script to start the server using this option:

```
JAVA_OPTIONS=-Djava.security.manager
```

3. Create a directory named `lib` in your domain directory.
4. Add this line to the `%WL_HOME%\server\lib\weblogic.policy` file:

```
add grant codeBase "file:/<Your user_projects dir>/YourDomain/lib/-" {
permission java.security.AllPermission; };
```

   This is necessary because the EJB stub's classpath is `lib`.

security-role-assignment

Range of values:	n/a
Default value:	n/a
Parent elements:	`weblogic-ejb-jar`

Function

Maps application roles in the `ejb-jar.xml` file to the names of security principals available in WebLogic Server.

Example

```
<security-role-assignment>
       <role-name>PayrollAdmin</role-name>
       <principal-name>Tanya</principal-name>
```

```
        <principal-name>system</principal-name>
        <externally-defined>True</externally-defined>
        ...
</security-role-assignment>
```

session-timeout-seconds

Range of values:	
Default:	`idle-timeout-seconds`

Parent elements:	`weblogic-enterprise-bean`
	`stateful-session-descriptor`
	`stateful-session-cache`

Function

Determines how long the EJB container leaves a passivated stateful session bean on disk. The container removes a passivated EJB `session-timeout-seconds` after passivating the bean instance to disk. If `session-timeout-seconds` is not specified, the default is the value specified by `idle-timeout-seconds`.

Example

```
<stateful-session-descriptor>
     <stateful-session-cache>
          <max-beans-in-cache>4</max-beans-in-cache>
          <idle-timeout-seconds>5</idle-timeout-seconds>
          <session-timeout-seconds>120</session-timeout-seconds>
          <cache-type>LRU</cache-type>
     </stateful-session-cache>
</stateful-session-descriptor>
```

stateful-session-cache

Range of values:	n/a
Default:	n/a
Parent elements:	`weblogic-enterprise-bean`, `stateful-session-descriptor`

Function

Defines the following options used to cache stateful session EJB instances.

- `max-beans-in-cache`
- `idle-timeout-seconds`
- `session-timeout-sections`
- `cache-type`

See "Caching and Passivating Stateful Session EJBs" on page 5-4 for more information about caching of stateful session beans.

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>
       <session-timeout-seconds>...</session-timeout-seconds>
       <cache-type>...</cache-type>
</stateful-session-cache>
```

stateful-session-clustering

Range of values:	n/a
Default:	n/a
Parent elements:	`weblogic-enterprise-bean`, `stateful-session-descriptor`

Function

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

```
<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-name>
     <replication-type>InMemory</replication-type>
</stateful-session-clustering>
```

stateful-session-descriptor

Range of values:	n/a
Default:	n/a
Parent elements:	`weblogic-enterprise-bean`

Function

Defines deployment behaviors, such as caching, clustering, and persistence, for stateless session EJBs in WebLogic Server.

:Example

```
<stateful-session-descriptor>
       <stateful-session-cache>...</stateful-session-cache>
       <allow-concurrent-calls>...</allow-concurrent-calls>
       <allow-remove-during-transaction>...
       </allow-remove-during-transaction>
       <persistent-store-dir>/myPersistenceStore</persistent-store-dir>
       <stateful-session-clustering>...</stateful-session-clustering>
</stateful-session-descriptor>
```

stateless-bean-call-router-class-name

Range of values:	Valid custom class name.
Default value:	.n/a

Parent elements:	`weblogic-enterprise-bean,`
	`stateless-session-descriptor`
	`stateless-clustering`

Function

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 A-80.

stateless-bean-is-clusterable

Range of values:	`True | False`
Default value:	`True`

Parent elements:	`weblogic-enterprise-bean,`
	`stateless-session-descriptor`
	`stateless-clustering`

Function

When `stateless-bean-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.

Example

See "stateless-clustering" on page A-80.

stateless-bean-load-algorithm

Range of values:	`round-robin` | `random` | `weight-based` | `RoundRobinAffinity` | `RandomAffinity` | `WeightBasedAffinity`
Default value:	Value of `weblogic.cluster.defaultLoadAlgorithm`
Parent elements:	`weblogic-enterprise-bean`, `stateless-session-descriptor`, `stateless-clustering`

Function

Specifies the algorithm to use for load balancing between replicas of the EJB home.

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.
- `round-robin-affinity`—Server affinity governs connections between external Java clients and server instances; round robin load balancing is used for connections between server instances.
- `weight-based-affinity`—Server affinity governs connections between external Java clients and server instances; weight-based load balancing is used for connections between server instances.
- `random-affinity`—Server affinity governs connections between external Java clients and server instances; random load balancing is used for connections between server instances.

For more information, see "Load Balancing for EJBs and RMI Objects" in *Using WebLogic Server Clusters*.

Example

See "stateless-clustering" on page A-80.

stateless-bean-methods-are-idempotent

Note: This element is deprecated in this release.

Range of values:	`True | False`
Default value:	`False`

Parent elements:	`weblogic-enterprise-bean,`
	`stateless-session-descriptor`
	`stateless-clustering`

Function

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 A-80.

stateless-clustering

Range of values:	n/a
Default:	n/a
Parent elements:	`weblogic-enterprise-bean,` `stateless-session-descriptor`

Function

Specifies options that determine how WebLogic Server replicates stateless session EJB instances in a cluster.

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-methods-are-idempotent>
</stateless-clustering>
```

stateless-session-descriptor

Range of values:	n/a
Default:	n/a
Parent elements:	`weblogic-enterprise-bean`

Function

Defines deployment parameters, such as caching, clustering, and persistence for stateless session EJBs in WebLogic Server.

Example

```
<stateless-session-descriptor>
       <pool>...</pool>
       <stateless-clustering>...</stateless-clustering>
</stateless-session-descriptor>
```

transaction-descriptor

Range of values:	n/a
Default:	n/a
Parent elements:	`weblogic-enterprise-bean`

Function

Specifies options that define transaction behavior in WebLogic Server. Currently, this stanza includes only one element: `trans-timeout-seconds`.

Example

```
<transaction-descriptor>
       <trans-timeout-seconds>20</trans-timeout-seconds>
</transaction-descriptor>
```

transaction-isolation

Range of values:	n/a
Default:	n/a
Parent elements:	`weblogic-ejb-jar`

Function

Defines method-level transaction isolation settings for an EJB.

Example

```
<transaction-isolation>
       <isolation-level>...</isolation-level>
       <method>
              <description>...</description>
              <ejb-name>...</ejb-name>
```

```
                <method-intf>...</method-intf>
                <method-name>...</method-name>
                <method-params>...</method-params>
        </method>
</transaction-isolation>
```

For more information, see "isolation-level" on page A-44.

transport-requirements

Range of values:	n/a
Default:	n/a
Parent elements:	`weblogic-enterprise-bean,` `iiop-security-descriptor`

Function

Provides the transport requirements for the EJB.

Example

```
<iiop-security-descriptor>
        <transport-requirements>
                <confidentiality>supported</confidentiality>
                <integrity>supported</integrity>
                <client-cert-authorization>suppoted
                </client-cert-authentication>
        </transport-requirements>
</iiop-security-descriptor>
```

trans-timeout-seconds

Range of values:	0 to *max*
Default value:	30
Parent elements:	weblogic-enterprise-bean, transaction-descriptor

Function

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 A-82.

type-identifier

Range of values:	Valid string. `WebLogic_CMP_RDBMS` specifies WebLogic Server RDBMS-based persistence.
Default value:	n/a
Parent elements:	weblogic-enterprise-bean, entity-descriptor, persistence persistence-use

Function

Required only for entity EJBs that use container-managed persistence services. Specifies 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-use" on page A-57 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
Parent elements:	`weblogic-enterprise-bean, entity-descriptor, persistence persistence-use`

Function

Required only for entity EJBs that use container-managed persistence services. 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-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-use" on page A-57 for an example that shows the complete persistence type definition for WebLogic Server RDBMS-based persistence.

type-version

Range of values:	Valid string
Default value:	n/a

Parent elements:	`weblogic-enterprise-bean,`
	`entity-descriptor,`
	`persistence`
	`persistence-use`

Function

Required for entity EJBs that use container-managed persistence service, if multiple versions of the same persistence type are installed. Identifies the version of the specified persistence type. For example, for WebLogic 2.0 CMP persistence, use the value:

```
6.0
```

For WebLogic 1.1 CMP persistence, use the value:

5.1.0

This element is necessary if multiple versions of the same persistence type are installed.

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-use" on page A-57 for an example that shows the complete persistence type definition for WebLogic Server RDBMS-based persistence.

use-serverside-stubs

Range of values:	`true | false`
Default value:	`false`

Parent elements:	`weblogic-enterprise-bean,`
	`  entity-descriptor,`
	`  entity-clustering`
	`and`
	`weblogic-enterprise-bean`
	`  stateful-session-descriptor`
	`  stateful-session-clustering`
	`and`
	`weblogic-enterprise-bean,`
	`  entity-descriptor,`
	`  entity-clustering`

Function

Causes the bean home to use server-side stubs in the context of server.

Example

See the example for "entity-clustering" on page A-29.

weblogic-ejb-jar

Requirements:	n/a
Default value:	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:	n/a
Default value:	n/a
Parent elements:	`weblogic-ejb-jar`

Function

Contains the deployment information for a bean that is available in WebLogic Server.

APPENDIX B

weblogic-cmp-jar.xml Deployment Descriptor Reference

The following sections describe the EJB 2.0 deployment descriptor elements found in the `weblogic-cmp-jar.xml` file, the WebLogic-specific XML document type definitions (DTD) file. Use these definitions to create the WebLogic-specific `weblogic-cmp-jar.xml` file that is part of your EJB deployment. Use this deployment descriptor file to specify container-managed-persistence (CMP) behavior.

For information on the EJB 1.1 deployment descriptor elements see Appendix F, "Important Information for EJB 1.1 Users".

- "2.0 weblogic-cmp-jar.xml Deployment Descriptor File Structure" on page B-1
- "Changes to weblogic-cmp-jar.xml in WebLogic Server 8.1" on page B-2
- "2.0 weblogic-cmp-jar.xml Deployment Descriptor Elements" on page B-3

2.0 weblogic-cmp-jar.xml Deployment Descriptor File Structure

The `weblogic-cmp-jar.xml` file defines deployment descriptors for entity EJBs that use WebLogic Server RDBMS-based persistence services. The EJB container uses a version of `weblogic-cmp-jar.xml` that is different from the XML shipped with WebLogic Server Version 6.x.

You can continue to use the earlier `weblogic-cmp-jar.xml` DTD for EJB 1.1 beans that you will deploy on the WebLogic Server Version 8.1. 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 8.1 `weblogic-cmp-jar.xml` consists of a `weblogic-rdbms-jar` stanza:

```
description

weblogic-version

weblogic-rdbms-jar

        weblogic-rdbms-bean
                ejb-name
                data-source-name
                table-map
                field-group
                relationship-caching
                weblogic-query
                delay-database-insert-until
                automatic-key-generation
                check-exists-on-method

        weblogic-rdbms-relation
                relation-name
                table-name
                weblogic-relationship-role
        order-database-operations

        enable-batch-operations

        create-default-dbms-tables

        validate-db-schema-with

        database-type

        default-dbms-tables-ddl

        compatibility
```

Changes to weblogic-cmp-jar.xml in WebLogic Server 8.1

These changes were made to `weblogic-cmp-jar.xml` in WebLogic Server 8.1:

- "enable-batch-operations" on page B-21 was added in WebLogic Server 8.1.
- "instance-lock-order" on page B-29 was added in WebLogic Server 8.1 SP01.

- "order-database-operations" on page B-34 was added in WebLogic Server 8.1.
- "serialize-byte-array-to-oracle-blob" on page B-40 was added in WebLogic Server SP02.
- "verify-rows" on page B-46 was added in WebLogic Server 8.1.

2.0 weblogic-cmp-jar.xml Deployment Descriptor Elements

This list of the elements in `weblogic-cmp-jar.xml` includes all elements that were supported in any service pack of WebLogic Server 8.1. The previous section, "Changes to weblogic-cmp-jar.xml in WebLogic Server 8.1" lists elements that were new, changed, or deprecated in Weblogic Server 8.1, or a subsequent service pack.

- "allow-readonly-create-and-remove" on page B-5
- "automatic-key-generation" on page B-6
- "caching-element" on page B-7
- "caching-name" on page B-8
- "check-exists-on-method" on page B-8
- "cmp-field" on page B-9
- "cmr-field" on page B-10
- "column-map" on page B-10
- "create-default-dbms-tables" on page B-12
- "database-type" on page B-15
- "data-source-name" on page B-15
- "db-cascade-delete" on page B-16
- "dbms-column" on page B-17
- "dbms-column-type" on page B-18
- "default-dbms-tables-ddl" on page B-19
- "delay-database-insert-until" on page B-19
- "description" on page B-20

- “ejb-name” on page B-21
- “enable-batch-operations” on page B-21
- “field-group” on page B-22
- “field-map” on page B-23
- “foreign-key-column” on page B-24
- “foreign-key-table” on page B-24
- “generator-name” on page B-25
- “generator-type” on page B-26
- “group-name” on page B-27
- “include-updates” on page B-28
- “instance-lock-order” on page B-29
- “key-cache-size” on page B-30
- “key-column” on page B-31
- “lock-order” on page B-31
- “max-elements” on page B-32
- “method-name” on page B-32
- “method-param” on page B-33
- “method-params” on page B-33
- “optimistic-column” on page B-34
- “order-database-operations” on page B-34
- “primary-key-table” on page B-35
- “query-method” on page B-36
- “relation-name” on page B-36
- “relationship-caching” on page B-37
- “relationship-role-map” on page B-38

- "relationship-role-name" on page B-40
- "serialize-byte-array-to-oracle-blob" on page B-40
- "sql-select-distinct" on page B-41
- "table-map" on page B-42
- "table-name" on page B-43
- "use-select-for-update" on page B-44
- "validate-db-schema-with" on page B-45
- "verify-columns" on page B-45
- "verify-rows" on page B-46
- "weblogic-ql" on page B-47
- "weblogic-query" on page B-48
- "weblogic-rdbms-bean" on page B-49
- "weblogic-rdbms-jar" on page B-49
- "weblogic-rdbms-relation" on page B-50
- "weblogic-relationship-role" on page B-54

allow-readonly-create-and-remove

Range of Values:	`true | false`
Default value:	`false`
Parent elements:	`weblogic-rdbms-jar` `compatibility`

Function

This element, introduced in WebLogic Server 8.1 SP02, is a backward compatibility flag. It is used to enable create and remove operations for an EJB that uses `ReadOnly` concurrency.

Prior to version 8.1 SP2, these operations were allowed, although they had no transactional meaning. They have been disallowed so that more efficient code can be generated for `ReadOnly` beans, and because using them is a bad practice.

Example

```
<compatibility>
     <allow-readonly-create-and-remove>
     true
     </allow-readonly-create-and-remove>
</compatibility>
```

automatic-key-generation

Range of Values	n/a
Default value:	n/a
Parent elements:	`weblogic-rdbms-bean`

Function

The `automatic-key-generation` element specifies how primary keys will be automatically generated. For more information about this feature, see "Automatically Generating Primary Keys" on page 6-9.

Example

The following code samples show the `automatic-key-generation` stanza for different primary key generation methods. For supported generation methods, see "generator-type" on page B-26

Listing B-1 automatic-key-generation With generator-type=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>
```

Listing B-2 automatic-key-generation With generator-type=SQL-SERVER

```
<automatic-key-generation>
        <generator-type>SQL-SERVER</generator-type>
</automatic-key-generation>
```

Listing B-3 automatic-key-generation With generator-type=NamedSequenceTable

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

caching-element

Range of Values	n/a
Default value:	n/a

Parent elements:	`weblogic-rdbms-jar`
	`weblogic-rdbms-bean`
	`relationship-caching`

Function

Specifies the `cmr-field` and the `group-name` in the related bean. If `group-name` is not specified, the default `group-name` (load all fields) is used. For more information, see "group-name" on page B-27.

The `caching-element` stanza can contain nested caching element stanzas, as in the example shown in "relationship-caching" on page B-37.

For information about relationship caching, see "Relationship Caching" on page 6-33.

Example

See "relationship-caching" on page B-37:

caching-name

Range of Values	n/a
Default value:	n/a

Parent elements:	`weblogic-rdbms-jar`
	`weblogic-rdbms-bean`
	`relationship-caching`

Function

The `caching-name` element specifies the name of a relationship cache. For more information about relationship caching, see "Relationship Caching" on page 6-33.

Example

See "relationship-caching" on page B-37:

check-exists-on-method

Range of values:	`True | False`
Default value:	`True`
Parent elements:	`weblogic-rdbms-bean`

Function

By default, the EJB container checks that a container-managed persistence (CMP) entity bean exists before any business method invoked on the bean completes. This means the container notifies an application as soon as any business method is invoked on a container-managed entity bean that has been removed.

To specify that the EJB container wait for transaction completion to perform the existence check, set `check-exists-on-method` to `False`. This results in high performance and still provides a sufficient level of checking for most applications.

Example

The following example specifies that WebLogic Server notify the application that a business method has been invoked on a CMP entity bean that has been removed.

