BEA WebLogic Server™

Programming WebLogic Enterprise JavaBeans

Release 7.0
Document Revised: February 18, 2005

Contents

About This Document

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

1. Introducing WebLogic Server Enterprise JavaBeans

Overview of Enterprise JavaBeans 1-2
- EJB Components 1-2
- Types of EJBs 1-2

Implementation of Java Specifications 1-3
- J2EE Specification 1-3
- EJB 2.0 Specification 1-4

Securing WebLogic Server EJB Resources 1-4
WebLogic Server EJB 2.0 Support 1-4
EJB Roles 1-5
- Application Roles 1-5
- Infrastructure Roles 1-6
- Deployment and Management Roles 1-7

EJB Enhancements in WebLogic Server 7.0 1-7
- Changed Deployment Elements in WebLogic Server 7.0 1-7
- Dynamic Query Support 1-8
- Message-Driven Bean Migratable Service Support 1-8
- EJB CMP Multiple Table Mapping Support 1-8
- EJB WebLogic QL Enhancements Support 1-8

Optimistic Concurrency Support 1-9
ReadOnly Entity Concurrency Support 1-9
Combined Caching Support 1-10
Relationship Caching Support 1-10
EJB Links Support 1-10
Bulk Insert Support 1-10
EJB Developer Tools 1-11
ANT Tasks to Create Skeleton Deployment Descriptors 1-11
WebLogic Builder 1-11
EJBGen 1-12
weblogic.Deployer 1-12
WebLogic EJB Deployment Descriptor Editor 1-12
XML Editor 1-13

2. Designing EJBs

Designing Session Beans 2-1
Designing Entity Beans 2-2
Entity Bean Home Interface 2-2
Make Entity EJBs Coarse-Grained 2-3
Encapsulate Additional Business Logic in Entity EJBs 2-3
Optimize Entity EJB Data Access 2-3
Designing Message-Driven Beans 2-4
Using WebLogic Server Generic Bean Templates 2-4
Using Inheritance with EJBs 2-5
Accessing Deployed EJBs 2-6
Differences Between Accessing EJBs from Local Clients and Remote Clients 2-6
Restrictions on Concurrency Access of EJB Instances 2-7
Storing EJB References in Home Handles 2-7
Using Home Handles Across a Firewall 2-8
Preserving Transaction Resources 2-8
Allowing the Datastore to Manage Transactions 2-9
Using Container-Managed Transactions Instead of Bean-Managed Transactions for EJBs 2-9
Never Demarcate Transactions from Application 2-10

Always Use A Transactional Datasource for Container-Managed EJBs.. 2-10

3. Designing Message-Driven Beans

What Are Message-Driven Beans? 3-2
Differences Between Message-Driven Beans and Standard JMS Consumers. 3-2
Differences Between Message-Driven Beans and Stateless Session EJBs 3-3
Concurrent Processing for Topics and Queues 3-3
Developing and Configuring Message-Driven Beans....................................... 3-4
Message-Driven Bean Class Requirements 3-7
Using the Message-Driven Bean Context 3-8
Implementing Business Logic with onMessage() 3-9
Handling Exceptions 3-9
Invoking a Message-Driven Bean 3-9
Creating and Removing Bean Instances... 3-10
Deploying Message-Driven Beans in WebLogic Server................................ 3-11
Using Transaction Services with Message-Driven Beans.............................. 3-11
Message Receipts 3-12
Message Acknowledgment 3-12
Message-Driven Bean Migratable Service.. 3-13
Enabling the Message-Driven Bean Migratable Service 3-13
Migrating Message-Driven Beans.. 3-14
Configuring Message-Driven Beans for non-BEA JMS Providers.. 3-15
Specifying an MDB as Transactional.. 3-15
Specifying an MDB as Non-Transactional 3-16
Reconnecting to a JMS Server or Non-BEA Service Provider 3-16
Configuring an MDB to Listen on a JMS Distributed Destination................ 3-17
Configuring a Security Identity for a Message-Driven Bean......................... 3-18

4. The WebLogic Server EJB Container and Supported Services

EJB Container.. 4-2
EJB Life Cycle 4-2
Entity Bean Lifecycle and Caching and Pooling 4-2
Initializing Entity EJB Instances (Free Pool)..................................... 4-3

READY and ACTIVE Entity EJB Instances (Cache)........................ 4-3
Removing Beans from Cache.. 4-4
Entity EJB Lifecycle Transitions 4-5
Stateless Session EJB Life Cycle 4-6
Initializing Stateless Session EJB Instances...................................... 4-6
Activating and Pooling Stateless Session EJBs 4-7
Stateful Session EJB Life Cycle... 4-7
Stateful Session EJB Creation 4-8
Stateful Session EJB Passivation.. 4-9
Controlling Passivation 4-9
Concurrent Access to Stateful Session Beans 4-11
ejbLoad() and ejbStore() Behavior for Entity EJBs 4-12
Using is-modified-method-name to Limit Calls to ejbStore() (EJB 1.1 Only) 4-12
Warning for is-modified-method-name.. 4-13
Using delay-updates-until-end-of-tx to Change ejbStore() Behavior 4-14
EJB Concurrency Strategy.. 4-14
Concurrency Strategy for Read-Write EJBs... 4-15
Specifying the Concurrency Strategy.. 4-16
Exclusive Concurrency Strategy 4-16
Database Concurrency Strategy 4-17
Optimistic Concurrency Strategy 4-17
ReadOnly Concurrency Strategy 4-20
Read-Only Entity Beans and ReadOnly Concurrency 4-21
Restrictions for ReadOnly Concurrency Strategy 4-21
Read-Only Multicast Invalidation.. 4-21
Read-Mostly Pattern.. 4-23
Combined Caching with Entity Beans.. 4-24
Caching Between Transactions 4-25
Caching Between Transactions with Exclusive Concurrency................. 4-26
Caching Between Transactions with ReadOnly Concurrency 4-27
Caching Between Transactions with Optimistic Concurrency................ 4-27
Enabling Caching Between Transactions... 4-27
Using cache-between-transactions to Limit Calls to ejbLoad() 4-28
Restrictions for cache-between-transactions 4-29

EJBs in WebLogic Server Clusters 4-29
Clustered Homes and EJBObjects 4-29
Clustered EJB Home Objects 4-30
Clustered EJBObjects 4-30
Clustering Support for Different Types of EJBs 4-31
Stateless Session EJBs in a Cluster 4-31
Stateful Session EJBs in a Cluster 4-33
Entity EJBs in a Cluster 4-35
Cluster Address 4-38
Transaction Management 4-38
Transaction Management Responsibilities 4-38
Using javax.transaction.UserTransaction 4-39
Restriction for Container-Managed EJBs 4-40
Transaction Isolation Levels 4-40
Setting Bean-Managed Transaction Isolation Levels 4-40
Setting Container-Managed Transaction Isolation Levels 4-41
Distributing Transactions Across Multiple EJBs 4-42
Calling Multiple EJBs from a Single Transaction Context 4-42
Encapsulating a Multi-Operation Transaction 4-43
Distributing Transactions Across EJBs in a WebLogic Server Cluster 4-43
Database Insert Support 4-44
Delay-Database-Insert-Until 4-44
Bulk Insert 4-45
Resource Factories 4-46
Setting Up JDBC Data Source Factories 4-46
Setting Up URL Connection Factories 4-48

5. WebLogic Server Container-Managed Persistence Service

Overview of Container Managed Persistence Service 5-2
EJB Persistence Services 5-3
Using WebLogic Server RDBMS Persistence 5-3
Writing for RDBMS Persistence for EJB 1.1 CMP 5-5
Finder Signature 5-5
finder-list Stanza 5-6

finder-query Element 5-6
Using WebLogic Query Language (WLQL) for EJB 1.1 CMP 5-7
WLQL Syntax 5-7
WLQL Operators 5-7
WLQL Operands 5-8
Examples of WLQL Expressions 5-9
Using SQL for CMP 1.1 Finder Queries 5-10
Using EJB QL for EJB 2.0 5-11
EJB QL Requirement for EJB 2.0 Beans 5-12
Migrating from WLQL to EJB QL 5-12
Using EJB 2.0 WebLogic QL Extension for EJB QL 5-13
upper and lower Functions 5-13
Using SELECT DISTINCT 5-14
Using ORDERBY 5-14
Using SubQueries 5-15
Using Aggregate Functions 5-20
Using Queries that Return ResultSets 5-22
Properties-Based Methods of the Query Interface 5-24
Using Dynamic Queries 5-24
Enabling Dynamic Queries 5-25
Executing Dynamic Queries 5-25
Using Oracle SELECT HINTS 5-26
“get” and “set” Method Restrictions 5-27
BLOB and CLOB DBMS Column Support for the Oracle DBMS 5-27
Specifying a BLOB Using the Deployment Descriptor 5-28
Controlling Serialization of cmp-fields Mapped to OracleBlobs 5-28
Specifying a CLOB Using the Deployment Descriptors 5-28
Tuned EJB 1.1 CMP Updates in WebLogic Server 5-29
Optimized Database Updates for CMP 2.0 Entity Beans 5-30
Flushing the CMP Cache 5-30
Using Primary Keys 5-31
Primary Key Mapped to a Single CMP Field 5-31
Primary Key Class That Wraps Single or Multiple CMP Fields 5-32
Anonymous Primary Key Class 5-32
Hints for Using Primary Keys 5-32

Mapping to a Database Column 5-33
Automatic Primary Key Generation for EJB 2.0 CMP 5-33
Valid Key Field Types 5-35
Specifying Primary Key Support for Oracle 5-35
Specifying Primary Key Support for Microsoft SQL Server 5-36
Specifying Primary Key Named Sequence Table Support 5-36
Multiple Table Mapping for EJB 2.0 CMP 5-38
Multiple Table Mappings for cmp-fields 5-38
Automatic Table Creation 5-40
Container-Managed Relationships 5-42
Understanding CMRs 5-43
Requirements and Limitations 5-43
Relationship Cardinality 5-44
Relationship Direction 5-44
Removing Relationships 5-44
Defining Container-Managed Relationships 5-45
Specifying Relationship in ejb-jar.xml 5-45
Specifying Relationships in weblogic-cmp-jar.xml 5-47
Using Relationship Caching for CMRs 5-51
Nested caching-elements 5-52
Relationship Caching Limitations 5-53
Cascade Delete 5-53
Cascade Delete Method 5-54
Database Cascade Delete Method 5-54
CMRs and Local Interfaces 5-55
Using the Local Client 5-56
Changes to the Container for Local Interfaces 5-57
Groups 5-58
Specifying Field Groups 5-58
Using EJB Links 5-59
Java Data Types for CMP Fields 5-59

6. Packaging EJBs for the WebLogic Server Container

Required Steps for Packaging EJBs 6-2
Reviewing the EJB Source File Components 6-2

WebLogic Server EJB Deployment Files 6-3
- ejb-jar.xml 6-4
- weblogic-ejb-jar.xml 6-4
- weblogic-cmp-rdbms.xml 6-4
- Relationships Among the Deployment Files 6-4

Specifying and Editing the EJB Deployment Descriptors 6-5
Creating the Deployment Files 6-6
- Manually Editing EJB Deployment Descriptors 6-6
- Using the EJB Deployment Descriptor Editor 6-7

Referencing Other EJBs and Resources 6-8
- Referencing External EJBs 6-8
- Referencing Application-Scoped EJBs 6-9
- Referencing Application-Scoped JDBC DataSources 6-10

Packaging EJBs into a Deployment Directory 6-10
- ejb.jar file 6-12

Compiling EJB Classes and Generating EJB Container Classes 6-12
- Possible Generated Class Name Collisions 6-13

Loading EJB Classes into WebLogic Server 6-14
Specifying an ejb-client.jar 6-14
Manifest Class-Path 6-16

7. Deploying EJBs to WebLogic Server

Roles and Responsibilities 7-1
Deploying EJBs at WebLogic Server Startup 7-2
- Deploying EJBs in Different Applications 7-3

Deploying EJBs on a Running WebLogic Server 7-3
- EJB Deployment Names 7-4
- Deploying New EJBs into a Running Environment 7-4
- Deploying Pinned EJBs - Special Step Required 7-5

Viewing Deployed EJBs 7-6
Undeploying Deployed EJBs 7-6
- Undeploying EJBs 7-6

Redeploying EJBs 7-7
- The Redeploy Process 7-7
- Steps to Redeploy 7-8

Deploying Compiled EJB Files 7-8
Deploying Uncompiled EJB Files 7-9
Deployment Restriction with Container Managed Relationships 7-10

8. WebLogic Server EJB Utilities

EJBGen 8-1
- EJBGen Syntax 8-2
  - Surround Attributes that Contain Spaces With Double Quotes 8-4
- EJBGen Example 8-5
- EJBGen Tags 8-7
  - @ejbgen:automatic-key-generation 8-7
  - @ejbgen:cmp-field 8-7
  - @ejbgen:cmr-field 8-7
  - @ejbgen:create-default-rdbms-tables 8-8
  - @ejbgen:ejb-client-jar 8-8
  - @ejbgen:ejb-local-ref 8-8
  - @ejbgen:ejb-ref 8-9
  - @ejbgen:entity 8-9
  - @ejbgen:env-entry 8-11
  - @ejbgen:finder 8-11
  - @ejbgen:jndi-name 8-12
  - @ejbgen:local-home-method 8-12
  - @ejbgen:local-method 8-13
  - @ejbgen:message-driven 8-13
  - @ejbgen:primkey-field 8-14
  - @ejbgen:relation 8-14
  - @ejbgen:remote-home-method 8-15
  - @ejbgen:remote-method 8-15
  - @ejbgen:resource-env-ref 8-16
  - @ejbgen:resource-ref 8-16
  - @ejbgen:role-mapping 8-17
  - @ejbgen:select 8-17
  - @ejbgen:session 8-17
  - @ejbgen:value-object 8-19

ejbc 8-19

Advantages of Using ejbc 8-20
ejbc Syntax 8-20
ejbc Arguments 8-21
ejbc Options 8-21
ejbc Examples 8-23
DDConverter 8-23
Conversion Options Available with DDConverter 8-24
Using DDConverter to Convert EJBs 8-26
DDConverter Syntax 8-26
DDConverter Arguments 8-26
DDConverter Options 8-27
DDConverter Examples 8-27
weblogic.Deployer 8-28
weblogic.deploy 8-28
deploy Syntax 8-28
deploy Arguments 8-29
deploy Options 8-29

9. weblogic-ejb-jar.xml Document Type Definitions

EJB Deployment Descriptors 9-2
DOCTYPE Header Information 9-2
Document Type Definitions (DTDs) for Validation 9-4
weblogic-ejb-jar.xml 9-4
ejb-jar.xml 9-5
Changed Deployment Elements in WebLogic Server 7.0 EJB 9-5
2.0 weblogic-ejb-jar.xml Deployment Descriptor File Structure 9-6
2.0 weblogic-ejb-jar.xml Deployment Descriptor Elements 9-7
allow-concurrent-calls 9-11
allow-remove-during-transaction 9-12
cache-between-transactions 9-13
cache-type 9-14
client-authentication 9-15
client-cert-authentication 9-16
clients-on-same-server 9-17
concurrency-strategy 9-18

confidentiality 9-20
connection-factory-jndi-name 9-21
delay-updates-until-end-of-tx 9-22
description 9-23
destination-jndi-name 9-24
ejb-name 9-25
ejb-reference-description 9-26
ejb-ref-name 9-27
    Example 9-27
ejb-local-reference-description 9-28
enable-call-by-reference 9-29
enable-dynamic-queries 9-30
entity-cache 9-31
entity-cache-name 9-32
entity-cache-ref 9-33
entity-clustering 9-34
entity-descriptor 9-35
estimated-bean-size 9-36
externally-defined 9-37
finders-load-bean 9-38
global-role 9-39
home-call-router-class-name 9-40
home-is-clusterable 9-41
home-load-algorithm 9-42
idempotent-methods 9-43
identity-assertion 9-44
idle-timeout-seconds 9-45
iiop-security-descriptor 9-46
initial-beans-in-free-pool 9-47
initial-context-factory 9-48
integrity 9-49
invalidation-target 9-50
is-modified-method-name 9-51
isolation-level 9-52
jms-polling-interval-seconds 9-54

jms-client-id 9-55
jndi-name 9-56
local-jndi-name 9-57
max-beans-in-cache 9-58
max-beans-in-free-pool 9-59
message-driven-descriptor 9-60
method 9-61
method-intf 9-62
method-name 9-63
method-param 9-64
method-params 9-65
persistence 9-66
persistence-use 9-67
persistent-store-dir 9-68
pool 9-69
principal-name 9-70
provider-url 9-71
read-timeout-seconds 9-72
reference-descriptor 9-73
relationship-description 9-74
replication-type 9-74
res-env-ref-name 9-75
res-ref-name 9-76
resource-description 9-77
resource-env-description 9-78
role-name 9-79
security-permission 9-80
security-permission-spec 9-81
security-role-assignment 9-82
session-timeout-seconds 9-83
stateful-session-cache 9-84
stateful-session-clustering 9-85
stateful-session-descriptor 9-86
stateless-bean-call-router-class-name 9-87
stateless-bean-is-clusterable 9-88

stateless-bean-load-algorithm 9-89
stateless-bean-methods-are-idempotent 9-90
stateless-clustering 9-91
stateless-session-descriptor 9-92
transaction-descriptor 9-93
transaction-isolation 9-94
transport-requirements 9-95
trans-timeout-seconds 9-96
type-identifier 9-97
type-storage 9-98
type-version 9-99
weblogic-ejb-jar 9-100
weblogic-enterprise-bean 9-100
5.1 weblogic-ejb-jar.xml Deployment Descriptor File Structure 9-101
5.1 weblogic-ejb-jar.xml Deployment Descriptor Elements 9-101
- caching-descriptor 9-102
  - max-beans-in-free-pool 9-102
  - initial-beans-in-free-pool 9-102
  - max-beans-in-cache 9-103
  - idle-timeout-seconds 9-103
  - cache-strategy 9-103
  - read-timeout-seconds 9-103
- persistence-descriptor 9-104
  - is-modified-method-name 9-104
  - delay-updates-until-end-of-tx 9-104
  - persistence-use 9-105
  - db-is-shared 9-106
  - stateful-session-persistent-store-dir 9-106
- clustering-descriptor 9-106
  - home-is-clusterable 9-107
  - home-load-algorithm 9-107
  - home-call-router-class-name 9-107
  - stateless-bean-is-clusterable 9-108
  - stateless-bean-load-algorithm 9-108
  - stateless-bean-call-router-class-name 9-108

stateless-bean-methods-are-idempotent 9-108
transaction-descriptor 9-108
trans-timeout-seconds 9-109
reference-descriptor 9-109
resource-description 9-110
ejb-reference-description 9-110
enable-call-by-reference 9-110
jndi-name 9-110
transaction-isolation 9-111
isolation-level 9-111
method 9-113
security-role-assignment 9-114
.... 9-114

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

EJB Deployment Descriptors 10-2
DOCTYPE Header Information 10-2
Document Type Definitions (DTDs) for Validation 10-4
weblogic-cmp-rdbms-jar.xml 10-4
ejb-jar.xml 10-5
2.0 weblogic-cmp-rdbms-jar.xml Deployment Descriptor File Structure 10-5
2.0 weblogic-cmp-rdbms-jar.xml Deployment Descriptor Elements 10-6
automatic-key-generation 10-9
caching-element 10-10
caching-name 10-11
check-exists-on-method 10-12
cmp-field 10-13
cmr-field 10-14
column-map 10-15
create-default-dbms-tables 10-16
database-type 10-17
data-source-name 10-18
db-cascade-delete 10-19
dbms-column 10-20

dbms-column-type 10-21
description 10-22
delay-database-insert-until 10-23
    Example 10-23
ejb-name 10-24
enable-tuned-updates 10-25
field-group 10-26
field-map 10-27
foreign-key-column 10-28
foreign-key-table 10-29
generator-name 10-30
generator-type 10-31
group-name 10-32
include-updates 10-33
    Function 10-33
key-cache-size 10-34
    Example 10-34
key-column 10-35
max-elements 10-36
method-name 10-37
method-param 10-38
method-params 10-39
optimistic-column 10-40
primary-key-table 10-41
query-method 10-42
relation-name 10-43
relationship-caching 10-44
relationship-role-map 10-45
relationship-role-name 10-46
sql-select-distinct 10-47
table-map 10-48
table-name 10-50
use-select-for-update 10-51
validate-db-schema-with 10-52
verify-columns 10-53

weblogic-ql 10-54
weblogic-query 10-55
weblogic-rdbms-bean 10-56
weblogic-rdbms-jar 10-57
weblogic-rdbms-relation 10-58
weblogic-relationship-role 10-59
1.1 weblogic-cmp-rdbms-jar.xml Deployment Descriptor File Structure 10-60
1.1 weblogic-cmp-rdbms-jar.xml Deployment Descriptor Elements 10-61
- RDBMS Definition Elements 10-61
  - pool-name 10-61
  - schema-name 10-62
  - table-name 10-62
- EJB Field-Mapping Elements 10-62
  - attribute-map 10-62
  - object-link 10-62
  - bean-field 10-63
  - dbms-column 10-63
- Finder Elements 10-63
  - finder-list 10-63
  - finder 10-64
  - method-name 10-64
  - method-params 10-64
  - method-param 10-64
  - finder-query 10-64
  - finder-expression 10-65

About This Document

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

- Chapter 1, "Introducing WebLogic Server Enterprise JavaBeans," is an overview of EJB features supported in WebLogic Server.
- Chapter 2, "Designing EJBs," is an overview of design techniques developers can use to create EJBs.
- Chapter 3, "Designing Message-Driven Beans," explains how to develop and deploy message-driven beans in the WebLogic Server container.
- Chapter 4, "The WebLogic Server EJB Container and Supported Services," describes the services available to the EJB with the WebLogic Services container.
- Chapter 5, "WebLogic Server Container-Managed Persistence Service," describes the EJB container-managed persistence services available for entity EJBs in the WebLogic Server container.
- Chapter 6, "Packaging EJBs for the WebLogic Server Container," describes the steps necessary to package EJBs for deployment to WebLogic Server.
- Chapter 7, "Deploying EJBs to WebLogic Server," describes the process for deploying EJBs in the EJB container.
- Chapter 8, "WebLogic Server EJB Utilities," describes the utilities, shipped with WebLogic Server, that are used with EJBs.
- Chapter 9, "weblogic-ejb-jar.xml Document Type Definitions," describes the WebLogic-specific deployment descriptors found in the `weblogic-ejb-jar.xml` file shipped with WebLogic Server.

- Chapter 10, "weblogic-cmp-rdbms- jar.xml Document Type Definitions," describes the WebLogic-specific deployment descriptors found in `weblogic-cmp-rdbms-jar.xml` file, shipped with WebLogic Server.

Audience

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

e-docs Web Site

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

How to Print the Document

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

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

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

Related Information

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

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

Contact Us!

Your feedback on the BEA WebLogic Server documentation is important to us. Send us e-mail at **docsupport@bea.com** if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the WebLogic Server documentation.

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

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

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

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

Documentation Conventions

The following documentation conventions are used throughout this document.

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

Convention	Item

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

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

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

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

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

CHAPTER

1 Introducing WebLogic Server Enterprise JavaBeans

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

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

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

- Overview of Enterprise JavaBeans
- Implementation of Java Specifications
- WebLogic Server EJB 2.0 Support
- EJB Roles
- EJB Enhancements in WebLogic Server 7.0
- EJB Developer Tools

Overview of Enterprise JavaBeans

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

EJB Components

An EJB consists of three main components:

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

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

Types of EJBs

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

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

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

Implementation of Java Specifications

WebLogic Server is compliant with the following Java Specifications.

J2EE Specification

WebLogic Server 7.0 is compliant with the J2EE 1.3 specification.

EJB 2.0 Specification

The Enterprise JavaBeans 2.0 implementation in WebLogic Server is fully compliant and can be used in production.

Securing WebLogic Server EJB Resources

The vast majority of business applications apply some means of securing application resources so that only certain users can access such resources under certain conditions. WebLogic Server includes robust functionality to protect your EJB resources in this way.

The Securing WebLogic Resources guide gives background information application on and steps you through securing all WebLogic Server application resources, including EJBs.

WebLogic Server EJB 2.0 Support

WebLogic Server supports an implementation of Sun Microsystems's EJB 2.0 specification and is compliant with the Sun Microsystem's EJB 1.1 specification. In most cases, you can use EJB 1.1 beans with this version of WebLogic Server. However, in a few cases you may need to migrate existing EJB deployments from earlier versions of WebLogic Server to this version of the EJB container. If necessary, see the information on "DDConverter" on page 8-23 for instructions on converting the beans.

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

- New type of EJB called message-driven bean that is a Java Messaging Service (JMS) consumer. See Chapter 3, "Designing Message-Driven Beans," for more information.

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

EJB Roles

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

Application Roles

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

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

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

  For more information in the design process, see Chapter 2, "Designing EJBs." For more information in the assembly process, see Chapter 6, "Packaging EJBs for the WebLogic Server Container." For more information on the deployment descriptors, see "weblogic-ejb-jar.xml Document Type Definitions" on page 9-1and Chapter 10, "weblogic-cmp-rdbms- jar.xml Document Type Definitions."

Infrastructure Roles

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

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

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

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

Deployment and Management Roles

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

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

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

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

EJB Enhancements in WebLogic Server 7.0

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

Changed Deployment Elements in WebLogic Server 7.0

For information about new, changed, or deprecated deployment elements in `weblogic-ejb-jar.xml`, see "Changed Deployment Elements in WebLogic Server 7.0 EJB" on page 9-5.

Dynamic Query Support

Dynamic queries allow you to construct and execute queries programmatically in your application code. Therefore, all finder queries no longer need to be static and hardcoded into an EJB's deployment descriptor. This gives you the ability to create and execute new queries without having to update and redeploy an EJB and reduces the size of the EJB's deployment descriptor. See "Using Dynamic Queries" on page 5-24 for more information on dynamic queries.

Message-Driven Bean Migratable Service Support

Message-driven bean migratable service support allows the message-driven bean and the Java Messaging Service (JMS) server to migrate to another server in the same cluster, to expedite message-driven bean recovery. In previous versions of WebLogic Server, there was no recovery mechanism for the JMS server and the message-driven bean when a server failed. See "Message-Driven Bean Migratable Service" on page 3-13 for more information on the message-driven bean migration service.

EJB CMP Multiple Table Mapping Support

Multiple table mapping allows you to map a single EJB to multiple DBMS tables within a single database. You use the WebLogic Server-specific CMP container XML deployment descriptors to map multiple DBMS tables and columns to the EJB and its fields. See "Multiple Table Mapping for EJB 2.0 CMP" on page 5-38 for more information on multiple table mapping.

EJB WebLogic QL Enhancements Support

EJB WebLogic QL enhancements support in this release of WebLogic Server includes a set of enhancements to the WebLogic QL query language. WebLogic QL is a WebLogic-specific extension of the EJB 2.0 Query Language known as EJB QL. This enhancement supports the following features:

- Subqueries

- Aggregate functions
- Queries returning ResultSets

  See "Using EJB QL for EJB 2.0" on page 5-11 for more information on these three EJB WebLogic QL enhancements.

- SELECT FOR UPDATE with NO WAIT

  See "Special Note for Oracle Databases" on page 4-41 for more information on this EJB WebLogic QL enhancement in TRANSACTION_READ_COMMITTED UPDATES.

Optimistic Concurrency Support

Optimistic concurrency support is a new concurrency strategy offered by WebLogic Server. Provides optimistic support with and without caching. The EJB guarantees that the data is consistent by making sure that it was not modified before committing the transaction. See "Optimistic Concurrency Strategy" on page 4-17 for more information on this feature.

ReadOnly Entity Concurrency Support

ReadOnly entity bean concurrency support allows WebLogic Server to activate separate instance of a read-only bean for each transaction that needs concurrent access to that bean. This will eliminate the need for exclusive locking in the EJB Container and allow concurrent requests that access the same bean to proceed in parallel. See "ReadOnly Concurrency Strategy" on page 4-20 for more information on read-only entity bean caching.

Combined Caching Support

Combined caching support allows you to configure a single cache for use with multiple entity beans. This will help solve usability and performance problems. Previously, you were required to configure a separate cache for each entity bean that was part of an application. See "Combined Caching with Entity Beans" on page 4-24 for more information on combined caching.

Relationship Caching Support

Relationship caching support will improve entity bean performance by loading related entity beans into a cache and avoid multiple queries by issuing a join query for the related beans. See "Combined Caching with Entity Beans" on page 4-24 for more information on relationship caching.

EJB Links Support

EJB links support allows you to link EJB references declared in one application component to an enterprise bean contained in the same J2EE application. See "Using EJB Links" on page 5-59 for more information on EJB links.

Bulk Insert Support

Bulk insert support increases the performance of container managed persistence (CMP) bean creation by enabling the EJB container to perform multiple database inserts for CMP beans in one SQL statement. This feature allows the container to avoid multiple database inserts. See "Database Insert Support" on page 4-44 for more information on this feature.

EJB Developer Tools

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

ANT Tasks to Create Skeleton Deployment Descriptors

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

For more information on using ANT utilities to create deployment descriptors, see Packaging Enterprise JavaBeans at `http://e-docs.bea.com/wls/docs70/programming/packaging.html` in the `Developing WebLogic Server Applications`.

WebLogic Builder

WebLogic Builder is a development tools that provides a visual environment for you to edit an application's deployment descriptor XML files. You can use WebLogic Builder's interface to view these XML files as you edit them, but you will not need to make textual edits to the XML files. For instructions on how to use the WebLogic Builder tool, see WebLogic Builder.

EJBGen

`EJBGen` is an Enterprise JavaBeans 2.0 code generator. You can annotate your Bean class file with javadoc tags and then use `EJBGen` to generate the Remote and Home classes and the deployment descriptor files for an EJB application. For more information on EJBGen and a list of the supported javadoc tags, see "EJBGen" on page 8-1.

weblogic.Deployer

The `weblogic.Deployer` command-line utility is new in WebLogic Server 7.0 and replaces the deprecated `weblogic.deploy` utility. The `weblogic.Deployer` utility allows your to initiate deployment from the command line, a shell script, or any automated environment other than Java.

For instructions on using weblogic.Deployer and a list of the commands, see Deploying Using weblogic.Deployer.

WebLogic EJB Deployment Descriptor Editor

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

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

For more information, see "Specifying and Editing the EJB Deployment Descriptors" on page 6-5 .

XML Editor

The XML editor is a simple, user-friendly tool from Ensemble for creating and editing XML files. It can validate XML code according to a specified DTD or XML Schema. You can use the XML editor on Windows or Solaris machines and download it from the Dev2Dev Online .

CHAPTER

2 Designing EJBs

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

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

Designing Session Beans

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

Have the model reside on the server, in the form of a session bean. (This is analogous to having a servlet providing support for an HTML form, except that a model session bean does not affect the final presentation.) There should be one model session bean instance for each GUI form instance, which acts as the form's representative on the server. For example, if you have a list of 100 network nodes to display in a form, you might have a method called `getNetworkNodes()` on the corresponding EJB that returns an array of values relevant to that list.

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

Designing Entity Beans

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

Entity Bean Home Interface

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

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

Make Entity EJBs Coarse-Grained

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

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

Encapsulate Additional Business Logic in Entity EJBs

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

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

Optimize Entity EJB Data Access

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

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

Designing Message-Driven Beans

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

Using WebLogic Server Generic Bean Templates

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

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

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

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

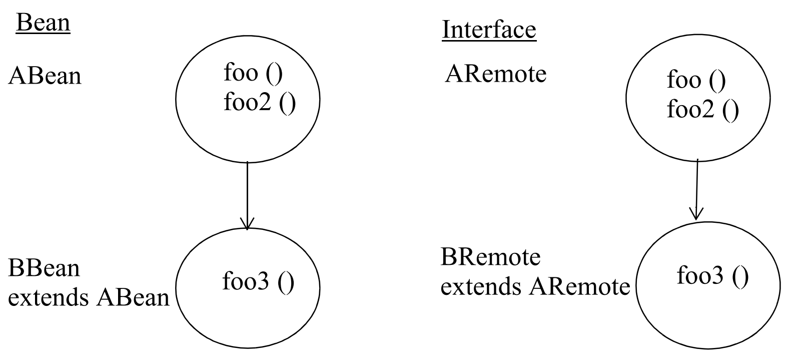
Using Inheritance with EJBs

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

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

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

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



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

Accessing Deployed EJBs

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

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

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

Differences Between Accessing EJBs from Local Clients and Remote Clients

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

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

```
...
Context ctx = getInitialContext("t3://localhost:7001", "user1", "user1Password");
...
static Context getInitialContext(String url, String user, String password) {

   Properties h = new Properties();
   h.put(Context.INITIAL_CONTEXT_FACTORY,
      "weblogic.jndi.WLInitialContextFactory");
   h.put(Context.PROVIDER_URL, url);
   h.put(Context.SECURITY_PRINCIPAL, user);
h.put(Context.SECURITY_CREDENTIALS, password);

   return new InitialContext(h);

}
```

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

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

Restrictions on Concurrency Access of EJB Instances

Although database concurrency is the default and recommended concurrency access option, multiple clients can use the exclusive concurrency option to access EJBs in a serial fashion. Using this exclusive option means that if two clients simultaneously attempt to access the same entity EJB instance (an instance having the same primary key), the second client is blocked until the EJB is available. For more information on the database concurrency option, see "Exclusive Concurrency Strategy" on page 4-16.