```
<check-exists-on-method>True</check-exists-on-method>
```

cmp-field

Range of Values	Valid name of field in the bean. Field must have matching `cmp-entry` entry in `ejb-jar.xml`. Field name is case-sensitive.
Default value:	n/a

Parent elements:	`weblogic-rdbms-bean`
	`field-map`
	`weblogic-rdbms-relation`
	`field-group`

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 B-23.

cmr-field

Range of Values	Valid name of field in the bean. Field must have matching `cmr-field` entry in `ejb-jar.xml`.
Default value:	n/a

Parent elements:	`weblogic-rdbms-relation`
	`field-group`
	and
	`relationship-caching`
	`caching-element`

Function

Specifies the name of a container-managed relationship field.

Example

```
<cmr-field>stock options</cmr-field>
```

column-map

Range of Values	n/a
Default value:	n/a
Requirements:	

Parent elements:	`<weblogic-rdbms-relation>`
	`<weblogic-relationship-role>`
	`<relationship-role-map>`

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 columns belong are implicit from the context in which the `column-map` element appears in the deployment descriptor.

Note: The `key-column` element is not specified if the `foreign-key-column` refers to a remote bean.

Example

The `XML` stanza can contain the elements shown here:

```
<column-map
       <foreign-key-column>account-id</foreign-key-column>
       <key-column>id</key-column>
</column-map>
```

compatibility

Range of values:	n/a
Default value:	n/a
Parent elements:	`weblogic-rdbms-jar`

Function

The `<compatibility>` stanza, introduced in WebLogic Server 8.1 SP02, contains elements that specify compatibility flags for all of the cmp beans described in the descriptor file.

Example

```
<compatibility>
     <serialize-byte-array-to-oracle-blob>
     <allow-readonly-create-and-remove>
</compatibility>
```

create-default-dbms-tables

Range of values:	`Disabled | CreateOnly | DropAndCreate | DropAndCreateAlways |AlterOrCreate`
Default value:	`Disabled`
Parent elements:	`weblogic-rdbms-jar`

Function

The `create-default-dbms-table` element performs two functions:

- It determines whether or how WebLogic Server automatically creates a default table based on the descriptions in the deployment files and the bean class, when underlying table schema have changed.
- It determines whether or how WebLogic Server automatically creates an Oracle SEQUENCE.

Use this element only for convenience during development phases. This is because the table schema in the `DBMS CREATE` statement used are the EJB container's best approximation of the definition. A production environment typically requires more precise schema definition.

Automatic Table Creation

The following table describes how WebLogic Server handles automatic table creation, based on the value of `create-default-dbms-tables`.

Setting <create-default-dbms-tables> to this value	Results in this behavior with respect to automatic table creation
`Disabled`	The EJB container takes no action with respect to automatic table creation. This is the default value.
`CreateOnly`	The EJB container automatically generates the table upon detecting changed schema. The container attempts to create the table based on information found in the deployment files and in the bean class. If table creation fails, a `'Table Not Found'` error is thrown, and the user must create the table manually.
`DropAndCreate`	The EJB container automatically generates the table upon detecting changed schema. The container drops and creates the table during deployment if columns have changed. The container does not save data.
`DropAndCreateAlways`	The EJB container automatically generates the table upon detecting changed schema. The container drops and creates the table during deployment whether or not columns have changed. The container does not save the data.
`AlterOrCreate`	The EJB container automatically generates the table upon detecting changed schema. The container creates the table if it does not yet exist. If the table does exist, the container alters the table schema. Table data is saved. **Note:** Do not choose this setting if a new column is specified as a primary key or if a column with null values is specified as the new primary key column.

If `TABLE CREATION` fails, the server throws a `Table Not Found` error and the table must be created by hand.

See "Automatic Table Creation (Development Only)" on page 6-14.

Automatic Oracle SEQUENCE Generation

Note: Automatic Oracle SEQUENCE generation works only with servers running in development mode.

The following table describes how WebLogic Server handles automatic SEQUENCE generation, based on the value of `create-default-dbms-tables`.

Setting <create-default-dbms-tables> to this value	Results in this behavior:
`Disabled`	The EJB container takes no action with respect to SEQUENCE generation. This is the default value.
`CreateOnly`	The EJB container creates a SEQUENCE, and constructs its name by appending "_WL" to the value of the `generator-name` element.
`DropAndCreate`	The EJB container creates a SEQUENCE, and constructs its name by appending "_WL" to the value of the `generator-name` element. If the SEQUENCE's increment value does not match the value of the key-cache-size element, the container alters the increment value to match the value of key-cache-size.
`DropAndCreateAlways`	The EJB container creates a SEQUENCE, and constructs its name by appending "_WL" to the value of the `generator-name` element. If the SEQUENCE's increment value does not match the value of the key-cache-size element, the container alters the increment value to match the value of key-cache-size.
`AlterOrCreate`	The EJB container creates a SEQUENCE, and constructs its name by appending "_WL" to the value of the `generator-name` element. If the SEQUENCE's increment value does not match the value of the key-cache-size element, the container alters the increment value to match the value of key-cache-size.

For more information on automatic generation of an Oracle SEQUENCE, see "Support for Oracle SEQUENCE" on page 6-12.

Example

The following example specifies the `create-default-dbms-tables` element.

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

database-type

Range of values:	`DB2 | Informix| Oracle | SQLServer | SQLServer2000| Sybase | POINTBASE.`
Default value:	
Parent elements:	`weblogic-rdbms-jar`

Function

The `database-type` element specifies the database used as the underlying dbms.

Example

```
<database-type>POINTBASE</database-type>
```

data-source-name

Range of values	Valid WebLogic Server JDBC datasource name.
Default value	n/a
Parent elements:	`weblogic-rdbms-bean`

Function

Specifies the JDBC data source name to be used for database connectivity for this bean. For more information on datasources, see "Programming WebLogic JDBC."

Example

See "table-name" on page B-43.

db-cascade-delete

Range of Values	n/a
Default	By default, database cascade delete is not used. The EJB container performs cascade deletes by issuing an individual SQL `DELETE`.
Parent elements:	`weblogic-rdbms-bean` `weblogic-relationship-role`

Function

Allows an application to take advantage of a database's built-in support for cascade delete, and possibly improve performance. This functionality is supported only for:

- Oracle databases
- One-to-one or one-to-many relationships.

If `db-cascade-delete` is enabled in `weblogic-cmp-rdbms-jar.xml`, you must

- Enable `cascade-delete` in `ejb-jar.xml`
- Enable cascade delete in the database table definition.

Note: If `db-cascade-delete` is not specified, do not enable the database's cascade delete functionality, as this will produce incorrect results.

Setting up Oracle for Cascade Delete

The following Oracle table definition will cause deletion all of the `emp` rows if the owning `dept` is deleted in the database.

```
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 );
```

Example

```
</weblogic-relationship-role>
    <db-cascade-delete/>
</weblogic-relationship-role>
```

dbms-column

Range of values:	Valid database column.
Default value:	n/a
Parent elements:	`weblogic-rdbms-bean` `field-map`

Function

The name of the database column to which the field should be mapped.

Note: `dbms-column` is case maintaining, although not all database are case sensitive.

Example

See "field-map" on page B-23.

dbms-column-type

Range of values:	`OracleBlob` | `OracleClob` | `LongString` | `SybaseBinary`
Default value:	
Parent elements:	`weblogic-rdbms-bean` `field-map`

Function

Specifies the type of the `cmp-field`. the current field to a Blob or Clob in an Oracle database or a LongString or SybaseBinary in a Sybase database.

- `OracleBlob`—maps the field to an Oracle Blob.
- `OracleClob`—maps the field to an Oracle Clob.
- `LongString`—tells the container to use `setCharacterStream` to write String data into the database. Some JDBC drivers have problems writing more than 4K of data using `setString`.
- `SybaseBinary`—tells the container to use `setBytes` to write bytes into the binary column, because `setBinaryStream` does not work with `SybaseXADriver`.

Example

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

default-dbms-tables-ddl

Range of values:	Valid file name.
Default value:	
Parent elements:	`weblogic-rdbms-jar`

Function

Specifies the `DDL` file name to which the EJB container writes the table creation scripts.

delay-database-insert-until

Range of values:	`ejbCreate | ejbPostCreate` The `commit` option is removed in WebLogic Server 8.1
Default value:	`ejbPostCreate`
Requirements:	
Parent elements:	`weblogic-rdbms-bean`

Function

Specifies when a new CMP bean is inserted into the database. The allowable values cause the following behavior:

- `ejbCreate` - perform database insert immediately after `ejbCreate`. This setting yields better performance than `ejbCreate` by avoiding an unnecessary store operation.
- `ejbPostCreate` - perform insert immediately after `ejbPostCreate`.

This element has an effect only when `order-database-operations` is `False`. By default, `order-database-operations` is `true`, which causes new beans to be inserted at the transaction commit time.

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, the `cmr-field` must be set to a non-null value in `ejbPostCreate` before the bean is inserted into the database.

For maximum flexibility, avoid creating related beans in your `ejbPostCreate` method. If `ejbPostCreate` creates related beans, and database constraints prevent related beans from referring to a bean that has not yet been created, it is not possible to database insert until after the method completion.

Note: `cmr-field`s may not be set during `ejbCreate`, before the primary key of the bean is known.

Example

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

description

Range of values:	n/a
Default value:	n/a

Parent elements:	`weblogic-rdbms-jar`
	`weblogic-rdbms-bean`
	`weblogic-query`

Function

The `description` element provides text that describes the parent element.

Example

```
<dscription>Contains a description of parent element</description>
```

ejb-name

Range of values:	Must match the `ejb-name` of a cmp entity bean defined in the `ejb-jar.xml`.
Default value:	n/a
Parent elements:	`weblogic-rdbms-bean`

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 B-43.

enable-batch-operations

Range of values:	`True | False`
Default value:	`True`
Parent elements:	`weblogic-rdbms-jar`

Function

This element, introduced in WebLogic Server 8.1, controls whether or not the EJB container allows batch database operations, including batch inserts, batch updates and batch deletes.

If this element is set to `True`, the EJB delays database operations in a transaction until commit time.

Note: In WebLogic Server 8.1, this element replaces the functionality of the `commit` setting, available in WebLogic Server 7.0, for the `delay-database-insert-until` element.

Example

The following XML sample demonstrates use of the `enable-batch-operations` element:

```
<enable-batch-operations>True</enable-batch-operations>
```

field-group

Range of values:	n/a
Default value:	A special group named `default` is used for finders and relationships that have no `field-group` specified. The default group contains all of a bean's cmp-fields, but none of its cmr-fields.
Parent elements:	`weblogic-rdbms-relation`

Function

The `field-group` element represents a subset of the `cmp-field`s and `cmr-field`s 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:	n/a
Default value:	n/a
Parent elements:	`weblogic-rdbms-bean`

Function

The `field-map` element represents a mapping between a particular column in a database and a `cmp-field` in the bean instance.

The optional `group-name` element specifies a field group that is to be loaded when the `getXXX` method for the `cmp-field` is called and the EJB container needs to read the value from the DBMS because it is not in memory. If `group-name` is omitted, the default group, which contains all `cmp-field`s, is used when the `cmp-field` is not explicitly listed in any field groups, otherwise a field group that contains the `cmp-field` is chosen. Thus, developers should specify a `group-name` if the `cmp-field` is listed in multiple field groups or the container will pick one of the groups arbitrarily.

The `dbms-column-type` element is optional.

Example

The `XML` stanza can contain the elements shown here:

```
<field-map>
       <cmp-field>....</cmp-field>
       <dbms-column>...</dbms-column>
       <dbms-column-type>...</dbms-column-type>
       <group-name>...</group name>
</field-map>
```

foreign-key-column

Range of values:	Valid foreign key database column name.
Default value:	n/a
Parent elements:	`weblogic-rdbms-bean` `column-map`

Function

The `foreign-key-column` element represents a column of a foreign key in the database.

Example

See "column-map" on page B-10.

foreign-key-table

Range of values:	Valid database table name.
Default value:	n/a

Parent elements:	`weblogic-rdbms-jar`
	`weblogic-rdbms-relation`
	`weblogic-relationship-role`
	`relationship-role-map`

Function

The `foreign-key-table` element specifies the name of a DBMS table that contains a foreign key.

Example

See "relationship-role-map" on page B-38.

generator-name

Range of values:	
Default value:	
Parent elements:	`weblogic-rdbms-bean` `automatic-key-generation`

Function

The `generator-name` element is used to specify the name of the primary key generator.

- If the `generator-type` element is `Oracle`, and WebLogic Server is running in development mode, then the EJB container constructs the Oracle SEQUENCE name by appending "_WL" to the `generator-name` as part of the container's automatic SEQUENCE generation feature.

  For more information on automatic Oracle SEQUENCE generation, see "Support for Oracle SEQUENCE" on page 6-12.

- If the `generator-type` element is `Oracle`, and WebLogic Server is running in production mode, then the EJB container sets the name of the Oracle SEQUENCE to the value of `generator-name`.

- If the `generator-type` element is `NamedSequenceTable`, then the `generator-name` element would be the name of the `SEQUENCE_TABLE` to be used.

Example

See "automatic-key-generation" on page B-6.

generator-type

Range of values:	`Oracle | SQLServer | SQLServer2000 | NamedSequenceTable`
Default value:	
Parent elements:	`weblogic-rdbms-bean` `automatic-key-generation`

Function

The `generator-type` element specifies the primary key generation method to use.

In addition, `generator-type` is used in conjunction with automatic Oracle SEQUENCE generation. See "Support for Oracle SEQUENCE" on page 6-12.

Example

See "automatic-key-generation" on page B-6.

group-name

Range of values:	n/a

Default value:	`field-group`
	and
	`caching-element`
	and
	`weblogic-query`
	and
	`field-map`
	and
	`weblogic-relationship-role`

Parent elements:	`weblogic-rdbms-relation` `field-group`

Function

Specifies the name of a field group.

Example

See "field-group" on page B-22.

include-updates

Range of values:	`True` | `False`
Default value:	`False` for beans that use optimistic concurrency. `True` for beans that use other concurrency types.
Parent elements:	`weblogic-rdbms-bean` `weblogic-query`

Function

Specifies whether 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. A value of `False` provides best performance.

Example

```
<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>
```

instance-lock-order

Range of values:	`AccessOrder | ValueOrder`
Default value:	`AccessOrder`
Parent elements:	`weblogic-rdbms-bean`

Function

Specifies a locking or processing order for instances of a particular EJB. This element can be used to prevent deadlocks in an application that would otherwise experience deadlocks. `instance-lock-order` is used whenever database operations (update, for example) that apply to multiple instances of the same EJB are performed by the container. It specifies an order for operations that can cause a database lock to be acquired for a bean instance.

For example, `instance-lock-order` could be used to specify the order in which the EJB container calls `ejbStore` for instances of a particular EJB that uses database concurrency; `ejbStore` may acquire an exclusive lock when a database update is done. `instance-lock-order` also controls the order in which beans using optimistic concurrency are locked when optimistic checking is performed.

- `AccessOrder`—The container will process beans so that locks are acquired (or upgraded) in the order in which the application originally accessed the beans during the transaction. This is the recommended value when all transactions in the system access instances of the bean, and ultimately rows in a database table, in the same order.
- `instance-lock-order`—`ValueOrder`-Beans are processed in order based on the value of their primary key. `ValueOrder` should be specified to avoid deadlocks when concurrent transactions access instances of the same EJB in different orders.

Note: The EJB's primary key class is not required to implement the `java.lang.Comparable` interface when `ValueOrder` is specified, although this will result in a total ordering. Beans are ordered partially using the hash code value of the primary key when the primary key does not implement `java.lang.Comparable`.

Example

```
<instance-lock-order>ValueOrder</instance-lock-order>
```

key-cache-size

Range of values:	
Default value:	1
Parent elements:	`weblogic-rdbms-bean` `automatic-key-generation`

Function

Specifies the optional size of the primary key cache available in the automatic primary key generation feature. In addition, the EJB container uses this value to calculate the increment value for an Oracle SEQUENCE when automatic SEQUENCE generation is enabled. See "Support for Oracle SEQUENCE" on page 6-12.

If `generator-type` is

- `Oracle`—`key-cache-size` must match the Oracle SEQUENCE INCREMENT value. If there is a mismatch between this value and the Oracle SEQUENCE INCREMENT value, then there will likely be duplicate key problems.
- `NamedSequenceTable`—`key-cache-size` specifies how many keys the container will fetch in a single DBMS call
- SQLServer—`key-cache-size` is ignored.

Example

See "automatic-key-generation" on page B-6.

key-column

Range of values:	Valid primary key column name.
Default value:	n/a
Parent elements:	`weblogic-rdbms-bean` `column-map`

Function

The `key-column` element represents a column of a primary key in the database.

Example

See "column-map" on page B-10.

lock-order

Range of values:	All positive integers.
Default value:	0
Parent elements:	`weblogic-rdbms-bean`

Function

Use this flag to specify the database locking order of an entity bean when a transaction involves multiple beans and exclusive concurrency. The bean with lowest number is locked first.

This flag should only be used to prevent a deadlock situation and, currently, only applies when a transaction performs cascade deletes.This flag is only used for cascade delete currently.

Example

The `XML` stanza can contain the elements shown here:

```
<lock-order>1</lock-order>
     <!ELEMENT lock-order (PCDATA)>
```

max-elements

Range of values:	n/a
Default value:	n/a/
Parent elements:	`weblogic-rdbms-bean` `weblogic-query`

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
Parent elements:	`weblogic-rdbms-bean` `query-method`

Function

The `method-name` element specifies the name of a finder or `ejbSelect` method.

Note: The '*' character may not be used as a wildcard.

Example

See "weblogic-query" on page B-48.

method-param

Range of values:	n/a
Default value:	n/a
Parent elements:	`weblogic-rdbms-bean` `method-params`

Function

The `method-param` element contains the fully qualified Java type name of a method parameter.

Example

```
<method-param>java.lang.String</method-param>
```

method-params

Range of values:	n/a
Default value:	n/a
Parent elements:	`weblogic-rdbms-bean` `query-method`

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 B-48.

optimistic-column

Range of values:	Valid database column name.
Default value:	n/a
Parent elements:	`weblogic-rdbms-bean` `table-map`

Function

The `optimistic-column` element denotes a database column that contains a version or timestamp value used to implement optimistic concurrency. For more information on optimistic concurrency, see "Choosing a Concurrency Strategy" on page 6-35.

Note: Although not all databases are case sensitive, this element is case maintaining.

Example

The following sample XML shows the use of the `optimistic-column` element.

```
<optimistic-column>ROW_VERSION</optimistic-column>
```

where `ROW_VERSION` is the name of a database column that contains the value used for concurrency checking.

order-database-operations

Range of values:	`True | False`
Default value:	`True`
Parent elements:	`weblogic-rdbms-jar`

Function

This element, introduced in WebLogic Server 8.1, determines whether the EJB container delays all database operations in a transaction until commit time, automatically sorts the database dependency between the operations, and sends these operations to the database in such a way to avoid any database constraint errors.

If enable-batch-operations is set to True, the container automatically sets order-database-operations to True. To turn off order-database-operations, set both order-database-operations and enable-batch-operations to False.

See also enable-batch-operations and delay-database-insert-until.

Example

The following sample XML demonstrates the use of the order-database-operations element.

```
<order-database-operations>True</order-database-operations>
```

primary-key-table

Range of values:	Valid database table name.
Default value:	n/a
Parent elements:	weblogic-rdbms-jar weblogic-rdbms-relation weblogic-relationship-role relationship-role-map

Function

The primary-key-table element specifies the name of a DBMS table that contains a primary key. For more information about primary keys, see "Using Primary Keys" on page 6-7.

Note: Although not all databases are case sensitive, this element is case maintaining.

Example

For examples, see "relationship-role-map" on page B-38 and "Mapping a Bean on Primary Key Side of a Relationship to Multiple Tables" on page B-39.

query-method

Range of values:	n/a
Default value:	n/a
Parent elements:	`weblogic-rdbms-bean`

Function

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 B-48.

relation-name

Range of values:	Must match the `ejb-relation-name` of an `ejb-relation` in the associated `ejb-jar.xml` deployment descriptor file. The `ejb-relation-name` is optional, but is required for each relationship defined in the associated `ejb-jar.xml` deployment descriptor file
Default value:	n/a
Parent elements:	`weblogic-rdbms-relation`

Function

The `relation-name` element specifies the name of a relation.

For more information about container-managed relationships, see "Using Container-Managed Relationships (CMRs)" on page 6-23.

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-caching

Range of values:	n/a
Default value:	n/a
Parent elements:	`weblogic-rdbms-jar` `weblogic-rdbms-bean`

Function

The `relation-caching` element specifies relationship caching. For more information about relationship caching, see "Relationship Caching" on page 6-33.

Example

The `XML` stanza can contain the elements shown here:

```
<relationship-caching>
       <caching-name>cacheMoreBeans</caching-name>
       <caching-element>
              <cmr-field>accounts<</cmr-field>
              <group-name>acct_group</group-name>
              <caching-element>
                     <cmr-field>address</cmr-field>
                     <group-name>addr_group</group-name>
              </caching-element>
       </caching-element>
       <caching-element>
              <cmr-field>phone</cmr-field>
              <group-name>phone_group</group-name>
```

```
        </caching-element>
    </relationship-caching>
```

relationship-role-map

Range of values:	n/a
Default value:	n/a
Parent elements:	`weblogic-rdbms-relation` `weblogic-relationship-role`

Function

The `relationship-role-map` element specifies foreign key column to key column mapping for beans involved in a relationship.

A CMP bean that is involved in a relationship may be mapped to multiple DBMS tables (see the `table-map` element for more details). If the bean on the foreign key side of a one-to-one or one-to-many relationship is mapped to multiple tables, then the name of the table containing the foreign-key columns must be specified using the `foreign-key-table` element.

Conversely, if the bean on the primary key side of a one-to-one or one-to-many relationship or a bean participating in a m-n relationship is mapped to multiple tables, then the name of the table containing the primary key must be specified using the `primary-key-table` element.

If neither of the beans in a relationship is mapped to multiple tables, then the `foreign-key-table` and `primary-key-table` elements can be omitted because the tables being used are implicit.

For more information about container-managed relationships, see "Using Container-Managed Relationships (CMRs)" on page 6-23.

Example

Mapping a Bean on Foreign Key Side of a Relationship to Multiple Tables

The bean on the foreign-key side of a one-to-one relationship, `Fk_Bean`, is mapped to multiple tables. The table that holds the foreign key columns must be specified in the `foreign-key-table` element.

Fk_Bean is mapped to two tables: Fk_BeanTable_1 and Fk_BeanTable_2. The foreign key columns for the relationship are located in table Fk_BeanTable_2. The foreign key columns are named Fk_column_1 and Fk_column_2. The bean on the primary key side, Pk_Bean, is mapped to a single table with primary key columns Pk_table_pkColumn_1 and Pk_table_pkColumn_2:

```
<relationship-role-map
       <foreign-key-table>Fk_BeanTable_2</foreign-key-table>
       <column-map>
              <foreign-key-column>Fk_column_1</foreign-key-column>
              <key-column>Pk_table_pkColumn_1</key-column>
       </column-map>
       <column-map>
              <foreign-key-column>Fk_column_2</foreign-key-column>
              <key-column>Pk_table_pkColumn_2</key-column>
       </column-map>
</relationship-role-map>
```

The foreign-key-table element must be specified so that the container can know which table contains the foreign key columns.

Mapping a Bean on Primary Key Side of a Relationship to Multiple Tables

The bean on the primary key side of a one-to-one relationship, Pk_bean, is mapped to multiple tables, but the bean on the foreign key side of the relationship, Fk_Bean, is mapped to one table, Fk_BeanTable. The foreign key columns are named Fk_column_1 and Fk_column_2.

Pk_bean is mapped to tables:

- Pk_BeanTable_1 with primary key columns Pk_table1_pkColumn_1 and Pk_table1_pkColumn_2 and
- Pk_BeanTable_2 with primary key columns Pk_table2_pkColumn_1 and Pk_table2_pkColumn_2.

```
<relationship-role-map>
     <primary-key-table>Pk_BeanTable_1</primary-key-table>
     <column-map>
          <foreign-key-column>Fk_column_1</foreign-key-column>
          <key-column>Pk_table1_pkColumn_1</key-column>
     </column-map>
     <column-map>
          <foreign-key-column>Fk_column_2</foreign-key-column>
```

```
            <key-column>Pk_table1_pkColumn_2</key-column>
        </column-map>
    </relationship-role-map>
```

relationship-role-name

Range of values:	Must match the `ejb-relationship-role-name` of an `ejb-relationship-role` in the associated `ejb-jar.xml`.
Default value:	n/a
Parent elements:	`weblogic-rdbms-relation` `weblogic-relationship-role`

Function

The `relationship-role-name` element specifies the name of a relationship role.

For more information about container-managed relationships, see "Using Container-Managed Relationships (CMRs)" on page 6-23.

Example

See the examples for "weblogic-relationship-role" on page B-54.

serialize-byte-array-to-oracle-blob

Range of values:	n/a
Default value:	false
Parent elements:	`weblogic-rdbms-jar` `compatibility`

Function

This element, introduced in WebLogic Server 8.1 SP02, is a compatibility flag. It is used to specify whether a `cmp-field` of type `byte[]` mapped to a OracleBlob should be serialized. By default, the value of the tag is `false`, which means that the container will persist the `byte[]` directly and not serialize it.

In versions prior to WebLogic Server 8.1 SP02, the default behavior was to serialize a `cmp-field` of type byte[] mapped to an OracleBlob. To revert to the old behavior, set the value of `serialize-byte-array-to-oracle-blob` to true.

Example

```
<compatibility>
  <serialize-byte-array-to-oracle-blob>
  true
  </serialize-byte-array-to-oracle-blob>
</compatibility>
```

sql-select-distinct

Note: This element is deprecated. To achieve the same functionality, use the `SELECT DISTINCT` clause directly in finder queries.

Range of values:	`True | False`
Default value:	`False`
Requirements:	
Parent elements:	`weblogic-query`

Function

The `sql-select-distinct` element controls whether the generated SQL SELECT statement will contain a a DISTINCT qualifier. Using the DISTINCT qualifier caused the database to return unique rows.

Oracle database does not allow use of a `SELECT DISTINCT` 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 `isolation-level` of `TransactionReadCommittedForUpdate`. You specify the `transaction-isolation` element in the `weblogic-ejb-jar.xml`.

Example

The `XML` example contains the element shown here:

```
<sql-select-distinct>True</sql-select-distinct>
```

table-map

Range of values:	
Default value:	
Requirements:	Each `table-map` element must contain a mapping for the bean's primary key fields.
Parent elements:	`weblogic-rdbms-bean`

Function

A CMP bean can be mapped to one or more DBMS tables. The `table-map` element specifies a mapping between the `cmp-field`s of a bean and the columns of a table for all of the `cmp-fields` mapped to that table. If you map a CMP bean to multiple DBMS tables, then you must specify a `table-map` element for each the tables.

When you map a CMP bean to multiple tables, each table contains a row that maps to a particular bean instance. Consequently, all tables will contain the same number of rows at any point in time. In addition, each table contains the same set of homogeneous primary key values. Therefore, each table must have the same number of primary key columns and corresponding primary key columns in different tables must have the same type, although they may have different names.

Each `table-map` element must specify a mapping from the primary key column(s) for a particular table to the primary key field(s) of the bean. You can only map non-primary key fields to a single table.

For information about using the `verify-rows`, `verify-columns`, and `optimistic-column` elements, see "Check Data for Validity with Optimistic Concurrency" on page 6-37

Example

The `XML` stanza can contain the elements shown here:

```
<table-map>
       <table-nme>DeptTable</table-name>
       <field-map>
              <cmp-field>deptId1</cmp-field>
              <dbms-column>t1_deptId1_column</dbms-column>
       </field-map>
       <field-map>
```

```
            <cmp-field>deptId2</cmp-field>
            <dbms-column>t1_deptId2_column</dbms-column>
        </field-map>
        <field-map>
            <cmp-field>location</cmp-field>
            <dbms-column>location_column</dbms-column>
        </field-map>
            <cmp-field>budget</cmp-field>
            <dbms-column>budget</dbms-column>
        </field-map>
        <verify-rows>Read</verify-rows>
        <verify-columns>Version</verify-columns>
        <optimistic-column>ROW_VERSION</optimistic-column>
</table-map>
```

table-name

Range of values:	Valid, fully qualified SQL name of the source table in the database.
Default value:	
Requirements:	`table-name` must be set in all cases.
Parent elements:	`weblogic-rdbms-bean` `weblogic-rdbms-relation`

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

```
<weblogic-rdbms-jar>
     <weblogic-rdbms-bean>
        <ejb-name>containerManaged</ejb-name>
        <data-source-name>examples-dataSource-demoPool</data-source-name>
        <table-name>ejbAccounts</table-name>
```

```
     </weblogic-rdbms-bean>
</weblogic-rdbms-jar>
```

use-select-for-update

Range of values:	True | False
Default value:	False
Parent elements:	`weblogic-rdbms-bean`

Function

Enforces pessimistic concurrency on a per-bean basis. Specifying `True` causes `SELECT ... FOR UPDATE` to be used whenever the bean is loaded from the database. This is different from the transaction isolation level of `TransactionReadCommittedForUpdate` in that this is set at the bean level rather than the transaction level.

Example

```
<weblogic-rdbms.jar>
     <weblogic-rdbms-bean>
          ejb-name>containerManaged</ejb-name>
          <use-select-for-update>True</use-select-for-update>
     </weblogic-rdbms-bean>
</weblogic-rdbms-jar>
```

validate-db-schema-with

Range of values:	`MetaData | TableQuery`
Default value:	`TableQuery`
Parent elements:	`weblogic-rdbms-jar`

Function

The `validate-db-schema-with` element specifies that container-managed persistence checks that beans have been mapped to a valid database schema during deployment.

If you specify `MetaData` WebLogic Server uses the JDBC metadata to validate the schema.

If you specify `TableQuery`, the default setting, WebLogic Server queries the tables directly to verify that they have the schema expected by CMP runtime.

Example

The `XML` stanza can contain the elements shown here:

```
<validate-db-schema-with>TableQuery</validate-db-schema-with>
```

verify-columns

Range of values:	`Read | Modified | Version | Timestamp`
Default value:	`none`
Requirements:	`table-name` must be set in all cases.
Parent elements:	`weblogic-rdbms-bean` `table-map`

Function

The `verify-columns` element specifies the columns in a table that you want WebLogic Server to check for validity when you use the `optimistic` concurrency strategy. WebLogic Server

checks columns at the end of a transaction, before committing it to the database, to make sure that no other transaction has modified the data.

See "Choosing a Concurrency Strategy" on page 6-35 for more information.

Example

```
<verify-columns>Modified</verify-columns>
```

verify-rows

Range of values:	`Read | Modified`
Default value:	`Modified`
Requirements:	`table-name` must be set in all cases.
Parent elements:	`weblogic-rdbms-bean` `table-map`

Function

The `verify-rows` element specifies the rows in a table that the EJB container should check when optimistic concurrency is used.

- `Modified`— only rows that are updated or deleted by a transaction are checked. This value ensures that two transactions do not update the same row concurrently, resulting in a lost update, but allows reads and updates of different transactions to be interleaved. This results in a level of consistency that falls between the ANSI `READ_COMMITTED` and `REPEATABLE_READ` levels of consistency.
- `Read`—specifies that any row that is read by the transaction should be checked. This includes both rows that are simply read and rows that are read and then updated or deleted by the transaction. Specifying a value of `Read` results in additional overhead since it generally increases the amount of optimistic checking the EJB container must perform. With the `Read` option, committed transactions read a set of rows that are guaranteed not to be modified by another transaction until after the transaction commits.This results in a high level of consistency which is very close to the ANSI definition of `SERIALIZABLE` consistency.

Note: If `verify-rows` is set to `Read` then the `verify-columns` element may not have a value of `Modified`, as this combination would result in the EJB container checking only the modified rows.

See "Choosing a Concurrency Strategy" on page 6-35 for more information.

Example

```
<verify-rows>Modified</verify-rows>
```

weblogic-ql

Range of values:	
Default value:	
Parent elements:	`weblogic-rdbms-bean` `weblogic-query`

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 B-48.

weblogic-query

Range of values:	n/a
Default value:	mA
Parent elements:	`weblogic-rdbms-bean`

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>
   <description>...</description>
   <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>
   <group-name>...</group-name>
   <caching-name>...</caching-name>
   <max-elements>...</max-elements>
   <include-updates>...</include-updates>
   <sql-select-distinct>...</sql-select-distinct>
</weblogic-query>
```

weblogic-rdbms-bean

Range of values:	n/a
Default value:	n/a
Parent elements:	`weblogic-rdbms-jar`

Function

The `weblogic-rdbms-bean` specifies an entity bean that is managed by the WebLogic RDBMS CMP persistence type.

Example

```
weblogic-rdbms-bean
        ejb-name
        data-source-name
                table-map
        field-group
        relationship-caching
        weblogic-query
        dalay-database-insert-until
        automatic-key-generation
        check-exists-on-method
```

weblogic-rdbms-jar

Function

The `weblogic-rdbms-jar` element is the root level element of a WebLogic RDBMS CMP deployment descriptor. This element contains the deployment information for one or more entity beans and an optional set of relations.

Example

The `XML` structure of `weblogic-rdbms-jar` is:

```
weblogic-rdbms-jar
        weblogic-rdbms-bean
        weblogic-rdbms-relation
        create-default-dbms-tables
        validate-db-schema-with
        database-type
```

weblogic-rdbms-relation

Range of values:	n/a
Default value:	n/a
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 "Using Container-Managed Relationships (CMRs)" on page 6-23.

Examples

See the sections that following for examples of how one-to-one, one-to-many, and many-to-many relationships are configured.

Defining a One-to-One Relationship

Listing 8-3 shows the `weblogic-rdbms-bean` stanza that defines a one-to-one relationship between the entities defined in Listing 8-1 and Listing 8-2. The `weblogic-rdbms-relation` stanza is in the `weblogic-cmp-jar.xml` file, after the `weblogic-rdbms-bean` stanzas.