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

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

Storing EJB References in Home Handles

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

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

Using Home Handles Across a Firewall

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

1. Start WebLogic Server.
2. Start the WebLogic Server Administration Console.
3. From the left pane, expand the Servers node and select a server.
4. In the right pane, select the Configuration tab for that server and then the Network tab.
5. Check the Reverse DNS Allowed box to enable reverse DNS lookups.

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

Preserving Transaction Resources

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

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

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

Allowing the Datastore to Manage Transactions

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

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

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

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

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

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

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

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

Never Demarcate Transactions from Application

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

For more information on demarcating transaction see "Transaction Management Responsibilities" on page 4-38.

Always Use A Transactional Datasource for Container-Managed EJBs

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

CHAPTER

3 Designing Message-Driven Beans

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

- What Are Message-Driven Beans?
- Developing and Configuring Message-Driven Beans
- Invoking a Message-Driven Bean
- Creating and Removing Bean Instances
- Deploying Message-Driven Beans in WebLogic Server
- Using Transaction Services with Message-Driven Beans
- Message-Driven Bean Migratable Service
- Configuring Message-Driven Beans for non-BEA JMS Providers
- Reconnecting to a JMS Server or Non-BEA Service Provider
- Configuring an MDB to Listen on a JMS Distributed Destination

What Are Message-Driven Beans?

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

EJB deployers create listeners to a Queue or Topic at deployment time, and WebLogic Server automatically creates and removes message-driven bean instances as needed to process incoming messages.

Differences Between Message-Driven Beans and Standard JMS Consumers

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

The WebLogic Server container provides other standard EJB services to message-driven beans, such as security services and automatic transaction management. These services are described in more detail in "Transaction Management" on page 4-38 and in "Using Transaction Services with Message-Driven Beans" on page 3-11.

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

Note: One limitation of message-driven beans compared to standard JMS listeners is that you can associate a given message-driven bean deployment with only one Queue or Topic, as described in "Invoking a Message-Driven Bean" on page

3-9. If your application requires a single JMS consumer to service messages from multiple Queues or Topics, you must use a standard JMS consumer, or deploy multiple message-driven bean classes.

Differences Between Message-Driven Beans and Stateless Session EJBs

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

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

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

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

Concurrent Processing for Topics and Queues

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

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

Note: The maximum number of MDBs configured—via the `max-beans-in-free-pool` deployment descriptor element—to receive messages at one time cannot exceed the maximum number of execution threads. For example, if `max-beans-in-free-pool` is set to 50 but 25 is the maximum number of execution threads allowed, only 25 of the MDBs will actually receive messages.

For more information on `max-beans-in-free-pool`, see, "max-beans-in-free-pool" on page 9-59.

Developing and Configuring Message-Driven Beans

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

To develop a message-driven bean:

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

   The message-driven bean class must define the following methods:

   - One `ejbCreate()` method that the container uses to create an instance of the message-driven bean on the free pool.
   - One `onMessage()` method that is called by the bean's container when a message is received. This method contains the business logic that handles processing of the message.
   - One `setMessageDrivenContext{}` method that provides information to the bean instance about its environment (certain deployment descriptor values); the Context is also the mechanism the bean class uses to access some services provided by the EJB container.

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

  For an example of output for a message-driven bean class, see "Message-Driven Bean Class Requirements" on page 3-7.

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

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

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

3. Set the `message-driven` element in the bean's `ejb-jar.xml` file to declare the bean.

4. Set the `message-driven-destination` element in the bean's `ejb-jar.xml` file to specify whether the bean is intended for a Topic or Queue.

5. Set the `subscription-durability` sub-element in the bean's `ejb-jar.xml` file when you want to specify whether an associated Topic should be durable.

6. If your bean will demarcate its own transaction boundaries, set the `acknowledge-mode` sub-element to specify the JMS acknowledgment semantics to use. This element has two possible values: `AUTO_ACKNOWLEDGE` (the default) or `DUPS_OK_ACKNOWLEDGE`.

7. If the container will manage the transaction boundaries, set the `transaction-type` element in the bean's `ejb-jar.xml` file to specify how the container must manage the transaction boundaries when delegating a method invocation to an enterprise bean's method.

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

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

```
<enterprise-beans>

       <message-driven>

              <ejb-name>exampleMessageDriven1</ejb-name>
```

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

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

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

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

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

9. Compile and generate the message-driven bean class using instructions in "Packaging EJBs into a Deployment Directory" on page 6-10.

10. Deploy the bean on WebLogic Server using the instructions in "Deploying Compiled EJB Files" on page 7-8.

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

Message-Driven Bean Class Requirements

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

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

```
public class MessageTraderBean implements MessageDrivenBean,
MessageListener{

       public MessageTraderBean() {...};

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

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

              // final or abstract.

       public void ejbCreate() (...)

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

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

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

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

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

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

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

              // declared as final or static.

       public void ejbRemove() {...}

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

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

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

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

}
```

Using the Message-Driven Bean Context

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

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

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

Implementing Business Logic with onMessage()

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

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

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

Handling Exceptions

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

Invoking a Message-Driven Bean

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

1. WebLogic Server obtains a new bean instance.

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

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

   If no bean instance is available in the free pool and the limit specified by `max-beans-in-free-pool`) has been reached, WebLogic Server waits until a bean instance is free. See "max-beans-in-free-pool" on page 9-59 for more information about this attribute.

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

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

   See "Implementing Business Logic with onMessage()" on page 3-9.

Note: These instances can be pooled.

Creating and Removing Bean Instances

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

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

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

Deploying Message-Driven Beans in WebLogic Server

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

Using Transaction Services with Message-Driven Beans

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

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

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

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

```
<assembly-descriptor>

       <container-transaction>

              <method>

<ejb-name>MyMessageDrivenBeanQueueTx</ejb-name>

                     <method-name>*</method-name>

              </method>

       <trans-attribute>NotSupported</trans-attribute>

       </container-transaction>

</assembly-descriptor>
```

Message Receipts

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

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

Message Acknowledgment

For message-driven beans that use container-managed transaction demarcation, WebLogic Server automatically acknowledges a message when the EJB transaction commits. If the EJB uses bean-managed transactions, both the receipt and the acknowledgment of a message occur outside the EJB transaction context. WebLogic

Server automatically acknowledges messages for EJBs with bean-managed transactions, but you can configure acknowledgment semantics using the `acknowledge-mode` deployment descriptor element defined in the `ejb-jar.xml` file.

Message-Driven Bean Migratable Service

WebLogic Server supports migratable and recovery services for message-driven beans. To provide these migratable and recovery services, WebLogic JMS uses the migration framework provided by WebLogic Server to respond to migration requests and bring a JMS server back online after a failure. Once the JMS server migrates to an available server, you should manually migrate the associated message-driven beans from a failed server in a WebLogic Server cluster to the same available server. The Message-driven bean can only use the Migratable Service when they are on clustered servers At this time, the Migratable Service cannot span multiple clusters.

If WebLogic Server does not migrate the message-driven bean along with the JMS Server to an available server in the cluster, the JMS destination will be flooded with messages. To expedite message-driven bean recovery until the original server recovers, the message-driven bean marks itself as migratable and WebLogic Server implements the Migratable Service process. After you migrate the bean to another server, it connects to its JMS server and continues to pull messages from the JMS destination on behalf of the failed server.

Enabling the Message-Driven Bean Migratable Service

To enable the message-driven bean Migratable Service:

1. Configure the message-driven bean as described in "Developing and Configuring Message-Driven Beans" on page 3-4.
2. Specify the message-driven bean's JMS destination type as either topic or queue by setting the `destination-type` element in the `ejb-jar.xml` file. For instructions on configuring JMS destinations, see Configuring Destinations.
3. Specify one of the following deployment schemes for the JMS destination:

- Simple destination - EJB container deploys the message-driven bean with the JMS destination when the topic or queue destination isn't distributed.
- Distributed destination - EJB container deploys the message-driven bean with the JMS destination on every server when the topic or queue destination is distributed.

For instructions, see Configuring Distributed Destinations.

4. Use the WebLogic Server Administration Console, configure a JMS server. For instructions see Configuring JMS Servers.

   A JMS server is deployed on a server in a WebLogic Server cluster and handles requests for a set of JMS destinations.

5. Configure JMS migratable targets for the JMS server. For instructions, see Configuring JMS Migratable Targets.

Migrating Message-Driven Beans

To migrate message-driven bean from a failed server in a WebLogic Server cluster to an available server:

1. Start the WebLogic Server Administration Console.
2. Specify one of the following deployment schemes for the JMS destination:
   - Simple destination - EJB container deploys the message-driven bean with the JMS destination when the JMS destination isn't distributed.
   - Distributed destination - EJB container deploys the message-driven bean with the JMS destination on every server when the JMS destination is distributed.

Because the message-driven bean can detect the migration target for the JMS server, you do not need to change the migration target for the message-driven bean.

However, the message-driven bean must be deployed in the cluster or all of the servers on the JMS server migration target lists because message-driven bean is not possible during migration.The message-driven bean is deployed with the a JMS destination on all servers in the migration target list, and remain inactive when the JMS destination is inactive.

When WebLogic Server activates a message-driven bean, it detects the JMS server and starts pulling the message from the JMS destination that is specified for the bean.

Configuring Message-Driven Beans for non-BEA JMS Providers

You can configure message-driven beans to work with non-BEA JMS providers such as IBM MQSeries. Beginning with WebLogic Server 7.0, you can do this for MDBs supporting container-managed transactions ("transactional MDBs") in addition to MDBs supporting bean-managed transactions ("non-transactional MDBs").

This means that it is now possible to for applications with transactional MDBs to achieve exactly-once semantics with a non-BEA JMS provider for messages processed by an MDB. In addition, WebLogic Server will use XA to automatically enlist the non-BEA JMS provider in a transaction.

For applications with non-transactional MDBs, the MDB will provide at-least-once message processing semantics and XA is not required.

If the non-BEA JMS provider does not support XA, then you cannot deploy an MDB that supports container-managed transactions with that provider. Furthermore, if the JMS provider does support XA, you must ensure that the JMS connection factory that you specify in the `weblogic-ejb-jar.xml` file supports XA—each JMS provider has a different way to specify this.

Specifying an MDB as Transactional

To specify an MDB as transactional, do the following:

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

Also, sure to set destination-jndi-name, initial-context-factory, provider-url, and connection-factory-jndi-name elements in weblogic-ejb-jar.xml appropriately for the non-BEA JMS provider.

For transactional MDBs, the JMS connection factory specified in connection-factory-jndi-name must support the optional XA extensions to JMS.

Specifying an MDB as Non-Transactional

To specify an MDB as non-transactional, do the following:

- Set the `transaction-type` element inside the `message-driven` element in the `ejb-jar.xml` file to `Bean`.
- Set the `trans-attribute` element inside the `container-transaction` attribute in the `ejb-jar.xml` file to `NotRequired`.

See the white paper, "Using Foreign JMS Providers with WLS Message Driven Beans" (`jmsmdb.pdf`) at http://dev2dev.bea.com/resourcelibrary/whitepapers/jmsproviders.jsp for an example of how to configure an MDB to use a non-BEA provider.

Reconnecting to a JMS Server or Non-BEA Service Provider

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

Configuring an MDB to Listen on a JMS Distributed Destination

WebLogic JMS supports service continuity in the event of a WebLogic Server instance failure within a cluster through the configuration of multiple physical destinations (queues and topics) as members of a single distributed destination set. Once configured, your producers and consumers send and receive messages through what appears to be a single destination.

However, WebLogic JMS actually distributes the messaging load across all the available destination members within the distributed destination. In the event that a member becomes unavailable due to a server failure, traffic is then redirected toward the other available destination members in the set.

Beginning with this release, when an MDB is deployed to a server in a cluster, WebLogic Server automatically enumerates the distributed destination members and ensures that there's an MDB listening on each member.

When an MDB is deployed to a server in a cluster, WebLogic Server automatically enumerates the distributed destination members and ensures that there's an MDB listening on each member.

Follow these steps to configure a message-driven bean for a distributed destination:

1. Configure a JMS distributed destination, as described in Configuring Distributed Destinations at http://edocs.bea.com/wls/docs70/adminguide/jms.html#config_distributed_destinations.
2. Set the MDB's destination-jndi-name in `weblogic-ejb-jar.xml` to the name used to bind the distributed topic or queue into the JNDI namespace.
3. Set the target for MDB to be the same as the distributed destination. The MDB need to be deployed wherever that distributed destination exists.
4. Deploy the MDB.

At deployment time, the MDB detects the member of the distributed destination the exists on that WebLogic Server instance, pin itself to the member, and start processing messages.

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. The security identity is especially important when using an MDB to receive messages from a non-BEA JMS provider (either a JMS provider from another vendor or a WebLogic Server JMS provider running in another WebLogic Server domain).

To configure a security identity for an MDB:

1. Create WebLogic user for the MDB. For more information, see "Users and Groups" in *Securing WebLogic Resources*. This WebLogic user should have the username and password required by the non-BEA JMS provider 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.

   The Deployments node expands to show the types of WebLogic resources that can be deployed.

b. Right-click the EJB resource (in this case, the MDB) for which you want to create a credential map.

c. Choose the Define Policies and Roles for Individual Beans... option.

d. Click the [Define Credential Maps] link for the MDB for which you want to create a credential map.

e. Click the Configure a New Credential Map link.

f. Enter the WebLogic Server username and password you defined for the MDB in step 1 in the WLS User field.

g. Click Apply to save your changes.

CHAPTER

4 The WebLogic Server EJB Container and Supported Services

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

- EJB Container
- EJB Life Cycle
- ejbLoad() and ejbStore() Behavior for Entity EJBs
- EJB Concurrency Strategy
- Combined Caching with Entity Beans
- Caching Between Transactions
- EJBs in WebLogic Server Clusters
- Transaction Management
- Database Insert Support
- Resource Factories

EJB Container

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

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

EJB Life Cycle

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

Entity Bean Lifecycle and Caching and Pooling

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

- Free pool—stores anonymous entity beans that are used for invoking finders, home methods, and creating entity beans.
- Cache—contains instances that have an identity—a primary key, or are currently enlisted in a transaction (READY and ACTIVE entity EJB instances).

The sections that follow describe the lifecycle of an entity bean instance, and how the container populates and manages the free pool and the cache. For an illustration, see Figure 4-2.

Initializing Entity EJB Instances (Free Pool)

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

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

An attempt to obtain an entity bean instance from the free pool will always succeed, even if the pool is empty. If the pool is empty, a new bean instance is 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 value of the `max-beans-in-free-pool` element in `weblogic-ejb-jar.xml`.

READY and ACTIVE Entity EJB Instances (Cache)

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

A READY instance is in the cache, has an identity—an associated primary key, but is not currently enlisted in a transaction. WebLogic maintains READY entity EJB instances in least-recently-used (LRU) order. Current Beans in Cache field in the monitoring tab displays the count of active and ready beans.

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

Cache Rules Vary by Concurrency Strategy

Table 4-1 lists, for each concurrency strategy:

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

Figure 4-1 Entity EJB Caching Behavior by Concurrency Type

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

Removing Beans from Cache

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

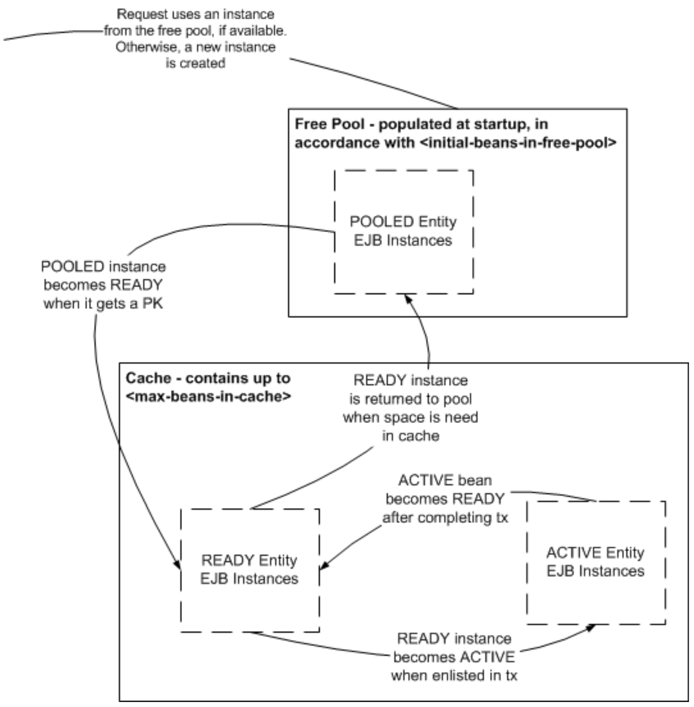
When the container tries to return an instance to the free pool and the pool already contains `max-beans-in-free-pool` instances, the instance is discarded.

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

Entity EJB Lifecycle Transitions

Figure 4-2 illustrates the EJB free pool and cache, and the transitions that occur throughout an entity bean instance's lifecycle.

Figure 4-2 Entity Bean Lifecycle

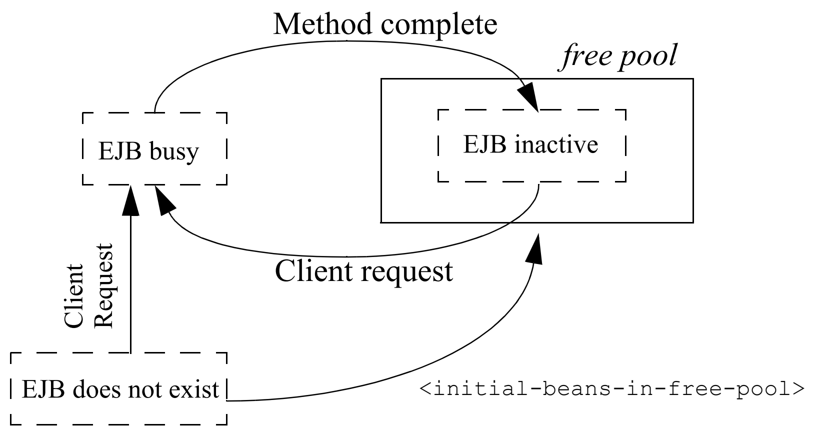

Request uses an instance from the free pool, if available. Otherwise, a new instance is created
Free Pool - populated at startup, in accordance with <initial-beans-in-free-pool>
POOLED Entity EJB Instances
POOLED instance becomes READY when it gets a PK
Cache - contains up to <max-beans-in-cache>
READY instance is returned to pool when space is need in cache
ACTIVE bean becomes READY after completing tx
READY Entity EJB Instances
ACTIVE Entity EJB Instances
READY instance becomes ACTIVE when enlisted in tx

Stateless Session EJB Life Cycle

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

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

Figure 4-3 WebLogic Server free pool showing stateless session EJB life cycle



Initializing Stateless Session EJB Instances

By default, no stateless session EJB instances exist in WebLogic Server at startup time. As clients access individual beans, WebLogic Server initializes new instances of the EJB. However, if you want inactive instances of the EJB to exist in WebLogic Server when it is started, specify how many in the `initial-beans-in-free-pool` deployment descriptor element, in the `weblogic-ejb-jar.xml` file.

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

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

Activating and Pooling Stateless Session EJBs

When a client calls a method on a stateless session EJB, WebLogic Server obtains an instance from the free pool. 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.

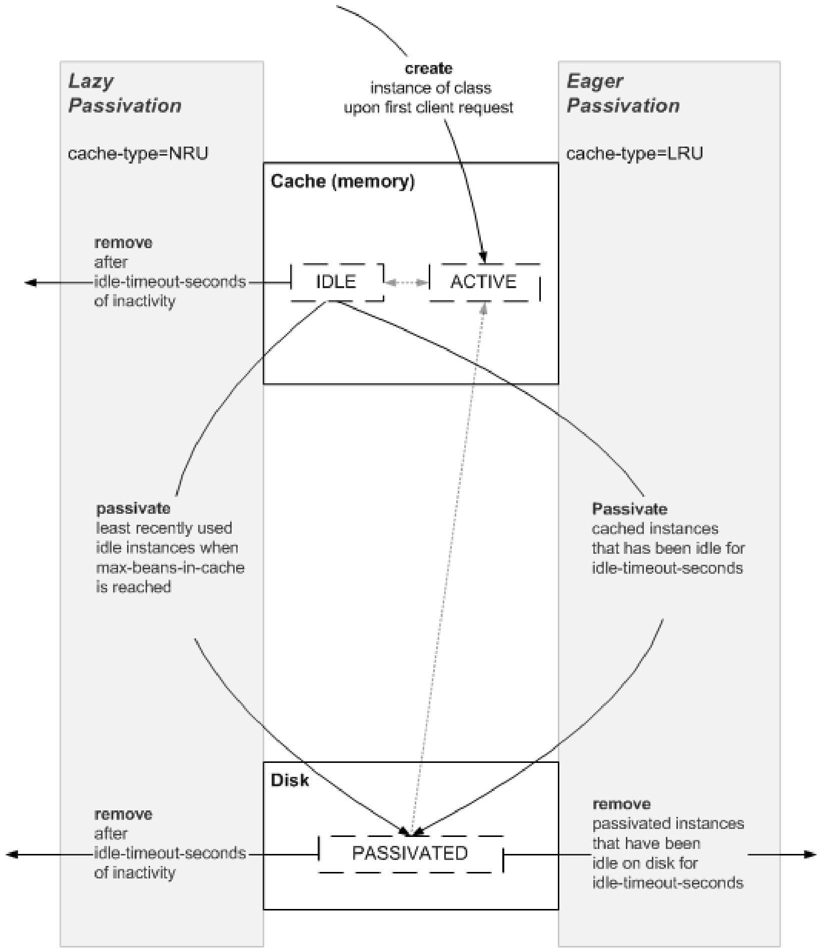
If all instances of an EJB class are active and `max-beans-in-free-pool` has been reached, new clients requesting the EJB class will be blocked until an active EJB completes a method call. If the transaction times out (or, for non-transactional calls, if five minutes elapse), WebLogic Server throws a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

Stateful Session EJB Life Cycle

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

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

Figure 4-4 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, referred to as eager passivation.
- `NRU`—not recently used, referred to as lazy passivation

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

Eager Passivation (LRU)

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

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

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.

Preventing Removal of Idle EJBs

Setting `idle-timeout-seconds` to 0 stops WebLogic Server from removing EJBs that are idle for a period of time. However, EJBs may still be passivated if cache resources become scarce.

Managing EJB Cache Size

For a discussion of managing cache size to optimize performance in a production environment see "Setting EJB Pool 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\700\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\700\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\700\bea\user_domains\mydomain\myserver\pstore\14t89ge
x0m2fr
```

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.

ejbLoad() and ejbStore() Behavior for Entity EJBs

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

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

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

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

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

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. We recommend that you do not use `is-modified-method-name` with bean-managed-persistence (BMP) because you would need to create both the `is-modified-method-name` element. and the `ejbstore` method.

By default, WebLogic Server calls the `ejbStore()` method at the successful completion (commit) of each transaction. `ejbStore()` is called at commit time regardless of whether the EJB's persistent fields were actually updated, 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 9-51.

Warning for is-modified-method-name

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

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

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

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

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

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

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

EJB Concurrency Strategy

The concurrency strategy specifies how the EJB container should manage concurrent access to an entity bean. Although the `Database` option is the default concurrency strategy for WebLogic Server, you may want to specify other options for your entity bean depending on the type of concurrency access the bean requires. WebLogic Server provides the following concurrency strategy options:

Concurrency Option	Description
`Exclusive`	Places an exclusive lock on cached entity EJB instances when the bean is associated with a transaction. Other requests for the EJB instance are block until the transaction completes. This option was the default locking behavior for WebLogic Server versions 3.1 through 5.1
`Database`	Defers locking requests for an entity EJB to the underlying datastore. WebLogic Server allocates a separate entity bean instance and allows locking and caching to be handled by the database. This is the default option.
`Optimistic`	Holds no locks in the EJB container or database during a transaction. The EJB container verifies that none the data updated by the transaction has changed before committing the transaction. If any updated data changed, the EJB container rolls back the transaction.
`ReadOnly`	Used only 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.

Concurrency Strategy for Read-Write EJBs

You can use the `Exclusive`, `Database`, and `Optimistic` concurrency strategies for `read-write` EJBs. WebLogic Server loads EJB data into the cache at the beginning of each transaction, or as described in "Using cache-between-transactions to Limit Calls to ejbLoad()" on page 4-28. WebLogic Server calls `ejbStore()` at the successful commit of a transaction, or as described under "Using is-modified-method-name to Limit Calls to ejbStore() (EJB 1.1 Only)" on page 4-12.

Specifying the Concurrency Strategy

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

The following excerpt from `weblogic-ejb-jar.xml` shows how to set the concurrency strategy for an EJB. In the following sample XML, the code specifies the default locking mechanism, `Database`.

Figure 4-5 Sample XML specifying the concurrency strategy

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

If you do not specify a `concurrency-strategy`, WebLogic Server performs database locking for entity EJB instances.

A description of each concurrency strategy is covered in the following sections.

Exclusive Concurrency Strategy

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

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

Database Concurrency Strategy

The `Database` concurrency strategy is the default option for WebLogic Server and the *recommended* mechanism for EJB 1.1 and EJB 2.0 beans. It improves concurrent access for entity EJBs. The WebLogic Server container defers locking services to the underlying database. Unlike exclusive locking, the underlying data store can provide finer granularity for locking EJB data, and deadlock detection.

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

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

When using the Database concurrency strategy instead of Optimistic with the `cache-between-transactions` element set to `"True,"` you will receive a warning message from the compiler indicating that `cache-between-transactions` should be disabled. If this condition exists, WebLogic Server automatically disables `cache-between-transactions`.

Optimistic Concurrency Strategy

The `Optimistic` concurrency strategy does not hold any locks in the EJB container or the database while the transaction is in process. When you specify this option, the EJB container ensures that the data being updated by a transaction has not changed. It performs a "smart update" by checking the fields before it commits the transaction.

Note: The EJB container does not check Blob/Clob fields for optimistic concurrency. The work-around is to use version or timestamp checking.

Limitations of Optimistic Concurrency

If you use optimistic concurrency, BEA recommends that the `include-updates` element in `weblogic-cmp-jar.xml` be set to `false`. Using optimistic concurrency with `include-updates` set to `true` is inefficient—it is equivalent to using pessimistic concurrency. If you need to set `include-updates true`, use the database concurrency strategy.

Using optimistic concurrency with `include-updates` set to `true` is not supported for databases that hold locks during transactions (non-Oracle databases) This is because: optimistic transactions read using a local transaction to avoid holding locks until the end of the transaction. However, optimistic transactions write using the current JTA transaction so that the updates can be rolled back, if necessary. In general, updates made by the JTA transaction are not visible to the read transactions until the JTA transaction commits.

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 work-around is to use version or timestamp checking.

Configuring Optimistic Checking

Configure validity checking for a bean with `Optimistic` concurrency using the `verify-columns` element in the `table-name` stanza for the bean in `weblogic-cmp-jar.xml`.

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 or timestamp column, updating its value as appropriate upon completion of the transaction.

Note: The version or timestamp column is *not* updated if the transaction did not modify and 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. 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.

By default, caching between transactions is not enabled for optimistic beans. You must explicitly enable it. See "Using cache-between-transactions to Limit Calls to ejbLoad()" on page 4-28. When you enable long term caching for an entity bean with an optimistic concurrency strategy the EJB container reuses the cached values from previous transactions. The container ensures that the updates are transactionally consistent by checking for optimistic conflicts at the end of the transaction. In addition, notifications for updates of optimistic data are broadcast to other cluster members to help avoid optimistic conflicts and keep cached data fresh.

Optimistic Checking 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();
  ```

ReadOnly Concurrency Strategy

WebLogic Server provides support for concurrent access to read-only entity beans. This concurrency strategy activates an instance of a read-only entity bean for each transaction so that requests may be processed in parallel.

Prior to WebLogic Server 7.0 read-only entity beans used the exclusive locking concurrency strategy. This strategy places an exclusive lock on cached entity bean instances when the bean is associated with a transaction. Other requests for the entity bean instance are block until the transaction completes.

To avoid reading from the database, WebLogic Server copies the state for an EJB 2.0 CMP bean from the existing instance in the cache. For this release, the default concurrency strategy for read-only entity beans is the `ReadOnly` option.

You can specify read-only entity bean caching at the application-level or the component-level.

To enable read-only entity bean caching:

1. Specify the `ReadOnly` option in the `concurrency-strategy` deployment descriptor element for either a JAR file or an EAR file.
   - Specify the `concurrency-strategy` element for application-level caches (EARS) in the `entity-cache-ref` element in the `weblogic-ejb-jar.xml` file.
   - Specify the `concurrency-strategy` element for component-level caches (JARS) in the `weblogic-ejb-jar.xml` file.
2. For instructions on specifying deployment descriptors, see "Specifying and Editing the EJB Deployment Descriptors" on page 6-5.

Read-Only Entity Beans and ReadOnly Concurrency

Previous versions of read-only entity beans will work in this version of WebLogic Server. As in previous versions, you can set the `read-timeout-seconds` element set in `weblogic-ejb-jar.xml`. If an EJB's concurrency strategy is `ReadOnly` and `read-timeout-seconds` is set, when a read-only bean is invoked, WebLogic Server checks whether the cached data is older than the `read-timeout-seconds` setting. If it is, the bean's `ejbLoad` is called. Otherwise, the cached data is used.

Restrictions for ReadOnly Concurrency Strategy

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

- They cannot require updates to the EJB data, because WebLogic Server never calls `ejbStore()` for read-only entity EJBs.
- The EJB's method calls must be idempotent. See "Stateless Session EJBs in a Cluster" on page 4-31 for more information.
- Because the bean's underlying data may be updated by an external source, calls to `ejbLoad()` are governed by the deployment parameter, `read-timeout-seconds`.

Read-Only Multicast Invalidation

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

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

Figure 4-6 Sample code showing CachingHome and CachingLocalHome interfaces

```
package weblogic.ejb;

public interface CachingHome {

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

public interface CachingLocalHome {

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

}
```

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

Figure 4-7 Sample code showing how to cast the home and call the method

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

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

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

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

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

Read-Mostly Pattern

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

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

%SAMPLES_HOME%`/server/config/examples/ejb/extensions/readMostl`
`y`

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

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

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

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

  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 "Optimistic Concurrency Strategy" on page 4-17.

Note: In a WebLogic Server cluster, clients of the `read-only` EJB benefit from using cached EJB data. Clients of the `read-write` EJB benefit from true transactional behavior, because the `read-write` EJB's state always matches the state of its data in the underlying datastore. See "Entity EJBs in a Cluster" on page 4-35 for more information.

Combined Caching with Entity Beans

Combined caching allows multiple entity beans that are part of the same J2EE application to share a single runtime cache. Previously, you had to configure a separate cache for each entity bean that was part of an application. This caused some usability and performance problems in that it took more time to configure caches for each entity bean and more memory to run the application. This feature will help solve those problems.

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. Provide an entry in the `weblogic-application.xml` file as follows:

```
<weblogic-application>
 <ejb>
    <entity-cache>
        <entity-cache-name>large_account</entity-cache-name>
        <max-cache-size>
           <megabytes>1</megabytes>
        </max-cache-size>
    </entity-cache>
 </ejb>
</weblogic_application>
```

   Use the `entity-cache` element to define a named application level cache that will be used to cache entity bean instances at runtime. There are no restrictions on the number of different entity beans that may reference an individual cache.

The sub elements of `entity-cache` have the same basic meaning as they do in the `weblogic-ejb-jar.xml` deployment descriptor file.

3. Specify an `entity-descriptor` element in `weblogic-ejb-jar.xml` file.

   Use the `entity-descriptor` element to configure an entity bean to use an application level cache.

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

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

Caching Between Transactions

Use caching between transactions or long tern caching to enable the EJB container to cache an entity bean's persistent data between transactions. Whether you can set caching between transactions for an entity bean depends on its concurrency strategy, as summarized in the following three tables:

Table 4-1 Permitted `cache-between-transactions` values, by concurrency strategy, for BMP beans

A BMP bean using this concurrency strategy	Can set `cache-between-transactions` to
`Database`	`False` only
`Exclusive`	`True` or `False`
`Read-Only`	`True` or `False`
`Optimistic`	Not applicable. `Optimistic` concurrency is not available for BMP beans.

Table 4-2 Permitted cache-between-transactions values, by concurrency strategy, for CMP 2.0 beans

A CMP 2.0 bean using this concurrency strategy	**Can set** `cache-between-transactions` **to this value**
`Database`	`True` only
`Exclusive`	`True` or `False`
`Read-Only`	`True` or `False`
`Optimistic`	`True` or `False`

Table 4-3 Permitted cache-between-transactions values, by concurrency strategy, for CMP 1.1 beans

A CMP 1.1 bean using this concurrency strategy	**Can set** `cache-between-transactions` **to this value**
`Database`	`True` only
`Exclusive`	`True` or `False`
`Read-Only`	`True` or `False`
`Optimistic`	Not applicable. `Optimistic` concurrency is not available for CMP 1.1 beans.