Listing 8-1 Bean 1

```
<weblogic-rdbms-bean>
  <ejb-name>CountryEJB</ejb-name>
  <data-source-name>wlsd21-datasource</data-source-name>
  <table-map>
    <table-name>EXAMPLE07_COUNTRY</table-name>
    <field-map>
      <cmp-field>name</cmp-field>
      <dbms-column>NAME</dbms-column>
    </field-map>
    <field-map>
      <cmp-field>continent</cmp-field>
      <dbms-column>CONTINENT</dbms-column>
    </field-map>
  </table-map>
</weblogic-rdbms-bean>
```

Listing 8-2 Bean 2

```
<weblogic-rdbms-bean>
   <ejb-name>CapitalEJB</ejb-name>
   <data-source-name>wlsd21-datasource</data-source-name>
      <table-map>
      <table-name>EXAMPLE07_CAPITAL</table-name>
      <field-map>
         <cmp-field>CAPITAL_NAME</cmp-field>
         <dbms-column>NAME</dbms-column>
      </field-map>
      <field-map>
         <cmp-field>continent</cmp-field>
```

```
         <dbms-column>CONTINENT</dbms-column>
      </field-map>
   </table-map>
</weblogic-rdbms-bean>
```

Listing 8-3 <weblogic-rdbms-relation> Stanza for a One-to-One Relationship

```
<weblogic-rdbms-relation>
   <relation-name>CountryCapitalRel</relation-name>
   <weblogic-relationship-role>
      <relationship-role-name>CountryRole</relationship-role-name>
      <relationship-role-map>
          <column-map>
             <foreign-key-column>CAPITAL_NAME</foreign-key-column>
             <key-column>NAME</key-column>
          </column-map>
      </relationship-role-map>
   </weblogic-relationship-role>
</weblogic-rdbms-relation>
```

Note: `CAPITAL_NAME` is the column name for the foreign key in the Country table.

`NAME` is the column name for the primary key located in the Capital table

`<relationship-role-name>` contains the relation field specified in `<cmr-field>` in the `<ejb-relationship-role>` stanza in `ejb-jar.xml`.

Defining a One-to-Many Relationship

Listing 8-4 contains a sample `<weblogic-rdbms-relation>` stanza that defines a one-to-many relationship:

Listing 8-4 <weblogic-rdbms-relation> Stanza for a One-to-Many Relationship

```
<weblogic-rdbms-relation>
   <relation-name>OwnerDogRel</relation-name>
   <weblogic-relationship-role>
      <relationship-role-name>DogRole</relationship-role-name>
      <relationship-role-map>
```

```
            <column-map>
               <foreign-key-column>OWNER_NAME</foreign-<key-column>
               <key-column>NAME</key-column>
            </column-map>
         <relationship-role-map>
      </weblogic-relationship-role>
</weblogic-rdbms-relation>
```

Note: `<relationship-role-name>` contains the relation field specified in `<cmr-field>` in the `<ejb-relationship-role>` stanza in `ejb-jar.xml`.

`<foreign-key-column>` must specify the column in the table on the "many" side of the relationship.

Defining a Many-to-Many Relationship

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 maps to the primary keys of the entities involved in the relationship.

The following example shows a many-to-many relationship between the `FRIENDS` bean and the `EMPLOYEES` bean.

Listing 8-5 <weblogic-rdbms-relation> Stanza for a Many-to-Many Relationship

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

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

In Figure 8-4, the FRIENDS join table has two columns, called 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 (key-column) of the EMPLOYEES table is id. For this example, assume that the EMPLOYEES bean is mapped to a single table. If the EMPLOYEES bean is mapped to multiple tables, then the table containing the primary key column (key-column) must be specified in the relationship-role-map. For more information, see "relationship-role-map" on page B-38.

weblogic-relationship-role

Range of values:	n/a
Default value:	n/a
Parent elements:	weblogic-rdbms-jar weblogic-rdbms-relation

Function

The weblogic-relationship-role element specifies the following DBMS schema information for an ejb-relationship-role specified in ejb-jar.xml:

- The relationship-role-map sub-element specifies the mapping between a foreign key and a primary key, for one side of a relationship. For a 1-1 or 1-n relationship, only the role on the foreign-key side of the relationship specifies a mapping. Both roles specify a mapping for a m-m relationship. For details and examples, see "relationship-role-map" on page B-38.
- A group-name can be used to indicate the field-group to be loaded when the bean corresponding to the role is loaded as a result of traversing the relationship, i.e. calling a cmr getXXX method.

- The `db-cascade-delete` tag can to used to specify that cascade deletes use the built-in cascade delete facilities of the underlying DBMS. For more information, see "db-cascade-delete" on page B-16.

For more information about container-managed relationships, see "Using Container-Managed Relationships (CMRs)" on page 6-23.

Example

```
<weblogic-relationship-role>
    <relationship-role-name>...</relationship-role-name>
    <group-name> ....</group-name>
    <relationship-role-map>...
       ....
    </relationship-role-map>
    <db-cascade-delete/>
</weblogic-relationship-role>
```

appc and ejbc Reference

appc

The `appc` compiler generates and compiles the classes needed to deploy EJBs and JSPs to WebLogic Server. It also validates the deployment descriptors for compliance with the current specifications at both the individual module level and the application level. The application-level checks include checks between the application-level deployment descriptors and the individual modules as well as validation checks across the modules.

Advantages of Using appc

The `appc` tool offers the following benefits:

- The flexibility of compiling an entire application, rather than compiling individual modules separately and combining them into an EAR after the fact.
- Validation checks across all modules and validation of application-level deployment descriptors against the various modules, because WebLogic Server has access to all modules during EAR compilation.

  Previously, a user wanting to compile all modules within an EAR file had to extract the individual components of an EAR and manually execute the appropriate compiler (`jspc` or `ejbc`) to prepare the module for deployment. `appc` automates this process and makes additional pre-deployment validation checks not previously possible.
- It is easy to identify and correct errors `appc` produces.

If an error occurs while running `appc` from the command line, appc exits with an error message.

By contrast, if you defer compilation to the time of deployment and a compilation error occurs, the server fails the deployment and goes on with its work. To determine why deployment failed, you must examine the server output, fix the problem and then redeploy.

- By running `appc` prior to deployment, you potentially reduce the number of time a bean is compiled.

  For example, if you deploy a JAR file to a cluster of 3 servers, the JAR file is copied to each of the three servers for deployment. If the JAR file wasn't precompiled, each of the three servers will have to compile the file during deployment.

appc Syntax

Use the following syntax to run `appc`:

```
prompt>java weblogic.appc [options] <ear, jar, or war file or directory>
```

Designating Alternative Deployment Descriptors

J2EE allows you to designate an alternative J2EE deployment descriptor for an EJB or web application module, using the `<alt-dd>` element in the `<module>` stanza of `application.xml`.

You can use `<alt-dd>` to specify an alternate deployment descriptor only for the J2EE deployment descriptors, `web.xml` and `ejb-jar.xml`. As of WebLogic Server 8.1 SP01, if you specify an alternative deployment descriptor for a module in `alt-dd`, `appc` will compile it using the EJB using the alternative descriptor file.

For more information about the `<alt-dd>` element, see "module" in *Developing WebLogic Server Applications*.

In WebLogic Server 8.1 SP01and later, you can use `appc` command line options to designate alternative J2EE and WebLogic Server deployment descriptors for an application, as shown below:

- `-altappdd <file>`—Use this option to specify the full path and file name of an alternative J2EE deployment descriptor file, `application.xml`.
- `-altwlsappdd <file>`—Use this option to specify the full path and file name of an alternative WebLogic application deployment descriptor., `weblogic-application.xml`

appc Options

Table C-1 lists `appc` command line options

Table C-1 appc Command Line Options

Option	Description
`-advanced`	Prints advanced usage options.
`-altappdd`	Designates an alternative J2EE application deployment descriptor.
`-altwlsappdd`	Designates an alternative WebLogic Server application deployment descriptor.
`-basicClientJar`	Does not include deployment descriptors in client JARs generated for EJBs.
`-classpath <path>`	Selects the classpath to use during compilation.
`-compiler <javac>`	Selects the Java compiler to use.
`-convertDD`	Attempts to update deployment descriptors to the latest version.
`-deprecation`	Warns about deprecated calls.
`-forceGeneration`	Forces generation of EJB and JSP classes. Without this flag, the classes will not be regenerated unless a checksum indicates that it is necessary.
`-g`	Compiles debugging information into a class file.
`-idl`	Generates IDL for EJB remote interfaces.
`-idlDirectory <dir>`	Specifies the directory where IDL files will be created (default: target directory or JAR)
`-idlFactories`	Generates factory methods for valuetypes.
`-idlMethodSignatures <>`	Specifies the method signatures used to trigger IDL code generation.
`-idlNoAbstractInterfaces`	Does not generate abstract interfaces and methods/attributes that contain them.

Option	Description
`-idlNoValueTypes`	Does not generate valuetypes and the methods/attributes that contain them.
`-idlOrbix`	Generates IDL somewhat compatible with Orbix 2000 2.0 C++.
`-idlOverwrite`	Always overwrites existing IDL files.
`-idlVerbose`	Displays verbose information for IDL generation.
`-idlVisibroker`	Generates IDL somewhat compatible with Visibroker 4.5 C++.
`-iiop`	Generates CORBA stubs for EJBs.
`-iiopDirectory <dir>`	Specifies the directory where IIOP stub files will be written (default: target directory or JAR)
`-timeout`	Used in conjunction with remote-client-timeout.
`-J<option>`	Passes flags through to Java runtime.
`-keepgenerated`	Keeps the generated `.java` files.
`-lineNumbers`	Adds line numbers to generated class files to aid in debugging.
`-normi`	Passes flags through to Symantec's sj.
`-nowarn`	Compiles without warnings.
`-O`	Compiles with optimization on.
`-output <file>`	Specifies an alternate output archive or directory. If not set, the output is placed in the source archive or directory.
`-print`	Prints the standard usage message.
`-verbose`	Compiles with verbose output.
`-version`	Prints `appc` version information.

appc and EJBs

`weblogic.appc` performs the following EJB-related functions:

- Generates WebLogic Server container classes for the EJBs.
- Checks all EJB classes and interfaces for compliance with the EJB specification.
- Checks deployment descriptors for potential configuration problems. For example, if there is a cmp field declared in `ejb-jar.xml`, `appc` verifies that the column is mapped in the `weblogic-cmp-rdbms.xml` deployment descriptor.
- Runs each EJB container class through the RMI compiler to create RMI descriptors necessary to dynamically generate stubs and skeletons.

By default, `appc` uses `javac` as a compiler. For faster performance, specify a different compiler (such as Symantec's `sj`) using the command-line `-compiler` flag or via the Administration Console. See Configuring Compiler Options at http://e-docs.bea.com/wls/docs81/ConsoleHelp/ejb.html/#configuring_compiler_options.

For the location of the public version of `weblogic-ejb-jar.xml`, see Chapter , "." For the location of the public version of `weblogic-cmp-jar.xml`, see Chapter , "."

ejbc

Note: `ejbc` is deprecated. Use `appc` instead. See "appc" on page C-1.

Use the `weblogic.ejbc` tool for generating and compiling EJB container classes. If you compile JAR files for deployment into the EJB container, you must use `weblogic.ejbc` to generate the container classes.

`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.

Note: `ejbc` accepts both JAR files and exploded directories as input.

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 or via the Administration Console. See Configuring Compiler Options.

Although versions of the WebLogic-specific XML deployment descriptor files are published on or web site for your convenience, an internal version is shipped with the product for use by `weblogic.ejbc`.

For the location of the public version of `weblogic-ejb-jar.xml and weblogic-cmp-jar.xml`, see "Document Type Definitions and DOCTYPE Header Information" on page A-1.

Advantages of Using ejbc

The `ejbc` tool offers the following benefits:

- It is easy to identify and correct errors `ejbc` produces.

  If an error occurs while running `ejbc` from the command line, ejbc exits with an error message.

  By contrast, if you defer compilation to the time of deployment and a compilation error occurs, the server fails the deployment and goes on with its work. To determine why deployment failed, you must examine the server output, fix the problem and then redeploy.

- By running `ejbc` prior to deployment, you potentially reduce the number of time a bean is compiled.

  For example, if you deploy a JAR file to a cluster of 3 servers, the JAR file is copied to each of the three servers for deployment. If the JAR file wasn't precompiled, each of the three servers will have to compile the file during deployment.

ejbc Syntax

```
prompt> java weblogic.ejbc [options] <source directory or 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

Argument	Description
<source directory or jar file>	Specifies the exploded source directory or 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

Option	Description
`-help`	Prints a list of all options available for the compiler.
`-version`	Prints `ejbc` version information.
-basicClientJar	Does not include deployment descriptors in client JARs generated for EJBs.
`-dispatchPolicy <queueName>`	Specifies a configured execute queue that the EJB should use for obtaining execute threads in WebLogic Server. For more information, see Using Execute Queues to Control Thread Usage.
`-forceGeneration`	Forces generation of EJB classes. Without this flag, the classes may not be regenerated (if determined to be unnecessary).
`-idl`	Generates CORBA Interface Definition Language for remote interfaces.
-idlNoValueTypes	Does not generate valuetypes and the methods/attributes that contain them.
-idlFactories	Generates factory methods for valuetypes.
-idlVisibroker	Generates IDL somewhat compatible with Visibroker 4.5 C++.

-idlOrbix	Generates IDL somewhat compatible with Orbix 2000 2.0 C++.
`-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.
-idlMethodSignatures <>	Specifies the method signatures used to trigger IDL code generation.
-iiop	Generates CORBA stubs for EJBs.
-iiopDirectory <dir>	Specifies the directory where IIOP stub files will be written (default: target directory or JAR)
`-timeout`	Used in conjunction with remote-client-timeout.
`-J`	Specifies the heap size for `weblogic.ejbc`. Use as follows: `java weblogic.ejbc -J-mx256m input.jar output.jar`
`-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 augments the system or shell CLASSPATH.
-convertDD	Attempts to update deployment descriptors to the latest version.

APPENDIX D

EJBGen Reference

EJBGen is an Enterprise JavaBeans 2.0 code generator. If you annotate your Bean class file with javadoc tags, you can use EJBGen to generate the Remote and Home classes and the deployment descriptor files for an EJB application.

BEA recommends that you use EJBGen to generate deployment descriptors; this BEA best practice allows for easier and simpler maintenance of EJBs. If you use EJBGen, you have to write and annotate only one bean class file, which simplifies writing, debugging and maintenance. If you use WebLogic Workshop as a development environment, WebLogic Workshop automatically inserts EJBGen tags for you. These sections provide instructions on using EJBGen and define EJBGen command options and tag.

- "Running EJBGen" on page D-1
- "Examples of EJBGen Tag Usage" on page D-13
- "EJBGen Tag Reference" on page D-17

Running EJBGen

These sections describe how to process an annotated bean file using EJBGen:

- "EJBGen Distribution and Examples" on page D-2
- "Invoking EJBGen" on page D-2
- "Using Key EJBGen Features" on page D-8

EJBGen Distribution and Examples

As of WebLogic Server 8.1 SP02, EJBGen and associated classes are not included in `weblogic.jar`; instead, they are provided as a separate archive—`ejbgen.jar` in the `WL_HOME/server/lib` of the WebLogic Server distribution. To use EJBGen, put `ejbgen.jar` in your CLASSPATH.

If you have installed BEA WebLogic 8.1 examples, see `WL_HOME\samples\server\examples\src\examples\ejb20\relationships` for an example application called Bands that uses EJBGen.

Invoking EJBGen

Use this command to run EJBGen on a bean source file:

```
javadoc -docletpath ejbgen.jar -doclet weblogic.tools.ejbgen.EJBGen
(YourBean).java
```

If you do not have `ejbgen.jar` in your classpath, add the path to `weblogic.jar` as follows:

```
javadoc -docletpath <path_to_ejbgen.jar> ejbgen.jar -doclet
weblogic.tools.ejbgen.EJBGen (YourBean).java
```

If you are invoking EJBGen for an EJB that has relationships with other EJBs, invoke the related EJBs by naming them, following your EJB, in the invocation, as follows:

```
javadoc -docletpath ejbgen.jar -doclet weblogic.tools.ejbgen.EJBGen
(YourBean).java (RelatedBean).java
```

EJBGen Command Options

Table D-1 defines EJBGen command options.

Table D-1 EJBGen Command Options

Option	Default	Definition
`-checkTags`	N/A	If invoked with this option, EJBGen will not generate any classes but will search the classes supplied on the command line for tags that are not valid EJBGen tags.
`-classpath`	N/A	Class path.
`-d [directory]`		The directory under which all the files will be created.

Option	Default	Definition
`-descriptorDir [directory]`		The directory under which all the descriptor files will be created, relative to the output directory specified with the -d [directory] option.
`-docTag tagName`		If specified, EJBGen prints out the detailed documentation for this tag, including all the recognized attributes. Note that even though this option does not need any source file, you still need to specify an existing `.java` class on the command line, or Javadoc will emit an error message even though it recognized the flag.
`-docTags`		Print out all the tags known by EJBGen. Note that even though this option does not need any source file, you still need to specify an existing `.java` class on the command line, or Javadoc will emit an error message even though it recognized the flag.
`-docTagsHtml`		Same as `-docTags`, but generates an HTML document.
`-ejbPrefix [string]`	""	The prefix to use when generating the EJB class.
`-ejbSuffix [string]`	Bean or EJB	The suffix to use when generating the EJB class.
`-ejbgenQuiet [boolean]`	N/A	Make EJBGen less verbose.
`-ejbgenVerbose [boolean]`	N/A	Make EJBGen less verbose.
`errorFile`	N/A	The path of the file where errors will be redirected.
`exitOnError [boolean]`	N/A	EJBGen should exit in case of an error.
`-homeBaseClass [string (className)]`	N/A	The base class for home remote interfaces.
`-experimental [boolean]`	N/A	Enable experimental features.

Option	Default	Definition
`-extractTemplates [directory]`	N/A	The directory where to extract the templates.
`-forceGeneration [boolean]`	N/A	Force the generation of all files regardless of time stamps.
`-homeBaseClass [string (className)]`	N/A	The base class for home remote interfaces.
`-ignorePackage`		If this flag is set, EJBGen ignores the package name of the Java files it generates and creates those in the output directory as specified by the -d flag (or in the current directory if no -d was specified).
`-jndiPrefix [string]`	“”	The prefix to use for `@remote-jndi-name` and `@local-jndi-name`
`-jndiSuffix [string]`	“”	The suffix to use for `@remote-jndi-name` and `@local-jndi-name`
`-localBaseClass [string (className)]`	N/A	The base class for local interfaces.
`localHomeBaseClass [className]`	N/A	The base class for local home interfaces
`-localHomePrefix [string]`	“”	The prefix to use when generating the local EJB class.
`-localHomeSuffix [string]`	LocalHome	The suffix to use when generating the local EJB class.
`-localPrefix [string]`	“”	The prefix to use when generating the local EJB class.
`-localSuffix [string] (`		The suffix to use when generating the local EJB class.
`-noImports [boolean]`	false	Disable propagation of java imports to generated classes.
`-noLocalInterfaces`		If specified, local interfaces are not generated.

Option	Default	Definition
`-noRedirection noRedirection ejbgen.noRedirection [boolean]`	N/A	Cancel all redirection performed by EJBGen
`-noRemoteInterfaces [boolean]`		If specified, remote interfaces are not generated.
`-noticeFile [file]`	N/A	The path of the file where notices will be redirected.
`-noValueClasses [boolean]`	N/A	If specified, value classes will not be generated.
`-pfd1`	N/A	If this flag is set, EJBGen will generate deployment descriptors compatible with the Public Final Draft 1 of the EJB 2.0 specification. You should use this flag if you are using any version anterior to Weblogic 6.1.
`-pfd2`	N/A	If this flag is set, EJBGen will generate deployment descriptors compatible with the Public Final Draft 2 of the EJB 2.0 specification. You should use this flag if you are using any version anterior to Weblogic 6.1.
`-primaryKeyBaseClass [string (class name)]`	N/A	The base class for generated primary keys.
`-printDiagnostic [boolean]`	N/A	Prints a diagnostic of all the tags found by EJBGen
`-printTags [boolean]`	N/A	Display all the tags recognized by EJBGen
`-propertyFile [fileName]`		The name of a property file that EJBGen reads to define substitution variables. See "Using Property Files with EJBGen" on page D-9.
`-remoteBaseClass [string (className)]`	N/A	The base class for remote interfaces.
`-remoteHomePrefix [string]`	""	The prefix to use when generating the remote EJB home class.

Option	Default	Definition
`-remoteHomeSuffix [string]`		The suffix to use when generating the remote EJB home class.
`-remotePrefix [string]`	""	The prefix to use when generating the remote EJB class.
`-remoteSuffix [string]`	""	The suffix to use when generating the remote EJB class.
`source [version number]`	1.3	Version of the JDK to use.
`-sourcepath [directory]`	N/A	Source path.
`-templateDir [directory]`	N/A	The directory where the templates are stored.
`-toStringForPrimitivesOnly`		If specified, the `toString()` methods of value objects only display container-managed persistence fields that are primitives. This flag fixes the problem of circular references between value objects (`AValue.toString()` invoking `BValue.toString()` invoking `AValue.toString()`, etcetera).
`-valueBaseClass [string (class name)]`	N/A	The suffix to use when generating the value object class.
`-valueObjectPrefix [string]`	N/A	The prefix to use when generating the value object class.
`-valueObjectSuffix [string]`	N/A	The suffix to use when generating the value object class.
`-valuePackage [string (package name)]`	N/A	Package the value classes belong to.
`verbose [boolean]`	false	Display more information.
`-wls6 [boolean]`	N/A	Use this flag to specify that EJBGen generate WebLogic Server 7.0 deployment descriptors.

Option	Default	Definition
`-wls7 [boolean]`	N/A	Use this flag to specify that EJBGen generate WebLogic Server 7.0 deployment descriptors.
`-wls71 [boolean]`	N/A	Use this flag to specify that EJBGen generate WebLogic Server 7.1 deployment descriptors.
`-wls81 [boolean]`	N/A	Use this flag to specify that EJBGen generate WebLogic Server 8.1 deployment descriptors.
`-wls9 [boolean]`	N/A	Use this flag to specify that EJBGen generate WebLogic Server 9.0 deployment descriptors.
`-workshopDocTags [boolean]`	N/A	Display the help in Workshop HTML format.
`-xmlEncoding [string]`	N/A	The encoding to use when generating XML files (default: UTF-8).

Javadoc Warnings with Uncompiled Value Type Classes

If an EJBGen class contains value types (e.g., parameter values, return values) that are Java classes that have not been compiled, then Javadoc will output what appear to be `javac` compilation errors, as in Listing 8-6:

Listing 8-6 Sample Javadoc Output When Value Type Classes Are Uncompiled

```
[javadoc]
C:\toddk\dev\issues\cr100528\src\test\po\PurchaseOrderBean.java:29: cannot
resolve symbol [javadoc] symbol : class PurchaseOrder [javadoc] location:
class test.po.PurchaseOrderBean [javadoc] public boolean
submitPO(PurchaseOrder po) throws SubmitException {

[javadoc]
[javadoc]
C:\toddk\dev\issues\cr100528\src\test\po\PurchaseOrderBean.java:29: cannot
resolve symbol [javadoc] symbol : class SubmitException [javadoc] location:
class test.po.PurchaseOrderBean [javadoc] public boolean
submitPO(PurchaseOrder po) throws SubmitException {
```

These are Javadoc warnings, not errors. Because the error messages resemble javac compiler errors, it is easy to interpret them incorrectly as errors. Javadoc returns with an exit status that indicates success; therefore the Ant EJBGen task does not cause the build to fail.

Using Key EJBGen Features

These sections describe key EJBGen features and how to use them:

- "Controlling the Files EJBGen Generates" on page D-8
- "Using Property Files with EJBGen" on page D-9
- "Using User Variables" on page D-9
- "Surround Attributes that Contain Spaces With Double Quotes" on page D-11
- "Using Predefined Variables" on page D-10
- "Surround Attributes that Contain Spaces With Double Quotes" on page D-11
- "EJBGen Supports Tag Inheritance" on page D-11
- "EJBGen Supports Attribute Inheritance" on page D-12

Controlling the Files EJBGen Generates

By default, EJBGen generates the following files:

- Remote bean and home interfaces (if @ejbgen:jndi-name with "remote" was specified)
- Local bean and home interfaces (if @ejbgen:jndi-name with "local" was specified)
- Value object classes
- Primary key classes (if applicable)

You can selectively disable the generation of these files by using the `@ejbgen:file-generation` tag. This can be done on a per-bean basis. The following code sample suppresses value class generation:

```
/**
*   @ejbgen:file-generation
*   value-class = False
*/
```

Note: The `-noValueClasses`, `-noRemoteInterfaces` or `-noLocalInterfaces`, these command-line options override any tag found on the beans.

Using Property Files with EJBGen

EJBGen can gather information for generation from property files, as discussed in this section.

Use the -`propertyFile` option to tell `EJBGen` to parse a properties file. Listing 8-7 illustrates a sample property file.

Listing 8-7 Sample EJBGen Property File

```
# property-file
#
remote-jndi-name = Account
```

Use the following syntax to invoke EJBGen with the `-propertyFile` option:

```
javadoc -docletpath ejbgen.jar -doclet EJBGen -propertyFile property-file
AccountBean.java
```

EJBGen recognizes two kinds of variables in a properties file: user variables and predefined variables.

Using User Variables

EJBGen tags can use variables instead of strings. These variables must be enclosed with "${" and "}", as in Listing 8-8.

Listing 8-8 User Variables in Place of Strings

```
@ejbgen:jndi-name
remote = ${remote-jndi-name}
```

Variables can be used anywhere after an EJBGen tag, so they can contain whole tag definitions, as in Listing 8-9.

Listing 8-9 User Variables as a Whole Tag Definition

```
@ejbgen:jndi-name

  ${jndi-name-tag}
```

```
#
#
# property-file
#
jndi-name-tag = remote = RemoteAccount local = LocalAccount
```

Using Predefined Variables

EJBGen recognizes a number of predefined variables. These variables are not supposed to be used in tags but EJBGen will use them at various places depending on their role. Here is the list of recognized variables:

- `remote.baseClass`

  If specified, the value of this variable will be used as the base class for all generated remote classes.

- `home.baseClass`

  If specified, the value of this variable will be used as the base class for all generated remote home classes.

- `local.baseClass`

  If specified, the value of this variable will be used as the base class for all generated local classes.

- `localHome.baseClass`

  If specified, the value of this variable will be used as the base class for all generated local home classes.

- `value.baseClass`

  If specified, the value of this variable will be used as the base class for all generated value classes.

- `value.package`

  If specified, the value of this variable will be used as the package for all generated value classes.

- `value.interfaces`

  If specified, the value of this variable will be used as the interfaces the value class should implement. This variable can be a list of interfaces separated by commas.

You can make these variables more specific by prefixing them with an `EJBName`. For example, consider the following property file:

```
#
# property-file
#
Account.home.baseClass = BaseAccountHome
home.baseClass = BaseHome
value.package = value
containerManaged.value.package=valuePackageForContainerManaged
```

All homes generated by EJBGen extend the class `BaseHome` except the home of EJB "Account", which extends `BaseAccountHome`.

Surround Attributes that Contain Spaces With Double Quotes

When you specify an attribute that contains spaces, surround the attribute value with double quotes. For example:

group-names = "group1, group2"

EJBGen Supports Tag Inheritance

One of the main features in EJBGen 2.0 is tag inheritance. You inherit `EJBGen` annotations the same way you do it in Java. For example, assume that you have an EJB named `AccountEJB` with a base class `BaseAccount`, as in Listing 8-10:

Listing 8-10 Tag Inheritance

```
/**
*   @ejbgen:jndi-name
*    remote="BaseRemote"
*/

public class BaseAccount implements EntityBean {
// ...
}
```

Now, define your EJB class:

```
/**
*   @ejbgen:entity
```

```
*    ejb-name = containerManaged
*    table-name = ejbAccounts
*    data-source-name = examples-dataSource-demoPool
*
*    Note that we inherit the JNDI name defined in BaseAccount
*/
public class AccountEJB extends BaseAccount {
// ...
```

EJBGen Supports Attribute Inheritance

You can also inherit attributes. This is a very powerful feature with a lot of potential uses. For example, you could define a setting common to all your EJB's, such as `max-beans-in-cache`, as in Listing 8-11.

Listing 8-11 Attribute Inheritance

```
/**
*   @ejbgen:entity
*    max-beans-in-cache = 300
*/
public class BaseAccount implements EntityBean {
// ...
}
```

Then, you define your entity bean as follows:

```
/**
*    @ejbgen:entity
*    ejb-name = containerManaged
*    table-name = ejbAccounts
*    data-source-name = examples-dataSource-demoPool
*
*    // automatically inherit the attribute max-beans-in-cache = 300
*/
public class AccountEJB extends BaseAccount {
// ...
```

Of course, you still have the ability to modify the inherited value. For example, for a specific bean, you could change the default value of `max-beans-in-cache` from 300 to 400:

```
/**
*   @ejbgen:entity
*    ejb-name = containerManaged
*    table-name = ejbAccounts
*    data-source-name = examples-dataSource-demoPool
*    max-beans-in-cache = 400
*
*    // the above setting overrides the one defined in the base class
*/
public class AccountEJB extends BaseAccount {
// ...
```

Examples of EJBGen Tag Usage

These sections contain example source code annotated with EJBGen tags.

- "A Sample Bean File Annotated with EJBGen Tags" on page D-13
- "Mapping an Entity Bean to Several Tables with EJBGen" on page D-15
- "Specifying Relationship Caching with EJBGen Tags" on page D-15
- "Specifying Relationships with EJBGen Tags" on page D-16

A Sample Bean File Annotated with EJBGen Tags

This example shows an CMP EJB 2.0 entity bean file with annotations that EJBGen can use to generate the remote and home interfaces and the deployment descriptor files. `AccountBean.java` is the main bean class.

```
/**
*   @ejbgen:entity
*    ejb-name = AccountEJB-OneToMany
*    data-source-name = examples-dataSource-demoPool
*    table-name = Accounts
*    prim-key-class = java.lang.String
*
```

```
*   @ejbgen:jndi-name
*    local = one2many.AccountHome
*
*   @ejbgen:finder
*    signature = "Account findAccount(double balanceEqual)"
*    ejb-ql = "WHERE balance = ?1"
*
*   @ejbgen:finder
*    signature = "Collection findBigAccounts(double balanceGreaterThan)"
*    ejb-ql = "WHERE balance > ?1"
*
*   @ejbgen:relation
*    name = Customer-Account
*    target-ejb = CustomerEJB-OneToMany
*    multiplicity = many
*    cmr-field = customer
*
*/
abstract public class AccountBean implements EntityBean {

/**
*   @ejbgen:cmp-field column = acct_id
*    @ejbgen:primkey-field
*    @ejbgen:remote-method transaction-attribute = Required
*/
   abstract public String getAccountId();
   abstract public void setAccountId(String val);
// ....
}
```

This example illustrates the two types of EJBGen tags: class tags and method tags, depending on where you can use them.

After completing the file, invoke EJBGen through the following Javadoc command:

```
javadoc -docletpath weblogic.jar -doclet weblogic.tools.ejbgen.EJBGen
AccountBean.java
```

The Javadoc command generates the following files:

- `Account.java`
- `AccountHome.java`
- `ejb-jar.xml`
- `weblogic-ejb-jar.xml`
- `weblogic-cmp-jar.xml`

"EJBGen Tag Reference" on page D-17 defines the EJBGen tags you can use to annotate your bean class file.

Mapping an Entity Bean to Several Tables with EJBGen

By default, entity beans are mapped to one table, with the attribute `table-name` on the tag `@ejbgen:entity`. If you want to map your entity bean to more than one table, you can use the `table-name` attribute on individual `@ejbgen:cmp-fields`. All the container-managed persistence fields that do not have a `table-name` attribute will use the table specified on `@ejbgen:entity` (which can therefore be considered as the "default" table).

If you want to map an entity bean to several tables, you need to specify a comma-separated list of tables in the `table-name` attribute (and also on a column). For example:

Make sure that the number of tables matches the number of columns, and that the columns exist in the corresponding table.

Specifying Relationship Caching with EJBGen Tags

You can specify relationship caching elements with the `@ejbgen:relationship-caching` tag. This XML element can be nested (recursive). In other words, you can specify a tree representing all the relationship elements that you want to see cached.

In order to support this tree structure, `@ejbgen:relationship-caching` has two attributes, `id` and `parent-id`. These attributes do not correspond to any XML, they simply allow you to specify your tree structure.

For example, if a caching-element A needs to have two children elements B and C, all you need to specify is an `id` for A (say "root0") and a `parent-id` for B and C equal to that name.

Listing 8-12 illustrates specifying relationship caching.

Listing 8-12 Specifying Relationship Caching

```
*   @ejbgen:relationship-caching-element
*    caching-name = cacheMoreBeans
*    cmr-field = employee
*    group-name = emp-group
*    id = A
*
*   @ejbgen:relationship-caching-element
*    caching-name = cacheMoreBeans
*    cmr-field = employee_ssn
*    group-name = emp-group
*    parent-id = A
*
*   @ejbgen:relationship-caching-element
*   caching-name = cacheMoreBeans
*   cmr-field = employee_zip
*   group-name = emp-group
*   *   parent-id = A
```

Specifying Relationships with EJBGen Tags

Unidirectional relationships are achieved by specifying only the first two parameters of the relation tag (no CMR_field, no joint table).

Many-to-many relationships must specify an extra table (parameter “joint table”) which must contain at least two columns. The names of these columns must match the names of the columns containing the foreign keys of the two beans being joined.

Upgrading Relationships to Use Local Interfaces

If you have EJB's based on a version of the EJB 2.0 specification that predates PFD2, you will need to update them. As of PFD2, relationships must be based on local interfaces, not remote ones. Here are the steps you need to take up upgrade your EJB's:

1. Add a JNDI name for your local home interface:

   ```
   @ejbgen:jndi-name
   remote = AccountHome
   local = AccountLocalHome
   ```

2. Add `@ejbgen:local-method` tags to any method you want to see appear on the local home interface. A method can have both `@ejbgen:local-method` and `@ejbgen:remote-method`.
3. Change the return type of your container-managed relationship (CMP) accessing methods to that of the local EJB object (this only applies to single-valued CMR fields since multi-valued CMR fields are collections. Keep in mind that these collections contain Local EJB objects):