Caching Between Transactions with Exclusive Concurrency

When you enable long term caching for an entity bean with an `Exclusive` concurrency strategy the EJB container must have exclusive update access to the underlying data. This means that another application outside of the EJB container must not be updating the data. If you deploy an EJB with an Exclusive concurrency strategy in a cluster, long term caching is disabled automatically because any node in the cluster may update the data. This would make long term caching impossible.

In previous versions of WebLogic Server, this feature was controlled by the `db-is-shared` element of `weblogic-ejb-jar.xml`.

Note: Exclusive concurrency is a single-server feature. Do not attempt to use it with clustered servers.

Caching Between Transactions with ReadOnly Concurrency

When you disable long term caching for an entity bean with a `ReadOnly` concurrency strategy it ignores the value of the `cache-between-transactions` setting because the EJB container always performs long term caching of read-only data.

Caching Between Transactions with Optimistic Concurrency

When you enable long term caching for an entity bean with an `Optimistic` concurrency strategy the EJB container reuses the cached values from previous transactions. The container ensures that the updates are transactionally consistent by checking for optimistic conflicts at the end of the transaction. See "Optimistic Concurrency Strategy" on page 4-17 for instructions on setting optimistic checking.

In addition, notifications for updates of optimistic data are broadcast to other cluster members to help avoid optimistic conflicts.

Enabling Caching Between Transactions

To enable caching between transactions:

1. Set the `cache-between-transactions` element in the `weblogic-ejb-jar.xml` file by choosing one of the following options:
   - Specify `True` to enable the EJB container performs long term caching of the data.

- Specify `False` to enable the EJB container performs short caching of the data. This is the default setting.

2. For instructions on specifying deployment descriptors, see "Specifying and Editing the EJB Deployment Descriptors" on page 6-5.

Using cache-between-transactions to Limit Calls to ejbLoad()

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

In the special circumstance where only a single WebLogic Server transaction ever accesses a particular EJB concurrently, such as when you use exclusive concurrency for a single server; not a cluster, calling `ejbLoad()` by default is unnecessary. Because no other clients or systems update the EJB's underlying data, WebLogic Server's cached version of the EJB data is always up-to-date. Calling `ejbLoad()` in this case simply creates extra overhead for WebLogic Server clients that access the bean.

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

- A client first references the EJB
- The EJB's transaction is rolled back

Restrictions for cache-between-transactions

The following restrictions apply to cache-between-transactions:

In a single-server deployment, enable cache-between-transactions only with Exclusive, Optimistic and Read-Only concurrency strategies. You cannot use cache-between transactions with a Database concurrency strategy.

In a clustered deployment, enable `cache-between-transactions` only with Optimistic and Read-Only concurrency strategies. You cannot use cache-between-transactions with an Exclusive or Database Concurrency strategy.

EJBs in WebLogic Server Clusters

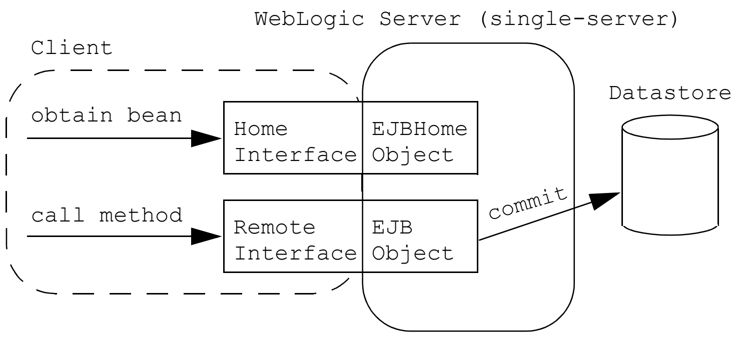
This section describes clustering support for EJBs.

Clustered Homes and EJBObjects

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

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

Figure 4-8 Single server behavior



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

Clustered EJB Home Objects

All EJB types—stateless session, stateful session, and entity EJBs—can have cluster-aware home stubs. Whether or not a cluster-aware home stub is created is determined by the `home-is-clusterable` deployment element in `weblogic-ejb-jar.xml`.

When an EJB bean is deployed to a cluster, its home is bound into the cluster-wide naming service. Each server can bind an instance of the home under the same name. When a client looks up this home, it gets a replica-aware stub that has a reference to the home on each server that deployed the bean. When `create()` or `find()` is called, the replica-aware stub routes the call to one of the replicas. The home replica receives the `find()` results or creates an instance of the bean on this server.

The clustered home stub provides load balancing by distributing EJB lookup requests to available servers. It can also provide failover support for lookup requests, because it routes those requests to available servers when other servers have failed.

Clustered EJBObjects

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

Whether or not an EJB can use a replica-aware `EJBObject` stub depends on the type of EJB deployed and, for entity EJBs, the concurrency strategy selected at deployment time. For more information, see "Clustering Support for Different Types of EJBs" on page 4-31.

Clustering Support for Different Types of EJBs

These sections describe the clustering support for session and entity EJBs.

- "Stateless Session EJBs in a Cluster" on page 4-31
- "Stateful Session EJBs in a Cluster" on page 4-33
- "Entity EJBs in a Cluster" on page 4-35

Stateless Session EJBs in a Cluster

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

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

If methods are written in such a way that repeated calls to the same method do not cause duplicate updates, the method is said to be "idempotent." For idempotent methods, WebLogic Server provides two `weblogic-ejb-jar.xml` deployment properties, one at the bean level and one at the method level.

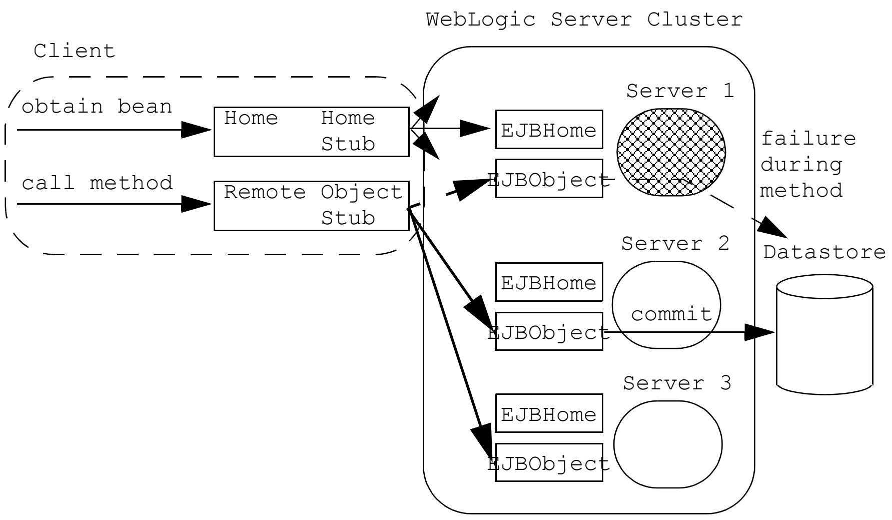
At the bean level, if you set `stateless-bean-methods-are-idempotent` to “true”, WebLogic Server assumes that the method is idempotent and *will* provide failover services for the EJB method, even if a failure occurs during a method call.

At the method level, you can use the `idempotent-methods` deployment property to accomplish the same thing:

```
<idempotent-methods>
        <method>
                <description>...</description>
                <ejb-name>...</ejb-name>
                <method-intf>...</method-intf>
                <method-name>...</method-name>
                <method-params>...</method-params>
        </method>
</idempotent-methods>
```

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

Figure 4-9 Stateless session EJBs in a clustered server environment



Stateful Session EJBs in a Cluster

To enable stateful session EJBs to use cluster-aware home stubs, set `home-is-clusterable` to "true." This provides failover and load balancing for stateful EJB lookups. Stateful session EJBs configured this way use replica-aware EJBObject stubs. For more information on in-memory replication for stateful session EJBs, see "In-Memory Replication for Stateful Session EJBs" on page 4-34.

Note: Load balancing and failover are discussed extensively in *Using WebLogic Server Clusters*. See these three sections: "EJB and RMI Objects", "Load Balancing for EJBs and RMI Objects" and "Replication and Failover for EJBs and RMIs".

In-Memory Replication for Stateful Session EJBs

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

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

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

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

Clients of a stateful session EJB are therefore guaranteed to have quick access to the latest committed state of the EJB, except under the special circumstances described in "Limitations of In-Memory Replication" on page 4-35. For more information on the use of replication groups, see Using Replication Groups.

Requirements and Configuration for In-Memory Replication

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

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

Figure 4-10 XML sample enabling replication

```
<stateful-session-clustering>
       ...
       <replication-type>InMemory</replication-type>
</stateful-session-clustering>
```

Limitations of In-Memory Replication

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

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

Entity EJBs in a Cluster

As with all EJBs, entity EJBs can utilize cluster-aware home stubs once you set `home-is-clusterable` to "true."

The behavior of the `EJBObject` stub depends on the `concurrency-strategy` deployment element in `weblogic-ejb-jar.xml`. `concurrency-strategy` can be set to `Read-Write` or `Read-Only`. The default value is `Read-Write`.

Fore details, see:

- "Read-Only Entity EJBs in a Cluster" on page 4-36
- "Read-Write Entity EJBs in a Cluster" on page 4-36

Read-Only Entity EJBs in a Cluster

When a home finds or creates a read-only entity bean, it returns a replica-aware an `EJBObject` stub. This stub load balances on every call but does not automatically fail over in the event of a recoverable call failure. Read-only beans are also cached on every server to avoid database reads.

Read-Write Entity EJBs in a Cluster

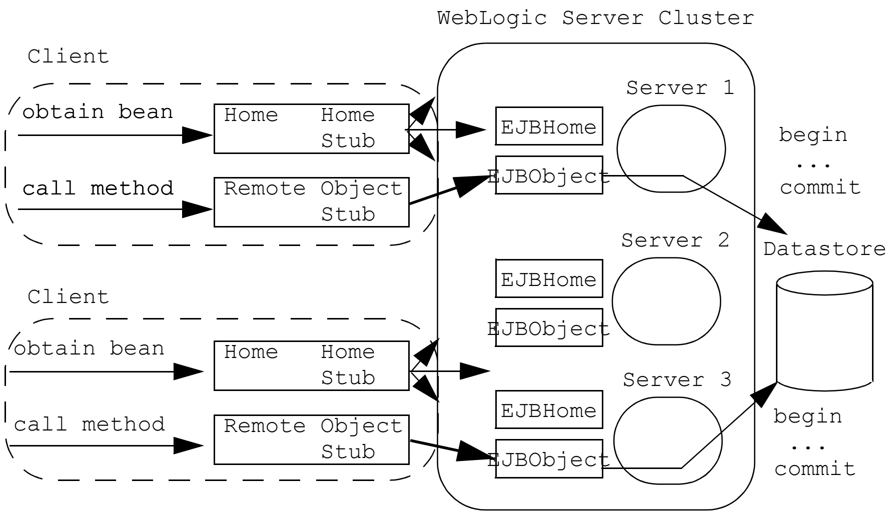
When a home finds or creates a read-write entity bean, it obtains an instance on the local server and returns an `EJBObject` stub pinned to that server. Load balancing and failover occur only at the home level. Because it is possible for multiple instances of the entity bean to exist in the cluster, each instance must read from the database before each transaction and write on each commit.

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

- Multiple clients can use the bean in transactions.
- `ejbLoad()` is always called at the beginning of each transaction when `cache-between-transactions` is set to `false`.
- `ejbStore()` behavior is governed by the rules described in "ejbLoad() and ejbStore() Behavior for Entity EJBs" on page 4-12.

Figure 4-11 shows read-write entity EJBs in a WebLogic Server clustered environment. The three arrows on Home Stub point to all three servers and show multiple client access.

Figure 4-11 Read-write entity EJBs in a clustered server environment



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

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

Cluster Address

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

Transaction Management

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

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

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

Transaction Management Responsibilities

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

- In **bean-managed transactions**, the EJB' code manages the transaction demarcation. If bean- or client-managed transactions are required, you must provide the java code and use the `javax.transaction.UserTransaction` interface. The EJB or client can then access a `UserTransaction` object through JNDI and specify transaction boundaries with explicit calls to `tx.begin()`, `tx.commit()`, `tx.rollback()`. See "Using javax.transaction.UserTransaction" on page 4-39 for more information on defining transaction boundaries.

- In **container-managed transactions**, the WebLogic Server EJB container manages the transaction demarcation. For EJBs that use container-managed transactions, you can use several deployment elements to control the transactional requirements for individual EJB methods. For more information about the deployment descriptors, see *Programming WebLogic EJB*.

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

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

Using javax.transaction.UserTransaction

To define transaction boundaries in EJB or client code, you must obtain a `UserTransaction` object and begin a transaction *before* you obtain a Java Transaction Service (JTS) or JDBC database connection. To obtain the `UserTransaction` object, use this command:

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

If you start a transaction after obtaining a database connection, the connection has no relationship to the new transaction, and there are no semantics to "enlist" the connection in a subsequent transaction context. If a JTS connection is not associated with a transaction context, it operates similarly to a standard JDBC connection that has autocommit equal to `true`, and updates are automatically committed to the datastore.

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

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

Restriction for Container-Managed EJBs

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

Transaction Isolation Levels

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

Setting Bean-Managed Transaction Isolation Levels

You set the isolation level for bean-managed transactions in the EJB's java code. When the application runs, the transaction is explicitly started. Allowable isolation levels are defined in "transaction-isolation" on page 9-94.

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 Figure 4-12 for a code sample.

Figure 4-12 Sample Code Setting Transaction Isolation Level

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

User Transaction tx = (UserTransaction)

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

//Begin user transaction

       tx.begin();

//Set transaction isolation level to TRANSACTION_READ_COMMITED
```

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

//perform transaction work

       tx.commit();
```

Setting Container-Managed Transaction Isolation Levels

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

Limitations of TransactionSerializable

Many datastores provide limited support for detecting serialization problems, even for a single user connection. Therefore, even if you set `transaction-level` to `TransactionSerializable` setting for an EJB, you may receive exceptions or rollbacks in the EJB client if contention occurs between clients for the same rows. To prevent these problems, make sure that the code in your client application catches and examines the SQL exceptions, and that you take the appropriate action to resolve the exceptions, such as restarting the transaction.

WebLogic Server provides special `isolation-level` settings designed to prevent this problem with Oracle databases, as described in "Special Note for Oracle Databases" on page 4-41.

For other database vendors, refer to your database documentation for more details about isolation level support.

Special Note for Oracle Databases

Even with an `isolation-level` setting of `TransactionSerializable`, Oracle does not detect serialization problems until commit time. The error message returned is:

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

WebLogic Server provides special `isolation-level` settings to prevent this. For more information, see "isolation-level" on page 9-111.

Distributing Transactions Across Multiple EJBs

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

Calling Multiple EJBs from a Single Transaction Context

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

Figure 4-13 Beginning and committing a transaction

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

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

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

Encapsulating a Multi-Operation Transaction

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

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

Distributing Transactions Across EJBs in a WebLogic Server Cluster

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

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

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

Database Insert Support

WebLogic Server allows you to control when and how the EJB container inserts newly created beans into the database.You specify your preference by setting the `delay-database-insert-until` deployment descriptor element in the `weblogic-cmp-rdbms-jar.xml` file. This element allows you to choose:

- To delay the database insert until after the EJB Container performs either an `ejbCreate` or `ejbPostCreate`, as described in the following section.
- To insert multiple entries into the database in one SQL statement, as described in "Bulk Insert" on page 4-45.

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

- `ejbCreate`—This method performs a database insert immediately after `ejbCreate.`
- `ejbPostCreate`—This method performs an insert immediately after `ejbPostCreate` (default).
- `commit`—performs a bulk insert when the transaction commits

Figure 4-14 Sample xml specifying delay-database-insert-until

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

Delay-Database-Insert-Until

By default, the database insert occurs after the client calls the `ejbPostCreate` method. The EJB container delays inserting the new bean when you specify either the `ejbCreate` or `ejbPostCreate` options for the `delay-database-insert-until` element in the `weblogic-cmp-rdbms-jar.xml` file. Setting either of these options specifies the precise time at which the EJB Container inserts a new bean that uses RDBMS CMP into the database.

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

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

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

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

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.

Bulk Insert

Bulk insert support increases the performance of container-managed persistence (CMP) bean creation by enabling the EJB container to perform multiple database inserts for CMP beans in one SQL statement. This feature allows the container to avoid making multiple database inserts.

The EJB container performs bulk database inserts when you specify the `commit` option for the `delay-database-insert-until` element in the `weblogic-cmp-rdbms-jar.xml` file.

When using bulk insert, you must set the boundary for the transaction as bulk insert only applies to the inserts between `transaction begin` and `transaction commit`.

Note: Bulk insert only works with drivers that support the `addBatch()` and `executeBatch()` methods. For example, the Oracle thin driver supports these methods but the WebLogic Oracle JDBC driver does not.

The two limitations on using bulk insert are:

The total number of entries you create in a single bulk insert cannot exceed the `max-beans-in-cache` setting, which is specified in the `weblogic-ejb-jar.xml` file. See "max-beans-in-cache" on page 9-58 for more information on this element.

If you set the `dbms-column-type` element in the `weblogic-cmp-rdbms-jar.xml` file to either `OracleBlob` or `OracleClob`, bulk insert 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.

Resource Factories

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

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

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

Note: WebLogic Server also supports JMS connection factories.

Setting Up JDBC Data Source Factories

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

With a non-transactional data source, the JDBC connection operates in auto commit mode, committing each insert and update operation to the database immediately, rather than as part of a container-managed transaction.

With a transactional data source, multiple insert and update operations in a method can be submitted as a single, container-managed transaction that either commits or rolls back as a logical unit.

Note: Entity beans that use container-managed persistence should always use a transactional data source, rather than a non-transactional data source, to preserve data consistency.

To create a JDBC data source factory:

1. Set up a JDBC connection pool in the Administration Console. See Managing JDBC Connectivity in the *Administration Guide* for more information.
2. Start WebLogic Server.
3. Start WebLogic Server Administration Console.
4. In the left pane of the Console, click the Services node and expand JDBC.
5. Select JDBC Data Source Factory and click the Configure a New JDBC Data Source Factory option in the right pane.
6. Enter values in the Name, User Name, URL, Driver Class Name, and Factory Name, attribute fields.
7. Enter any connection properties in the Properties attribute field.

   a. *For non-transactional JDBC datasources*, enter:

   ```
   weblogic.jdbc.DataSource.jndi_name=pool_name
   ```

   where `jndi_name` is the full WebLogic Server JNDI name to bind to the datasource and `pool_name` is the name of the WebLogic Server connection pool you created in step 1.

   b. *For transactional JDBC datasources*, select Tx Data Sources from the left pane of the Administration Console, click Configure a New JDBC Tx Data Source in the right pane, and enter:

   ```
   weblogic.jdbc.TXDataSource.jndi_name=pool_name
   ```

where *jndi_name* is the full WebLogic Server JNDI name to bind to the transactional datasource and *pool_name* is the name of the WebLogic Server connection pool you created in step 1.

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

8. Click Create to create the JDBC Data Source Factory. The new Data Source Factory is added under the JDBC Data source Node in the left pane.

9. Click Apply to save the changes.

10. Bind the JNDI name of the datasource to the EJB's local JNDI environment by doing one of the following:

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

Setting Up URL Connection Factories

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

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

    - `WebServer`
    - `VirtualHost:`

2. Set the `URLResource` attribute for the `WebServer` element using the following syntax:

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

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

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

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

CHAPTER

5 WebLogic Server Container-Managed Persistence Service

The following sections describe the container-managed persistence (CMP) features supported by the WebLogic Server EJB container.

- Overview of Container Managed Persistence Service
- Writing for RDBMS Persistence for EJB 1.1 CMP
- Using WebLogic Query Language (WLQL) for EJB 1.1 CMP
- Using SQL for CMP 1.1 Finder Queries
- Using EJB QL for EJB 2.0
- Using Dynamic Queries
- Using Oracle SELECT HINTS
- "get" and "set" Method Restrictions
- BLOB and CLOB DBMS Column Support for the Oracle DBMS
- Tuned EJB 1.1 CMP Updates in WebLogic Server
- Tuned EJB 1.1 CMP Updates in WebLogic Server
- Flushing the CMP Cache
- Using Primary Keys

- Automatic Primary Key Generation for EJB 2.0 CMP
- Multiple Table Mapping for EJB 2.0 CMP
- Automatic Table Creation
- Container-Managed Relationships
- Groups
- Using EJB Links
- Java Data Types for CMP Fields

Overview of Container Managed Persistence Service

The EJB container provides a uniform interface between the EJB and WebLogic Server. The container creates new instances of the EJBs, manages these bean resources, and provides persistent services such as, transactions, security, concurrency, and naming at runtime. In most cases, EJBs from earlier version of WebLogic Server run in the container. However, see the Migration Guide for information on when you would need to migrate your bean code. See "DDConverter" on page 8-23 for instructions on using the conversion tool.

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

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

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

EJB Persistence Services

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

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

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

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

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

Using WebLogic Server RDBMS Persistence

To use WebLogic Server RDBMS-based persistence service with your EJBs:

1. Create a dedicated XML deployment file.
2. Define the persistence elements for each EJB that will use container-managed persistence.
3. For instructions on creating deployment descriptor files, see "Specifying and Editing the EJB Deployment Descriptors" on page 6-5.

If you use WebLogic Server's utility, DDConverter to create this file, it is named `weblogic-cmp-rdbms-jar.xml`. If you create the file from scratch, you can save it to a different filename. However, you must ensure that the `persistence-type` and `persistence-use` elements in `weblogic-ejb-jar.xml` refer to the correct file.

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

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

- EJB connection pools or data source for EJB 2.0 CMP
- EJB field to database element mappings
- Query Language
  - WebLogic Query Language (WLQL) for EJB 1.1 CMP
  - WebLogic EJB-QL with WebLogic QL extension for EJB 2.0 CMP (optional)
- Finder method definitions (CMP 1.1)
- Foreign key mappings for relationships
- WebLogic Server-specific deployment descriptors for queries

Writing for RDBMS Persistence for EJB 1.1 CMP

Clients use finder methods to query and receive references to entity beans that fulfill query conditions. This section describes how to write finders for WebLogic-specific 1.1 EJBs that use RDBMS persistence. You use EJB QL, a portable query language, to define finder queries for 2.0 EJBs with container-managed persistence. For more information about on EJB QL, see "Using EJB QL for EJB 2.0" on page 5-11.

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.

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

The key components of a finder for RDBMS persistence are:

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

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

Finder Signature

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

Note: 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, `ejbc` generates an error message when you compile the deployment unit.

finder-query Element

The `finder-query` element defines the WebLogic Query Language (WLQL) expression 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 5-7 for more information on WLQL.

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

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

Using WebLogic Query Language (WLQL) for EJB 1.1 CMP

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

Note: For queries to 2.0 EJBs, see "Using EJB QL for EJB 2.0" on page 5-11. Using the weblogic-ql query completely overrides the ejb-ql query.

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)`

Operator	Description	Sample Syntax
`!`	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-rdbms-jar.xml` file

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

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

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

Examples of WLQL Expressions

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

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

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

  The sample `<finder>` stanza is:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Using 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 5-7.

To specify this SQL finder query:

1. In the `weblogic-cmp-rdbms-jar.xml` file write a SQL query using the `finder-sql` element in the `weblogic-cmp-rdbms-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 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.

Note: You can use any amount of vendor-specific SQL in the SQL query.

Using EJB QL for EJB 2.0

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

EJB QL Requirement for EJB 2.0 Beans

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

Migrating from WLQL to EJB QL

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

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

Using EJB 2.0 WebLogic QL Extension for EJB QL

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

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

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.

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';
```

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';
```

Note: The `upper` and `lower` extensions were added in WebLogic Server 7.0 SP03.

Using SELECT DISTINCT

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

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

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

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.

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

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

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

Also, you can specify an ORDERBY with ascending [ASC] or descending [desc] order for multiple fields as follows:.

Figure 5-2 WebLogic QL 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
```

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 EJB 2.0 WebLogic QL Extension for EJB QL" on page 5-13. 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 5-20.

Beans with Simple Primary Key

WebLogic Server supports a return type consisting of a `cmp-bean` with a simple primary key.

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 result in a failure when you compile the query.

An example of a subquery that returns the value(s) of the type bean with a simple primary key is as follows:

```
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.

Subqueries as Comparison Operands

Use subqueries as the operands of comparison operators. WebLogic QL supports subqueries as the operands of the following *Comparison Operators*: *[NOT]IN, [NOT]EXISTS*, and the following *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 5-19 for more information. This query returns all customers that have not placed an order.

```
SELECT (cust) FROM CustomerBean AS cust
       WHERE cust.num NOT IN
              (SELECT order.cust_num 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.quantity Ordered 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.

All of the arithmetic operators, <, >, <= >=, =, <> are use, as in the above examples.

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 DISTICNT clause in a main query; whereas the DISTICT clause in the subquery is enforced by the generated SQL, SELECT DISTINCT. An example of a DISTINCT clause in a 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 selects the last 10 shipped Orders.

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 EJB 2.0 WebLogic QL Extension for EJB QL" on page 5-13. 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.
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 EJB 2.0 WebLogic QL Extension for EJB QL" on page 5-13. Use those instructions with a query specifying a ResultSet as shown in the above query to specify WebLogic QL, see "Using EJB 2.0 WebLogic QL Extension for EJB QL" on page 5-13. 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:

CMP fields showing salaries of employees in California

Name	Location	Salary
Matt	CA	110,000
Rob	CA	100,000

CMP fields showing salaries of employees in Arizona

Name	Location	Salary
Dan	AZ	120,000
Dave	AZ	80,000

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 AVEAGE 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.

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 Figure 5-3, which sets field group settings:

Figure 5-3 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);
```

Using Dynamic Queries

Dynamic queries allow you to construct and execute EJB-QL queries programmatically in your application code. Queries are expressions that allow you to request information of EJB objects from the RDBMS. This feature is only available for use with EJB 2.0 CMP beans. Using dynamic queries provides the following benefits:

- Allows you to create and execute new queries without having to update and deploy an EJB.
- Allows you 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. Specify the `enable-dynamic-queries` element in the EJB's `weblogic-ejb-jar.xml` deployment descriptor file as follows:

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

2. For instructions on how to add or edit the `enable-dynamic-queries` element, see "Specifying and Editing the EJB Deployment Descriptors" on page 6-5.
3. 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;
Sring ejbql="SELECT OBJECT(e)FROM EmployeeBean e WHERE
e.name='rob'"
Query query=qh.createQuery();
```

```
query.setMaxElements(10)
Collection results=query.find(ejbql);
```

Using Oracle SELECT HINTS

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

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

The WebLogic QL keyword and usage is as follows:

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

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

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

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

"get" and "set" Method Restrictions

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

BLOB and CLOB DBMS Column Support for the Oracle DBMS

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

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

To enable BLOB/CLOB support:

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

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

Specifying a BLOB Using the Deployment Descriptor

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

Figure 5-4 Specifying a BLOB object

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

Controlling Serialization of cmp-fields Mapped to OracleBlobs

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

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

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

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

Specifying a CLOB Using the Deployment Descriptors

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

Figure 5-5 Specifying a CLOB object

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

```
</field-map>
```

Tuned EJB 1.1 CMP Updates in WebLogic Server

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

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

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

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

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

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

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

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

Optimized Database Updates for CMP 2.0 Entity Beans

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 transactions.

Flushing the CMP Cache

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

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

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

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

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

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

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

Using Primary Keys

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

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

Primary Key Mapped to a Single CMP Field

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

Primary Key Class That Wraps Single or Multiple CMP Fields

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

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-rdbms-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.

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).
```

Hints for Using Primary Keys

Some hints for using primary keys with WebLogic Server include:

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

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

- Do not construct a new primary key class with an `ejbCreate`. Instead, allow the container to create the primary key class internally.
- Set the values of the primary key cmp-fields using the `setXXX` methods within the `ejbCreate` method.
- Do not use a cmp field of the type `BigDecimal` as a primary key field for CMP beans. The `boolean BigDecimal.equals (object x)` method considers two `BigDecimal` equal only if they are equal in value and scale. This is because there are differences in precision between the Java language and different databases. For example, the method does not consider `7.1` and `7.10` to be equal. Consequently, this method will most likely return false or cause the CMP bean to fail.

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

  a. Implement a primary key class.

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

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

Mapping to a Database Column

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

Automatic Primary Key Generation for EJB 2.0 CMP

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

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

Generated key support is provided in two ways:

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

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

  For more information on this feature, see "Specifying Primary Key Support for Oracle" on page 5-35 and "Specifying Primary Key Support for Microsoft SQL Server" on page 5-36.

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

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

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

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

In the `weblogic-cmp-rdbms-jar.xml` file, set the `key_cache_size` element to specify how many primary key values a database SELECT and UPDATE will fetch at one time. The default value of `key_cache_size` is `1`. BEA recommends that you set this element to a value of `>1`, to minimize database accesses and to improve performance. For more information in this feature, see "Specifying Primary Key Named Sequence Table Support" on page 5-36.

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

Valid Key Field Types

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

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

Specifying Primary Key Support for Oracle

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

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

Figure 5-7 Specifying automatic key generation for Oracle

```
<automatic-key-generation>
       <generator-type>ORACLE</generator-type>
       <generator_name>test_sequence</generator-name>
       <key-cache-size>10</key-cache-size>
</automatic-key-generation>
```

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

Warning: Do not use the generator type USER_DESIGNATED_TABLE with Oracle, as doing so 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
NAMED_SEQUENCE_TABLE, but encountered SQLException
java.sql.SQLException: ORA-08177: can't serialize access
for this transaction.
```

USER_DESIGNATED_TABLE mode sets the TX ISOLATION LEVEL to SERIALIZABLE which can cause problems with Oracle.

Instead, use the AutoKey option ORACLE.

Specifying Primary Key Support for Microsoft SQL Server

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

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

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

Figure 5-8 Specifying automatic key generation for Microsoft SQL

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

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

Specifying Primary Key Named Sequence Table Support

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

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

To use Named Sequence Table support, make sure that the underlying database supports the transaction isolation level, `TRANSACTION_SERIALIZABLE`. You specify this option for the `isolation-level` element, in the `weblogic-ejb.xml` file. The `TRANSACTION_SERIALIZABLE` option specifies that simultaneously executing a transaction multiple times has the same effect as executing the transaction multiple times in a serial fashion. If the database doesn't support the transaction isolation level, `TRANSACTION_SERIALIZABLE`, then you cannot use Named Sequence Table support.

Note: See the documentation for the underlying database to determine the type of isolation level support it provides and see "Specifying and Editing the EJB Deployment Descriptors" on page 6-5 for instructions on setting the isolation level.

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

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

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

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

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

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

Multiple Table Mapping for EJB 2.0 CMP

Multiple table mapping allows you to map a single EJB to multiple DBMS tables within a single database for EJB 2.0 CMP beans. You configure this feature by mapping multiple DBMS tables and columns to the EJB and its fields in the EJB's `weblogic-cmp-rdbms-xml` file. This includes the following types of mappings:

- EJB container-managed persistence (cmp) fields - These fields describe which of the EJB's cmp-fields are mapped to which DBMS tables.
- EJB container-managed relationship (cmr) fields - These fields describes which of the EJBs DBMS tables contain the foreign key columns required for mapping the relationships in the DBMS.

When enabling multiple table mappings, the following requirement applies:

- If the EJB is a participant in a container-managed relationship and the relationship requires that the DBMS tables maintain foreign keys, then those foreign keys will reside on only one of the EJB's multiple tables.

Previously, you could associate an EJB with a single table and a list of fields and columns. Now, you can associate sets of fields and columns for as many tables as the EJB maps to.

Note this restriction for multiple mapped tables on a single bean:

Tables that are mapped to a single entity bean must not have referential integrity constraints declared between their primary keys. Doing so may result in a runtime error upon bean removal.

Multiple Table Mappings for cmp-fields

Configure multiple table mappings for `cmp-fields`, in a `weblogic-rdbms-bean` stanza of the EJB's `weblogic-cmp-rdbms-xml` file, as follows:

1. Specify the following elements in the `weblogic-cmp-rdbms-jar.xml` file:
    - `table-field-map` element
    - `table-name` element

- `field-map` element

2. For instructions on specifying deployment descriptors, see "Specifying and Editing the EJB Deployment Descriptors" on page 6-5.

The following sample XML shows an EJB that maps to a single DBMS table:

Figure 5-10 Mapping a single DBMS table

```
<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>
```

The following sample XML shows an EJB that maps to two different tables:

Figure 5-11 Mapping 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>
```

Note: As shown in the above XML sample for a table mapping, you must map the primary key field to each table's primary key column.

For information about specifying CMRs when one of the beans in the relationship maps to multiple tables, see "Specifying CMRs for EJBs that Map to Multiple Tables" on page 5-50.