```
/**
*   @ejbgen:cmr-field
*
*   @ejbgen:remote-method
*   @ejbgen:local-method
*/
abstract public LocalCustomer getCustomer();
```

EJBGen Tag Reference

This section provides a complete reference for EJBGen tags you can use to annotate your bean class file.

@ejbgen:automatic-key-generation

Where: Class

Applicable to: Entity bean

Attribute	Description	Required
cache-size	The size of the key cache.	Yes
name	The name of the generator.	Yes
type	The type of the generator.	Yes

@ejbgen:cmp-field

Where: Method

Applicable to: Entity bean

Attribute	Description	Required
column	The column where this CMP field will be mapped. See "Mapping an Entity Bean to Several Tables with EJBGen" on page D-15.	Yes
column-type	`(OracleClob|OracleBlob)` The type of this column.	No
exclude-from-value-object	`(True|False)` If True, this field will not be generated in the value object.	No
group-names	Comma-delimited names of the groups this field belongs to. Surround the list of group names with double quotes.	No
ordering-number (0..n)	The number where this field must appear in signatures and constructors. For this ordering to work, all cmr and cmp fields must set this attribute to a distinct numeric value.	No
primkey-field	`(True|False)` Whether this field is part of the compound primary key.	No
read-only-in-value-object	`(True|False)` If True, only the getter will be generated for this field in the value object.	No
table-name	The table(s) where this field should be mapped. "Mapping an Entity Bean to Several Tables with EJBGen" on page D-15.	No

@ejbgen:cmr-field

*Where:*Method

Applicable to: Entity bean

Attribute	Description	Required
exclude-from-value-object	(True|False) If True, this field will not be generated in the value object.	No
group-names	Comma-delimited names of the groups this field belongs to. Surround the list of group names with double quotes.	No

Attribute	Description	Required
ordering-number (0..n)	The number where this field must appear in signatures and constructors. For this ordering to work, all cmr and cmp fields must have this attribute to a distinct numeric value.	No
read-only-in-value-object	`(True|False)` If True, only the getter will be generated for this field in the value object.	No

@ejbgen:compatibility

*Where:*Class

Applicable to: All

Attribute	Description	Required
serialize-byte-array-to-oracle-blob	`(True|False)` Whether a cmp-field of type byte[] mapped to a OracleBlob should be serialized.	No

@ejbgen:create-default-rdbms-tables

Where: Class

Applicable to: Entity bean

Attribute	Description	Required
value	(`CreateOnly|Disabled|DropAndCreate|DropAndCreateAlways|AlterOrCreate`)	No

@ejbgen:ejb-client-jar

Where: Class

Applicable to: All bean types

Attribute	Description	Required
file-name	The name of the client JAR to generate. If more than one EJB has this tag, only one of the specified JAR files is included in the deployment descriptor.	Yes

@ejbgen:ejb-interface

Where: Class

Applicable to: All bean types

Attribute	Description	Required
ejb-type	`(javax.ejb.MessageDrivenBean|javax.ejb.EntityBean|javax.ejb.SessionBean)` Specifies the type of this EJB. This tag is not normally used.	Yes

@ejbgen:ejb-local-ref

Where: Class

Applicable to: Session and entity beans

Attribute	Description	Required
home	Local class of the bean.	No
id	The identity of this tag (used for tag inheritance only).	No
jndi-name	The JNDI name of the reference.	No
link	Link to the bean.	No
local	Home class of the bean.	No
name	Name of the reference.	No
type	(Entity | Session)	No

@ejbgen:ejb-ref

Where: Class

Applicable to: All bean types

Attribute	Description	Required
home	Remote class of the bean.	No
id	The identity of this tag (used for tag inheritance only).	No
jndi-name	The JNDI name of the reference.	No
link	Link of the bean.	No
name	Name of the reference.	No
remote	Home class of the bean.	No
type	(`Entity`|`Session`)	No

@ejbgen:entity

Where: Class

Applicable to: Entity beans

Attribute	Description	Required
ejb-name	The name of this entity bean.	Yes
prim-key-class	null	Yes
abstract-schema-name	The abstract schema name for this EJB. If not specified, the `ejb-name` value will be used.	No
cache-between-transactions	(`True`|`False`) Whether to cache the persistent data of an entity bean across (between) transactions.	No
check-exists-on-method	(`True`|`False`) Whether the container checks for the existence of a bean for each method call.	No
concurrency-strategy	(`Optimistic`|`ReadOnly`|`Exclusive`|`Database`) Defines the concurrency strategy for this bean.	No

Attribute	Description	Required
data-source-name	The name of the DataSource (as it was declared in your `config.xml` file).	No
database-type	The type of the database.	
db-is-shared	(`True`|`False`)	No
default-dbms-tables-ddl	Specifies the DDL file name.	No
default-transaction	The transaction attribute to be applied to all methods that do not have a more specific transaction attribute setting.	No
delay-database-insert-until	(`ejbCreate`|`ejbPostCreate`)	No
delay-updates-until-end-of-tx	`(True|False)` Determines whether updates will be sent after the transaction has been committed.	No
dispatch-policy	The JMS dispatch policy queue for this bean	No
enable-batch-operations	`(True|False)` Determines whether to perform batch operations.	No
enable-call-by-reference	(`True`|`False`) Whether the container will call this EJB by reference	No
enable-dynamic-queries	(`True`|`False`) Whether dynamic queries are enabled.	No
finders-load-bean	(`True`|`False`) If this is set to True, the beans will immediately be loaded into the cache by the finder.	No
home-call-router-class-name	Class to be used for routing home method calls.	No
home-is-clusterable	(`True`|`False`) Whether this bean can be deployed from multiple servers in a cluster.	No
home-load-algorithm	(`RoundRobin`|`Random`|`WeightBased`) The algorithm to use for load-balancing between replicas of this home.	No
idle-timeout-seconds	Maximum duration an EJB should stay in the cache.	No
initial-beans-in-free-pool	Specifies the initial number of beans in the free pool.	No
instance-lock-order	`(AccessOrder|ValueOrder)` The locking order for this Entity bean.	No

Attribute	Description	Required
invalidation-target	The `ejb-name` of a read-only entity bean that should be invalidated when this container-managed persistence entity EJB has been modified.	No
lock-order	Sets the database locking order of this bean when a transaction involves multiple beans.	No
max-beans-in-cache	The maximum number of beans in the cache.	No
max-beans-in-free-pool	The maximum number of beans in the free pool.	No
optimistic-column	The column that holds the timestamp for optimistic concurrency	No
persistence-type	(`CMP`|`BMP`) Whether this bean's persistence is container managed or bean-managed.	No
prim-key-class-nogen	(`True`|`False`). If this keyword is specified, EJBGen does not generate the primary key class—it is assumed that you are providing it yourself.	No
read-timeout-seconds	The number of seconds between each `ejbLoad()` call on a read-only entity bean.	No
reentrant	(`True`|`False`)	No
run-as	Specifies the `role-name` for this EJB.	No
run-as-identity-principal	The name of the principal in case the role maps to several principals. **Note:** run-as-identity-principal was deprecated in WebLogic Server 8.1.	No
table-name	The Java class of the primary key. In case of a compound primary key, this class is generated by EJBGen.	No
trans-timeout-seconds	The transaction timeout, in seconds.	No
use-caller-identity	(`True`|`False`) Whether this EJB uses caller's identity.	No
use-select-for-update	(`True`|`False`) Causes `SELECT ... FOR UPDATE` to be used whenever the bean is loaded from the database.	No

Attribute	Description	Required
validate-db-schema-with	(`MetaData`|`TableQuery`) The method used to validate the tables created by the EJB container.	No
verify-columns	(`Read`|`Modified`|`Version`|`Timestamp`) How optimistic concurrency verifies that the columns modified during the transactions have not been modified.	No
verify-rows	(`Read`|`Modified`) How optimistic concurrency verifies that the rows modified during the transactions have not been modified.	No

@ejbgen:entity-cache-ref

Where: Class

Applicable to: Entity beans

Attribute	Description	Required
cache-between-transactions	(`True`|`False`) Whether to cache the persistent data of an entity bean across (between) transactions.	No
concurrency-strategy	(`Optimistic`|`ReadOnly`|`Exclusive`|`Database`) Defines the concurrency strategy for this bean.	No
estimated-bean-size	The estimated average size of the instances of an entity bean in bytes.	No
name	The name of the cache.	No

@ejbgen:env-entry

*Where:*Class

Applicable to: All bean types

Attribute	Description	Required
name	The name of this environment entry.	Yes

Attribute	Description	Required
type	The Java type for this environment entry (must be fully qualified, even if `java.lang`).	Yes
value	The value for this environment entry.	Yes

@ejbgen:file-generation

Where: Class

Applicable to: All bean types

Attribute	Description	Required
local-class	(`True`|`False`) Whether to generate the local interface for this EJB.	No
local-class-name	The name of the local class to be generated.	No
local-home	(`True`|`False`) Whether to generate the local home interface for this EJB.	No
local-home-name	The name of the local home class to be generated.	No
local-home-package	The package for the local home interface.	No
local-package	The package for the local interface.	No
pk-class	(`True`|`False`) Whether to generate the primary key class for this EJB.	No
remote-class	(`True`|`False`) Whether to generate the remote interface for this EJB.	No
remote-class-name	The name of the remote class to be generated.	
remote-home	(`True`|`False`) Whether to generate the remote home interface for this EJB.	No
remote-home-name	The name of the remote home class to be generated.	No

Attribute	Description	Required
remote-home-package	The package for the remote home interface.	No
remote package	The package for the remote interface.	No
value-class	(`True`|`False`) Whether to generate the value class for this EJB.	No
value-class-name	The name of the value class to be generated.	No

@ejbgen:finder

Where: **Class**

Applicable to: Entity beans

Attribute	Description	Required
caching-name	The name of an eager relationship caching.	No
comment	A comment that will be reproduced above the generated finder Java method.	No
ejb-ql	The EJB QL request as it will appear in the deployment descriptor.	No
generate-on	(`Local`|`Remote`) On which home this finder will be generated (if unspecified, both)	No
group-name	Name of the group for the WebLogic query.	No
include-updates	(`True`|`False`) Whether updates made during the current transaction must be reflected in the result of a query.	No
isolation-level	The type of transaction isolation for this method.	No
max-elements	The maximum number of elements that should be returned by a multi-valued query.	No
signature	It must match exactly the signature as you want it generated on the Home class. EJBGen adds the conformant exceptions, but you must make sure that you specify the fully qualified type of each parameter, even if it belongs to `java.lang`.	No

Attribute	Description	Required
sql-select-distinct	(`True`|`False`) Whether the generated `SQL 'SELECT'` contains a '`DISTINCT`' qualifier.	No
transaction-attribute	(`NotSupported`|`Supports`|`Required`|`RequiresNew`|`Mandatory`|`Never`) The transaction attribute for this local method. If not specified, the default transaction attribute will be used. Methods with this tag will are generated on the Local class.	No
weblogic-ejb-ql	The Weblogic EJB Query Language (QL) request as it will appear in the deployment descriptor. Note: if this request is needed, you need to enclose both EJB QL and Weblogic EJB QL within double quotes.	No

@ejbgen:foreign-jms-provider

Where: Class

Applicable to: Message-driven beans

Attribute	Description	Required
connection-factory-jndi-name	The JNDI name for the connection factory.	No
initial-context-factory	The initial JNDI context factory.	No
provider-url	The URL of the foreign JMS provider.	Yes

@ejbgen:jar-settings

Where: Class

Applicable to: All bean types

Attribute	Description	Required
create-tables	Whether to create tables.	No
disable-warning	A comma-separated line of warnings to disable when running `ejbc`.	No

Attribute	Description	Required
ejb-client-jar	Specifies the name of the client JAR to be generated. If multiple EJBs have this tag, only one of the specified JAR files will be included in the deployment descriptor.	No
enable-bean-class-redeploy	(True|False) Determines whether this EJB can be redeployed without redeploying the entire module.	No

@ejbgen:jndi-name

Where: Class

Applicable to: All bean types

Attribute	Description	Required
local	The local JNDI name of this EJB. It not specified, no local interfaces is generated.	No
remote	The remote JNDI name of this EJB. It not specified, no remote interfaces is generated.	No

@ejbgen:local-home-method

Where: Method

Applicable to: Entity and Session beans

Attribute	Description	Required
is-idempotent	`(True|False)` Specifies whether this method is idempotent.	No
isolation-level	`(TransactionSerializable|TransactionReadCommitted|TransactionReadUncommitted|TransactionRepeatableRead)` Specifies the type of transaction isolation for this method.	No
ordering-number	`(0..n)` Specifies the number where this method must appear in the generated class. No	

Attribute	Description	Required
roles	Comma-delimited list of roles that are allowed to invoke this method.	No
transaction-attribute	The transaction attribute for this local method. If not specified, the default transaction attribute will be used. Methods with this tag are generated on the Local class.	No

@ejbgen:local-method

Where: Method

Applicable to: Entity and Session beans

Attribute	Description	Required
is-idempotent	(`True`|`False`) Whether this method is idempotent.	No
isolation-level	The type of transaction isolation for this method.	No
ordering-number	(`0..n`) The number where this method must appear in the generated class.	No
roles	Comma-delimited list of roles that are allowed to invoke this method.	No
transaction-attribute	(`NotSupported`|`Supports`|`Required`|`RequiresNew`|`Mandatory`|`Never`) The transaction attribute for this local method. If not specified, the default transaction attribute will be used. Methods with this tag will be generated on the Local class.	No

@ejbgen:message-driven

Where: Class

Applicable to: Message-Driven beans

Attribute	Description	Required
acknowledge-mode	(`auto-acknowledge`|`dups-ok-acknowledge`) The acknowledgement mode.	No

Attribute	Description	Required
clients-on-same-server	(`True`|`False`) Specifies whether the clients are collocated with the EJB on the same server.	No
create-as-principal-name	Names the principal to be used when `ejbCreate` would otherwise run with an anonymous principal.	No
default-transaction	The transaction attribute to be applied to all methods that do not have a more specific transaction attribute setting.	No
destination-jndi-name	The JNDI name of the destination.	Yes
destination-type	(`javax.jms.Queue`|`javax.jms.Topic`)The JMS destination type.	Yes
dispatch-policy	Specifies the dispatch policy queue for this bean.	No
durable	(`True`|`False`) If the destination-type is `Topic`, setting this attribute to `True` will make the subscription durable.	No
ejb-name	The name of this message-driven bean.	Yes
enable-call-by-reference	(`True`|`False`) Whether the container will call this EJB by reference	No
initial-beans-in-free-pool	The initial number of beans in the free pool.	No
jms-client-id	The client id of this EJB.	No
jms-polling-interval-seconds	The number of seconds between each attempt to reconnect to the JMS destination.	No
max-beans-in-free-pool	The maximum number of beans in the free pool.	No
message-selector	The JMS message selector.	No
passivate-as-principal-name	Names the principal to be used when `ejbPassivate` would otherwise run with an anonymous principal.	No
remove-as-principal-name	Names the principal to be used when `ejbRemove` would otherwise run with an anonymous principal.	No
run-as	Specifies the role-name for this EJB.	No

Attribute	Description	Required
run-as-identity-principal	The name of the principal in case the role maps to several principals. **Note:** This attribute was deprecated in WebLogic Server 8.1	No
transaction-type	`(Bean|Container)` Specifies where the transactions for this EJB are managed.	No
trans-timeout-seconds	The transaction timeout (in seconds).	No
use-caller-identity	(`True`|`False`) Whether this EJB uses caller's identity.	No

@ejbgen:method-isolation-level-pattern

Where: Class

Applicable to: All beans

Attribute	Description	Required
id	The identity of this tag (used for tag inheritance only).	No
isolation-level	The isolation level for the methods specified in the pattern tag.	Yes
pattern	The pattern that matches all methods that will receive this isolation level (for example, "*").	Yes

@ejbgen:method-permission-pattern

Where: Class

Applicable to: All beans

Attribute	Description	Required
id	The identity of this tag (used for tag inheritance only).	No
interface	(`Home`|`Remote`|`LocalHome`|`Local`) The interface where this permission pattern applies.	No

Attribute	Description	Required
itf	(Home|Remote|LocalHome|Local) Specifies the interface to which this permission pattern applies.	No
pattern	The pattern that matches all methods that will receive this isolation level (for example, "*").	Yes
roles	Comma-delimited list of roles for the methods specified in the pattern tag.	Yes

@ejbgen:relation

Where: Class

Applicable to: Entity beans

Attribute	Description	Required
cascade-delete	(`True`|`False`)	No
cmr-field	The cmr field where this relationship will be kept. This field is optional. If it not present, the relationship is unidirectional. If it is present, the attribute `fk-column` must be specified as well.	No
db-cascade-delete	(`True`|`False`) Whether a cascade delete will use the built-in cascade delete facilities of the underlying DBMS.	No
fk-column	Only needed in a relationship having at least one One side. In that case, the non-One side EJB must declare a column that it will use to store the primary key of its counterpart.	No
foreign-key-table	The name of a DBMS table that contains a `foreign-key`.	No
join-table	Only needed in a many-to-many relationship. It must be the name of an existing table that will be used to hold the joint table containing the relationships. In case you are using a compound primary key, you need to specify a set of corresponding foreign keys separated by a comma.	No
multiplicity	(`One`|`Many`)	Yes
name	The name of the relationship. Make sure you use the same name on both ends of a relationship for the roles to be generated properly (note that this constraint applies to unidirectional as well).	Yes

Attribute	Description	Required
primary-key-table	The name of a DBMS table that contains a `primary-key`.	No
role-name	The name of this role (such as `ParentHasChildren`). If no role name is given, EJBGen will generate one for you. Note that you have to specify a `role-name` if you are going to inherit relations.	No
target-ejb	The EJB name of the target of this relationship.	Yes

@ejbgen:relationship-caching-element

Where: Method

Applicable to: Entity beans

Attribute	Description	Required
caching-name	The name of an eager relationship caching.	Yes
cmr-field	A comma-delimited list of CMR field names.	Yes
group-name	The name of the group to be loaded for the CMR field.	No
id	An id that allows a child to use this element as a parent. See "Specifying Relationship Caching with EJBGen Tags" on page D-15.	No
parent-id	The parent id of this element.	No

@ejbgen:remote-home-method

Where: Method

Applicable to: Entity and Session beans

Attribute	Description	Required
is-idempotent	(`True`|`False`) Specifies whether this method is idempotent. No	No

Attribute	Description	Required
isolation-level	`(TransactionSerializable|TransactionReadCommitted|TransactionReadUncommitted|TransactionRepeatableRead)` Specifies the type of transaction isolation for this method.	No
ordering-number	(0..n) Specifies the number where this method must appear in the generated class	No
roles	Comma-delimited list of roles that are allowed to invoke this method.	No
transaction-attribute	The transaction attribute for this remote method. If not specified, the default transaction attribute will be used. Methods with this tag will be generated on the Remote class.	No

@ejbgen:remote-method

Where: Method

Applicable to: Entity and Session beans

Attribute	Description	Required
is-idempotent	(True|False) Whether this method is idempotent.	No
isolation-level	The type of transaction isolation for this method.	No
ordering-number	(0..n) The number where this method must appear in the generated class.	No
roles	Comma-delimited list of roles that are allowed to invoke this method.	No
transaction-attribute	(NotSupported|Supports|Required|RequiresNew|Mandatory|Never) The transaction attribute for this local method. If not specified, the default transaction attribute will be used. Methods with this tag will be generated on the Local class.	No

@ejbgen:resource-env-ref

Where: Class

Applicable to: All bean types

Attribute	Description	Required
id	The identity of this tag (used for tag inheritance only).	No
jndi-name	JNDI name of the resource.	No
name	Name of the resource environment reference.	Yes
type	Type of the environment resource references (e.g. javax.jms.Queue).	Yes

@ejbgen:resource-ref

Where: Class

Applicable to: All bean types

Attribute	Description	Required
auth	`(Application|Container)`	Yes
id	The identity of this tag (used for tag inheritance only).	No
jndi-name	JNDI name of the resource.	Yes
name	Name of the resource.	Yes
sharing-scope	(Shareable|Unshareable)	No
type	Type of the resource (e.g. javax.sql.DataSource).	Yes

@ejbgen:role-mapping

Where: Class

Applicable to: All bean types

Note:

Attribute	Description	Required
externally-defined	`(True|False)` True if this role is defined externally.	No

Attribute	Description	Required
global-role	(True|False) True if this role is global. **Note:** global-role was deprecated in WebLogic Server 7.0	No
id	The identity of this tag (used for tag inheritance only).	No
principals	The names of the principals in this role (separated by commas).	No
role-name	The name of the role.	Yes

@ejbgen:security-role-ref

Where: Method

Applicable to: Entity and session beans

Attribute	Description	Required
id	The identity of this tag (used for tag inheritance only).	No
role-link	A reference to a defined security role.	No
role-name	The name of the security role.	Yes

@ejbgen:select

Where: Method

Applicable to: Entity beans

Attribute	Description	Required
caching-name	The name of an eager relationship caching.	No
ejb-ql	The EJB-QL defining this select method. The method name must start with `ejbSelect`.	Yes
group-name	Name of the group for the query.	No
include-updates	(`True`|`False`) Whether updates made during the current transaction must be reflected in the result of a query.	No

Attribute	Description	Required
max-elements	The maximum number of elements that should be returned by a multi-valued query.	No
ordering-number	(`0..n`) The number where this method must appear in the generated class.	No
result-type-mapping	(`Remote`|`Local`) Whether the returned objects are mapped to `EJBLocalObject` or `EJBObject`.	No
sql-select-distinct	(`True`|`False`) Determines whether the generated `SQL SELECT` will contain a `DISTINCT` qualifier.	No
weblogic-ejb-ql	The Weblogic EJB QL request as it will appear in the deployment descriptor. **Note:** If this request is needed, you need to enclose both EJB QL and Weblogic EJB QL within double quotes.	No

@ejbgen:session

Where: Class

Applicable to: Session beans

Attribute	Description	Required
allow-concurrent-calls	(`True`|`False`) Whether to allow concurrent calls on that EJB.	No
allow-remove-during-transaction	(`True`|`False`) Specifies whether `remove()` can be invoked during a transaction.	No
bean-load-algorithm	The algorithm to use for load-balancing between replicas of this bean.	No
cache-type	(`NRU`|`LRU`) Specifies the type of the cache for this stateful Session bean.	No
call-router-class-name	Class name to be used for routing home method calls.	No
clients-on-same-server	(`True`|`False`) Specifies whether the clients are collocated with the EJB on the same server.	No

Attribute	Description	Required
create-as-principal-name	Names the principal to be used when `ejbCreate` would otherwise run with an anonymous principal.	No
default-transaction	The transaction attribute to be applied to all methods that do not have a more specific transaction attribute setting.	No
dispatch-policy	The JMS dispatch policy queue for this bean	No
ejb-name	The name of this session bean.	Yes
enable-call-by-reference	(`True`|`False`) Whether the container will call this EJB by reference	No
home-call-router-class-name	Class to be used for routing home method calls.	No
home-is-clusterable	(`True`|`False`) Whether this bean can be deployed from multiple servers in a cluster.	No
home-load-algorithm	(`RoundRobin`|`Random`|`WeightBased`) The algorithm to use for load-balancing between replicas of this home.	No
idle-timeout-seconds	Maximum duration an EJB should stay in the cache.	No
initial-beans-in-free-pool	The initial number of beans in the free pool.	No
is-clusterable	(`True`|`False`) Whether this bean is clusterable	No
load-algorithm	(`RoundRobin`|`Random`|`WeightBased`) The name of the algorithm used to balance replicas of this home	No
max-beans-in-cache	The maximum number of beans in the cache.	No
max-beans-in-free-pool	The maximum number of beans in the free pool.	No
methods-are-idempotent	(`True`|`False`) Whether the methods for this stateless session bean are idempotent or not.	No
passivate-as-principal-name	Names the principal to be used when `ejbPassivate` would otherwise run with an anonymous principal.	No
persistent-store-dir	The directory in which to store the passivated beans.	No
remove-as-principal-name	Names the principal to be used when ejbRemove would otherwise run with an anonymous principal.	No

Attribute	Description	Required
replication-type	(`InMemory`|`None`) How to replicate stateful session beans in a cluster.	No
run-as	Specifies the `role-name` for this EJB.	No
run-as-identity-principal	The name of the principal in case the role maps to several principals. **Note:** Deprecated in WebLogic Server 8.1.	No
transaction-type	(`Bean`|`Container`) Whether transactions for this EJB are bean- managed or container-managed.	No
trans-timeout-seconds	The transaction timeout, in seconds.	No
type	(`Stateless` | `Stateful`) The type of the session bean. If this attribute is not specified, EJBGen guesses the right type by looking at the `ejbCreate()` methods on your class.	No
use-caller-identity	(`True` | `False`) Whether this EJB uses caller's identity.	No

@ejbgen:value-object

Where: Class

Applicable to: All bean types

Attribute	Description	Required
reference	(`Local`|`Value`) Specifies which objects the value object class should reference when accessing other EJB's.	Yes

APPENDIX E

EJB Query Language (EJB-QL) and WebLogic Server

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. You can create queries in the deployment descriptor for any finder method other than `findByPrimaryKey()`. `findByPrimaryKey` is automatically handled by the EJB container.

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 container-managed persistence. For more information on WLQL and EJB 1.1 container-managed persistence, see "Using WebLogic Query Language (WLQL) for EJB 1.1 CMP" on page F-3.

Using the 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-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-jar.xml` file. However, the value of the `weblogic-cmp-jar.xml` query element overrides the value of the `ejb-jar.xml` query element.

These topics provide guidelines for using the WebLogic QL extension to EJB 2.0 QL:

- "upper and lower Functions" on page E-2
- "Using ORDERBY" on page E-2
- "Using Subqueries" on page E-3
- "Using Aggregate Functions" on page E-9
- "Using Queries that Return ResultSets" on page E-10
- "Using Oracle SELECT HINTS" on page E-12
- "Visual Indicator of Error in Query" on page E-15
- "Multiple Errors Reported after a Single Compilation" on page E-15

upper and lower Functions

The EJB WebLogic QL `upper` and `lower` extensions convert the case of arguments to allow finder methods to return results that match the characters in an search expression but not the case. The case change is transient, for the purpose of string matching, and is not persisted in database. The underlying database must also support `upper` and `lower` functions.

upper

The `upper` function converts characters in its arguments from any case to upper case before string matching is performed. Use the `upper` function with an upper-case expression in a query to return all items that match the expression, regardless of case. For example:

```
select name from products where upper(name)='DETERGENT';
```

lower

The `lower` function converts characters in its arguments from any case to lower case before string matching is performed. Use the `lower` function with an lower-case expression in a query to return all items that match the expression, regardless of case.

```
select type from products where lower(name)='domestic';
```

Using 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.

Listing 8-13 ORDERBY Extension Showing Order by id

```
ORDERBY

       SELECT OBJECT(A) from A for Account.Bean

              ORDERBY A.id
```

You can specify an `ORDERBY` with ascending [`ASC`] or descending [`DESC`] order for multiple fields as follows:.

Listing 8-14 ORDERBY Extension Showing Order by id with ASC and DESC

```
ORDERBY <field> [ASC|DESC], <field> [ASC|DESC]

       SELECT OBJECT(A) from A for Account.Bean, OBJECT(B) from B for
Account.Bean

              ORDERBY A.id ASC; B.salary DESC
```

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 Subqueries

WebLogic Server supports the use of the following features with subqueries in EJB QL:

- Subquery return type
  - Single `cmp-fields`
  - Aggregate functions
  - Beans with simple primary keys
- Subqueries as comparison operands
- Correlated subqueries
- Uncorrelated subqueries
- `DISTINCT` clauses with subqueries

The relationship between WebLogic QL and subqueries is similar to the relationship between SQL queries and subqueries. Use WebLogic QL subqueries in the `WHERE` clause of an outer

WebLogic QL query. With a few exceptions, the syntax for a subquery is the same as a WebLogic QL query.

To specify WebLogic QL, see "Using the EJB 2.0 WebLogic QL Extension for EJB QL" on page E-1. Use those instructions with a `SELECT` statement that specifies a subquery as shown the following sample.

The following query selects all above average students as determined by the provided grade number:

```
SELECT OBJECT(s) FROM studentBean AS s WHERE s.grade > (SELECT AVG(s2.grade)
FROM StudentBean AS s2)
```

Note that in the above query the subquery, `(SELECT AVG(s2.grade) FROM StudentBean AS s2)`, has the same syntax as an EJB QL query.

You can create nested subqueries.The depth is limited by the underlying database's nesting capabilities.

In a WebLogic QL query, the identifiers declared in the `FROM` clauses of the main query and all of its subqueries must be unique. This means that a subquery may not re-declare a previously declared identifier for local use within that subquery.

For example, the following example is not legal because employee bean is being declared as emp in both the query and the subquery:

```
SELECT OBJECT(emp)
       FROM EmployeeBean As emp
              WHERE emp.salary=(SELECT MAX(emp.salary) FROM
                     EmployeeBean AS emp WHERE employee.state=MA)
```

Instead, this query should be written as follows:

```
SELECT OBJECT(emp)
       FROM EmployeeBean As emp
              WHERE emp.salary=(SELECT MAX(emp2.salary) FROM
                     EmployeeBean AS emp2 WHERE emp2.state=MA)
```

The above examples correctly declares the subquery's employee bean to have a different identifier from the main query's employee bean.

Subquery Return Types

The return type of a WebLogic QL subquery can be one of a number of different types, such as:

Single cmp-field Type Subqueries

WebLogic Server supports a return type consisting of a `cmp-field`. The results returned by the subquery may consists of a single value or collection of values. An example of a subquery that returns value(s) of the type `cmp-field` is as follows:

```
SELECT emp.salary FROM EmployeeBean AS emp WHERE emp.dept = 'finance'
```

This subquery selects all of the salaries of employees in the finance department.

Aggregate Functions

WebLogic Server supports a return type consisting of an aggregate of a `cmp-field`. As an aggregate always consist of a single value, the value returned by the aggregate is always a single value. An example of a subquery that return a value of the type aggregate (`MAX`) of a `cmp-field` is as follows:

```
SELECT MAX(emp.salary) FROM EmployeeBean AS emp WHERE emp.state=MA
```

This subquery selects the single highest employee salary in Massachusetts.

For more information on aggregate functions, see "Using Aggregate Functions" on page E-9.

Beans with Simple Primary Key

WebLogic Server supports a return type consisting of a `cmp-bean` with a simple primary key.

The following example illustrates a subquery that returns the value(s) of the type bean with a simple primary key:

```
SELECT OBJECT(emp) FROM EMployeeBean As emp WHERE
emp.department.budget>1,000,000
```

This subquery provides a list of all employee in departments with budgets greater than $1,000,000.

Note: Beans with compound primary keys are NOT supported. Attempts to designate the return type of a subquery to a bean with a compound primary key will fail when you compile the query.

Subqueries as Comparison Operands

Use subqueries as the operands of comparison operators and arithmetic operators. WebLogic QL supports subqueries as the operands of:

- these comparison operators: `[NOT]IN, [NOT]EXISTS`

and

- these arithmetic operators: `<, >, <=, >=, =, <> with ANY and ALL`

[NOT]IN

The [NOT]IN comparison operator tests whether the left-had operand is or is not a member of the subquery operand on the right-hand side.

An example of a subquery which is the right-hand operand of the NOT IN operator is as follows:

```
SELECT OBJECT(item)
       FROM ItemBean AS item
              WHERE item.itemId NOT IN
                     (SELECT oItem2.item.itemID
                                   FROM OrderBean AS orders2,
IN(orders2.orderItems)oIttem2
```

The subquery selects all items from all orders.

The main query's NOT IN operator selects all the items that are not in the set returned by the subquery. So the end result is that the main query selects all unordered items.

[NOT]EXISTS

The [NOT]EXISTS comparison operator tests whether the set returned by the subquery operand is or is not empty.

An example of a subquery which is the operand of the NOT EXISTS operand is as follows:

```
SELECT (cust) FROM CustomerBean AS cust
       WHERE NOT EXISTS
              (SELECT order.cust_num FROM OrderBean AS order
                     WHERE cust.num=order_num)
```

This is an example of a query with a correlated subquery. See "Correlated and Uncorrelated Subqueries" on page E-7 for more information. the following query returns all customers that have not placed an order.