Automatic Table Creation

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

If you enable automatic table creation, WebLogic Serve examines the value of the `database-type` element in `weblogic-cmp-rdbms-jar.xml` to determine the correct syntax and datatype conversions to use to create a table in your database. WebLogic Server version 7.0 uses the vendor-specific `CREATE TABLE` syntax and datatype conversions for the following databases and vendors:

- Informix
- Oracle
- PointBase
- SQL Server
- Sybase

For all other database systems, WebLogic Server makes a best attempt to create the new table using a basic syntax and the datatype conversions shown in the following table:

Table 5-1 Generic Java Field to DBMS Column Type Conversion

Java Type	DBMS Column Type
boolean	INTEGER
byte	INTEGER
char	CHAR
double	DOUBLE PRECISION
float	FLOAT
int	INTEGER
long	INTEGER
short	INTEGER
java.lang.string	VARCHAR (150)
java.lang.BigDecimal	DECIMAL (38, 19)
java.lang.Boolean	INTEGER
java.lang.Byte	INTEGER
java.lang.Character	CHAR (1)
java.lang.Double	DOUBLE PRECISION
java.lang.Float	FLOAT
java.lang.Integer	INTEGER
java.lang.Long	INTEGER
java.lang.Short	INTEGER
java.sql.Date	DATE
java.sql.Time	DATE

Java Type	DBMS Column Type
java.sql.Timestamp	DATETIME
byte[ ]	RAW (1000)
Any serializable Class that is not a valid SQL type:	RAW (1000)

If, based on the descriptions in the deployment files, a field cannot be successfully mapped to an appropriate column type in the database, the `CREATE TABLE` fails, an error is thrown, and you must create the table yourself.

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

To define automatic table creation:

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

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

2. Specify the correct database system or database vendor name in the `database-type` element of `weblogic-cmp-rdbms-jar.xml`. `CREATE TABLE` syntax and datatype mapping is provide for the following `database-type` values: `INFORMIX`, `ORACLE`, `POINTBASE`, `SQL_SERVER`, and `SYBASE`. All other DBMS systems use a basic syntax and the datatype conversions shown in the table above.

Container-Managed Relationships

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.

Understanding CMRs

This section describes the features and limitations of WebLogic Server CMRs. For instruction on configuring CMRs, see "Defining Container-Managed Relationships" on page 5-45.

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 Server 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 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.

Relationship Cardinality

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

Relationship 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 Relationships

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

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

- "Specifying Relationship in ejb-jar.xml" on page 5-45
- "Specifying Relationships in weblogic-cmp-jar.xml" on page 5-47

Specifying Relationship in ejb-jar.xml

Container-managed relationships are defined in the `ejb-relation` stanza of `ejb-jar.xml`. Figure 5-12 shows the `ejb-relation` stanza for a relationship between two entity EJBs: `TeacherEJB` and `StudentEJB`.

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

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

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

```
            <cmr-field-type>java.util.Collection
         <cmr-field>
      </ejb-relationship-role>
```

Specifying Relationship Cardinality

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

In Figure 5-12, the cardinality of the `TeacherEJB-StudentEJB` relationship is one-to-many—it is specified by setting `multiplicity` to `one` on the `TeacherEJB` side and `Many` on the `StudentEJB` side.

The cardinality of the CMR in Figure 5-13, is one-to-one—the `multiplicity` is set to `one` in both role stanza for the relationship

Figure 5-13 One-to-One, Unidirectional CMR in ejb-jar.xml

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

If a side of a relationship of a relationship has a `<multiplicity>` of `Many`, its `<cmr-field>` is a collection, and you must specify its `<cmr-field-type>` as `java.util.Collection`, as shown in the `StudentEJB` side of the relationship in Figure 5-12. It is not necessary to specify the `cmr-field-type` when the `cmr-field` is a single valued object.

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

Table 5-2 Cardinality and cmr-field-type

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

Specifying Relationship Directionality

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

A bidirectional CMR has a `cmr-field` element in the `ejb-relationship-role` stanza for both sides of the relationship, as shown in Figure 5-12.

A unidirectional relationship has a `cmr-field` in only one of the role stanzas for the relationship. The `ejb-relationship-role` for the starting EJB contains a `cmr-field`, but the role stanza for the target bean does not. Figure 5-13 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 5-14 is the `weblogic-rdbms-relation` stanza for a the one-to-one relationship between `MentorEJB` and `StudentEJB`, whose `<ejb-relation>` is shown in Figure 5-13.

Figure 5-14 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.

Figure 5-15 is the `weblogic-rdbms-relation` stanza for a the one-to-many relationship between `TeacherEJB` and `StudentEJB`, whose `<ejb-relation>` is shown in Figure 5-12.

Figure 5-15 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.

Figure 5-16 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 5-50.

Figure 5-16 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.

Figure 5-17 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.

Figure 5-17 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>
```

Using Relationship Caching for CMRs

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 example, given 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, a SQL query is issued by the first line of code to load `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 the `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, its a good idea to let the finder method know via the relationship caching feature.

Specify relationship caching using these stanzas in `weblogic-cmp-jar.xml`

- `relationship-caching`—define the `caching-name` and the `caching-element` for the related bean, as shown in Figure 5-18.

- `caching-name` element in the `weblogic-query`—if specified, when the finder query is executed, WebLogic Server loads the related beans into cache. For an example, see Figure 5-19.
- `database-type` in `weblogic-rdbms-jar`, the root of `weblogic-cmp-jar.xml` —relationship caching uses outer joins for queries, the syntax for which can vary from database to database.

Note: Make sure that the `finders-load-bean` element, specified in the `weblogic-ejb-jar.xml` file, in the bean that specifies an relationship (for example, `customerBean` in the above sample XML code) is not set to `False` or relationship caching will not be enabled. The `finder-load-bean` element's default is `True`.

Figure 5-18 relationship-caching in weblogic-cmp-jar.xml

```
<relationship-caching>
  <caching-name>cacheMoreBeans</caching-name>
  <caching-element>
    <cmr-field>accounts</cmr-field>
  </caching-element>
</relationship-caching>
```

Figure 5-19 weblogic-query in weblogic-cmp-jar.xml

```
<weblogic-query>
  <query-method>
    <method-name>findBigAccounts</method-name>
      <method-params>
        <method-param>java.lang.String</method-param>
        <method-param>java.lang.Integer</method-param>
     </method-params>
  </query-method>
  <caching-name>cacheMoreBeans</caching-name>
</weblogic-query>
```

Nested caching-elements

Using nested caching-elements enables the bean to load more than one level of related beans. 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.

Since relationship caching uses join queries, and a join query might duplicate results for a table in the ResultSet, the number of `caching-element` elements specified will have a direct impact on the number of duplicate results in the ResultSet. For one-to-many relationships, do not specify too many `caching-element` deployment descriptors in the `relationship-caching` element because the number of duplicate results might multiply for each `caching-element` deployment descriptor.

Relationship Caching Limitations

The relationship caching feature has the following limitations:

1. Relationship caching only works with `one-to-one` and `one-to-many` relationships.
2. When using `weblogic-ql`, this feature only works with finder methods that return references to either `EJBObject` or `EJBLocalObject` beans.
3. 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 because there is no way to identify the duplicate in the ResultSet is the result of the original data or the result of the outer join.

Cascade Delete

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

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

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

Cascade Delete Method

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

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

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

Figure 5-20 Specifying a cascade delete

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

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

Database Cascade Delete Method

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

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

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

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

Figure 5-21 Oracle table setup for cascade delete

```
CREATE TABLE dept

  (deptno   NUMBER(2) CONSTRAINT pk_dept PRIMARY KEY,

   dname    VARCHAR2(9) );

CREATE TABLE emp

  (empno    NUMBER(4) PRIMARY KEY,

   ename    VARCHAR2(10),

   deptno   NUMBER(2)  CONSTRAINT fk_deptno

            REFERENCES dept(deptno)

            ON DELETE CASCADE );
```

CMRs and Local Interfaces

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

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

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

The EJB container makes the local home interface accessible to local clients through JNDI. To reference a local interface you need to have a local JNDI name. The objects that implement the entity beans' local home interface are called `EJBLocalHome` objects. You can specify either a `jndi-name` or `local-jndi-name` in the `weblogic-ejb-jar.xml` file. For more information on how to specify deployment descriptors, see "Specifying and Editing the EJB Deployment Descriptors" on page 6-5

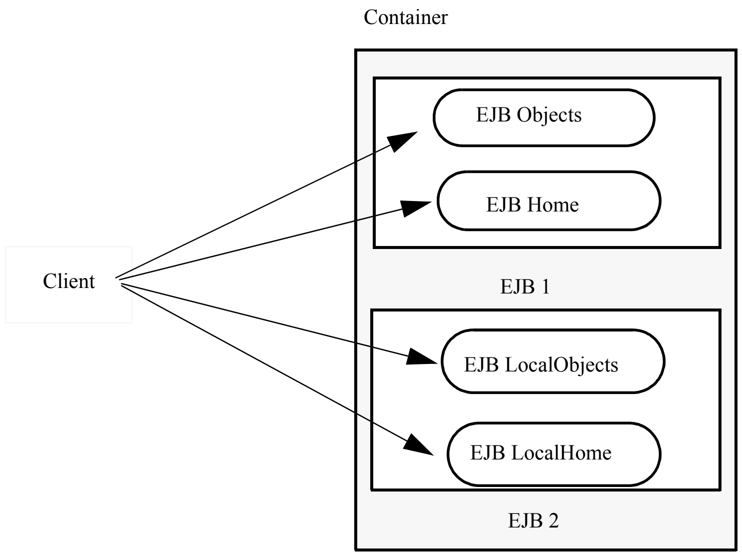
In earlier versions of WebLogic Server, `ejbSelect` methods were used to return remote interfaces. Now you can specify a `result-type-mapping` element in the `ejb-jar.xml` file that indicates whether the result returned by the query will be mapped to a local or remote object.

Using the Local Client

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

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

Figure 5-22 Local client and local interfaces



WebLogic Server provides support for both local and uni-directional remote relationships between EJBs. If the EJBs are on the same server and are part of the same JAR file, they can have local relationships. If the EJBs are not on the same server, the relationships must be remote. For a relationship between local beans, multiple column mappings are specified if the key implementing the relation is a compound key. For a remote bean, only a single column-map is specified, since the primary key of the remote bean is opaque. No column-maps are specified if the role just specifies a group-name. No group-name is specified if the relationship is remote.

Changes to the Container for Local Interfaces

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

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

Groups

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

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

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

Specifying Field Groups

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

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

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

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.

To create an `ejb-link`:

1. Specify the link to the EJB using the optional `ejb-link` deployment descriptor element of the `ejb-ref` element of the referencing application component.

   The value of the `ejb-link` element must be the `ejb-name` 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.

2. 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.

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

Java Data Types for CMP Fields

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

Table 5-3 Java data types for CMP fields

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

Java Types for CMP Fields	Oracle Data Types
serializable	**RAW, LONG RAW**

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

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

CHAPTER

6 Packaging EJBs for the WebLogic Server Container

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

- Required Steps for Packaging EJBs
- Reviewing the EJB Source File Components
- WebLogic Server EJB Deployment Files
- Specifying and Editing the EJB Deployment Descriptors
- Creating the Deployment Files
- Referencing Other EJBs and Resources
- Packaging EJBs into a Deployment Directory
- Compiling EJB Classes and Generating EJB Container Classes
- Loading EJB Classes into WebLogic Server
- Specifying an ejb-client.jar
- Manifest Class-Path

Required Steps for Packaging EJBs

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

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

Reviewing the EJB Source File Components

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

Component	**Description**
Bean Class	The *bean class* implements the bean's business and life cycle methods.
Remote Interface	The *remote interface* defines the beans's business methods that can be accessed from applications outside of the bean's EJB container.
Remote Home Interface	The *remote home interface* defines the bean's life cycle methods that can be accessed from applications outside of the bean's EJB container.

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

WebLogic Server EJB Deployment Files

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

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

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

ejb-jar.xml

The `ejb-jar.xml` file contains the Sun Microsystem-specific EJB DTD. The deployment descriptors in this file describe the enterprise bean's structure and declares its internal dependences and the application assembly information, which describes how the enterprise bean in the `ejb-jar` file is assembled into an application deployment unit. For a description of the elements in this file, see the JavaSoft specification.

weblogic-ejb-jar.xml

The `weblogic-ejb-jar.xml` file contains the WebLogic Server-specific EJB DTD that defines the concurrency, caching, clustering, and behavior of EJBs. It also contains descriptors that map available WebLogic Server resources to EJBs. WebLogic Server resources include security role names and data sources such as JDBC pools, JMS connection factories, and other deployed EJBs. For a description of the elements in this file, see Chapter 9, "weblogic-ejb-jar.xml Document Type Definitions."

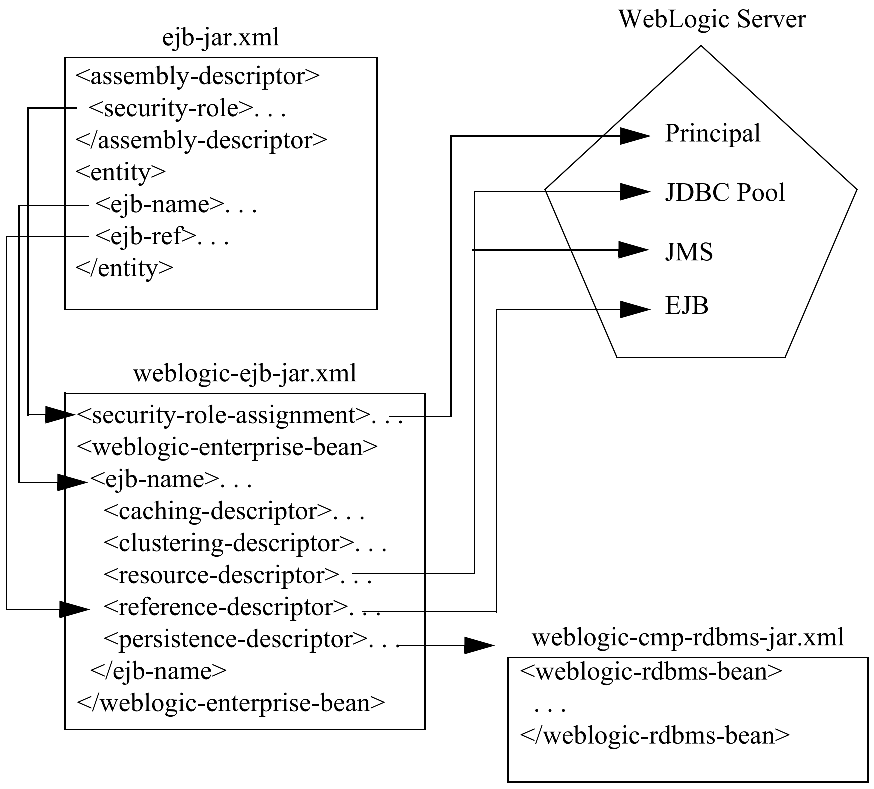
weblogic-cmp-rdbms.xml

The `weblogic-cmp-rdbms.xml` file contains the WebLogic Server-specific EJB DTD that defines container-managed persistence services. Use this file to specify how the container handles synchronizing the entity beans's instance fields with the data in the database. For a description of the elements in this file, see Chapter 10, "weblogic-cmp-rdbms- jar.xml Document Type Definitions."

Relationships Among the Deployment Files

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

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



Specifying and Editing the EJB Deployment Descriptors

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

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

- Using the EJB Deployment Descriptor Editor in the WebLogic Server Administration Console to edit the bean's deployment files. For instructions on using the EJB Deployment Descriptor Editor, see "Using the EJB Deployment Descriptor Editor" on page 6-7.
- Using a WebLogic Server command line utility tool called `DDConverter` to convert EJB 1.1 deployment descriptors to EJB 2.0 XML. For instructions on using the `DDConverter` tool, see "DDConverter" on page 8-23.

Creating the Deployment Files

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

```
SAMPLES_HOME\server\config\examples\applications
```

Manually Editing EJB Deployment Descriptors

To edit XML deployment descriptor elements manually:

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

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

Using the EJB Deployment Descriptor Editor

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

1. Start WebLogic Server.
2. Start the Administration Console and select the `Deployments` node from the right pane and then click `EJB`.
3. From the expanded list of deployed EJBs, right-click the bean you want to edit and choose `Edit EJB Descriptor...`
4. When the EJB Deployment Descriptor Editor displays, click either the Persist or Validate button.
   - Choose Persist to save your changes to the EJB's deployment descriptor file.
   - Choose Validate to specify that WebLogic Server check the structure of the EJB's deployment descriptor files to verify that the files XML can be parsed correctly.
5. In the left pane, click the EJB to expand the node.

   You should see the following nodes that represent the EJB's three deployment descriptor files:
   - **EJB Jar:** represents the `ejb-jar.xml` file deployment descriptors for this EJB.
   - **WebLogic EJB Jar:** represents the `weblogic-ejb-jar.xml` file deployment descriptors for this EJB.
   - **CMP:** represents the `weblogic-cmp-rdbms-jar.xml` file deployment descriptors for this EJB.
6. Expand the node for the deployment descriptors that you want to edit.

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

You can change the settings in the dialog window to edit those deployment descriptors.

8. Clicking on the folders displays tables in the right pane where you can view your settings.

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

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

Note: For more information on the EJB deployment descriptors, see either the online help in the Administration Console or Chapter 9, "weblogic-ejb-jar.xml Document Type Definitions," and Chapter 10, "weblogic-cmp-rdbms- jar.xml Document Type Definitions."

Referencing Other EJBs and Resources

An EJB can look up and use other EJBs deployed in WebLogic Server by specifying an EJB reference in the deployment descriptor. The requirements for creating an EJB reference differ depending on whether the referenced EJB is external to the calling EJB (deployed independently of the calling EJB's application EAR file) or deployed as part of the same application EAR file.

Referencing External EJBs

To reference an external EJB, you add a `<reference-descriptor>` stanza to the calling EJB's `weblogic-ejb-jar.xml` file. The following XML code shows a sample stanza that references an external EJB:

Figure 6-2 Sample XML code referencing an external EJB

```
<reference-descriptor>
   <ejb-reference-description>
      <ejb-ref-name>AdminBean</ejb-ref-name>
      <jndi-name>payroll.AdminBean</jndi-name>
   </ejb-reference-description>
76</reference-descriptor>
```

In the sample stanza, the `ejb-ref-name` element specifies the name that the calling EJB uses to look up the external EJB. The `jndi-name` element specifies the global JNDI name to use when looking up the specified `ejb-ref-name`.

Referencing Application-Scoped EJBs

When you deploy multiple EJBs as part of the same EAR file, WebLogic Server adds the EJB names to the application's local JNDI tree. EJBs and other components of the application can look up other application-scoped components directly in the JNDI tree relative to `java:comp/env`.

An EJB that references other EJBs deployed as part of the same EAR file does not need to specify a global JNDI name in the `weblogic-ejb-jar.xml` file. In fact, you can omit the `weblogic-ejb-jar.xml` file entirely if you do not need other WebLogic-specific features of the deployment descriptor.

To reference an EJB deployed as part of the same EAR file, add an `<ejb-local-ref>` stanza to the calling EJB's `ejb-jar.xml` deployment descriptor file. For example:

Figure 6-3 Sample XML code referencing an EJB in the same EAR file

```
<ejb-local-ref>
   <description>Reference to application EJB</description>
   <ejb-ref-name>ejb1</ejb-ref-name>
   <ejb-ref-type>Session</ejb-ref-type>
   <local-home>mypackage.ejb1.MyHome</home>
   <local>mypackage.ejb1.MyRemote</local>
   <ejb-link>ejb1.jar#myejb</ejb-link>
</ejb-local-ref>
```

In this example, the `ejb-ref-name` element indicates the name the calling EJB uses to look up the application-scoped EJB. The `ejb-link` element maps the indicated `<ejb-ref-name>` to the other EJB deployed in the EAR file. Note that this example qualifies the `<ejb-link>` name with the filename that stores the second EJB. Qualifying the EJB name in this manner is necessary when two or more EJBs in the EAR file use the same name; the filename qualifier ensures a unique reference.

For more information about EJB links, see "Using EJB Links" on page 5-59.

Referencing Application-Scoped JDBC DataSources

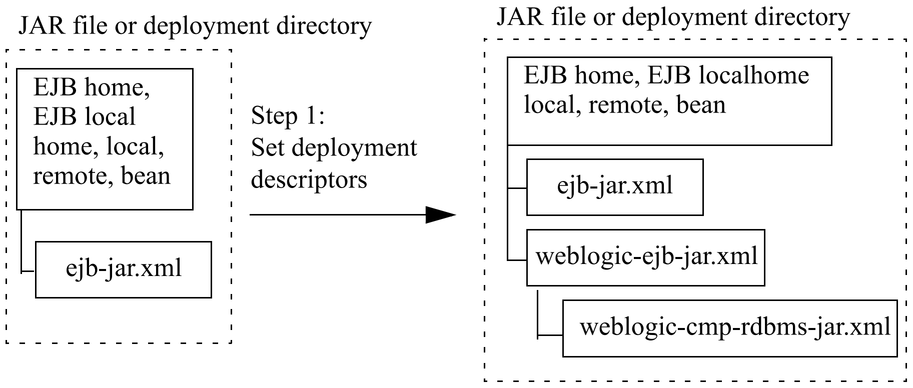
EJBs can also access JDBC DataSources that are deployed as part of the same EAR file. DataSources that are identified in the `weblogic-application.xml` deployment descriptor can be accessed locally from `java:comp/env` (without referencing the DataSource's global JNDI name). See Configuring Application-Scoped Resources in *Configuring Web Applications* for more information.

Packaging EJBs into a Deployment Directory

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

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

Figure 6-4 Packaging the EJB classes and deployment descriptors into a deployment directory


JAR file or deployment directory
EJB home, EJB local home, local, remote, bean
ejb-jar.xml
Step 1: Set deployment descriptors
JAR file or deployment directory
EJB home, EJB localhome local, remote, bean
ejb-jar.xml
weblogic-ejb-jar.xml
weblogic-cmp-rdbms-jar.xml

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

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

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

ejb.jar file

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

Compiling EJB Classes and Generating EJB Container Classes

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

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

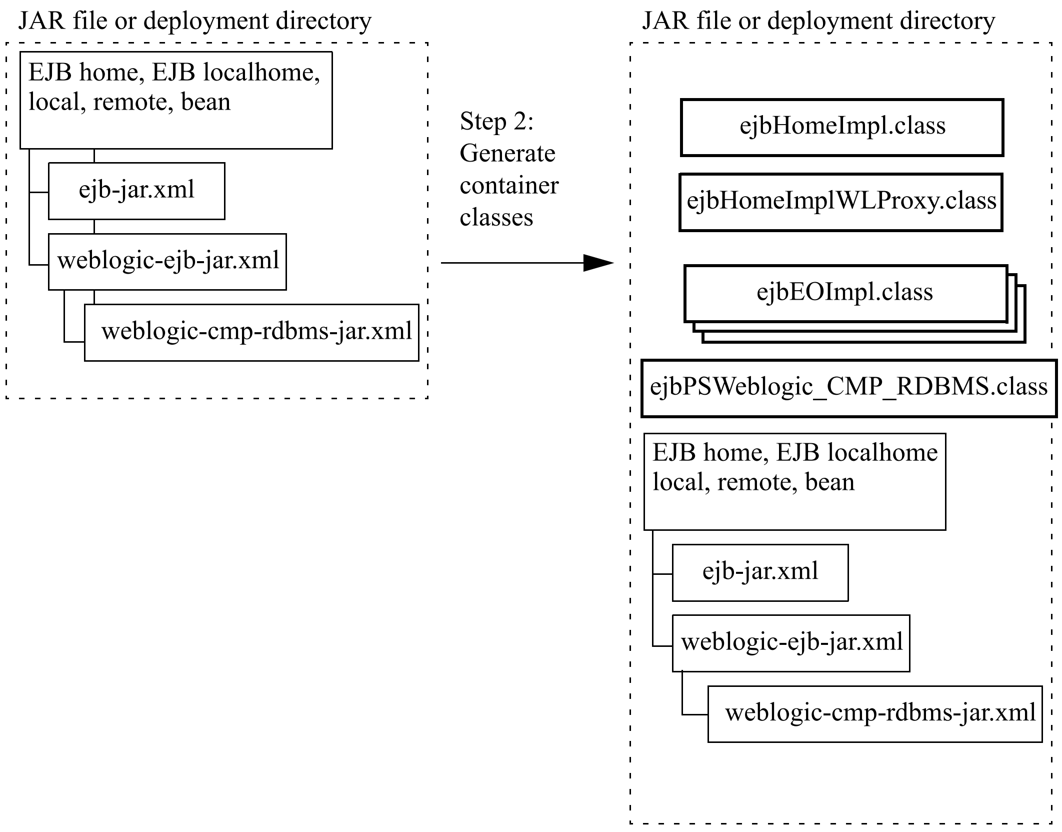
   Container classes include both the internal representation of the EJB that WebLogic Server uses, as well as implementation of the external interfaces (home, local, and/or remote) that clients use.

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

You can also use `ejbc` directly from the command line by supplying the required options and arguments. See "ejbc" on page 8-19 for more information.

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

Figure 6-5 Generating EJB container classes


JAR file or deployment directory
EJB home, EJB localhome, local, remote, bean
ejb-jar.xml
weblogic-ejb-jar.xml
weblogic-cmp-rdbms-jar.xml
Step 2: Generate container classes
JAR file or deployment directory
ejbHomeImpl.class
ejbHomeImplWLProxy.class
ejbEOImpl.class
ejbPSWeblogic_CMP_RDBMS.class
EJB home, EJB localhome local, remote, bean
ejb-jar.xml
weblogic-ejb-jar.xml
weblogic-cmp-rdbms-jar.xml

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

Possible Generated Class Name Collisions

Although infrequent, when you generate classes with ejbc, you may encounter a generated class name collision which could result in a `ClassCaastException` and other undesireable 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 contains 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.

Since 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, since each EAR file has its own classloader, this problem never occurs with two EJBs in different EAR files.

Loading EJB Classes into WebLogic Server

Classloaders in Weblogic Server are hierarchical. When you start WebLogic Server, the Java system classloader is active and is the parent of all subsequent classloaders that WebLogic Server creates. When WebLogic Server deploys an application, it automatically creates two new classloaders: one for EJBs and one for Web applications. The EJB classloader is a child of the Java system classloader and the Web application classloader is a child of the EJB classloader.

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

Specifying an ejb-client.jar

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

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

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

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

To specify an `ejb-client.jar`:

1. Compile the bean's Java classes into a directory, using the `javac` compiler from the command line.

2. Add the EJB XML deployment descriptor files to the compiled unit using the guidelines in "WebLogic Server EJB Deployment Files" on page 6-3.

3. Edit the `ejb-client-jar` deployment descriptor in the bean's `ejb-jar.xml` file, as follows, to specify support for `ejb-client.jar`:

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

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

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

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

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

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

Manifest Class-Path

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

To use the manifest file to reference another JAR file:

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

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

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

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

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

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

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

1. Use `ejbc` to compile the EJB into a JAR file.

2. Create a `client.jar` file. For instructions on using the `client.jar`, see "Specifying an ejb-client.jar" on page 6-14.

3. Place the `client.jar`, along with all the clients of the bean in an EAR.

4. Reference the EAR in the manifest file.

.

CHAPTER

7 Deploying EJBs to WebLogic Server

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

- Roles and Responsibilities
- Deploying EJBs at WebLogic Server Startup
- Deploying EJBs on a Running WebLogic Server
- Deploying New EJBs into a Running Environment
- Undeploying Deployed EJBs
- Redeploying EJBs
- Deploying Compiled EJB Files
- Deploying Uncompiled EJB Files
- Deployment Restriction with Container Managed Relationships

Roles and Responsibilities

The following sections are written primarily for:

- Deployers who configure EJBs to run in the WebLogic Server container

- Application assemblers who link multiple EJBs and EJB resources to create larger Web application systems
- EJB developers who create and configure new EJB JAR files

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

Deploying EJBs at WebLogic Server Startup

To deploy EJBs automatically when WebLogic Server starts:

1. Follow the instructions in "Specifying and Editing the EJB Deployment Descriptors" on page 6-5 to ensure that your deployable EJB JAR file or deployment directory contains the required WebLogic Server XML deployment files.

2. Use a text editor or the EJB Deployment Descriptor Editor in the Administration Console to edit the XML deployment descriptor elements, as necessary.

3. Follow the instructions in "Compiling EJB Classes and Generating EJB Container Classes" on page 6-12 to compile implementation classes required for WebLogic Server.

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

   `mydomain\applications` directory

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

4. Start WebLogic Server.

   When you boot WebLogic Server, it automatically attempts to deploy the specified EJB JAR file or deployment directory.

5. Launch the Administration Console.

6. In the left pane, click Deployments and then the EJB node.

   A list of the EJB deployments for the server displays under the node.

Deploying EJBs in Different Applications

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

Deploying EJBs on a Running WebLogic Server

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

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

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

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

EJB Deployment Names

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

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

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

Deploying New EJBs into a Running Environment

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

1. Start the WebLogic Server Administration Console.
2. Select the Domain in which you will be working.
3. In the left pane of the Console, click Deployments.
4. In the left pane of the Console, click EJB. A table displays in the right pane of the Console showing all the deployed EJBs.
5. Select the Configure a new EJB option.
6. Locate the EAR, WAR or JAR file you would like to configure. You can also configure an exploded application or component directory. Note that WebLogic Server will deploy all components it finds in and below the specified directory.

7. Click [select] to the left of a directory or file to choose desired file and proceed to the next step.
8. Select a Target Server from among Available Servers.
9. Enter a name for the EJB or application in the provided field.
10. Click Configure and Deploy to install the EJB or application. The Console will display the Deploy panel, which lists deployment status and deployment activities for the EJB.
11. Using the available tabs, enter the following information:
    - Configuration—Edit the staging mode and enter the deployment order.
    - Targets—Indicate the Targets-Server for this configured EJB or application by moving the server from the Available list to the Chosen list.
    - Deploy—Deploy the EJB or application to all or selected targets or undeploy it from all or selected targets.
    - Monitoring—Enable session monitoring for the EJB or application.
    - Notes—Enter notes related to the EJB or application.

Deploying Pinned EJBs - Special Step Required

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

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

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

For more information on pinned deployments, see Setting Up WebLogic Clusters at http://e-docs.bea.com/wls/docs70/cluster/setup.html#pinned_deployment.

Viewing Deployed EJBs

To view deployed EJBs:

1. Start the Administration Console.
2. Click the Deployments node in the left pane and then choose the EJB sub-node. A list of EJBs deployed on your domain displays under EJB and in the right pane.

Undeploying Deployed EJBs

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

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

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

Undeploying EJBs

To undeploy a deployed EJB, use the following steps:

From the WebLogic Server Administration Console:

1. Select the component in the left panel.

2. In the component Deployments table, select the component to undeploy.
3. Click Apply.

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

Redeploying EJBs

When you change a deployed EJB's classes, these changes are not reflected in WebLogic Server until:

- You reboot the server (if the JAR or directory is to be automatically deployed),

  or

- you redeploy the EJB deployment.

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.

The Redeploy Process

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.

Steps to Redeploy

You can redeploy an EJB that is standalone or part of an application, using the `weblogic.Deployer` tool or via the Administration Console.

To redeploy using `weblogic.Deployer`:

1. Use the `-deploy` flag:

   ```
   java weblogic.Deployer -deploy ejb_name
   ```

To redeploy via the WebLogic Server Administration Console:

1. Choose EJB from the Deployments node in the left pane of the Console, or—if the EJB is part of an application—choose Applications, then the application name.
2. Click the name of the EJB you want to redeploy.
3. Click the Deploy tab in the right pane.
4. Click Redeploy.

Deploying Compiled EJB Files

To create compiled EJB 2.0 JAR or EAR files:

1. Compile your EJB classes and interfaces using `javac`.
2. Package the EJB classes and interfaces into a valid JAR or EAR file.
3. Use the `weblogic.ejbc` compiler on the JAR file to generate WebLogic Server container classes. For instructions on using ejbc, see "ejbc" on page 8-19.

To create compiled EJBs from previous versions of WebLogic Server:

1. Run `weblogic.ejbc` against the ejb JAR file to generate EJB 2.0 container-classes.
2. Copy the compiled ejb JAR files into

   ```
   mydomain\applications\DefaultWebApp directory
   ```

Note: You should manually recompile any EJBs from previous versions before deploying then to the EJB container. Otherwise, WebLogic Server automatically recompiles the EJBs and if there are errors, the output from the compile is sent to a separate log file.

If you change the contents of a compiled `ejb.jar` file in `applications` (by repackaging, recompiling, or copying over the existing `ejb.jar`), WebLogic Server automatically attempts to redeploy the `ejb.jar` file using the automatic deployment feature.

Deploying Uncompiled EJB Files

The WebLogic Server container also enables you to automatically deploy JAR files that contain uncompiled EJB classes and interfaces. An uncompiled EJB file has the same structure as a compiled file, with the following exceptions:

- You do not have to compile individual class files and interfaces.
- You do not have to use `weblogic.ejbc` on the packaged JAR file to generate WebLogic Server container classes.

The `.java` or `.class` files in the JAR file must still be packaged in subdirectories that match their Java package hierarchy. Also, as with all `ejb.jar` files, you must include the appropriate XML deployment files in a top-level `META-INF` directory.

After you package the uncompiled classes, simply copy the JAR into the `wlserver\config\mydomain\applications` directory. If necessary, WebLogic Server automatically runs `javac` (or a compiler you specify) to compile the `.java` files, and runs `weblogic.ejbc` to generate container classes. The compiled classes are copied into a new JAR file in `mydomain\applications\DefaultWebApp`, and deployed to the EJB container.

Should you ever modify an uncompiled ejb `.jar` in the `applications` directory (either by repackaging or copying over the JAR file), WebLogic Server automatically recompiles and redeploys the JAR using the same steps.

Deployment Restriction with Container Managed Relationships

Two EJBs that have a container-managed relationship must be deployed in the same JAR file. For more information on container-managed relationships, see "Container-Managed Relationships" on page 5-42.

CHAPTER

8 WebLogic Server EJB Utilities

The following sections provide a complete reference to the utilities and support files supplied with WebLogic Server that are used with EJBs:

- EJBGen
- ejbc (`weblogic.ejbc`)
- DDConverter (`weblogic.ejb.utils.DDConverter`)
- weblogic.Deployer (`weblogic.Deployer`)
- weblogic.deploy (`weblogic.deploy`)

EJBGen

`EJBGen` is an Enterprise JavaBeans 2.0 code generator. You can annotate your Bean class file with javadoc tags and then use `EJBGen` to generate the Remote and Home classes and the deployment descriptor files for an EJB application, reducing to one the number of EJB files you need to edit and maintain.

If you have installed BEA WebLogic 7.0 examples, see *`SAMPLES_HOME`*`\server\src\examples\ejb20\ejbgen` for an example application called Bands that uses EJBGen.

EJBGen Syntax

```
javadoc -docletpath weblogic.jar -doclet
weblogic.tools.ejbgen.EJBGen (YourBean).java
```

If you do not have `weblogic.jar` in your classpath, add the path to `weblogic.jar` as follows:

```
javadoc -docletpath <path_to_weblogic.jar> weblogic.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 weblogic.jar -doclet
weblogic.tools.ejbgen.EJBGen (YourBean).java (RelatedBean).java
```

EJBGen uses the following options.

Option	Definition
`-d [directory]`	The directory under which all the files will be created.
`-ignorePackage`	If this flag is set, EJBGen will ignore the package name of the Java files it generates and will create those in the output directory as specified by the -d flag (or in the current directory if no -d was specified).
`-pfd1`	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.
`-ejbPrefix [string] (default: "")`	The prefix to use when generating the EJB class.
`-ejbSuffix [string] (default: "Bean" or "EJB")`	The suffix to use when generating the EJB class.
`-localHomePrefix [string] (default: "")`	The prefix to use when generating the local EJB class.

Option	Definition
`-localHomeSuffix [string] (default: "LocalHome")`	The suffix to use when generating the local EJB class.
`-remoteHomePrefix [string] (default: "")`	The prefix to use when generating the remote EJB home class.
`-remoteHomeSuffix [string] (default: "Home")`	The suffix to use when generating the remote EJB home class.
`-remotePrefix [string] (default: "")`	The prefix to use when generating the remote EJB class.
`-remoteSuffix [string] (default: "")`	The suffix to use when generating the remote EJB class.
`-localPrefix [string] (default: "")`	The prefix to use when generating the local EJB class.
`-localSuffix [string] (default: "Local")`	The suffix to use when generating the local EJB class.
`-valueObjectPrefix [string] (default: "")`	The prefix to use when generating the value object class.
`-valueObjectSuffix [string] (default: "Value")`	The suffix to use when generating the value object class.
`-jndiPrefix [string] (default: "")`	The prefix to use for `@remote-jndi-name` and `@local-jndi-name`
`-jndiSuffix [string] (default: "")`	The suffix to use for `@remote-jndi-name` and `@local-jndi-name`
`-checkTags`	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.

Option	Definition
`-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.
`-docTag tagName`	Print 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.
`-docTagsHtml`	Same as -docTags, but generate an HTML document.
`-propertyFile [fileName]`	The name of a property file that EJBGen will read to define substitution variables. See the substitution variable documentation
`-valueBaseClass [className]`	Removed. Use the variable `value.baseClass`.
`-noValueClasses`	If specified, value classes will not be generated.

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 Example

This example shows a Bean file annotated so that `EJBGen` will generate the Remote and Home interfaces and the deployment descriptor files. AccountBean.java is the main bean class. It is a CMP EJB 2.0 Entity bean:

```
/**

 * @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);
  // ....
}
```

As you can see from this example, there are two types of tags: class tags and method tags, depending on where you can use them.

Once you finish editing your file, you invoke EJBGen through the following javadoc command:

```
javadoc -docletpath weblogic.jar -doclet
weblogic.tools.ejbgen.EJBGen AccountBean.java
```

When javadoc exits, it will have generated the following files for you:

- `Account.java`
- `AccountHome.java`
- `ejb-jar.xml`
- `weblogic-ejb-jar.xml`
- `weblog-cmp-rdbms-jar.xml`

EJBGen Tags

Use the following tags to annotate your Bean file.

@ejbgen:automatic-key-generation

Where: Class

Applicable on: 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 on: Entity bean

Attribute	Description	Required
column	The column where this CMP field will be mapped.	Yes
column-type	The type of this column. (OracleClob|OracleBlob)	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 have this attribute to a distinct numeric value.	No

@ejbgen:cmr-field

*Where:*Method

Applicable on: Entity

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

@ejbgen:create-default-rdbms-tables

Where: Class

Applicable on: Entity bean

@ejbgen:ejb-client-jar

Where: Class

Applicable on: All types of beans

Attribute	Description	Required
file-name	The name of the client jar to generate. If more than one EJB's have this tag, only one of the specified jar files will be included in the deployment descriptor.	Yes

@ejbgen:ejb-local-ref

Where: Class

Applicable on: All types of beans

Attribute	Description	Required
home	Local class of the bean.	No
jndi-name	The JNDI name of the reference.	No
link	Link of the bean.	No
local	Home class of the bean.	No

Attribute	Description	Required
name	Name of the reference.	No
type	(Entity|Session)	No

@ejbgen:ejb-ref

Where: Class

Applicable on: All types of beans

Attribute	Description	Required
home	Remote class of the bean.	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 on: 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
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).	No
db-is-shared	(True|False)	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) Whether updates will be sent after the transaction has committed.	No
idle-timeout-seconds	Maximum duration an EJB should stay in the cache.	No
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
max-beans-in-cache	The maximum number of beans in the cache.	No
persistence-type	(cmp|bmp) The type of this Entity bean (default: cmp).	No
prim-key-class-nogen	(True|False). If this keyword is specified, EJBGen will 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.	No
table-name	The Java class of the primary key. In case of a compound primary key, this class will be generated by EJBGen.	No

Attribute	Description	Required
trans-timeout-seconds	The transaction timeout (in seconds).	No
use-caller-iden tity	(True|False) Whether this EJB uses caller's identity.	No

@ejbgen:env-entry

*Where:*Class

Applicable on: All types of beans

Attribute	Description	Required
name	The name of this environment entry.	Yes
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:finder

***Where:* Class**

Applicable on: Entity beans

Attribute	Description	Required
ejb-ql	The EJB QL request as it will appear in the deployment descriptor.	Yes
signature	It must match exactly the signature as you want it generated on the Home class. EJBGen will add 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.	Yes
isolation-le vel	The type of transaction isolation for this method.	No

Attribute	Description	Required
transaction-attribute	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
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 EJBQL and Weblogic EJBQL within double quotes.	No

@ejbgen:jndi-name

Where: Class

Applicable on: All types of beans

Attribute	Description	Required
local	The local JNDI name of this EJB. It not specified, no local interfaces will be generated.	No
remote	The remote JNDI name of this EJB. It not specified, no remote interfaces will be generated.	No

@ejbgen:local-home-method

Where: Method

Applicable on: Entity and Session beans

Attribute	Description	Required
transaction-attribute	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:local-method

Where: Method

Applicable on: Entity and Session beans

Attribute	Description	Requir ed
isolation-level	The type of transaction isolation for 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 will be generated on the Local class.	No

@ejbgen:message-driven

Where: Class

Applicable on: Message-Driven beans

Attribute	Description	Required
destination-jndi-name	The JNDI name of the destination.	Yes
ejb-name	The name of this Message-Driven bean.	Yes
acknowledge-mode	(auto-acknowledge|dups-ok-acknowledge) The acknowledgement mode.	No
default-transaction	The transaction attribute to be applied to all methods that do not have a more specific transaction attribute setting.	No
destination-type	(javax.jms.Queue|javax.jms.Topic).	No
durable	(True|False) If the destination-type is "Topic", setting this attribute to True will make the subscription durable.	No
initial-beans-in-free-p ool	The initial number of beans in the free pool.	No

Attribute	Description	Required
max-beans-in-free-pool	The maximum number of beans in the free pool.	No
message-selector	The JMS message selector.	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.	No
trans-timeout-seconds	The transaction timeout (in seconds).	No
use-caller-identity	(True|False) Whether this EJB uses caller's identity.	No

@ejbgen:primkey-field

Where: Method

Applicable on: Entity beans

@ejbgen:relation

Where: Class

Applicable on: Entity beans

Attribute	Description	Required
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
target-ejb	The EJB name of the target of this relationship.	Yes
cascade-delete	(True|False)	No

Attribute	Description	Required
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
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
joint-table	Only needed in a Many-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
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

@ejbgen:remote-home-method

Where: Method

Applicable on: Entity and Session beans

Attribute	Description	Required
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 on: Entity and Session beans

Attribute	Description	Required
isolation-level	The type of transaction isolation for 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:resource-env-ref

Where: Class

Applicable on: All types of beans

Attribute	Description	Required
name	Name of the resource environment reference.	Yes
type	Type of the environment resource references (e.g. javax.jms.Queue).	Yes
jndi-name	JNDI name of the resource.	No

@ejbgen:resource-ref

Where: Class

Applicable on: All types of beans

Attribute	Description	Required
auth	(Application|Container)	Yes
jndi-name	JNDI name of the resource.	Yes
name	Name of the resource.	Yes
type	Type of the resource (e.g. javax.sql.DataSource).	Yes

Attribute	Description	Required
sharing-scope	(Shareable|Unshareable)	No

@ejbgen:role-mapping

Where: Class

Applicable on: All types of beans

Attribute	Description	Required
principals	The names of the principals in this role (separated by commas).	Yes
role-name	The name of the role	Yes

@ejbgen:select

Where: Method

Applicable on: Entity beans

Attribute	Description	Required
ejb-ql	The EJB-QL defining this select method. Note: the method name must start with ejbSelect.	Yes
result-type-mapping	(Remote|Local) Whether the returned objects are mapped to EJBLocalObject or EJBObject.	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 EJBQL and Weblogic EJBQL within double quotes.	No

@ejbgen:session

Where: Class

Applicable on: Session beans

Attribute	Description	Required
ejb-name	The name of this Session bean.	Yes
call-router-class-name	Class name to be used for routing home method calls	No
default-transaction	The transaction attribute to be applied to all methods that do not have a more specific transaction attribute setting.	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
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.	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 will guess 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 on: All types of beans

Attribute	Description	Required
reference	(Local|Value) Specify what objects the value object class should reference when accessing other EJB's.	Yes

ejbc

Use the `weblogic.ejbc` utility for generating and compiling EJB 2.0 and 1.1 container classes. If you compile JAR files for deployment into the EJB container, you must use `weblogic.ejbc` to generate the container classes.

`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.

Although versions of the WebLogic-specific XML deployment descriptor files are published on or web site for your convience, 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`, see "DOCTYPE Header Information" on page 9-2; for the location of the public version of `weblogic-cmp-rdbms-jar.xml`, see "DOCTYPE Header Information" on page 10-2.

Advantages of Using ejbc

The `ejbc` utility 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)
-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.

ejbc Examples

The following example uses the `javac` compiler against an input JAR file in `c:\%SAMPLES_HOME%\server\src\examples\ejb\basic\containerManaged\build`. The output JAR file is placed in `c:\%SAMPLES_HOME%\server\config\examples\applications`.

```
prompt> java weblogic.ejbc -compiler javac
c:\%SAMPLES_HOME%\server\samples\src\examples\ejb\basic\container
Managed\build\std_ejb_basic_containerManaged.jar
c:%SAMPLES_HOME%\server\config\examples\ejb_basic_containerManage
d.jar
```

The following example checks a JAR file for compliance with the EJB 1.1 specification and generates WebLogic Server container classes, but does not generate RMI stubs:

```
prompt> java weblogic.ejbc -normi
c:%SAMMPLES_HOME%\server\src\examples\ejb\basic\containerManaged\
build\std_ejb_basic_containerManaged.jar
```

DDConverter

The `DDConverter` is a command line utility that converts earlier versions EJB deployment descriptors into EJB deployment descriptors that conform to this version of WebLogic Server. The WebLogic Server EJB container supports both the EJB 1.1 and EJB 2.0 specifications including the EJB 1.1 and EJB 2.0 document type definitions (DTD). Each WebLogic Server EJB deployment includes standard deployment descriptors in the following files:

- `ejb-jar.xml`

  This XML file contains the J2EE-specific EJB deployment descriptors.

- `weblogic-ejb-jar-.xml`

  This XML file contains the WebLogic-specific EJB deployment descriptors.

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

This XML file contains the WebLogic-specific container-managed persistence (CMP) deployment descriptors.

Conversion Options Available with DDConverter

The `DDConverter` command line utility includes the following conversion options:

- Converting beans from earlier versions of WebLogic Server (WLS).
- Converting CMP and non-CMP beans from earlier version of the EJB specification.

The following table lists the various conversion options for the DDconverter:

Table 8-1

Conversion Options for the DDConverter Utility

WLS		**EJB non-CMP**		**EJB CMP**	
From	**To**	**From**	**To**	**From**	**To**
WLS 4.5 -	WLS 7.0	See **Note 1**		EJB CMP 1.0 -	EJB CMP 1.1 See **Note 2**
WLS 4.5 -	WLS 7.0	EJB 1.1 -	EJB 2.0	EJB CMP 1.0 -	EJB CMP 2.0
WLS 5.x -	WLS 7.0	EJB 1.1 -	EJB 2.0	See **Note 3**	
WLS 6.x -	WLS 7.0	EJB 1.1 -	EJB 2.0	EJB CMP 1.1 -	EJB CMP 2.0

Note: Converting non-CMP EJB 1.0 beans to non-CMP EJB 1.1 beans is not necessary because the EJB 1.1 non-CMP deployment descriptors are the same as the EJB 2.0 non-CMP deployment descriptors.

Note: Use the DDConverter command line option `-EJBVer` for converting EJB CMP 1.0 to EJB CMP 1.1. See "DDConverter Options" on page 8-27 for a description of this option.

Note: Even though WLS 5.x CMP 1.1 beans and WLS 7.0 CMP 1.1 beans are different, WLS 5.1 CMP 1.1 beans can run in WebLogic Server 7.0 without any changes to the source code.

You should always recompile the beans after you use the DDConverter. We recommend that you use `weblogic.ejbc` and then deploy the new generated JAR file. Recompiling the bean makes sure that the code is compliant with the EJB Specifications and saves you time because you can skip the recompile process during server startup.

- When converting WLS 4.5 EJB 1.0 beans to WLS 7.0 EJB 1.1 beans, the input to `DDConverter` is the WebLogic 4.5 deployment descriptor text. The output is a JAR file that only includes the WebLogic 7.0 deployment descriptors. Run `weblogic-ejbc` to see if you need to make any additional changes to the source code following the steps in "Using DDConverter to Convert EJBs" on page 8-26. See the first row in the Conversion Options for the DDConverter Utility table.
- When converting WLS 4.5 EJB 1.1 beans to WLS 7.0 EJB 2.0 beans, the input to `DDConverter` is the WebLogic Server 4.5 deployment descriptor text. The output is a JAR file that only includes the WebLogic 7.0 deployment descriptors. Run `weblogic-ejbc` to see if you need to make any additional changes to the source code, follow the steps in "Using DDConverter to Convert EJBs" on page 8-26. See the second row in the Conversion Options for the DDConverter Utility table.
- You can deploy WLS 5.x EJB 1.1 beans to WLS 7.0 without any making changes to the source code because WLS 7.0 is backward compatible. WLS 7.0 detects, recompiles, and then deploys beans from previous versions of WLS. However, we recommend that you use the DDConverter to upgrade the WLS 5.x EJB 1.1 beans to WLS 7.0 EJB 2.0 beans.

  When converting WLS 5.x EJB 1.1 beans to WLS 7.0 EJB 2.0 beans, the input to `DDConverter` is the WebLogic 5.1 JAR file. This file contains the deployment descriptor files and class files. The output goes to a JAR file that includes the WebLogic 7.0 deployment descriptor files and all necessary class files. See the third row in the Conversion Options for the DDConverter Utility table.

  You can convert non-CMP beans to EJB 2.0 beans with little or no changes to the source code. To do this, run weblogic.ejbc on the `output.jar` file and then deploy the generated JAR file. With CMP beans, you must make changes to the source code using the steps in "Using DDConverter to Convert EJBs" on page 8-26.

Using DDConverter to Convert EJBs

To convert earlier versions of EJBs for use in WebLogic Server:

1. Input the EJB's deployment descriptor file into the `DDConverter` using the command line format shown in "DDConverter Syntax" on page 8-26.

   The output is a JAR file.

2. Extract the XML deployment descriptors from the JAR file.
3. Modify the source code according to the JavaSoft EJB Specification.
4. Compile the modified java file with the extracted XML deployment descriptors, using `weblogic.ejbc` to create a JAR file.
5. Deploy the JAR file.

DDConverter Syntax

```
prompt> java weblogic.ejb20.utils.DDConverter [options] file1
[file2...]
```

DDConverter Arguments

`DDConverter` takes the argument *`file1 [file2...]`*, where file is one of the following:

- A text file containing EJB 1.0-compliant deployment descriptors.
- A JAR file containing EJB 1.1 compliant deployment descriptors.

`DDConverter` uses the `beanHomeName` property of EJBs in the text deployment descriptor to define new `ejb-name` elements in the resultant `ejb-jar.xml` file.

DDConverter Options

The following table lists the DDConverter command-line options:

Option	Description
`-d` destDir	Specifies the destination directory for the output of the JAR files. This is a required option.
`-c` jar name	Specifies a JAR file in which you combine all beans in the source files.
`-EJBVer` *`output EJB version`*	Specifies the output EJB version number, such as 2.0 or 1.1. The default is 2.0.
`-log` log file	Specifies a file into which the log information can be placed instead of the `ddconverter.log`.
`-verboseLog`	Specifies that extra information on the conversion be placed in the `ddconverter.log`.
`-help`	Prints a list of all options available for the `DDConverter` utility.

DDConverter Examples

The following example converts a WLS 5.x EJB 1.1 bean into a WLS 7.0 EJB 2.0 bean.

The JAR file is created in the *`destDir`* subdirectory:

```
prompt> java weblogic.ejb20.utils.DDConverter -d destDir
Employee.jar
```

Where the `Employee` bean is a WLS 5.x EJB 1.1 JAR file.

weblogic.Deployer

The `weblogic.Deployer` command-line utility is new in WebLogic Server 7.0 and replaces the earlier weblogic.deploy utility, which has been deprecated. The `weblogic.Deployer` utility is a Java-based deployment tool that provides a command line interface to the WebLogic Server deployment API. This utility 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.

For instructions on using weblogic.Deployer and a list of the commands, see Deploying Using weblogic.Deployer.

weblogic.deploy

Note: This tool is being deprecated in this release. We strongly encourage you to use the command-line tool weblogic.Deployer. It provides more sophisticated deployment options.

Use the `weblogic.deploy` command-line utility tool to deploy EJBs. Given an EJB-compliant JAR file, the JAR's EJBs are deployed into a running WebLogic Server.

deploy Syntax

```
prompt> java weblogic.deploy [options]
[list|deploy|undeploy|update] password {name} {source}
```

deploy Arguments

Argument	Description
`list`	Lists all EJB deployment units in the specified WebLogic Server.
`deploy`	Deploys an EJB JAR to the specified server.
`undeploy`	Removes an existing EJB deployment unit from the specified server.
`password`	Specifies the system password for the WebLogic Server.
{*`name`*}	Identifies the name of the EJB deployment unit. This name can be specified at deployment time, either with the `deploy` or `console` utilities.
{*`source`*}	Specifies the exact location of the EJB JAR file, or the path to the top level of an EJB deployment directory.

deploy Options

Option	Description
`-help`	Prints a list of all options available for the `deploy` utility.
`-version`	Prints the version of the utility.

`-port <port>`	Specifies the port number of the WebLogic Server to use for deploying the JAR file. If you do not specify this option, the `deploy` utility attempts to connect using port number 7001.
`-host <host>`	Specifies the host name of the WebLogic Server to use for deploying the JAR file. If you do not specify this option, the `deploy` utility attempts to connect using host name `localhost`.
`-user`	Specifies the system username of the WebLogic Server to be used to deploy the JAR file. If you do not specify this option, `deploy` attempts to make a connection using the system username `system`. You use the `weblogic.system.user` property to define the system username.
`-debug`	Prints detailed debugging information during the deployment process.

CHAPTER

9 weblogic-ejb-jar.xml Document Type Definitions

The following sections describe the EJB 1.1 and 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.

- EJB Deployment Descriptors
- DOCTYPE Header Information
- Changed Deployment Elements in WebLogic Server 7.0 EJB
- 2.0 weblogic-ejb-jar.xml Deployment Descriptor File Structure
- 2.0 weblogic-ejb-jar.xml Deployment Descriptor Elements
- 5.1 weblogic-ejb-jar.xml Deployment Descriptor File Structure
- 5.1 weblogic-ejb-jar.xml Deployment Descriptor Elements

EJB Deployment Descriptors

The EJB deployment descriptors contain structural and application assembly information for an enterprise bean. You specify this information by specifying values for the deployment descriptors in three EJB XML DTD files. These files are:

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

You package these three XML files with the EJB and other classes into a deployable EJB component, usually a JAR file, called `ejb.jar`.

The `ejb-jar.xml` file is based on the deployment descriptors found in Sun Microsystems's `ejb.jar.xml` file. The other two XML files are weblogic-specific files that are based on the deployment descriptors found in `weblogic-ejb-jar.xml` and `weblogic-cmp-rdbms-jar.xml`.

DOCTYPE Header Information

When you edit or create XML deployment files, it is critical to include the correct `DOCTYPE` header for the deployment file. In particular, using an incorrect `PUBLIC` element within the `DOCTYPE` header can result in parser errors that may be difficult to diagnose.

WebLogic provides a public location for you to access the correct text for the WebLogic Server-specific DTD file, `weblogic-ejb-jar.xml`. However, an identical version of this DTD file is embedded in WebLogic Server for internal use. weblogic.ejbc uses this file when the XML parser checks the sequence of the deployment descriptors files.

The correct text for the `PUBLIC` elements for the `WebLogic Server-specific weblogic-ejb-jar.xml` file are as follows.

XML File	PUBLIC Element String
weblogic-ejb-jar.xml	'-//BEA Systems, Inc.//DTD WebLogic 7.0.0 EJB//EN' 'http://www.bea.com/servers/wls700/dtd/weblogic-ejb-jar.dtd'
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'
weblogic-ejb-jar.xml	'-//BEA Systems, Inc.//DTD WebLogic 5.1.0 EJB//EN' 'http://www.bea.com/servers/wls510/dtd/weblogic-ejb-jar.dtd'

The correct text for the `PUBLIC` elements for the Sun Microsystem-specific `ejb-jar.xml` file are as follows.

XML File	PUBLIC Element String
ejb-jar.xml	'-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN' 'http://java.sun.com/dtd/ejb-jar_2_0.dtd'
ejb-jar.xml	'-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN' 'http://www.java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'

For example, the entire `DOCTYPE` header for a `weblogic-ejb-jar.xml` file is as follows:

```
<!DOCTYPE weblogic-ejb-jar PUBLIC
'-//BEA Systems, Inc.//DTD WebLogic 7.0.0 EJB//EN'
'http://www.bea.com/servers/wls700/dtd/weblogic-ejb-jar.dtd'>
```

XML files with incorrect header information may yield error messages similar to the following, when used with a utility that parses the XML (such as `ejbc`):

```
SAXException: This document may not have the identifier 'identifier_name'
```

`identifier_name` generally includes the invalid text from the `PUBLIC` element.

Document Type Definitions (DTDs) for Validation

The contents and arrangement of elements in your XML files must conform to the Document Type Definition (DTD) for each file you use. WebLogic Server ignores the DTDs embedded within the `DOCTYPE` header of XML deployment files, and instead uses the DTD locations that were installed along with the server. However, the `DOCTYPE` header information must include a valid URL syntax in order to avoid parser errors.

Note: Most browsers do not display the contents of files having the `.dtd` extension. To view the DTD file contents in your browser, save the links as text files and view them with a text editor.

weblogic-ejb-jar.xml

The following links provide the new public DTD locations for the `weblogic-ejb-jar.xml` deployment files used with the WebLogic Server:

- For `weblogic-ejb-jar.xml` 7.0 DTD:

  `http://www.bea.com/servers/wls700/dtd/weblogic-ejb-jar.dtd` contains the DTD used for creating `weblogic-ejb-jar.xml`, which defines EJB properties used for deployment to WebLogic Server.

- For `weblogic-ejb-jar.xml` 6.0 DTD:

  `http://www.bea.com/servers/wls600/dtd/weblogic-ejb-jar.dtd` contains the DTD used for creating `weblogic-ejb-jar.xml`, which defines EJB properties used for deployment to WebLogic Server.

- For `weblogic-ejb-jar.xml` 5.1 DTD:

  `weblogic-ejb-jar.dtd` contains the DTD used for creating `weblogic-ejb-jar.xml`, which defines EJB properties used for deployment to WebLogic Server. This file is located at *http://www.bea.com/servers/wls510/dtd/weblogic-ejb-jar.dtd*

ejb-jar.xml

The following links provide the public DTD locations for the `ejb-jar.xml` deployment files used with WebLogic Server:

- For `ejb-jar.xml` 2.0 DTD:

  `http://www.java.sun.com/dtd/ejb-jar_2_0.dtd` contains the DTD for the standard `ejb-jar.xml` deployment file, required for all EJBs. This DTD is maintained as part of the JavaSoft EJB 2.0 specification; refer to the JavaSoft specification for information about the elements used in `ejb-jar.dtd`.

- For `ejb-jar.xml` 1.1 DTD:

  `ejb-jar.dtd` contains the DTD for the standard `ejb-jar.xml` deployment file, required for all EJBs. This DTD is maintained as part of the JavaSoft EJB 1.1 specification; refer to the JavaSoft specification for information about the elements used in `ejb-jar.dtd`.

Note: Refer to the appropriate JavaSoft EJB specification for a description of the `ejb-jar.xml` deployment descriptors.

Changed Deployment Elements in WebLogic Server 7.0 EJB

These changes were made to `weblogic-ejb-jar.xml` in WebLogic Server 7.0:

- "cache-between-transactions" on page 9-13 was added.
- "entity-cache-name" on page 9-32 was added.
- "entity-cache-ref" on page 9-33 was added.
- "estimated-bean-size" on page 9-36 was added.
- "externally-defined" on page 9-37 was added.
- "global-role" on page 9-39 was deprecated.
- "idempotent-methods" on page 9-43
- "isolation-level" on page 9-52—the new `TransactionReadCommittedForUpdateNoWait` value was added.

2.0 weblogic-ejb-jar.xml Deployment Descriptor File Structure

The WebLogic Server 7.0 `weblogic-ejb-jar.xml` deployment descriptor file describes the elements that are unique to WebLogic Server. Although you can use both WebLogic 7.0 and WebLogic 6.0 deployment descriptor elements in the EJB container, the WebLogic Server 7.0 version of `weblogic-ejb-jar.xml` is different from the version shipped with WebLogic Server Version 6.0.

The top level elements in the WebLogic Server 7.0 `weblogic-ejb-jar.xml` are as follows:

- `description`
- weblogic-version
- `weblogic-enterprise-bean`
  - `ejb-name`
  - `entity-descriptor | stateless-session-descriptor | stateful-session-descriptor | message-driven-descriptor`
  - `transaction-descriptor`
  - `reference-descriptor`
  - `enable-call-by-reference`
  - `clients-on-same-server`
  - `jndi-name`
- security-role-assignment
- transaction-isolation

2.0 weblogic-ejb-jar.xml Deployment Descriptor Elements

- "allow-concurrent-calls" on page 9-11
- "allow-remove-during-transaction" on page 9-12
- "cache-between-transactions" on page 9-13
- "cache-type" on page 9-14
- "client-authentication" on page 9-15
- "client-cert-authentication" on page 9-16
- "clients-on-same-server" on page 9-17
- "concurrency-strategy" on page 9-18
- "confidentiality" on page 9-20
- "connection-factory-jndi-name" on page 9-21
- "delay-updates-until-end-of-tx" on page 9-22
- "description" on page 9-23
- "destination-jndi-name" on page 9-24
- "ejb-local-reference-description" on page 9-28
- "ejb-name" on page 9-25
- "ejb-reference-description" on page 9-26
- "ejb-ref-name" on page 9-27
- "ejb-local-reference-description" on page 9-28
- "enable-call-by-reference" on page 9-29
- "enable-dynamic-queries" on page 9-30
- "entity-cache" on page 9-31

- "entity-cache-name" on page 9-32
- "entity-cache-ref" on page 9-33
- "entity-clustering" on page 9-34
- "entity-descriptor" on page 9-35
- "externally-defined" on page 9-37
- "estimated-bean-size" on page 9-36
- "finders-load-bean" on page 9-38
- "home-call-router-class-name" on page 9-40
- "home-is-clusterable" on page 9-41
- "home-load-algorithm" on page 9-42
- "idempotent-methods" on page 9-43
- "identity-assertion" on page 9-44
- "idle-timeout-seconds" on page 9-45
- "iiop-security-descriptor" on page 9-46
- "initial-beans-in-free-pool" on page 9-47
- "initial-context-factory" on page 9-48
- "integrity" on page 9-49
- "invalidation-target" on page 9-50
- "is-modified-method-name" on page 9-51
- "isolation-level" on page 9-52
- "jms-polling-interval-seconds" on page 9-54
- "jms-client-id" on page 9-55
- "jndi-name" on page 9-56
- "local-jndi-name" on page 9-57
- "max-beans-in-cache" on page 9-58

- "max-beans-in-free-pool" on page 9-59
- "message-driven-descriptor" on page 9-60
- "method" on page 9-61
- "method-intf" on page 9-62
- "method-name" on page 9-63
- "method-param" on page 9-64
- "method-params" on page 9-65
- "persistence" on page 9-66
- "persistence-use" on page 9-67
- "persistent-store-dir" on page 9-68
- "pool" on page 9-69
- "principal-name" on page 9-70
- "provider-url" on page 9-71
- "read-timeout-seconds" on page 9-72
- "reference-descriptor" on page 9-73
- "relationship-description" on page 9-74
- "replication-type" on page 9-74
- "res-env-ref-name" on page 9-75
- "res-ref-name" on page 9-76
- "resource-description" on page 9-77
- "resource-env-description" on page 9-78
- "role-name" on page 9-79
- "security-permission" on page 9-80
- "security-permission-spec" on page 9-81
- "security-role-assignment" on page 9-82

- "security-role-assignment" on page 9-82
- "stateful-session-cache" on page 9-84
- "stateful-session-clustering" on page 9-85
- "stateful-session-descriptor" on page 9-86
- "stateless-bean-call-router-class-name" on page 9-87
- "stateless-bean-is-clusterable" on page 9-88
- "stateless-bean-load-algorithm" on page 9-89
- "stateless-bean-methods-are-idempotent" on page 9-90
- "stateless-clustering" on page 9-91
- "stateless-session-descriptor" on page 9-92
- "transaction-descriptor" on page 9-93
- "transaction-isolation" on page 9-94
- "transport-requirements" on page 9-95
- "trans-timeout-seconds" on page 9-96
- "type-identifier" on page 9-97
- "type-storage" on page 9-98
- "type-version" on page 9-99
- "weblogic-ejb-jar" on page 9-100
- "weblogic-enterprise-bean" on page 9-100

allow-concurrent-calls

Range of values:	`true | false`
Default value:	`false`
Requirements:	Requires the server to throw a `RemoteException` when a stateful session bean instance is currently handling a method call and another (concurrent) method call arrives on the server.

Parent elements:	`weblogic-enterprise-bean`
	    `stateful-session-descriptor`

Function

The `allow-concurrent-calls` element specifies whether a stateful session bean instance allows concurrent method calls. By default, `allows-concurrent-calls` is `false`. However, when this value is set to `true`, the EJB container blocks the concurrent method call and allows it to proceed when the previous call has completed.

Example

See "stateful-session-descriptor" on page 9-86.

allow-remove-during-transaction

Range of values:	`True` | `False`
Default value:	`False`
Requirements:	
Parent elements:	`weblogic-enterprise-bean` `stateful-session-descriptor`

Function

The `allow-remove-during-transaction` element specifies that the remove method on a stateful session bean can be invoked within a transaction context.

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 9-86.

cache-between-transactions

Range of values:	`true	false`
Default value:	`false`
Requirements:	Optional element. Valid only for entity EJBs.
Parent elements:	`weblogic-enterprise-bean,` `entity-descriptor,` `persistence`

Function

The `cache-between-transactions` element, formerly the `db-is-shared` element, specifies whether the EJB container will cache the persistent data of an entity bean across (between) transactions.

The `cache-between-transactions` element applies only to entity beans. Specify `True` to enable the EJB container performs long term caching of the data. Specify `False` to enable the EJB container to perform short term caching of the data. This is the default setting.

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 "Caching Between Transactions" on page 4-25 for more information.

Example

See "persistence" on page 9-66.

cache-type

Range of values:	`NRU	LRU`
Default value:	`NRU`
Requirements:	

Parent elements:	`weblogic-enterprise-bean`
	 `stateful-session-cache`

Function

The `cache-type` element specifies the order in which EJBs are removed from the cache. The values are:

- Least recently used (LRU)
- Not recently used (NRU)

  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 `cache-type`.

Example

The following example shows the structure of the `cache-type` element.