```
SELECT (cust) FROM CustomerBean AS cust
       WHERE cust.num NOT IN
              (SELECT order.cust_um FROM OrderBean AS order
                     WHERE cust.num=order_num)
```

Arithmetic Operators

Use arithmetic operators for comparison when the right-hand subquery operand returns a single value. If the right hand subquery instead returns multiple values, then the qualifiers ANY or ALL must precede the subquery.

An example of a subquery which uses the '=' operator is as follows:

```
SELECT OBJECT (order)
       FROM OrderBean AS order, IN(order.orderItems)oItem
              WHERE oItem.quantityOrdered =
                     (SELECT MAX (subOItem.quantityOrdered)
                            FROM Order ItemBean AS subOItem
                                   WHERE subOItem,item itemID = ?1)
AND oItem.item.itemId = ?1
```

For a given `itemId`, the subquery returns the maximum quantity ordered of that item. Note that this aggregate returned by the subquery is a single value as required by the '=' operator.

For the same given `itemId`, the main query's `'='` comparison operator checks which order's `OrderItem.quantityOrdered` equals the maximum quantity returned by the subquery. The end result is that the query returns the OrderBean that contains the maximum quantity of a given item that has been ordered.

Use arithmetic operators in conjunction with `ANY` or `ALL`, when the right-hand subquery operand may return multiple values.

An example of a subquery which uses `ANY` and `ALL` is as follows:

```
SELECT OBJECT (order)
       FROM OrderBean AS order, IN(order.orderItems)oItem
              WHERE oItem.quantityOrdered > ALL
                     (SELECT subOItem.quantityOrdered
                            FROM OrderBean AS suborder IN
(subOrder.orderItems)subOItem
                                   WHERE subOrder,orderId = ?1)
```

For a given `orderId`, the subquery returns the set of `orderItem.quantityOrdered` of each item ordered for that `orderId`. The main query's '>' `ALL` operator looks for all orders whose `orderItem.quantityOrdered` exceeds all values in the set returned by the subquery. The end result is that the main query returns all orders in which all `orderItem.quantityOrdered` exceeds every `orderItem.quantityOrdered` of the input order.

Note that since the subquery can return multi-valued results that they '>'`ALL` operator is used rather then the `'>'` operator.

Correlated and Uncorrelated Subqueries

WebLogic Server supports both correlated and Uncorrelated subqueries.

UnCorrelated Subqueries

Uncorrelated subqueries may be evaluated independently of the outer query. An example of an uncorrelated subquery is as follows:

```
SELECT OBJECT(emp) FROM EmployeeBean AS emp
       WHERE emp.salary>
       (SELECT AVG(emp2.salary) FROM EmployeeBean AS emp2)
```

This example of a uncorrelated subquery selects the employees whose salaries are above average. This examples uses the '>' arithmetic operator.

Correlated

Correlated subqueries are subqueries in which values from the outer query are involved in the evaluation of the subquery. An example of a correlated subquery is as follows:

```
SELECT OBJECT (mainOrder) FROM OrderBean AS mainOrder
       WHERE 10>
              (SELECT COUNT (DISTINCT subOrder.ship_date)
       FROM OrderBean AS subOrder
       WHERE subOrder.ship_date>mainOrder.ship_date
       AND mainOrder.ship_date IS NOT NULL
```

This example of a correlated subquery selects the last 10 shipped orders. This example uses the `NOT IN` operator.

Note: Keep in mind that correlated subqueries can involve more processing overhead the uncorrelated subqueries.

DISTINCT Clause with Subqueries

Use the `DISTINCT` clause in a subquery to enable an SQL `SELECT DISTINCT` in the subquery's generated SQL. Using a `DISTINCT` clause in a subquery is different from using one in a main query because the EJB container enforces the `DISTINCT` clause in a main query; whereas the `DISTINCT` clause in the subquery is enforced by the generated SQL `SELECT DISTINCT`. The following is an example of a `DISTINCT` clause in a subquery:

```
SELECT OBJECT (mainOrder) FROM OrderBean AS mainOrder
       WHERE 10>
              (SELECT COUNT (DISTINCT subOrder.ship_date)
       FROM OrderBean AS subOrder
       WHERE subOrder.ship_date>mainOrder.ship_date
       AND mainOrder.ship_date IS NOT NULL
```

Using Aggregate Functions

WebLogic Server supports aggregate functions with WebLogic QL. You only use these functions as `SELECT` clause targets, not as other parts of a query, such as a `WHERE` clause. The aggregate functions behave like SQL functions. They are evaluated over the range of the beans returned by the `WHERE` conditions of the query

To specify WebLogic QL, see "Using the EJB 2.0 WebLogic QL Extension for EJB QL" on page E-1. Use those instructions with a SELECT statement that specifies an aggregate function as shown in the samples shown in the following table.

A list of the supported functions and sample statements follow:

Aggregate Function	Description	Sample Statement
MIN(x)	Returns the minimum value of this field.	SELECT MIN(t.price) FROM TireBean AS t WHERE t.size=?1 This statement selects the lowest price for a tire of a given input size.
MAX(x)	Returns the maximum value of this field.	SELECT MAX(s.customer_count) FROM SalesRepBean AS s WHERE s.city='Los Angeles' This statement selects the maximum number of customers served by any single sales representative in Los Angeles.
AVG([DISTINCT] x)	Returns the average value of this field	SELECT AVG(b.price) FROM BookBean AS b WHERE b.category='computer_science' This statement selects the Average Price of a book in the Computer Science category.

Aggregate Function	Description	Sample Statement
SUM([DISTINCT] x)	Returns the sum of this field.	SELECT SUM(s.customer_count) FROM SalesRepBean AS s WHERE s.city='Los Angeles' This statement retrieves the total number of customers served by sales representatives in Los Angeles.
COUNT([DISTINCT] x)	Returns the number of occurrences of a field.	SELECT COUNT(s.deal.amount) FROM SalesRepBean AS s, IN(deal)s WHERE s.deal.status='closed' AND s.deal.amount>=1000000 This statement retrieves the number of closed deals for at lease 1 million dollars.

You can return aggregate functions in ResultSets as described below.

Using Queries that Return ResultSets

WebLogic Server supports `ejbSelect()` queries that return the results of multi-column queries in the form of a `java.sql.ResultSet`. To support this feature, WebLogic Server now allows you to use the SELECT clause to specify a comma delimited list of target fields as shown in the following query:

```
SELECT emmp.name, emp.zip FROM EmployeeBean AS emp
```

This query returns a `java.sqlResultSet` with rows whose columns are the values Employee's Name and Employee's Zip.

To specify WebLogic QL, see "Using the EJB 2.0 WebLogic QL Extension for EJB QL" on page E-1. Use those instructions with a query specifying a ResultSet as shown in the above query to specify WebLogic QL, see "Using the EJB 2.0 WebLogic QL Extension for EJB QL" on page E-1. Use those instructions with a SELECT statement that specifies an aggregate query like the samples shown in the following table.

.

ResultSets created in EJB QL can only return `cmp-field` values or aggregates of `cmp-field` values, they cannot return beans.

In addition, you can create powerful queries, as described in the following example, when you combine `cmp-fields` and aggregate functions.

The following rows (beans) show the salaries of employees in different locations:

Table E-1 CMP Fields Showing Salaries of Employees in California

Name	Location	Salary
Matt	CA	110,000
Rob	CA	100,000

Table E-2 CMP Fields Showing Salaries of Employees in Arizona

Name	Location	Salary
Dan	AZ	120,000
Dave	AZ	80,000

Table E-3 CMP Fields Showing Salaries of Employees in Texas

Name	Location	Salary
Curly	TX	70,000
Larry	TX	180,000
Moe	TX	80,00

Note: Each row represents a bean.

The following SELECT statement shows a query that uses ResultSets and the aggregate function (AVG) along with a GROUP BY statement and an ORDER BY statement using a descending sort to retrieve results from a multi-column query.

```
SELECT e.location, AVG(e.salary)
       FROM Finder EmployeeBean AS e
              GROUP BY e.location
                     ORDER BY 2 DESC
```

The query shows the average salary in of employees at each location in descending order. The number, 2 means that the `ORDERBY` sort is on the second item in the `SELECT` statement. The `GROUP BY` clause specifies the `AVERAGE` salary of employees with a matching `e.location` attribute.

The ResultSet, in descending order is as follows:

Location	Average
TX	110,000
AZ	100,000
CA	105,000

Note: You can only use integers as `ORDERBY` arguments in queries that return ResultSets. WebLogic Server does not support the use of integers as `ORDERBY` arguments in any Finder or `ejbselect()` that returns beans.

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 in `weblogic-cmp-jar.xml`. 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 ....
```

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`.

Properties-Based Methods of the Query Interface

The `Query` interface contains both find and execute methods. The find methods work like standard EJB methods, in that they return EJBObjects. The execute methods work more like `Select` statements in that you can select individual fields.

The `Query` interface return type is a disconnected `ResultSet`, meaning you access the information from the returned object the same way you would access it from a `ResultSet`, except that the `ResultSet` does not hold open a database connection.

The `Query` interface’s properties-based methods offer an alternate way of specifying settings particular to a query. The `QueryProperties` interface holds standard EJB query settings while the `WLQueryProperties` interface holds WebLogic-specific query settings.

Although the `Query` interface extends `QueryProperties`, the actual `Query` implementation extends `WLQueryProperties` so it can be safely casted, as in the example in Listing 8-15, which sets field group settings:

Listing 8-15 Setting Field Group Settings with WLQueryProperties

```
Query query=qh.createQuery(); ((WLQueryProperties)
query).setFieldGroupName("myGroup"); Collection results=query.find(ejbql);
```

or

```
Query query=qh.createQuery(); Properties props = new Properties();
props.setProperty(WLQueryProperties.GROUP_NAME, "myGroup"); Collection
results=query.find(ejbql, props);
```

Migrating from WLQL to EJB QL

If you have an existing application that uses EJB 1.1, 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
`(< 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

Known Issue with Implied Cross Products

When an EJB QL query contains an implied cross product—as opposed to an explicit one—the EJB-QL query can return an empty result.

Consider this example query:

```
SELECT OBJECT(e) FROM EmployeeBean AS e WHERE e.name LIKE 'Joe' OR
e.acct.balance < 100
```

This query references `AccountEJB`, but `AccountEJB` is not listed in the `FROM` clause. The result of this query is identical to that of query with `AccountEJB` explicitly listed in the `FROM` clause.

EJB QL Error-Reporting

Compiler error messages in EJB QL provide a visual aid to identify which part of the query is in error and allow the reporting of more than one error per compilation.

Visual Indicator of Error in Query

When an error is reported, EJB QL indicates the location of the problem within these symbols: =>> <<=. These symbols are highlighted in red in the following sample compiler error report.

```
ERROR: Error from appc: Error while reading
'META-INF/FinderEmployeeBeanRDBMS.xml'. The error was:

Query:

EJB Name: FinderEmployeeEJB

Method Name: findThreeLowestSalaryEmployees

Parameter Types: (java.lang.String)

Input EJB Query: SELECT OBJECT(e) FROM FinderEmployeeBean e WHERE f.badField
= '2' O

R (e.testId = ?1) ORDERBY e.salary

SELECT OBJECT(e ) FROM FinderEmployeeBean e
WHERE =>> f.badField <<= = '2' OR ( e.testId = ?1 ) ORDERBY e.salary

Invalid Identifier in EJB QL expression:

Problem, the path expression/Identifier 'f.badField' starts with an
identifier: 'f'. The identifier 'f', which can be either a range variable
identifier or a collection member identifier, is required to be declared in
the FROM clause of its query or in the FROM clause of a parent query.

'f' is not defined in the FROM clause of either its query or in any parent
query.

Action, rewrite the query paying attention to the usage of 'f.badField'.
```

Multiple Errors Reported after a Single Compilation

If a query contains multiple errors, EJB QL is now capable of reporting more than one of these after a single compilation. Previously, the compiler could only report one error per compilation. Reporting of subsequent errors required recompilation.

The compiler is not guaranteed to report all errors after a single compilation.

APPENDIX F

Important Information for EJB 1.1 Users

BEA strongly recommends that new users implement their distributed business applications using EJB 2.0 beans. However, if your existing application implements EJB 1.1 beans, please read the following sections, which contain important design and implementation information specific to EJB 1.1. This section includes a detailed reference to EJB 1.1 deployment descriptors.

- "Writing for RDBMS Persistence for EJB 1.1 CMP" on page F-1
- "Using WebLogic Query Language (WLQL) for EJB 1.1 CMP" on page F-3
- "Using SQL for CMP 1.1 Finder Queries" on page F-7
- "Tuned EJB 1.1 CMP Updates in WebLogic Server" on page F-7
- "Using is-modified-method-name to Limit Calls to ejbStore()" on page F-8
- "5.1 weblogic-ejb-jar.xml Deployment Descriptor File Structure" on page F-9
- "5.1 weblogic-ejb-jar.xml Deployment Descriptor Elements" on page F-9
- "1.1 weblogic-cmp-jar.xml Deployment Descriptor File Structure" on page F-20
- "1.1 weblogic-cmp-jar.xml Deployment Descriptor Elements" on page F-22

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. You use EJB QL, a portable query language, to define finder queries for 2.0 EJBs with container-managed persistence.

WebLogic Server provides an easy way to write finders.

1. Write the method signature of a finder in the `EJBHome` interface.
2. Define the finder's query expressions in the `ejb-jar.xml` deployment file.

`appc` 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-jar.xml`.

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

Finder Signature

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

Note: You 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, `appc` generates an error message when you compile the deployment unit.

finder-query Element

The `finder-query` element defines the WebLogic Query Language (WLQL) expression you use 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 F-3 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-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.

WLQL Syntax

WLQL strings use the prefix notation for comparison operators, as follows:

```
(operator operand1 operand2)
```

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

WLQL 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`* or an input parameter	`(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')`

WLQL Operands

Valid WLQL operands include:

- Another WLQL expression
- A container-managed field defined elsewhere in the `weblogic-cmp-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-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-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 SQL for CMP 1.1 Finder Queries

WebLogic Server allows you to use a SQL string instead of the standard WLQL query language to write SQL for a CMP 1.1 finder query. The SQL statement retrieves the values from the database for the CMP 1.1 finder query. Use SQL to write a CMP 1.1 finder query when a more complicated finder query is required and you cannot use WLQL.

For more information on WLQL, see "Using WebLogic Query Language (WLQL) for EJB 1.1 CMP" on page F-3.

To specify this SQL finder query:

1. In the `weblogic-cmp-jar.xml` file write a SQL query using the `finder-sql` element in the `weblogic-cmp-jar.xml` file as follows.

   `findBigAccounts(double cutoff)` as follows:

   ```
   <finder-sql><![CDATA{balance >$0]]></finder-sql>
   ```

   Use values like $0, or $1 in the SQL string to reference the parameters to the finder method. The WebLogic Server EJB container replaces the `$` parameters but will not interpret the SQL query.

2. The Container emits the following SQL:

   ```
   SELECT <columns> FROM table WHERE balance > ?
   ```

   The SQL should be the WHERE clause of an SQL statement. The Container prepends the SELECT and FROM clauses. The WHERE clause may contain arbitrary SQL.

If you use characters in your SQL query that may confuse an XML parser, such as the.greater than (>) symbol and the less than (<) symbol, make sure that you declare the SQL query using the CDATA format shown in the preceding sample SQL statement.

You can use any amount of vendor-specific SQL in the SQL query.

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 BEA Systems recommends 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-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.

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

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. WebLogic Server CMP implementation automatically detects modifications of CMP fields and writes only those changes to the underlying datastore. BEA Systems recommends that you 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, and results in a DBMS update. 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 A-43.

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.

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.

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`.

`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`.

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.

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 `TransactionReadCommittedUncommitted`, 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.

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 F-13 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-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-jar.xml</type-storage>
</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.

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.extensions.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

This property is similar to `home-is-clusterable`, but it is applicable only to stateless session EJBs.

stateless-bean-load-algorithm

This property is similar to `home-load-algorithm`, but it is applicable only to stateless session EJBs.

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

The `transaction-isolation` element defines method-level transaction isolation settings for an EJB. Allowable values include:

- `TRANSACTION_SERIALIZABLE`—Simultaneously executing this transaction multiple times has the same effect as executing the transaction multiple times in a serial fashion.
- `TRANSACTION_READ_COMMITTED`—The transaction can view only committed updates from other transactions
- `TRANSACTION_READ_UNCOMMITTED`—The transaction can view uncommitted 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.

These additional values are supported only for Oracle databases, and only for container-managed persistence (CMP) EJBs:

- TRANSACTION_READ_COMMITTED_FOR_UPDATE—Supported only for Oracle databases, and only for container-managed persistence (CMP) EJBs. This value sets the isolation level to TRANSACTION_READ_COMMITTED, and for the duration of the transaction, all SQL SELECT statements executed in any method are executed with FOR UPDATE appended to them. This causes the selected rows to be locked for update. If Oracle cannot lock the rows affected by the query immediately, then it waits until the rows are free. This condition remains in effect until the transaction does a COMMIT or ROLLBACK.

  This isolation level can be used to avoid the error:

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

  which can occur when using the TRANSACTION_SERIALIZABLE isolation level with Oracle databases.

- TRANSACTION_READ_COMMITTED_FOR_UPDATE_NO_WAIT—Supported only for Oracle databases, and only for container-managed persistence (CMP) EJBs.

  This value sets the isolation level to TRANSACTION_READ_COMMITTED, and for the duration of the transaction, all SQL SELECT statements executed in any method are executed with FOR UPDATE NO WAIT appended to them. This causes the selected rows to be locked for update.

  In contrast to the TRANSACTION_READ_COMMITTED_FOR_UPDATE setting, TRANSACTION_READ_COMMITTED_FOR_UPDATE_NO_WAIT causes the Oracle DBMS to NOT WAIT if the required locks cannot be acquired immediately—the affected SELECT query will fail and an exception will be thrown by the Container.

Refer to your database documentation for more information 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 indicates whether the specified method(s) reside in the EJB's home or remote interface. The value of this element must be "Home" or "Remote". 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.

1.1 weblogic-cmp-jar.xml Deployment Descriptor File Structure

`weblogic-cmp-jar.xml` defines deployment elements for a single entity EJB that uses WebLogic Server RDBMS-based persistence services.

The top-level element of the WebLogic Server 1.1 `weblogic-cmp-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>
```

1.1 weblogic-cmp-jar.xml Deployment Descriptor Elements

RDBMS Definition Elements

This section describes the RDBMS definition elements.

enable-tuned-updates

`enable-tuned-updates` 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.

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.

`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, `appc` 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.

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.

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.