```
<stateful-session-cache>

<cache-type>NRU</cache-type>

</stateful-session-cache>
```

client-authentication

Range of values:	`none	supported	required`
Default value:	
Requirements:	n/a
Parent elements:	`weblogic-enterprise-bean` `iiop-security-descriptor`

Function

The `client-authentication` element specifies whether the EJB supports or requires client authentication.

Example

See "iiop-security-descriptor" on page 9-46.

client-cert-authentication

Range of values:	`none	supported	required`
Default value:	
Requirements:	n/a

Parent elements:	`weblogic-enterprise-bean`
	    `iiop-security-descriptor`
	        `transport-requirements`

Function

The `client-cert-authentication` element specifies whether the EJB supports or requires client certificate authentication at the transport level.

Example

See "transport-requirements" on page 9-95.

clients-on-same-server

Range of values:	`true	false`
Default value:	`false`
Requirements:	n/a
Parent elements:	`weblogic-enterprise-bean`

Function

The `clients-on-same-server` attribute 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

The following example enables pass-by-value for EJB methods:

```
<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`
Requirements:	Optional element. Valid only for entity EJBs.
Parent elements:	`weblogic-enterprise-bean, entity-descriptor, entity-cache`

Function

The `concurrency-strategy` element specifies how the container should manage concurrent access to an entity bean. Set this element to one of 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 block until the transaction completes. This option was the default locking behavior for WebLogic Server versions 3.1 through 5.1.
- `Database` causes WebLogic Server to defer locking requests for an entity EJB to the underlying datastore. With the `Database` concurrency strategy, WebLogic Server allocates a separate entity bean instance and allows locking and caching to be handled by the database. This is the default option.
- `ReadOnly` 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 "EJB Concurrency Strategy" on page 4-14 for more information on the `Exclusive` and `Database` locking behaviors. See "Read-Only Multicast Invalidation" on page 4-21 for more information about `read-only` entity EJBs.

Example

The following entry identifies the `AccountBean` class as a read-only entity EJB:

```
<weblogic-enterprise-bean>

       <ejb-name>AccountBean</ejb-name>

       <entity-descriptor>

              <entity-cache>


<concurrency-strategy>ReadOnly</concurrency-strategy>

              </entity-cache>

       </entity-descriptor>

</weblogic-enterprise-bean>
```

confidentiality

Range of values:	`none | supported | required`
Default value:	n/a
Requirements:	n/a

Parent elements:	`weblogic-enterprise-bean`
	`    iiop-security-descriptor`
	`        transport-requirements n`

Function

The `confidentiality` element 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 9-95.

connection-factory-jndi-name

Range of values:	`valid name`
Default value:	`weblogic.jms.MessageDrivenBeanConnectionFactory` in `config.xml`
Requirements:	Requires the server to throw a `RemoteException` when a stateful session bean instance is currently handling a method call and another (concurrent) method call arrives on the server.

Parent elements:	`weblogic-enterprise-bean`
	    `message-driven-descriptor`

Function

The `connection-factory-jndi-name` element specifies the JNDI name of the JMS ConnectionFactory that the MessageDriven Bean should look up to create its queues and topics. If this element is not specified, the default is the `weblogic.jms.MessageDrivenBeanConnectionFactory` in `config.xml.`

Example

The following example shows the structure of the `connection-factory-jndi-name` element:

```
<message-driven-descriptor>

<connection-factory-jndi-name>weblogic.jms.MessageDrivenBean
ConnectionFactory</connection-factory-jndi-name>

</message-driven-descriptor>
```

delay-updates-until-end-of-tx

Range of values:	`true	false`
Default value:	`true`
Requirements:	Optional element. Valid only for entity EJBs.
Parent elements:	`weblogic-enterprise-bean,` `entity-descriptor,` `persistence`

Function

Set the `delay-updates-until-end-of-tx` element to `true` (the default) to update the persistent store of all beans in a transaction at the completion of the transaction. This setting generally improves performance by avoiding unnecessary updates. However, it does not preserve the ordering of database updates within a database transaction.

If your datastore uses an isolation level of `TRANSACTION_READ_UNCOMMITTED`, you may want to allow other database users to view the intermediate results of in-progress transactions. In this case, set `delay-updates-until-end-of-tx` to `false` to update the bean's persistent store at the conclusion of each method invoke. See "ejbLoad() and ejbStore() Behavior for Entity EJBs" on page 4-12 for more information.

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

Example

The following example shows a `delay-updates-until-end-of-tx` stanza.

```
<entity-descriptor>
```

```
       <persistence>

<delay-updates-until-end-of-tx>false</delay-updates-until-end-of-
tx>

       </persistence>

</entity-descriptor>
```

description

Range of values:	n/a
Default value:	n/a
Requirements:	n/a

Parent elements:	`weblogic-ejb-jar`
	`     transaction-isolation`
	`          method`

Function

The `description` element is used to provide text that describes the parent element.

Example

The following example specifies the `description` element.

```
<description>Contains a description of parent element</description>
```

destination-jndi-name

Range of values:	Valid JNDI name
Default value:	n/a
Requirements:	Required in `message-driven-descriptor`.

Parent elements:	`weblogic-enterprise-bean`
	    `message-driven-descriptor`

Function

The `destination-jndi-name` element specifies the JNDI name used to associate a message-driven bean with an actual JMS Queue or Topic deployed in the WebLogic Server JNDI tree.

Example

See "message-driven-descriptor" on page 9-60.

ejb-name

Range of values:	Name of an EJB defined in `ejb-jar.xml`
Default value:	n/a
Requirements:	Required element in method stanza. The name must conform to the lexical rules for an NMTOKEN.

Parent elements:	`weblogic-enterprise-bean`
	    `method`

Function

`ejb-name` specifies the name of an EJB to which WebLogic Server applies isolation level properties. This name is assigned by the `ejb-jar` file's deployment descriptor. The name must be unique among the names of the enterprise beans in the same `ejb.jar` file. The enterprise bean code does not depend on the name; therefore the name can be changed during the application-assembly process without breaking the enterprise bean's function. There is no built-in relationship between the `ejb-name` in the deployment descriptor and the JNDI name that the deployer will assign to the enterprise bean's home.

Example

See "method" on page 9-61.

ejb-reference-description

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Optional element.
Parent elements:	`weblogic-enterprise-bean` `reference-descriptor`

Function

The `ejb-reference-description` element maps the JNDI name in the WebLogic Server of an EJB that is referenced by the bean in the `ejb-reference` element.

- `ejb-ref-name` specifies a resource reference name. This is the reference that the EJB provider places within the `ejb-jar.xml` deployment file.
- `jndi-name` specifies the JNDI name of an actual resource factory available in WebLogic Server.

Example

The `ejb-reference-description` stanza is shown here:

```
<ejb-reference-description>

       <ejb-ref-name>AdminBean</ejb-ref-name>

              <jndi-name>payroll.AdminBean</jndi-name>

       </ejb-reference-description>
```

ejb-ref-name

Range of values:	n/a
Default value:	n/a
Requirements:	Optional element.

Parent elements:	`weblogic-enterprise-bean`
	`reference-description`
	`ejb-reference-description`

Function

The `ejb-ref-name` element specifies a resource reference name. This element is the reference that the EJB provider places within the `ejb-jar.xml` deployment file.

Example

The `ejb-ref-name` stanza is shown here:

```
<reference-descriptor>

       <ejb-reference-description>

              <ejb-ref-name>AdminBean</ejb-ref-name>

              <jndi-name>payroll.AdminBean</jndi-name>

       </ejb-reference-description>

</reference-descriptor>
```

ejb-local-reference-description

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Optional element.
Parent elements:	`weblogic-enterprise-bean` `reference-descriptor`

Function

The `ejb-local-reference-description` element maps the JNDI name of an EJB in the WebLogic Server that is referenced by the bean in the `ejb-local-ref` element.

Example

The following example shows the `ejb-local-reference-description` element.

```
<ejb-local-reference-description>

       <ejb-ref-name>AdminBean</ejb-ref-name>

       <jndi-name>payroll.AdminBean</jndi-name>

</ejb-local-reference-description>
```

enable-call-by-reference

Range of values:	`true | false`
Default value:	`true`
Requirements:	Optional element.

Parent elements:	`weblogic-enterprise-bean`
	    `reference-descriptor`
	        `ejb-reference-description`

Function

By default, EJB methods called from within the same application (EAR) pass arguments by reference. This increases the performance of method invocation because parameters are not copied.

If you set `enable-call-by-reference` to `False`, parameters to the EJB methods are copied (pass-by-value) in accordance with the EJB 1.1 specification. Pass by value is always necessary when the EJB is called remotely (not from within the same application).

Example

The following example enables pass-by-value for EJB methods:

```
<weblogic-enterprise-bean>

       <ejb-name>AccountBean</ejb-name>

       ...

       <enable-call-by-reference>false</enable-call-by-reference>

</weblogic-enterprise-bean>
```

enable-dynamic-queries

Range of values:	`true | false`
Default value:	`true`
Requirements:	Optional element.
Parent elements:	`weblogic-enterprise-bean` `entity-descriptor`

Function

The optional enable-dynamic-queries element must be set to true to enable dynamic queries. Dynamic queries are only available for use with EJB 2.0 CMP beans.

Example

The following example enables dynamic queries:

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

entity-cache

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	The `entity-cache` stanza is optional, and is valid only for entity EJBs.
Parent elements:	`weblogic-enterprise-bean,` `entity-descriptor`

Function

The `entity-cache` element defines the following options used to cache entity EJB instances within WebLogic Server:

- `max-beans-in-cache`
- `idle-timeout-seconds`
- `read-timeout-seconds`
- `concurrency-strategy`

See "EJB Life Cycle" on page 4-2 for a general discussion of the caching services available in WebLogic Server.

Example

The `entity-cache` stanza is shown here:

```
<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:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	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.
Parent elements:	`weblogic-enterprise-bean,` `entity-descriptor` `entity-cache-ref`

Function

The `entity-cache-name` element 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.

For more information about the `weblogic-application.xml` file, see the application deployment descriptors.

Example

See "entity-cache-ref" on page 9-33.

entity-cache-ref

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	The `entity-cache-name` element in the `entity-cache-ref` stanza must contain the name of the application level cache.
Parent elements:	`weblogic-enterprise-bean,` `entity-descriptor`

Function

The `entity-cache-ref` element 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 the "concurrency-strategy" on page 9-18 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 banes with a `concurrency-strategy` of Exclusive. While a MultiVersion cache supports the Database, ReadOnly, and Optimistic concurrency strategies.

Example

The `entity-cache-ref` stanza is shown here:

```
<entity-cache-ref>
       <entity-cache-name>AllEntityCache</entity-cache-name>
       <concurrency-strategy>ReadOnly</concurrency-strategy>
       <estimated-bean-size>20</estimated-bean-size>
</entity-cache-ref>
```

entity-clustering

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Optional element. Valid only for entity EJBs in a cluster.
Parent elements:	`weblogic-enterprise-bean,` `entity-descriptor`

Function

The `entity-clustering` element uses the following options to specify how an entity bean will be replicated in a WebLogic cluster:

- `home-is-clusterable`
- `home-load-algorithm`
- `home call-router-class-name`

Example

The following excerpt shows the structure of a `entity-clustering` stanza:

```
<entity-clustering>

       <home-is-clusterable>true</home-is-clusterable>

       <home-load-algorithm>random</home-load-algorithm>


<home-call-router-class-name>beanRouter</home-call-router-class-n
ame>

</entity-clustering>
```

entity-descriptor

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	One `entity-descriptor` stanza is required for each entity EJB in the *.jar*.
Parent elements:	`weblogic-enterprise-bean`

Function

The `entity-descriptor` element specifies the following deployment parameters that are applicable to an entity bean:

- `pool`
- `entity-cache`
- `persistence`
- `entity-clustering`

Example

The following example shows the structure of the `entity-descriptor` stanza:

```
<entity-descriptor>

       <entity-cache>...</entity-cache>

       <persistence>...</persistence>

       <entity-clustering>...</entity-clustering>

</entity-descriptor>
```

estimated-bean-size

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	n/a

Parent elements:	`weblogic-enterprise-bean`
	 `entity-descriptor`

Function

The `estimated-bean-size-` element 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 ad 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 9-33.

externally-defined

Range of values:	`True | False`
Default value:	
Requirements:	n/a
Parent elements:	`weblogic-enterprise-bean` `security-role-assignment`

Function

The `externally-defined` element indicates that a particular security role is defined externally in a security realm, outside of the deployment descriptor. Because the security role and it's 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`
Requirements:	Optional element. Valid only for CMP entity EJBs.
Parent elements:	`weblogic-enterprise-bean,` `entity-descriptor,` `persistence`

Function

The `finders-load-bean` element determines whether WebLogic Server loads the EJB into the cache after a call to a finder method returns a reference to the bean. If you set this element to `true`, WebLogic Server immediately loads the bean into the cache if a reference to a bean is returned by the finder. If you set this element to `false`, WebLogic Server does not automatically load the bean into the cache until the first method invocation; this behavior is consistent with the EJB 1.1 specification.

Example

The following entry specifies that EJBs are loaded into the WebLogic Server cache automatically when a finder method returns a reference to the bean:

```
<entity-descriptor>

       <persistence>

              <finders-load-bean>true</finders-load-bean>

       </persistence>

</entity-descriptor>
```

global-role

Range of values:	`True	False`
Default value:	`True`
Requirements:	
Parent elements:	`weblogic-enterprise-bean` `security-role-assignment`

Function

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 router class name
Default value:	null
Requirements:	Optional element. Valid only for entity EJBs, stateful session EJBs, and stateless session EJBs in a cluster.

Parent elements:	`weblogic-enterprise-bean,`
	`    entity-descriptor,`
	`        entity-clustering`
	`and`
	`weblogic-enterprise-bean`
	`    stateful-session-descriptor`
	`        stateful-session-clustering`

Function

`home-call-router-class-name` specifies the name of a custom class to use for routing bean method calls. This class must implement `weblogic.rmi.cluster.CallRouter()`. If specified, an instance of this class is called before each method call. The router class has the opportunity to choose a server to route to based on the method parameters. The class returns either a server name or null, which indicates that the current load algorithm should select the server.

Example

See "entity-clustering" on page 9-34 and "stateful-session-clustering" on page 9-85.

home-is-clusterable

Range of values:	`true | false`
Default value:	`true`
Requirements:	Optional element. Valid for entity EJBs, stateless session EJBs, and stateful session EJBs in a cluster.

Parent elements:	`weblogic-enterprise-bean,`
	`     entity-descriptor,`
	`          entity-clustering`
	`and`
	`weblogic-enterprise-bean`
	`     stateful-session-descriptor`
	`          stateful-session-clustering`
	`and`
	`weblogic-enterprise-bean`
	`     stateless-session-descriptor`
	`          stateless-clustering`

Function

Use `home-is-clusterable` to specify whether the home interface of an entity, stateless session, or stateful session bean is clustered.

When `home-is-clusterable` is `true` for an EJB deployed to a cluster, each server instance binds the bean's home interface to its cluster JNDI tree under the same name. When a client requests the bean's home from the cluster, the server instance that does the look-up returns a `EJBHome` stub that has a reference to the home on each server.

When the client issues a `create()` or `find()` call, the stub routes selects a server from the replica list in accordance with the load balancing algorithm, and routes the call to the home interface on that server. The selected home interface receives the call, and creates a bean instance on that server instance and executes the call, creating an instance of the bean.

Example

See “entity-clustering” on page 9-34.

home-load-algorithm

Range of values:	`round-robin	random	weight-based`
Default value:	Value of `weblogic.cluster.defaultLoadAlgorithm`
Requirements:	Optional element. Valid only for entity EJBs and stateful session EJBs in a cluster.
Parent elements:	`weblogic-enterprise-bean,` `entity-descriptor,` `entity-clustering` and `weblogic-enterprise-bean` `stateful-session-descriptor` `stateful-session-clustering`

Function

`home-load-algorithm` specifies the algorithm to use for load balancing between replicas of the EJB home. If this 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.

Example

See "entity-clustering" on page 9-34 and "stateful-session-clustering" on page 9-85.

idempotent-methods

Range of values:	`n/a`
Default value:	`n/a`
Requirements:	Clustering must be enabled for the EJB.
Parent elements:	`weblogic-enterprise-bean`

Function

The `idempotent-methods` element defines list of methods 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.

To enable clustering, see "entity-clustering" on page 9-34, "stateful-session-clustering" on page 9-85, and "stateless-clustering" on page 9-91.

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

The `method` stanza can contain the elements shown here:

```
<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:	
Requirements:	n/a
Parent elements:	`weblogic-enterprise-bean` `iiop-security-descriptor`

Function

The `identity-assertion` element specifies whether the EJB supports or requires identity assertion.

Example

See "iiop-security-descriptor" on page 9-46.

idle-timeout-seconds

Range of values:	`1` to *`maxSeconds`*, where *`maxSeconds`* is the maximum value of an int.
Default value:	`600`
Requirements:	Optional element

Parent elements:	`weblogic-enterprise-bean,`
	     `entity-descriptor,`
	          `entity-cache`
	`and`
	`weblogic-enterprise-bean,`
	     `stateful-session-descriptor,`
	          `stateful-session-cache`

Function

`idle-timeout-seconds` defines the maximum length of time a stateful 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 "EJB Life Cycle" on page 4-2 for more information.

Note: Although `idle-timeout-seconds` appears in the `entity-cache` stanza, WebLogic Server 7. 0 SP01, SP02, SP03, and SP04 do not use its value in managing the lifecycle of entity EJBs—in those service pecks, `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`
Requirements:	n/a
Parent elements:	`weblogic-enterprise-bean`

Function

The `iiop-security-descriptor` element specifies security configuration parameters at the bean-level. These parameters determine the IIOP security information contained in the IOR.

Example

The `iiop-security-descriptor` stanza can contain the elements shown here

```
<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`
Requirements:	Optional element. Valid for stateless session, entity, and message-driven EJBs.

Parent elements:	`weblogic-enterprise-bean,`
	 `stateless-session-descriptor, message-bean-descriptor, entity-descriptor`
	 `pool`

Function

If you specify a value for `initial-beans-in-free-pool`, you set the initial size of the pool. WebLogic Server populates the free pool with the specified number of bean instances for every bean class at startup. Populating the free pool in this way improves initial response time for the EJB, because initial requests for the bean can be satisfied without generating a new instance.

Example

See “pool” on page 9-69.

initial-context-factory

Range of values:	`true | false`
Default value:	`weblogic.jndi.WLInitialContextFactory`
Requirements:	Requires the server to throw a `RemoteException` when a stateful session bean instance is currently handling a method call and another (concurrent) method call arrives on the server.

Parent elements:	`weblogic-enterprise-bean`
	    `message-driven-descriptor`

Function

The `initial-context-factory` element specifies the initial contextFactory that the container will use to create its connection factories. If initial-context-factory is not specified, the default will be `weblogic.jndi.WLInitialContextFactory`.

Example

The following example specifies the `initial-context-factory` element.

```
<message-driven-descriptor>

<initial-context-factory>weblogic.jndi.WLInitialContextFactory
</initial-context-factory>

</message-driven-descriptor>
```

integrity

Range of values:	`none	supported	required`
Default value:	
Requirements:	n/a.

Parent elements:	`weblogic-enterprise-bean`
	    `iiop-security-descriptor`
	        `transport-requirements`

Function

The `integrity` element specifies the transport integrity requirements for he 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 9-95.

invalidation-target

Range of values:	
Default value:	
Requirements:	The target `ejb-name` must be a Read-Only entity EJB and this element can only be specified for an EJB 2.0 container-managed persistence entity EJB.

Parent elements:	`weblogic-enterprise-bean`
	 `entity-descriptor`

Function

The invalidation-target element specifies a Read-Only entity EJB that should be invalidated when this container-managed persistence entity EJB has been modified.

Example

The following entry specifies that the EJB named `StockReaderEJB` should be invalidated when the EJB has been modified.

```
<invalidation-target>

       <ejb-name>StockReaderEJB</ejb-name>

</invalidation-target>
```

is-modified-method-name

Range of values:	Valid entity EJB method name
Default value:	None
Requirements:	Optional element. Valid only for entity EJBs.
Parent elements:	`weblogic-enterprise-bean,` `entity-descriptor,` `persistence`

Function

`is-modified-method-name` specifies a method that WebLogic Server calls when the EJB is stored. The specified method must return a `boolean` value. If no method is specified, WebLogic Server always assumes that the EJB has been modified and always saves it.

Providing a method and setting it as appropriate can improve performance for EJB 1.1-compliant beans, and for beans that use bean-managed persistence. However, any errors in the method's return value can cause data inconsistency problems. See "Using is-modified-method-name to Limit Calls to ejbStore() (EJB 1.1 Only)" on page 4-12 for more information.

Note: `isModified()` is no longer required for 2.0 CMP entity EJBs based on the EJB 2.0 specification However, it still applies to BMP and 1.1 CMP EJBs. When you deploy EJB 2.0 entity beans with container-managed persistence, WebLogic Server automatically detects which EJB fields have been modified, and writes only those fields to the underlying datastore.

Example

The following entry specifies that the EJB method named `semidivine` will notify WebLogic Server when the EJB has been modified:

```
<entity-descriptor>

       <persistence>

<is-modified-method-name>semidivine</is-modified-method-name>

       </persistence>

</entity-descriptor>
```

isolation-level

Range of values:	`TransactionSerializable | TransactionReadCommitted | TransactionReadUncommitted | TransactionRepeatableRead | TransactionReadCommittedForUpdate | TransactionReadCommittedForUpdateNoWait`
Default value:	default setting of the underlying database
Requirements:	Optional element.
Parent elements:	`weblogic-ejb-jar` `transaction-isolation`

Function

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 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) beans 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 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, for container-managed persistence (CMP) beans 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.

  See "Setting Container-Managed Transaction Isolation Levels" on page 4-41 for background information on the Oracle-specific `isolation-level` values.

Refer to your database documentation for more information support for different isolation levels.

Example

See “transaction-isolation” on page 9-94.

jms-polling-interval-seconds

Range of values:	n/a
Default value:	10 seconds
Requirements:	n/a
Parent elements:	`weblogic-enterprise-bean`

Function

The `jms-polling-interval-seconds` 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

The following entry specifies the jms polling intervals for message-driven beans:

```
<jms-polling-interval-seconds>5</jms-polling-interval seconds>
```

jms-client-id

Range of values:	n/a
Default value:	The default client identifier is the `ejb-name` for this EJB.
Requirements:	The `jms-client-id` is necessary for durable subscriptions to JMS topics.
Parent elements:	`weblogic-enterprise-bean`

Function

The `jms-client-id` specifies an associated id for the JMS consumers. A message-driven bean with a durable subscription needs an associated client id. If you use a separate connection factory, you can set the client id on the connection factory. In this case, the message-driven bean uses this client id.

If the associated connection factory does not have a client id or if you use the default connection factory, then the message-driven bean used the jms-client-id value as its client id.

Example

The following entry specifies an associated id for JMS consumers:

```
<jms-client-id>MyClientID</jms-client-id>
```

jndi-name

Range of values:	Valid JNDI name
Default value:	n/a
Requirements:	Required in `resource-description` and `ejb-reference-description`.

Parent elements:	`weblogic-enterprise-bean`
	 `and`
	 `weblogic-enterprise-bean`
	      `reference-descriptor`
	           `resource-description`
	 `and`
	 `weblogic-enterprise-bean`
	      `reference-descriptor`
	           `ejb-reference-description`

Function

`jndi-name` specifies the JNDI name of an actual EJB, resource, or reference available in WebLogic Server.

Example

See "resource-description" on page 9-77 and "ejb-reference-description" on page 9-26.

local-jndi-name

Range of values:	Valid JNDI name
Default value:	n/a
Requirements:	Required if the bean has a local home.
Parent elements:	`weblogic-enterprise-bean`

Function

The `local-jndi-name` element specifies a jndi-name for a bean's local home. If a bean has both a remote and a local home, then it must have two JNDI names; one for each home.

Example

The following example shows the specifies the `local-jndi-name` element.

```
<local-jndi-name>weblogic.jndi.WLInitialContext
</local-jndi-name>
```

max-beans-in-cache

Range of values:	`1` to *`maxBeans`*
Default value:	`1000`
Requirements:	Optional element

Parent elements:	`weblogic-enterprise-bean,`
	`     entity-descriptor,`
	`          entity-cache`
	`and`
	`weblogic-enterprise-bean`
	`     stateful-session-descriptor`
	`          stateful-session-cache`

Function

The `max-beans-in-cache` element specifies the maximum number of objects of this class that are allowed in memory. When `max-bean-in-cache` is reached, WebLogic Server passivates some EJBs that have not recently been used by a client. `max-beans-in-cache` also affects when EJBs are removed from the WebLogic Server cache, as described in "EJB Concurrency Strategy" on page 4-14.

Example

The following entry enables WebLogic Server to cache a maximum of 200 instances of the `AccountBean` class:

```
<weblogic-enterprise-bean>

        <ejb-name>AccountBean</ejb-name>

        <entity-descriptor>

                <entity-cache>
                <max-beans-in-cache>200</max-beans-in-cache>
            </entity-cache>
        </entity-descriptor>
    </weblogic-enterprise-bean>
```

max-beans-in-free-pool

Range of values:	`0` to *`maxBeans`*
Default value:	1000
Requirements:	Optional element.

Parent elements:	`weblogic-enterprise-bean,`
	`     stateless-session-descriptor,`
	`          pool`
	`weblogic-enterprise-bean,`
	`     message-bean-descriptor,`
	`          pool`
	`weblogic-enterprise-bean,`
	`     entity-descriptor,`
	`          pool`

Function

WebLogic Server maintains a free pool of EJBs for every entity, stateless session, and message-driven bean class. The `max-beans-in-free-pool` element defines the size of this pool. See "Stateless Session EJB Life Cycle" on page 4-6 and "Differences Between Message-Driven Beans and Stateless Session EJBs" on page 3-3 for more information.

Example

See "pool" on page 9-69.

message-driven-descriptor

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	
Parent elements:	`weblogic-enterprise-bean`

Function

The `message-driven-descriptor` element associates a message-driven bean with a JMS destination in WebLogic Server. This element specifies the following deployment parameters:

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

Example

The following example shows the structure of the `message-driven-descriptor` stanza:

```
<message-driven-descriptor>
```

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

method

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Optional element. You can specify more than one `method` stanza to configure multiple EJB methods.

Parent elements:	`weblogic-ejb-jar`
	 `transaction-isolation`

Function

The `method` element defines a method or set of methods for an enterprise bean's home or remote interface.

Example

The `method` stanza can contain the elements shown here:

```
<method>
        <description>...</description>
        <ejb-name>...</ejb-name>
        <method-intf>...</method-intf>
        <method-name>...</method-name>
        <method-params>...</method-params>
</method>
```

method-intf

Range of values:	`Home	Remote	Local	Localhome`
Default value:	n/a
Requirements:	Optional element.

Parent elements:	`weblogic-ejb-jar`
	`transaction-isolation`
	`method`

Function

`method-intf` specifies the EJB interface to which WebLogic Server applies isolation level properties, if the method has the same signature in multiple interfaces.

Example

See "method" on page 9-61.

method-name

Range of values:	Name of an EJB defined in `ejb-jar.xml	*`
Default value:	n/a
Requirements:	Required element in method stanza.

Parent elements:	`weblogic-ejb-jar`
	`    transaction-isolation`
	`        method`

Function

`method-name` specifies the name of an individual EJB method to which WebLogic Server applies isolation level properties. Use the asterisk (*) to specify all methods in the EJB's home and remote interfaces.

If you specify a `method-name`, the method must be available in the specified `ejb-name`.

Example

See "method" on page 9-61.

method-param

Range of values:	Fully qualified Java type of a method parameter
Default value:	n/a
Requirements:	Required element in `method-params`.

Parent elements:	`weblogic-ejb-jar`
	    `transaction-isolation`
	        `method`
	            `method-params`

Function

The `method-param` element specifies the fully qualified Java type name of a method parameter.

Example

See "method-params" on page 9-65.

method-params

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Optional stanza.

Parent elements:	`weblogic-ejb-jar`
	`     transaction-isolation`
	`          method`

Function

The `method-params` stanza contains one or more elements that define the Java type name of each of the method's parameters.

Example

The `method-params` stanza contains one or more `method-param` elements, as shown here:

```
<method-params>

       <method-param>java.lang.String</method-param>

       ...

</method-params>
```

persistence

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Required only for entity EJBs that use container-managed persistence services.
Parent elements:	`weblogic-enterprise-bean,` `entity-descriptor`

Function

The `persistence` element defines the following options that determine the persistence type, transaction commit behavior, and `ejbLoad()` and `ejbStore()` behavior for entity EJBs in WebLogic Server:

- `is-modified-method-name`
- `delay-updates-until-end-of-tx`
- `finders-load-bean`
- `db-is-shared`
- `persistence-use`

Example

The following example specifies the `persistence` element.

```
<entity-descriptor>
       <persistence>
              <is-modified-method-name>...</is-modified-
              method-name>
              <delay-updates-until-end-of-tx>
              </delay-updates-until-end-of-tx>
              <finders-load-beand>...</finders-load-bean>
              <db-is-shared>false</db-is-shared>
              <persistence-use>...</persistence-use>
```

```
        </persistence>
</entity-descriptor>
```

persistence-use

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Required only for entity EJBs that use container-managed persistence services.
Parent elements:	`weblogic-enterprise-bean,` `entity-descriptor,` `persistence`

Function

The `persistence-use` element stores an identifier of the persistence type to be used for this particular bean.

Example

The following excerpt shows a sample `persistence-use` stanza:

```
<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

Default value:	pstore
Requirements:	Optional element.

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 4-10.

Example

```
<stateful-session-descriptor>
   <stateful-session-cache>...</stateful-session-cache>
   <allow-concurrent-calls>...</allow-concurrent-calls>
   <persistent-store-dir>MyPersistenceDirr</persistent-store-dir>
   <stateful-session-clustering>...</stateful-session-clustering>
   <allow-remove-during-transaction>
</stateful-session-descriptor>
```

pool

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Optional element.
Parent elements:	`weblogic-enterprise-bean` `stateless-session-descriptor, message-bean-descriptor, entity-descriptor`

Function

The `pool` element configures the behavior of the WebLogic Server free pool for EJBs.

Example

The `pool` stanza can contain the elements shown here:

```
<stateless-session-descriptor>
   <pool>
     <max-beans-in-free-pool>500</max-beans-in-free-pool>
     <initial-beans-in-free-pool>250</initial-beans-in-free-pool>
   </pool>
</stateless-session-descriptor>
```

principal-name

Range of values:	valid WebLogic Server principal name
Default value:	n/a
Requirements:	At least one `principal-name` is required in the security-role-assignment stanza. You may define more than one `principal-name` for each role-name.
Parent elements:	`weblogic-enterprise-bean` `security-role-assignment`

Function

`principal-name` specifies the name of an actual WebLogic Server principal to apply to the specified `role-name`.

Example

See "security-role-assignment" on page 9-82.

provider-url

Range of values:	valid name
Default value:	n/a
Requirements:	Used in conjunction with `initial-context-factory` and `connection-factory-jndi-name`.

Parent elements:	`weblogic-enterprise-bean`
	    `message-driven-descriptor`

Function

The `provider-url` element specifies the URL provider to be used by the `InitialContext`. Typically, this is the host:port and used in conjunction with `initial-context-factory` and `connection-factory-jndi-name`.

Example

The following example specifies the `provider-url` element.

```
<message-driven-descriptor>

<provider-url>WeblogicURL:Port</provider-url>

</message-driven-descriptor>
```

read-timeout-seconds

Range of values:	`0` to *`maxSeconds`*, where *`maxSeconds`* is the maximum value of an int.
Default value:	`600`
Requirements:	Optional element. Valid only for entity EJBs.
Parent elements:	`weblogic-enterprise-bean,` `entity-descriptor,` `entity-cache`

Function

The `read-timeout-seconds` element specifies the number of seconds between `ejbLoad()` calls on a `Read-Only` entity bean. A setting of 0 causes WebLogic Server to calls `ejbLoad()` only when the bean is brought into the cache.

Example

The following entry causes WebLogic Server to call `ejbLoad()` for instances of the `AccountBean` class only when the instance is first brought into the cache:

```
<weblogic-enterprise-bean>
    <ejb-name>AccountBean</ejb-name>
        <entity-descriptor>
            <entity-cache>
                <read-timeout-seconds>0</read-timeout-seconds>
            </entity-cache>
        </entity-descriptor>
</weblogic-enterprise-bean>
```

reference-descriptor

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Optional element.
Parent elements:	`weblogic-enterprise-bean`

Function

The `reference-descriptor` element maps references in the `ejb-jar.xml` file to the JNDI names of actual resource factories and EJBs available in WebLogic Server.

Example

The `reference-descriptor` stanza contains one or more additional stanzas to define resource factory references and EJB references. The following shows the organization of these elements:

```
<reference-descriptor>

       <resource-description>

              ...

       </resource-description>

       <ejb-reference-description>

              ...

       </ejb-reference-description>

</reference-descriptor>
```

relationship-description

This element is no longer supported in WebLogic Server.

replication-type

Range of values:	`InMemory	None`
Default value:	`None`
Requirements:	Optional element. Valid only for stateful session EJBs in a cluster.

Parent elements:	`weblogic-enterprise-bean`
	    `stateful-session-descriptor`
	        `stateful-session-clustering`

Function

The `replication-type` element determines whether WebLogic Server replicates the state of stateful session EJBs across WebLogic Server instances in a cluster. If you select `InMemory`, the state of the EJB is replicated. If you select `None`, the state is not replicated.

See "In-Memory Replication for Stateful Session EJBs" on page 4-34 for more information.

Example

See "stateful-session-clustering" on page 9-85.

res-env-ref-name

Range of values:	A valid resource environment reference name from the `ejb-jar.xml` file
Default value:	n/a
Requirements:	n/a

Parent elements:	`weblogic-enterprise-bean`
	`    reference-descriptor`
	`        resource-env-description`

Function

The `res-env-ref-name` element specifies the name of a resource environment reference.

Example

See "resource-description" on page 9-77.

res-ref-name

Range of values:	A valid resource reference name from the `ejb-jar.xml` file
Default value:	n/a
Requirements:	Required element if the EJB specifies resource references in `ejb-jar.xml`

Parent elements:	`weblogic-enterprise-bean`
	    `reference-descriptor`
	        `resource-description`

Function

The `res-ref-name` element specifies the name of a `resourcefactory` reference. This is the reference that the EJB provider places within the `ejb-jar.xml` deployment file.

Example

See "resource-description" on page 9-77.

resource-description

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Optional element.
Parent elements:	`weblogic-enterprise-bean` `reference-descriptor`

Function

The `resource-description` element maps a resource reference defined in `ejb-jar.xml` to the JNDI name of an actual resource available in WebLogic Server.

Example

The `resource-description` stanza can contain additional elements as shown here:

```
<reference-descriptor>

       <resource-description>

              <res-ref-name>. . .</res-ref-name>

              <jndi-name>...</jndi-name>

       </resource-description>

       <ejb-reference-description>

              <ejb-ref-name>. . .</ejb-ref-name>

              <jndi-name>. . .</jndi-name>

       </ejb-reference-description>

</reference-descriptor>
```

resource-env-description

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Optional element.
Parent elements:	`weblogic-enterprise-bean` `reference-descriptor`

Function

The `resource-env-description` element maps a resource environment reference defined in `ejb-jar.xml` to the JNDI name of an actual resource available in WebLogic Server.

Example

The `resource-env-description` stanza can contain additional elements as shown here:

```
<reference-descriptor>
       <resource-env-description>
              <res-env-ref-name>. . .</res-env-ref-name>
              <jndi-name>...</jndi-name>
       <reference-env-description>
</reference-descriptor>
```

role-name

Range of values:	An EJB role name defined in `ejb-jar.xml`
Default value:	n/a
Requirements:	Required element in `security-role-assignment`.

Parent elements:	`weblogic-enterprise-bean`
	    `security-role-assignment`

Function

The `role-name` element identifies an application role name that the EJB provider placed in the `ejb-jar.xml` deployment file. Subsequent `principal-name` elements in the stanza map WebLogic Server principals to the specified `role-name`.

Example

See "security-role-assignment" on page 9-82.

security-permission

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	.
Parent elements:	n/a

Function

The `security-permission` element 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

The `security-permission` stanza can contain one or more of the following elements:

```
<security-permission> </security-permission>
```

security-permission-spec

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	n/a
Parent elements:	security-permission

Function

The `security-permission-spec` element specifies a security permission 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

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 (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Required element if `ejb-jar.xml` defines application roles.
Parent elements:	`n/a`

Function

The `security-role-assignment` element maps application roles in the `ejb-jar.xml` file to the names of security principals available in WebLogic Server.

Example

The `security-role-assignment` stanza can contain one or more of the following elements:

```
<security-role-assignment>
       <role-name>PayrollAdmin</role-name>
       <principal-name>Tanya</principal-name>
       <principal-name>system</principal-name>
       ...
</security-role-assignment>
```

session-timeout-seconds

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	The `session-timeout-seconds` stanza is valid only for stateful session EJBs.

Parent elements:	`weblogic-enterprise-bean,`
	    `stateful-session-descriptor`
	      `stateful-session-cache`

Function

The `session-timeout-seconds` element specifies how long the EJB container waits before removing a passivated stateful session EJB from disk.

The `idle-timeout-seconds` element determines how long the EJB container waits before passivating stateful session beans, that is, removing them from cache and writing them to disk.

In past releases, the EJB container also used `idle-timeout-seconds` to determine how long to wait before removing passivated EJBs from the disk. With the addition of `session-timeout-seconds`, you can specify how long stateful session beans stay idle in the cache and how long they stay idle on disk using two different elements.

Example

The following example shows how to specify the `session-timeout-seconds` element

```
<session-timeout-seconds>3600</session-timeout-seconds>
```

stateful-session-cache

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	The `stateful-session-cache` stanza is optional, and is valid only for stateful session EJBs.
Parent elements:	`weblogic-enterprise-bean,` `stateful-session-descriptor`

Function

The `stateful-session-cache` element defines the following options used to cache stateful session EJB instances within WebLogic Server.

- `max-beans-in-cache`
- `idle-timeout-seconds`
- `cache-type`

See "EJB Life Cycle" on page 4-2 for a general discussion of the caching services available in WebLogic Server.

Example

The following example shows how to specify the `stateful-session-cache` element

```
<stateful-session-cache>

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

        <idle-timeout-seconds>...</idle-timeout-seconds>

        <cache-type>...</cache-type>

</stateful-session-cache>
```

stateful-session-clustering

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Optional element. Valid only for stateful session EJBs in a cluster.
Parent elements:	`weblogic-enterprise-bean, stateful-session-descriptor`

Function

The `stateful-session-clustering` stanza element specifies clustering behaviors for a stateful session EJB instances in a cluster:

Example

The following excerpt shows the structure of a `entity-clustering` stanza:

```
<stateful-session-clustering>

       <home-is-clusterable>true</home-is-clusterable>

       <home-load-algorithm>random</home-load-algorithm>


<home-call-router-class-name>beanRouter</home-call-router-class-n
ame>

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

</stateful-session-clustering>
```

stateful-session-descriptor

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	One `stateful-session-descriptor` stanza is required for each stateful session EJB in the `.jar`.
Parent elements:	`weblogic-enterprise-bean`

Function

The `stateful-session-descriptor` element specifies deployment behavior for a stateful session EJB:

Example

The following example shows the structure of the `stateful-session-descriptor` stanza:

```
<stateful-session-descriptor>
   <stateful-session-cache>...</stateful-session-cache>
   <allow-concurrent-calls>...</allow-concurrent-calls>
   <persistent-store-dir>...</persistent-store-dir>
   <stateful-session-clustering>...</stateful-session-clustering>
   <allow-remove-during-transaction>...
   </allow-remove-during-transaction>
</stateful-session-descriptor>
```

stateless-bean-call-router-class-name

Range of values:	Valid router class name
Default value:	n/a
Requirements:	Optional element. Valid only for stateless session EJBs in a cluster.
Parent elements:	`weblogic-enterprise-bean,` `stateless-session-descriptor` `stateless-clustering`

Function

The `stateless-bean-call-router-class-name` element specifies the name of a custom class to use for routing bean method calls. This class must implement `weblogic.rmi.cluster.CallRouter()`. If specified, an instance of this class is called before each method call. The router class has the opportunity to choose a server to route to based on the method parameters. The class returns either a server name or null, which indicates that the current load algorithm should select the server.

Example

See "stateless-clustering" on page 9-91.

stateless-bean-is-clusterable

Range of values:	`true | false`
Default value:	`true`
Requirements:	Optional element. Valid only for stateless session EJBs in a cluster.

Parent elements:	`weblogic-enterprise-bean,`
	`    stateless-session-descriptor`
	`        stateless-clustering`

Function

Use `stateless-bean-is-clusterable` to specify whether a stateless session bean's `EJBObject` interface is clustered. Clustered `EJBObjects` support load balancing and failover.

If `stateless-bean-is-clusterable` is `true`, when a home interface of a clustered stateless session bean creates a bean instance, it returns a `EJBObject` stub to the client that lists all of the servers in the cluster. Given the stateless nature of the bean, any instance can service any client request.

Example

See "stateless-clustering" on page 9-91.

stateless-bean-load-algorithm

Range of values:	`round-robin | random | weight-based`
Default value:	Value of `weblogic.cluster.defaultLoadAlgorithm`
Requirements:	Optional element. Valid only for stateless session EJBs in a cluster.
Parent elements:	`weblogic-enterprise-bean,` `stateless-session-descriptor` `stateless-clustering`

Function

`stateless-bean-load-algorithm` specifies the algorithm to use for load balancing between replicas of the EJB home. If this property is not defined, WebLogic Server uses the algorithm specified by the server property, `weblogic.cluster.defaultLoadAlgorithm`.

You can define `stateless-bean-load-algorithm` as one of the following values:

- `round-robin`: Load balancing is performed in a sequential fashion among the servers hosting the bean.
- `random`: Replicas of the EJB home are deployed randomly among the servers hosting the bean.
- `weight-based`: Replicas of the EJB home are deployed on host servers according to the servers' current workload.

Example

See "stateless-clustering" on page 9-91.

stateless-bean-methods-are-idempotent

Range of values:	`true | false`
Default value:	`false`
Requirements:	Optional element. Valid only for stateless session EJBs in a cluster.
Parent elements:	`weblogic-enterprise-bean,` `stateless-session-descriptor` `stateless-clustering`

Function

DEPRECATED: The stateless-bean-methods-are-idempotent element is now deprecated and will be removed in a future version of WebLogic.

Please use the `idempotent-methods` element instead.

Example

See "stateless-clustering" on page 9-91.

stateless-clustering

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Optional element. Valid only for stateless session EJBs in a cluster.
Parent elements:	`weblogic-enterprise-bean,` `stateless-session-descriptor`

Function

The `stateless-clustering` element specifies the 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>
   <home-is-clusterable>.../home-is-clusterable>
   <home-load-algorithm>...</home-load-algorithm>
   <home-call-router-class-name>...</home-call-router-class-name>
   <use-serverside-stubs>...</use-serverside-stubs>
   <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>
</stateless-clustering>
```

stateless-session-descriptor

Requirements:	One `stateless-session-descriptor` element is required for each stateless session EJB in the JAR file.
Parent elements:	`weblogic-enterprise-bean`

Function

The `stateless-session-descriptor` element defines deployment behaviors, such as caching, clustering, and persistence, for stateless session EJBs in WebLogic Server.

Example

The following example shows the structure of the `stateless-session-descriptor` stanza:

```
<stateless-session-descriptor>
       <pool>...</pool>
       <stateless-clustering>...</stateless-clustering>
</stateless-session-descriptor>
```

transaction-descriptor

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Optional element.
Parent elements:	`weblogic-enterprise-bean`

Function

The `transaction-descriptor` element specifies options that define transaction behavior in WebLogic Server. Currently, this stanza includes only one element: `trans-timeout-seconds`.

Example

The following example shows the structure of the `transaction-descriptor` stanza:

```
<transaction-descriptor>

       <trans-timeout-seconds>20</trans-timeout-seconds>

</transaction-descriptor>
```

transaction-isolation

Range of values:	n/a (XML stanza)
Default value:	n/a
Requirements:	Optional element.
Parent elements:	`weblogic-ejb-jar`

Function

The `transaction-isolation` element specifies method-level transaction isolation settings for an EJB. Example

The `transaction-isolation` stanza can contain the elements shown here:

```
<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 9-52.

transport-requirements

Range of values:	`n/a`
Default value:	`n/a`
Requirements:	n/a
Parent elements:	`weblogic-enterprise-bean,` `iiop-security-descriptor`

Function

The `transport-requirements` element provides the transport requirements for the EJB.

Example

The `transport-requirements` stanza can contain the elements shown here

```
<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`
Requirements:	Optional element. Valid for any type of EJB.

Parent elements:	`weblogic-enterprise-bean,`
	`transaction-descriptor`

Function

The `trans-timeout-seconds` element specifies the maximum duration for an EJB's container-initiated transactions. If a transaction lasts longer than `trans-timeout-seconds`, WebLogic Server rolls back the transaction.

Example

See "transaction-descriptor" on page 9-93.

type-identifier

Range of values:	Valid string. `WebLogic_CMP_RDBMS` specifies WebLogic Server RDBMS-based persistence.
Default value:	n/a
Requirements:	Required only for entity EJBs that use container-managed persistence services.

Parent elements:	`weblogic-enterprise-bean,`
	`    entity-descriptor,`
	`        persistence`
	`            persistence-use`

Function

The `type-identifier` element contains text that identifies an entity EJB persistence type. WebLogic Server RDBMS-based persistence uses the identifier, `WebLogic_CMP_RDBMS`. If you use a different persistence vendor, consult the vendor's documentation for information on the correct `type-identifier`.

Example

See "persistence-use" on page 9-67 for an example that shows the complete `persistence-use` definition for WebLogic Server RDBMS-based persistence.

type-storage

Range of values:	`5.1.0` for WebLogic persistence, EJB 1.1 `6.0` for WebLogic persistence, EJB 2.0.
Default value:	n/a
Requirements:	Required only for entity EJBs that use container-managed persistence services.

Parent elements:	`weblogic-enterprise-bean,`
	    `entity-descriptor,`
	        `persistence`
	            `persistence-use`

Function

The `type-storage` element defines the full path of the file that stores data for this persistence type. The path must specify the file's location relative to the top level of the EJB's JAR deployment file or deployment directory.

WebLogic Server RDBMS-based persistence generally uses an XML file named `weblogic-cmp-rdbms-jar.xml` to store persistence data for a bean. This file is stored in the `META-INF` subdirectory of the JAR file.

Example

See "persistence-use" on page 9-67 for an example that shows the complete `persistence-use` definition for WebLogic Server RDBMS-based persistence.

type-version

Range of values:	Valid string
Default value:	n/a
Requirements:	Required only for entity EJBs that use container-managed persistence services.

Parent elements:	`weblogic-enterprise-bean,`
	`    entity-descriptor,`
	`        persistence`
	`            persistence-use`

Function

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.

Note: If you use WebLogic Server RDBMS-based persistence, the specified version must *exactly* match the RDBMS persistence version for the WebLogic Server release. Specifying an incorrect version results in the error:

```
weblogic.ejb.persistence.PersistenceSetupException: Error
initializing the CMP Persistence Type for your bean: No installed
Persistence Type matches the signature of (identifier
'Weblogic_CMP_RDBMS', version 'version_number').
```

Example

See persistence-use for an example that shows the complete `persistence-use` definition for WebLogic Server RDBMS-based persistence.

weblogic-ejb-jar

Range of values:	N/A
Default value:	N/A
Requirements:	N/A
Parent elements:	N/A

Function

`weblogic-ejb-jar` is the root element of the weblogic component of the EJB deployment descriptor.

weblogic-enterprise-bean

Range of values:	
Default value:	
Requirements:	
Parent elements:	`weblogic-ejb-jar`

Function

The `weblogic-enterprise-bean` element contains the deployment information for a bean that is available in WebLogic Server.

5.1 weblogic-ejb-jar.xml Deployment Descriptor File Structure

The WebLogic Server 5.1 weblogic-ejb-jar.xml file defines the EJB document type definitions (DTD)s you use with EJB 1.1 beans. These deployment descriptor elements are WebLogic-specific. The top level elements in the WebLogic Server 5.1 weblogic-ejb-jar.xml are as follows:

- description
- weblogic-version
- weblogic-enterprise-bean
  - ejb-name
  - caching-descriptor
  - persistence-descriptor
  - clustering-descriptor
  - transaction-descriptor
  - reference-descriptor
  - jndi-name
  - transaction-isolation
- security-role-assignment

5.1 weblogic-ejb-jar.xml Deployment Descriptor Elements

The following sections describe WebLogic-Server 5.1 weblogic-ejb-jar.xml deployment descriptor elements.

caching-descriptor

The `caching-descriptor` stanza affects the number of EJBs in the WebLogic Server cache as well as the length of time before EJBs are passivated or pooled. The entire stanza, as well as each of its elements, is optional. WebLogic Server uses default values where no elements are defined.

The following is a sample `caching-descriptor` stanza that shows the caching elements described in this section:

```
<caching-descriptor>

   <max-beans-in-free-pool>500</max-beans-in-free-pool>

   <initial-beans-in-free-pool>50</initial-beans-in-free-pool>

   <max-beans-in-cache>1000</max-beans-in-cache>

   <idle-timeout-seconds>20</idle-timeout-seconds>

   <cache-strategy>Read-Write</cache-strategy>

   <read-timeout-seconds>0</read-timeout-seconds>

</caching-descriptor>
```

max-beans-in-free-pool

Note: This element is valid only for stateless session EJBs.

WebLogic Server maintains a free pool of EJBs for every bean class. This optional element defines the size of the pool. By default, `max-beans-in-free-pool` has no limit; the maximum number of beans in the free pool is limited only by the available memory. See "EJB Life Cycle" on page 4-2.

initial-beans-in-free-pool

Note: This element is valid only for stateless session EJBs.

If you specify a value for `initial-bean-in-free-pool`, WebLogic Server populates the free pool with the specified number of bean instances at startup. Populating the free pool in this way improves initial response time for the EJB, since initial requests for the bean can be satisfied without generating a new instance.

`initial-bean-in-free-pool` defaults to 0 if the element is not defined.

max-beans-in-cache

Note: This element is valid only for stateful session EJBs and entity EJBs.

This element specifies the maximum number of objects of this class that are allowed in memory. When `max-bean-in-cache` is reached, WebLogic Server passivates some EJBs that have not been recently used by a client. `max-beans-in-cache` also affects when EJBs are removed from the WebLogic Server cache, as described in "EJB Life Cycle" on page 4-2.

The default value of `max-beans-in-cache` is 100.

idle-timeout-seconds

`idle-timeout-seconds` defines the maximum length of time a stateful EJB should remain in the cache. After this time has elapsed, WebLogic Server may remove the bean instance if the number of beans in cache approaches the limit of `max-beans-in-cache`. See "EJB Life Cycle" on page 4-2 for more information.

`idle-timeout-seconds` defaults to 600.

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-use>
      <type-identifier>...</type-identifier>
      <type-version>...</type-version>
      <type-storage>...</type-storage>
   </persistence-use>
   <db-is-shared>...</db-is-shared>
   <stateful-session-persistent-store-dir>
   </stateful-session-persistent-store-dir>
   <persistence-use>...</persistence-use>
</persistence-descriptor>
```

is-modified-method-name

`is-modified-method-name` specifies a method that WebLogic Server calls when the EJB is stored. The specified method must return a `boolean` value. If no method is specified, WebLogic Server always assumes that the EJB has been modified and always saves it.

Providing a method and setting it as appropriate can improve performance. However, any errors in the method's return value can cause data inconsistency problems. See "Using is-modified-method-name to Limit Calls to ejbStore() (EJB 1.1 Only)" on page 4-12 for more information.

delay-updates-until-end-of-tx

Set this property to `true` (the default), to update the persistent store of all beans in a transaction at the completion of the transaction. This generally improves performance by avoiding unnecessary updates. However, it does not preserve the ordering of database updates within a database transaction.

If your datastore uses an isolation level of `TRANSACTION_READ_UNCOMMITTED`, you may want to allow other database users to view the intermediate results of in-progress transactions. In this case, set `delay-updates-until-end-of-tx` to `false` to update the bean's persistent store at the conclusion of each method invoke. See "ejbLoad() and ejbStore() Behavior for Entity EJBs" on page 4-12 for more information.

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

persistence-use

A `persistence-use` defines a persistence service that can be used by an EJB. You can define multiple `persistence-use` entries in `weblogic-ejb-jar.xml` for testing with multiple persistence services.

`persistence-use` includes several elements that define the properties of a service:

- `type-identifier` contains text that identifies the specified persistence type. For example, WebLogic Server RDBMS persistence uses the identifier, `WebLogic_CMP_RDBMS`.
- `type-version` identifies the version of the specified persistence type.

Note: The specified version must *exactly* match the RDBMS persistence version for the WebLogic Server release. Specifying an incorrect version results in the error:

```
weblogic.ejb.persistence.PersistenceSetupException: Error
initializing the CMP Persistence Type for your bean: No installed
Persistence Type matches the signature of (identifier
'Weblogic_CMP_RDBMS', version 'version_number').
```

- `type-storage` defines the full path of the file that stores data for this persistence type. The path must specify the file's location relative to the top level of the EJB's JAR deployment file or deployment directory.

  WebLogic Server RDBMS-based persistence generally uses an XML file named `weblogic-cmp-rdbms-jar.xml` to store persistence data for a bean. This file is stored in the `META-INF` subdirectory of the JAR file.

The following shows an example `persistence-use` stanza with values appropriate for WebLogic Server RDBMS persistence:

```
<persistence-use>

    <type-identifier>WebLogic_CMP_RDBMS</type-identifier>

    <type-version>5.1.0</type-version>

    <type-storage>META-INF\weblogic-cmp-rdbms-jar.xml</type-stora
ge>

</persistence-use>
```

db-is-shared

The `db-is-shared` element applies only to entity beans. When set to `true` (the default value), WebLogic Server assumes that EJB data could be modified between transactions and reloads data at the beginning of each transaction. When set to `false`, WebLogic Server assumes that it has exclusive access to the EJB data in the persistent store. See "Using cache-between-transactions to Limit Calls to ejbLoad()" on page 4-28 for more information.

stateful-session-persistent-store-dir

`stateful-session-persistent-store-dir` specifies the file system directory where WebLogic Server stores the state of passivated stateful session bean instances.

clustering-descriptor

The `clustering-descriptor` stanza defines the replication properties and behavior for EJBs deployed in a WebLogic Server cluster. The `clustering-descriptor` stanza and each of its elements are optional, and are not applicable to single-server systems.

The following shows all elements contained in the `clustering-descriptor` stanza:

```
<clustering-descriptor>
   <home-is-clusterable>. . .</home-is-clusterable>
   <home-load-algorithm>. . .</home-load-algorithm>
   <home-call-router-class-name>...</home-call-router-class-name>
   <stateless-bean-is-clusterable>...</stateless-bean-is-
   clusterable>
   <stateless-bean-load-algorithm>
   </stateless-bean-load-algorithm>
```

```
    <stateless-bean-call-router-class-name>
    </stateless-bean-call-router-class-name>
    <stateless-bean-methods-are-idempotent>
    </stateless-bean-methods-are-idempotent>
</clustering-descriptor>
```

home-is-clusterable

You can set this element to either `true` or `false`. When `home-is-clusterable` is true, the EJB can be deployed from multiple WebLogic Servers in a cluster. Calls to the home stub are load-balanced between the servers on which this bean is deployed, and if a server hosting the bean is unreachable, the call automatically fails over to another server hosting the bean.

home-load-algorithm

`home-load-algorithm` specifies the algorithm to use for load balancing between replicas of the EJB home. If this property is not defined, WebLogic Server uses the algorithm specified by the server property, `weblogic.cluster.defaultLoadAlgorithm`.

You can define `home-load-algorithm` as one of the following values:

- `round-robin`: Load balancing is performed in a sequential fashion among the servers hosting the bean.
- `random`: Replicas of the EJB home are deployed randomly among the servers hosting the bean.
- `weight-based`: Replicas of the EJB home are deployed on host servers according to the servers' current workload.

home-call-router-class-name

`home-call-router-class-name` specifies the custom class to use for routing bean method calls. This class must implement `weblogic.rmi.cluster.CallRouter()`. If specified, an instance of this class is called before each method call. The router class has the opportunity to choose a server to route to based on the method parameters. The class returns either a server name or null, which indicates that the current load algorithm should select the server.

stateless-bean-is-clusterable

Use `stateless-bean-is-clusterable` to specify whether a stateless session bean's `EJBObject` interface is clustered. Clustered `EJBObject`s support load balancing and failover.

If `stateless-bean-is-clusterable` is `true`, when a home interface of a clustered stateless session bean creates a bean instance, it returns a `EJBObject` stub to the client that lists all of the servers in the cluster. Given the stateless nature of the bean, any instance can service any client request

stateless-bean-load-algorithm

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>TransactionSerializable</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

Range of values:	`TransactionSerializable` | `TransactionReadCommitted` | `TransactionReadUncommitted` | `TransactionRepeatableRead` | `TransactionReadCommittedForUpdate` | `TransactionReadCommittedForUpdateNoWait`
Default value:	default setting of the underlying database
Requirements:	Optional element.
Parent elements:	`weblogic-ejb-jar` `transaction-isolation`

Function

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 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) beans 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) beans 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.

  See “Setting Container-Managed Transaction Isolation Levels” on page 4-41 for background information on the Oracle-specific `isolation-level` values.

Refer to your database documentation for more information on the implications and support for different isolation levels.

Example

See "transaction-isolation" on page 9-111.

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.

CHAPTER

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

The chapter describes both the EJB 1.1and EJB 2.0 deployment descriptor elements found in the `weblogic-cmp-rdbms-jar.xml` file, the weblogic-specific XML document type definitions (DTD) file. Use these definitions to create the WebLogic-specific `weblogic-cmp-rdbms-jar.xml` file that is part of your EJB deployment.

The following sections provide a complete reference of both versions of the WebLogic-specific XML including the DOCTYPE header information. Use these deployment descriptor elements to specify container-managed-persistence (CMP).

- EJB Deployment Descriptors
- DOCTYPE Header Information
- 2.0 weblogic-cmp-rdbms-jar.xml Deployment Descriptor File Structure
- 2.0 weblogic-cmp-rdbms-jar.xml Deployment Descriptor Elements
- 1.1 weblogic-cmp-rdbms-jar.xml Deployment Descriptor File Structure
- 1.1 weblogic-cmp-rdbms-jar.xml Deployment Descriptor Elements

EJB Deployment Descriptors

The EJB deployment descriptors provide structural and application assembly information for an enterprise bean. You specify this information by specifying values for the deployment descriptors in three EJB XML DTD files. These files are:

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

You package these three XML files with the EJB and other classes into a deployable EJB component, usually a JAR file, called `ejb.jar`.

The `ejb-jar.xml` file is based on the deployment descriptors found in Sun Microsystems's `ejb.jar.xml` file. The other two XML files are weblogic-specific files that are based on the deployment descriptors found in `weblogic-ejb-jar.xml` and `weblogic-cmp-rdbms-jar.xml`.

DOCTYPE Header Information

When editing or creating XML deployment files, it is critical to include the correct `DOCTYPE` header for each deployment file. In particular, using an incorrect `PUBLIC` element within the `DOCTYPE` header can result in parser errors that may be difficult to diagnose. The correct text for the `PUBLIC` element for each XML deployment file is as follows.

The correct text for the `PUBLIC` element for the WebLogic Server-specific `weblogic-cmp-rdbms-jar.xml` files are as follows.

XML File	PUBLIC Element String
`weblogic-cmp-rdbms -jar.xml`	`'-// BEA Systems, Inc.//DTD WebLogic 7.0.0 EJB RDBMS Persistence//EN'` `'http://www.bea.com/servers/wls700/dtd/weblogic-rdbms 20-persistence-700.dtd'`

XML File	PUBLIC Element String
`weblogic-cmp-rdbms-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'`

The correct text for the `PUBLIC` elements for the Sun Microsystem-specific `ejb-jar` files are as follows.

XML File	PUBLIC Element String
`ejb-jar.xml`	`'-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN'` `'http://java.sun.com/dtd/ejb-jar_2_0.dtd'`
`ejb-jar.xml`	`'-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN'` `'http://www.java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'`

For example, the entire `DOCTYPE` header for a `weblogic-cmp-rdbms-jar.xml` file is as follows:

```
<!DOCTYPE weblogic-cmp-rdbms-jar PUBLIC
'-//BEA Systems, Inc.//DTD WebLogic 7.0.0 EJB RDBMS
Persistence//EN'
'http://www.bea.com/servers/wls700/dtd/weblogic-rdbms20-persisten
ce-700.dtd '>
```

XML files with incorrect header information may yield error messages similar to the following, when used with a utility that parses the XML (such as `ejbc`):

```
SAXException: This document may not have the identifier 'identifier_name'
```

`identifier_name` generally includes the invalid text from the `PUBLIC` element.

Document Type Definitions (DTDs) for Validation

The contents and arrangement of elements in your XML files must conform to the Document Type Definition (DTD) for each file you use. WebLogic Server utilities ignore the DTDs embedded within the `DOCTYPE` header of XML deployment files, and instead use the DTD locations that were installed along with the server. However, the DOCTYPE header information must include a valid URL syntax in order to avoid parser errors.

Note: Most browsers do not display the contents of files having the `.dtd` extension. To view the DTD file contents in your browser, save the links as text files and view them with a text editor.

weblogic-cmp-rdbms-jar.xml

The following links provide the public DTD locations for the `weblogic-cmp-rdbms-jar.xml` deployment files used with WebLogic Server:

- For `weblogic-cmp-rdbms-jar.xml` 2.0 DTD:

  `http://www.bea.com/servers/wls700/dtd/weblogic-rdbms20-persistence-700.dtd` contains the DTD that defines container-managed persistence properties for entity EJBs. This DTD is changed from WebLogic Server Version 6.0, and you must still include a `weblogic-cmp-rdbms-jar.xml` file for entity EJBs using WebLogic Server RDBMS-based persistence.

  Use the existing DTD file located at:

  `http://www.bea.com/servers/wls700/dtd/weblogic-rdbms-persistence-700.dtd`

  `http://www.bea.com/servers/wls600/dtd/weblogic-rdbms20-persistence-600.dtd` contains the DTD that defines container-managed persistence properties for entity EJBs. This DTD is changed from WebLogic Server Version 5.1, and you must still include a `weblogic-cmp-rdbms-jar.xml` file for entity EJBs using WebLogic Server RDBMS-based persistence.

  Use the existing DTD file located at:

  `http://www.bea.com/servers/wls600/dtd/weblogic-rdbms-persistence-600.dtd`

ejb-jar.xml

The following links provide the public DTD locations for the `ejb-jar.xml` deployment files used with WebLogic Server:

- For `ejb-jar.xml` 2.0 DTD:

  `http://www.java.sun.com/dtd/ejb-jar_2_0.dtd` contains the DTD for the standard `ejb-jar.xml` deployment file, required for all EJBs. This DTD is maintained as part of the JavaSoft EJB 2.0 specification; refer to the JavaSoft specification for information about the elements used in `ejb-jar.dtd`.

- For `ejb-jar.xml` 1.1 DTD:

  `ejb-jar.dtd` contains the DTD for the standard `ejb-jar.xml` deployment file, required for all EJBs. This DTD is maintained as part of the JavaSoft EJB 1.1 specification; refer to the JavaSoft specification for information about the elements used in `ejb-jar.dtd`.

Note: Refer to the appropriate JavaSoft EJB specification for a description of the `ejb-jar.xml` deployment descriptors.

2.0 weblogic-cmp-rdbms-jar.xml Deployment Descriptor File Structure

The `weblogic-cmp-rdbms-jar.xml` file defines deployment descriptors for a entity EJBs that uses WebLogic Server RDBMS-based persistence services. The EJB container uses a version of `weblogic-cmp-rdbms-jar.xml` that is different from the XML shipped with WebLogic Server Version 6.x.

You can continue to use the earlier `weblogic-cmp-rdbms-jar.xml` DTD for EJB 1.1 beans that you will deploy on the WebLogic Server Version 7.0. However, if you want to use any of the new CMP 2.0 features, you must use the new DTD described below.

The top-level element of the WebLogic Server 7.0 `weblogic-cmp-rdbms-jar.xml` consists of a `weblogic-rdbms-jar` stanza:

```
description
```

```
weblogic-version

weblogic-rdbms-jar

       weblogic-rdbms-bean
              ejb-name
              data-source-name
              table-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
       create-default-dbms-tables
       validate-db-schema-with
       database-type
```

2.0 weblogic-cmp-rdbms-jar.xml Deployment Descriptor Elements

- "automatic-key-generation" on page 10-9
- "caching-element" on page 10-10
- "caching-name" on page 10-11
- "check-exists-on-method" on page 10-12
- "cmp-field" on page 10-13
- "cmr-field" on page 10-14
- "column-map" on page 10-15
- "create-default-dbms-tables" on page 10-16

- "database-type" on page 10-17
- "data-source-name" on page 10-18
- "db-cascade-delete" on page 10-19
- "dbms-column" on page 10-20
- "dbms-column-type" on page 10-21
- "description" on page 10-22
- "delay-database-insert-until" on page 10-23
- "ejb-name" on page 10-24
- "enable-tuned-updates" on page 10-25
- "field-group" on page 10-26
- "field-map" on page 10-27
- "foreign-key-column" on page 10-28
- "foreign-key-table" on page 10-29
- "generator-name" on page 10-30
- "generator-type" on page 10-31
- "group-name" on page 10-32
- "include-updates" on page 10-33
- "key-cache-size" on page 10-34
- "key-column" on page 10-35
- "max-elements" on page 10-36
- "method-name" on page 10-37
- "method-param" on page 10-38
- "method-params" on page 10-39
- "optimistic-column" on page 10-40
- "primary-key-table" on page 10-41

- "query-method" on page 10-42
- "relation-name" on page 10-43
- "relationship-caching" on page 10-44
- "relationship-role-map" on page 10-45
- "relationship-role-name" on page 10-46
- "sql-select-distinct" on page 10-47
- "table-map" on page 10-48
- "table-name" on page 10-50
- "use-select-for-update" on page 10-51
- "validate-db-schema-with" on page 10-52
- "verify-columns" on page 10-53
- "weblogic-ql" on page 10-54
- "weblogic-query" on page 10-55
- "weblogic-rdbms-bean" on page 10-56
- "weblogic-rdbms-jar" on page 10-57
- "weblogic-rdbms-relation" on page 10-58
- "weblogic-relationship-role" on page 10-59

automatic-key-generation

Range of values:	n/a
Default value:	n/a
Requirements:	Optional.
Parent elements:	`weblogic-rdbms-bean`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `automatic-key-generation` element specifies the use of the Sequence/Key Generation feature.

Example

The `XML` stanza can contain the elements shown here:

```
<automatic-key-generation>
       <generator-type>ORACLE</generator-type>
       <generator-name>test_sequence</generator-name>
       <key-cache-size>10</key-cache-size>
</automatic-key-generation>
```

```
<automatic-key-generation>
       <generator-type>SQL-SERVER</generator-type>
</automatic-key-generation>
```

```
<automatic-key-generation>
       <generator-type>NAMED_SEQUENCE_TABLE</generator-type>
       <generator-name>MY_SEQUENCE_TABLE_NAME</generator-name>
```

```
        <key-cache-size>100</key-cache-size>
</automatic-key-generation>
```

caching-element

Range of values:	n/a
Default value:	n/a
Requirements:	n/a

Parent elements:	`weblogic-rdbms-jar`
	    `weblogic-rdbms-bean`
	        `relationship-caching`

Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

`caching-element` specifies the container-managed relationship (`cmr-field`) for the related bean, 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 about `group-name`, see "group-name" on page 10-32.

The `caching-element` descriptor specifies the container-managed relationship (`cmr-field`) for the related bean, 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 about `group-name`, see "group-name" on page 10-32.

As of WebLogic Server release 7.0 Service Pack 3, the EJB container now allows multiple caching-element sub-elements. The relevant DTD entry is this:

```
<!ELEMENT caching-element (
cmr-field,
group-name?,
caching-element*
)>
```

Previously, the DTD entry read this way:

```
<!ELEMENT caching-element (
cmr-field,
group-name?,
caching-element?
)>
```

Example

See “relationship-caching” on page 10-44:

caching-name

Range of values:	Valid name
Default value:	n/a
Requirements:	n/a

Parent elements:	`weblogic-rdbms-jar`
	`    weblogic-rdbms-bean`
	`        relationship-caching`

Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `caching-name` element specifies the name of a relationship cache.

Example

See “relationship-caching” on page 10-44:

check-exists-on-method

Range of values:	`True|False`
Default value:	`False`
Requirements:	.
Parent elements:	`weblogic-rdbms-bean`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

By default, the EJB container waits for a transaction to complete to check that a container-managed persistence (CMP) entity bean exists. This results in high performance and still provides a sufficient level of checking for most applications.

To specify that the EJB container check that bean exists before any business method invoked on the bean completes, set `check-exists-on-method` to `True`. 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.

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
Default value:	n/a
Requirements:	Field is case sensitive and must match the name of the field in the bean and must also have a `cmp-entry` entry in the `ejb-jar.xml`.

Parent elements:	`weblogic-rdbms-bean`
	     `field-map`
	`weblogic-rdbms-relation`
	     `field-group`

Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

This name specifies the mapped field in the bean instance which should be populated with information from the database.

Example

See "field-map" on page 10-27.

cmr-field

Range of values:	Valid name
Default value:	n/a
Requirements:	The field referenced in `cmr-field` must have a matching `cmr-field` entry in the `ejb-jar.xml`.

Parent elements:	`weblogic-rdbms-relation`
	`    field-group`

Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `cmr-field` element specifies the name of a container-managed relationship field (cmr-field.)

Example

The `XML` stanza can contain the elements shown here:

```
<weblogic-rdbms-jar>

       <weblogic-rdbms-relation>

              <field-group>employee</field-group>

                            <cmp-field>employee stock
purchases</cmp-field>

                            <cmr-field>stock options</cmr-field>

       </weblogic-rdbms-relation>

</weblogic-rdbms-jar>
```

column-map

Range of values:	n/a.
Default value:	n/a
Requirements:	The `key-column` element is not specified, if the `foreign-key-column` refers to a remote bean.
Parent elements:	`weblogic-rdbms-bean` `weblogic-relationship-role`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

This element represents the mapping of a foreign key column in one table in the database to a corresponding primary key. The two columns may or may not be in the same table. The tables to which the column belong are implicit from the context in which the `column-map` element appears in the deployment descriptor.

Example

The `XML` stanza can contain the elements shown here:

```
<weblogic-rdbms-jar>
       <weblogic-rdbms-bean>
              <column-map

<foreign-key-column>account-id</foreign-key-column>
                            <key-column>id</key-column>
              </column-map>


  </weblogic-rdbms-bean>
</weblogic-rdbms-jar>
```

create-default-dbms-tables

Range of values:	True | False.
Default value:	False
Requirements:	Use this element only for convenience during the development and prototyping phases. This is because the Table Schema in the DBMS CREATE statement used will be the container's best approximation of the definition. A production environment most likely, will require a more precise schema definition.
Parent elements:	`weblogic-rdbms-jar`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `create-default-dbms-table` element turns on or off a feature that automatically creates a default table based on the descriptions in the deployment files and the bean class. When set to False, this feature is turned off and table will not automatically be generated. When set to `True`, this feature is turned on and the table is automatically created. If `TABLE CREATION` fails, a `Table Not Found` error is thrown and the table must be created by hand.

Example

The following example specifies the `create-default-dbms-tables` element.

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

database-type

Range of values:	DB2| INFORMIX| ORACLE| SQL_SERVER| SYBASE| POINTBASE.
Default value:	NA
Requirements:	NA.
Parent elements:	`weblogic-rdbms-jar`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `database-type` element specifies the database used as the underlying dbms.

Example

The following example specifies the underlying dbms.

```
<database-type>POINTBASE</database-type>
```

data-source-name

Range of values:	Valid name of the data source used for all data base connectivity for this bean.
Default value:	n/a
Requirements:	Must be defined as a standard WebLogic Server JDBC data source for database connectivity. For more information on datasources, see Programming WebLogic JDBC.
Parent elements:	weblogic-rdbms-bean
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `data-source-name` that specifies the JDBC data source name to be used for all database connectivity for this bean.

Example

See "table-name" on page 10-50.

db-cascade-delete

Range of values:	
Default value:	n/a
Requirements:	Only supported for Oracle database. Can only be specified for one-to-one or one-to-many relationships.
Parent elements:	`weblogic-rdbms-bean` `weblogic-relationship-role`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `db-cascade-delete` element specifies whether the database cascade feature is turned on. If this element is not specified, WebLogic Server assumes that database cascade delete is not specified.

Example

See "Cascade Delete Method" on page 5-54.

dbms-column

Range of values:	Valid name
Default value:	n/a
Requirements:	`dbms-column` is case maintaining, although not all database are case sensitive.

Parent elements:	`weblogic-rdbms-bean`
	`    field-map`

Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The name of the database column to which the field should be mapped.

Example

See "field-map" on page 10-27.

dbms-column-type

Range of values:	Valid name
Default value:	n/a
Requirements:	`Available for use with Oracle database only.`
Parent elements:	`weblogic-rdbms-bean` `field-map`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `dbms-column-type` element maps the current field to a Blob or Clob in an Oracle database or a LongString or SybaseBinary in a Sybase database. This element can be one of the following:

- OracleBlob
- OracleCLob
- LongString
- SybaseBinary

Example

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

</field-map>
```

description

Range of values:	n/a
Default value:	n/a
Requirements:	n/a

Parent elements:	`weblogic-rdbms-jar`
	`     weblogic-rdbms-bean`
	`          weblogic-query`

Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `description` element is used to provide text that describes the parent element.

Example

The following example specifies the `description` element.

```
<dscription>Contains a description of parent element</description>
```

delay-database-insert-until

Range of values:	`ejbCreate	ejbPostCreate	commit`
Default value:	`ejbPostCreate`
Requirements:	Database insert is delayed until after `ejbPostCreate` when a `cmr-field` is mapped to a `foreign-key column` that does not allow null values. In this case, the `cmr-field` must be set to a non-null value in `ejbPostCreate` before the bean is inserted into the database. The `cmr-fields` may not be set during `ejbCreate`, before the primary key of the bean is known.
Parent elements:	`weblogic-rdbms-bean`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `delay-database-insert-until` element specifies the precise time when a new bean that uses RDBMS CMP is inserted into the database.

It is advisable to delay the database insert until after the `ejbPostCreate` method modifies the persistent fields of the bean. This can yield better performance by avoiding an unnecessary store operation.

For maximum flexibility, you should avoid creating related beans in your `ejbPostCreate` method. This may make delaying the database insert impossible if database constraints prevent related beans from referring to a bean that has not yet been created.

Example

The following example specifies the `delay-database-insert-until` element.

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

ejb-name

Range of values:	Valid name of an EJB.
Default value:	n/a
Requirements:	Must match the `ejb-name` of the cmp entity bean defined in the `ejb-jar.xml`.
Parent elements:	`weblogic-rdbms-bean`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The name that specifies an EJB as defined in the ejb-cmp-rdbms.xml. This name must match the `ejb-name` of a cmp entity bean contained in the `ejb-jar.xml`.

Example

See “table-name” on page 10-50.

enable-tuned-updates

Range of values:	`True/False`
Default value:	`True`
Requirements:	
Parent elements:	`weblogic-rdbms-bean`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `enable-tuned-updates` element specifies that when `ejbStore` is called that the EJB container automatically determine which container-managed fields have been modified and then writes only those fields back to the database.

Example

The following examples shows how to specify the `enable-tuned-updates` element.

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

field-group

Range of values:	Valid name
Default value:	A special group named `default` is used for finders and relationships that have no group specified.
Requirements:	The default group contains all of a bean's cmp-fields, but none of its cmr-fields.
Parent elements:	`weblogic-rdbms-relation`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `field-group` element represents a subset of the cmp and cmr-fields of a bean. Related fields in a bean can be put into groups that are faulted into memory together as a unit. A group can be associated with a finder or relationship, so that when a bean is loaded as the result of executing a finder or following a relationship, only the fields specified in the group are loaded.

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

Example

The `XML` stanza can contain the elements shown here:

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

field-map

Range of values:	Valid name
Default value:	n/a
Requirements:	Field mapped to the column in the database must correspond to a cmp field in the bean.
Parent elements:	`weblogic-rdbms-bean`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The name of the mapped field for a particular column in a database that corresponds to a cmp field in the bean instance.

Example

The `XML` stanza can contain the elements shown here:

```
<weblogic-rdbms-jar>
       <weblogic-rdbms-bean>

       <field-map>
              <cmp-field>accountId</cmp-field>
                     <dbms-column>id</dbms-column>
       </field-map>

       <field-map>
              <cmp-field>balance</cmp-field>
                     <dbms-column>bal</dbms-column>
       </field-map>

    <field-map>
              <cmp-field>accountType</cmp-field>
                     <dbms-column>type</dbms-column>
       </field-map>
```

```
  </weblogic-rdbms-bean>
</weblogic-rdbms-jar>
```

foreign-key-column

Range of values:	Valid name
Default value:	n/a
Requirements:	Must correspond to a column of a foreign key.
Parent elements:	`weblogic-rdbms-bean` `column-map`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `foreign-key-column` element represents a column of a foreign key in the database.

Example

See "column-map" on page 10-15.

foreign-key-table

Range of values:	Valid name
Default value:	n/a
Requirements:	n/a

Parent elements:	`weblogic-rdbms-jar`
	    `weblogic-rdbms-relation`
	        `weblogic-relationship-role`
	            `relationship-role-map`

Deployment file:	`weblogic-cmp-rdbms-jar.xml`

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 10-45.

generator-name

Range of values:	n/a
Default value:	n/a
Requirements:	Optional.

Parent elements:	`weblogic-rdbms-bean`
	    `automatic-key-generation`

Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `generator-name` element is used to specify the name of the generator.

For example;

- If the `generator-type` element is `ORACLE`, then the `generator-name` element would be the name of the `ORACLE_SEQUENCE` to be used.
- If the `generator-type` element is `NAMED_SEQUENCE_TABLE`, then the `generator-name` element would be the name of the `SEQUENCE_TABLE` to be used.

Example

See "automatic-key-generation" on page 10-9.

generator-type

Range of values:	n/a
Default value:	n/a
Requirements:	Optional

Parent elements:	`weblogic-rdbms-bean`
	 `automatic-key-generation`

Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The generator-type element specifies the key generation method to use. The options include:

- `ORACLE`
- `SQL_SERVER`
- `NAMED_SEQUENCE_TABLE`

Example

See "automatic-key-generation" on page 10-9.

group-name

Range of values:	Valid name
Default value:	n/a
Requirements:	n/a

Parent elements:	`weblogic-rdbms-relation`
	    `field-group`
	`weblogic-rdbms-bean`
	    `finder`
	        `finder-query`

Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `group-name` element specifies the name of a field group.

Example

The `XML` stanza can contain the elements shown here:

```
<weblogic-rdbms-jar>
       <weblogic-rdbms-relation>
                     <field-group>employee</field-group>
                            <cmp-field>employee stock
purchases</cmp-field>
                            <cmr-field>stock options</cmr-field>
                                   <group-name>financial
data</group-name>
```

```
        </weblogic-rdbms-relation>

    </weblogic-rdbms-jar>
```

include-updates

Range of values:	`True | False`
Default value:	`False`
Requirements:	The default value, which is `False`, provides the best performance.
Parent elements:	weblogic-rdbms-bean weblogic-query
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The include-updates element specifies that updates made during the current transaction must be reflected in the result of a query. If this element is set to `True`, the container will flush all changes made by the current transaction to disk before executing the query.

Example

The `XML` stanza can contain the elements shown here:

```
<include-updates>False</include_updates>
```

key-cache-size

Range of values:	n/a
Default value:	1
Requirements:	Optional

Parent elements:	`weblogic-rdbms-bean`
	`    automatic-key-generation`

Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The key-cache-size element specifies the optional size of the primary key cache available in the automatic primary key generation feature.

Example

See “automatic-key-generation” on page 10-9.

key-column

Range of values:	Valid name
Default value:	n/a
Requirements:	Must correspond to a column of a primary key.

Parent elements:	`weblogic-rdbms-bean`
	     `column-map`

Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `key-column` element represents a column of a primary key in the database.

Example

See "column-map" on page 10-15.

max-elements

Range of values:	n/a
Default value:	n/a
Requirements:	n/a
Parent elements:	weblogic-rdbms-bean weblogic-query
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

`max-elements` specifies the maximum number of elements that should be returned by a multi-valued query. This element is similar to the `maxRows` feature in JDBC.

Example

The `XML` stanza can contain the elements shown here:

```
<max-elements>100</max-elements>

      <!ELEMENT max-element (PCDATA)>
```

method-name

Range of values:	n/a
Default value:	n/a
Requirements:	The '*' character may not be used as a wildcard.

Parent elements:	`weblogic-rdbms-bean`
	    `query-method`

Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `method-name` element specifies the name of a finder or `ejbSelect` method.

Example

See "weblogic-query" on page 10-55.

method-param

Range of values:	Valid name
Default value:	n/a
Requirements:	n/a

Parent elements:	`weblogic-rdbms-bean`
	    `method-params`

Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `method-param` element contains the fully qualified Java type name of a method parameter.

Example

The `XML` stanza can contain the elements shown here:

```
<method-param>java.lang.String</method-param>
```

method-params

Range of values:	list of valid names
Default value:	n/a
Requirements:	n/a

Parent elements:	`weblogic-rdbms-bean`
	    `query-method`

Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `method-params` element contains an ordered list of the fully-qualified Java type names of the method parameters.

Example

See "weblogic-query" on page 10-55.

optimistic-column

Range of values:	n/a
Default value:	n/a
Requirements:	Although not all databases are case sensitive, this element preserves case.

Parent elements:	`weblogic-rdbms-bean`
	    `table-map`

Deployment file:	`weblogic-cmp-rdbms-jar.xml`

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 "Optimistic Concurrency Strategy" on page 4-17.

Example

The following sample XML shows the use of the `optimistic-column` element.

```
<optimistic-column>ROW_VERSION</optimistic-column>
```

primary-key-table

Range of values:	n/a
Default value:	n/a
Requirements:	Although not all databases are case sensitive, this element is case maintaining.

Parent elements:	`weblogic-rdbms-jar`
	    `weblogic-rdbms-relation`
	        `weblogic-relationship-role`
	            `relationship-role-map`

Deployment file:	`weblogic-cmp-rdbms-jar.xml`

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 5-31.

In the following XML stanza The bean on the `primary-key` side of a one-to-one relationship, called `Pk_bean` is mapped to multiple tables, but the bean on the `foreign-key` side of the relationship, called `Fk_Bean` is mapped to one table, called `Fk_BeanTable`. The foreign-key columns are named `Fk_column_1` and `Fk_column_2`.

For more information, see “Container-Managed Relationships” on page 5-42.

Example

The following sample XML shows the use of the `primary-key-table` element.

```
<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>
```

query-method

Range of values:	n/a
Default value:	n/a
Requirements:	n/a
Parent elements:	`weblogic-rdbms-bean`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `query-method` element specifies the method that is associated with a `weblogic-query`. It also uses the same format as the `ejb-jar.xml` descriptor.

Example

See "weblogic-query" on page 10-55.

relation-name

Range of values:	Valid name
Default value:	n/a
Requirements:	Must match the `ejb-relation-name` of an ejb-relation in the associated `ejb-jar.xml` 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
Parent elements:	`weblogic-rdbms-relation`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `relation-name` element specifies the name of a relation.

Example

The `XML` stanza can contain the elements shown here:

```
<weblogic-rdbms-jar>

       <weblogic-rdbms-relation>

              <relation-name>stocks-holders</relation-name>

                     <table-name>stocks</table-name>

       </weblogic-rdbms-relation>

</weblogic-rdbms-jar>
```

relationship-caching

Range of values:	n/a
Default value:	n/a
Requirements:	n/a
Parent elements:	`weblogic-rdbms-jar` `weblogic-rdbms-bean`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `relation-caching` element specifies relationship caching. For more information on relationship caching, see "Using Relationship Caching for CMRs" on page 5-51.

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 container-managed relationship (CMR) fields in the customerBean table; the address field is a CMR field in the accountBean table; and the addr_group and phone_group are groups in the addressBean and phoneBean.

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

Since relationship caching uses join queries, and a join query might duplicate results for a table in the ResultSet, the number of caching-element deployment descriptors specified in the relationship-caching element will have a direct impact on the number of duplicate results in the ResultSet. For one-to-many relationships, do not specify too many caching-element deployment descriptors in the relationship-caching element because the number of duplicate results might multiply for each caching-element deployment descriptor.

relationship-role-map

Default value:	n/a
Requirements:	The name must match the `ejb-relationship-role-name` of an `ejb-relationship-role` in the associated `ejb-jar.xml` descriptor file.

Parent elements:	`weblogic-rdbms-relation`
	      `weblogic-relationship-role`

Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The relationship-role-map element specifies foreign-key-column to key-column mapping for beans involved in a relationship.

For more information, see "Container-Managed Relationships" on page 5-42.

Example

The XML stanza can contain the elements shown here:

```
<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>
```

relationship-role-name

Default value:	n/a
Requirements:	The name must match the `ejb-relationship-role-name` of an `ejb-relationship-role` in the associated `ejb-jar.xml` descriptor file.
Parent elements:	`weblogic-rdbms-relation` `weblogic-relationship-role`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `relationship-role-name` element specifies the name of a relationship role.

Example

The `XML` stanza can contain the elements shown here:

```
<relationship-role-name>
</relationship- role-name>
```

sql-select-distinct

Range of values:	`True | False`
Default value:	`False`
Requirements:	The Oracle database does not allow you to use a SELECT DISTINCT in conjunction with a FOR UPDATE clause. Therefore, you cannot use the `sql-select-distinct` element if any bean in the calling chain has a method with a `transaction-isolation` element set to the `isolation-level` sub element with a value of `TransactionReadCommittedForUpdate`. You specify the `transaction-isolation` element in the `weblogic-ejb-jar.xml` file.
Parent elements:	`weblogic-query`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `sql-select-distinct` element controls whether the generated SQL SELECT statement will contain a a DISTINCT qualifier. Using the DISTINCT qualifier caused the database to return unique rows.

Example

The `XML` example contains the element shown here:

```
<sql-select-distinct>True</sql-select-distinct>
```

table-map

Range of values:	n/a
Default value:	n/a
Requirements:	Each `table-map element` must contain a mapping for the bean's primary key fields.
Parent elements:	`weblogic-rdbms-bean`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `table-map` element specifies a mapping between the cmp-fields 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 `n` DBMS tables, then you must specify `n` `table-map` elements for the bean, one for each n DBMS table.

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.

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>
       <fieldmap
</table-map>
```

table-name

Range of values:	Valid, fully qualified SQL name of the source table in the database.
Default value:	n/a
Requirements:	`table-name` must be set in all cases.
Parent elements:	`weblogic-rdbms-bean` `weblogic-rdbms-relation`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The fully qualified SQL name of the table. The user defined for the `data-source` for this bean must have read and write privileges for this table, but does not necessarily need schema modification privileges.

Example

The `XML` stanza can contain the elements shown here:

```
<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
Requirements:	`n/a`
Parent elements:	`weblogic-rdbms-bean`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

Enforces pessimistic concurrency on a per-bean basis. Specifying `"true"` for this flag causes `SELECT ... FOR UPDATE` to be used whenever the bean is loaded from the database. This is different from the transaction isolation level of `TRANSACTION_READ_COMMITTED_FOR_UPDATE` in that this is set at the bean level rather than the transaction level.

Example

The `XML` stanza can contain the elements shown here:

```
<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`
Requirements:	`n/a`
Parent elements:	`weblogic-rdbms-jar`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

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`

Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `verify-columns` element specifies the columns in a table 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.

- `Read`—all of the columns in the table that have been read during the transaction are checked.
- `Modified`—only the columns that have been updated by the current transaction are checked.
- `Version` and `Timestamp`—specify that a version or timestamp (pseudo)column exists in the table and that this column is used to implement optimistic concurrency. The EJB container will automatically increment the version or timestamp column when a row in the table is updated. The name of the version (or timestamp) column is specified using the `optimistic-column` element. It is not necessary to map this column to a cmp field, unless desired.

If a bean is mapped to multiple tables, checking is only performed on the tables that are updated during the transaction. The `verify-columns` elements for each table do not need to have the same value.

See "Optimistic Concurrency Strategy" on page 4-17 for more information.

Example

The `XML` stanza can contain the elements shown here:

```
<verify-columns>Modified</verify-columns>
```

weblogic-ql

Range of values:	n/a
Default value:	n/a
Requirements:	n/a

Parent elements:	`weblogic-rdbms-bean`
	`     weblogic-query`

Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `weblogic-ql` element specifies a query that contains a WebLogic specific extension to the ejb-ql language. You should specify queries that only use standard EJB-QL language features in the `ejb-jar.xml` deployment descriptor.

Example

See "weblogic-query" on page 10-55.

weblogic-query

Range of values:	n/a
Default value:	n/a
Requirements:	n/a
Parent elements:	`weblogic-rdbms-bean`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `weblogic-query` element allows you to associate WebLogic specific attributes with a query, as necessary. For example, `weblogic-query` can be used to specify a query that contains a WebLogic specific extension to EJB-QL. Queries that do not take advantage of WebLogic extensions to EJB-QL should be specified in the `ejb-jar.xml` deployment descriptor.

Also, the `weblogic-query` element is used to associate a `field-group` with the query if the query retrieves an entity bean that should be pre-loaded into the cache by the query.

Example

The `XML` stanza can contain the elements shown here:

```
<weblogic-query>

        <query-method>

          <method-name>findBigAccounts</method-name>

          <method-params>

                                   <method-param>double</method-param>
```

```
                </method-params>

            <query-method>

                                    <weblogic-ql>WHERE BALANCE>10000
ORDERBY NAME</weblogic-ql>

                    </weblogic-query>
```

weblogic-rdbms-bean

Range of values:	n/a
Default value:	n/a
Requirements:	n/a
Parent elements:	`weblogic-rdbms-jar`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `weblogic-rdbms-bean` represents a single entity bean that is managed by the WebLogic RDBMS CMP persistence type.

Example

The `XML` structure of `weblogic-rdbms-bean` is:

```
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-jar

Range of values:	n/a
Default value:	n/a
Requirements:	n/a
Parent elements:	n/a
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

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
Requirements:	n/a
Parent elements:	`weblogic-rdbms-jar`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `weblogic-rdbms-relation` element represents a single container-managed relationship (CMR).

For more information about CMRs, see "Container-Managed Relationships" on page 5-42.

Example

```
<weblogic-rdbms-relation>
   <relation-name>...</relation-name>
   <weblogic-relationship-role>...</weblogic-relationship-role>
</weblogic-rdbms-relation>
```

weblogic-relationship-role

Range of values:	Valid name
Default value:	n/a
Requirements:	The mapping of a role to a table is specified in the associated `weblogic-rdbms-bean` and `ejb-relation` elements.

Parent elements:	`weblogic-rdbms-jar`
	    `weblogic-rdbms-relation`

Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `weblogic-relationship-role` element is used to express a mapping from a foreign key to a primary key. Only one mapping is specified for one-to-one or a one-to-many relationships. With a many-to-many relationship, you must specify two mappings

Multiple column mappings are specified for a single role, if the key is complex. No `column-map` is specified if the role is just specifying a `group-name`.

For more information, see "Container-Managed Relationships" on page 5-42.

Example

The `XML` stanza can contain the elements shown here:

```
<weblogic-relationship-role>
   <relationship-role-name>...</relationhsip-role-name>
   <relationship-role-map>
                     <<column-map>
                            <foreign-key-column>manager-id
                                   </foreign-key-column>
                            <key-column>id</key-column>
```

```
                    </column-map>
             <relationship-role-name>
      </weblogic-relationship-role>
```

1.1 weblogic-cmp-rdbms-jar.xml Deployment Descriptor File Structure

`weblogic-cmp-rdbms-jar.xml` defines deployment elements for a single entity EJB that uses WebLogic Server RDBMS-based persistence services.

The top-level element of the WebLogic Server 1.1 `weblogic-cmp-rdbms-jar.xml` consists of a `weblogic-enterprise-bean` stanza:

```
description

weblogic-version

<weblogic-enterprise-bean>

   <pool-name>finance_pool</pool-name>

   <schema-name>FINANCE_APP</schema-name>

   <table-name>ACCOUNT</table-name>

   <attribute-map>

      <object-link>

         <bean-field>accountID</bean-field>

         <dbms-column>ACCOUNT_NUMBER</dbms-column>

      </object-link>

      <object-link>

         <bean-field>balance</bean-field>

         <dbms-column>BALANCE</dbms-column>

      </object-link>
```

```
    </attribute-map>

    <finder-list>

       <finder>

          <method-name>findBigAccounts</method-name>

          <method-params>

             <<method-param>double</method-param>

          </method-params>

       <finder-query><![CDATA[(> balance $0)]]></finder-query>

       <finder-expression>. . .</finder-expression>

       </finder>

    </finder-list>

</weblogic-enterprise-bean>
```

1.1 weblogic-cmp-rdbms-jar.xml Deployment Descriptor Elements

RDBMS Definition Elements

This section describes the RDBMS definition elements.

pool-name

`pool-name` specifies name of the WebLogic Server connection pool to use for this EJB's database connectivity. See Using connection pools for more information.

schema-name

`schema-name` specifies the schema where the source table is located in the database. This element is required only if you want to use a schema that is not the default schema for the user defined in the EJB's connection pool.

Note: This field is case sensitive, although many SQL implementations ignore case.

table-name

`table-name` specifies the source table in the database. This element is required in all cases.

Note: The user defined in the EJB's connection pool must have read and write privileges to the specified table, though not necessarily schema modification privileges. This field is case sensitive, although many SQL implementations ignore case.

EJB Field-Mapping Elements

This section describes the EJB field-mapping elements.

attribute-map

The `attribute-map` stanza links a single field in the EJB instance to a particular column in the database table. The `attribute-map` must have exactly one entry for each field of an EJB that uses WebLogic Server RDBMS-based persistence.

object-link

Each `attribute-map` entry consists of an `object-link` stanza, which represents a link between a column in the database and a field in the EJB instance.

bean-field

`bean-field` specifies the field in the EJB instance that should be populated from the database. This element is case sensitive and must precisely match the name of the field in the bean instance.

The field referenced in this tag must also have a `cmp-field` element defined in the `ejb-jar.xml` file for the bean.

dbms-column

`dbms-column` specifies the database column to which the EJB field is mapped. This tag is case sensitive, although many databases ignore the case.

Note: WebLogic Server does not support quoted RDBMS keywords as entries to `dbms-column`. For example, you cannot create an attribute map for column names such as “create” or “select” if those names are reserved in the underlying datastore.

Finder Elements

This section describes the finder elements.

finder-list

The `finder-list` stanza defines the set of all finders that are generated to locate sets of beans. See “Writing for RDBMS Persistence for EJB 1.1 CMP” on page 5-5 for more information.

`finder-list` must contain exactly one entry for each finder method defined in the home interface, except for `findByPrimarykey`. If an entry is not provided for `findByPrimaryKey`, one is generated at compilation time.

Note: If you do provide an entry for `findByPrimaryKey`, WebLogic Server uses that entry without validating it for correctness. In most cases, you should omit an entry for `findByPrimaryKey` and accept the default, generated method.

finder

The `finder` stanza describes a finder method defined in the home interface. The elements contained in the `finder` stanza enable WebLogic Server to identify which method in the home interface is being described, and to perform required database operations.

method-name

`method-name` defines the name of the finder method in the home interface. This tag must contain the exact name of the method.

method-params

The `method-params` stanza defines the list of parameters to the finder method being specified in method-name.

Note: WebLogic Server compares this list against the parameter types for the finder method in the EJB's home interface; the order and type for the parameter list must exactly match the order and type defined in the home interface.

method-param

`method-param` defines the fully-qualified name for the parameter's type. The type name is evaluated into a `java.lang.Class` object, and the resultant object must precisely match the respective parameter in the EJB's finder method.

You can specify primitive parameters using their primitive names (such as "double" or "int"). If you use a non-primitive data type in a `method-param` element, you must specify a fully qualified name. For example, use `java.sql.Timestamp` rather than `Timestamp`. If you do not use a qualified name, `ejbc` generates an error message when you compile the deployment unit.

finder-query

`finder-query` specifies the WebLogic Query Language (WLQL) string that is used to retrieve values from the database for this finder. See "Using WebLogic Query Language (WLQL) for EJB 1.1 CMP" on page 5-7 for more information.

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

finder-expression

`finder-expression` specifies a Java language expression to use as a variable in the database query for this finder.

Note: Future versions of the WebLogic Server EJB container will use the EJB QL query language (as required by the EJB 2.0 specification). EJB QL does not provide support for embedded Java expressions. Therefore, to ensure easier upgrades to future EJB containers, create entity EJB finders *without* embedding Java expressions in WLQL.

Index

A

acessing remote clients 2-6
aggregate functions
  EJB WebLogic QL 1-9
  subqueries 5-20
  with subquery return types 5-16
ANT tasks 1-11
application level cache
  configuring 4-24
applications
  building with EJBs 2-1
application-scoped
  EJBs 6-9
arguments
  ejbc 8-21
  weblogic.deploy (deprecated) 8-29
arithmetic operators 5-18
automatic
  table creation 5-40
automatic generation
  primary key 5-33
  primary key support for named sequence table 5-36
  primary key support for Oracle 5-35, 5-36

B

bean-managed transactions 4-38
bidirectional
  relationships 5-44
BLOB
  binary large object 5-27
  DBMS column support 5-27
  specifying with deployment descriptors 5-28
bulk insert support 1-10, 4-45
business logic
  modeling in entity EJBs 2-3

C

cache strategy
  Read-Mostly 4-23
caching
  application level cache 4-24
  between transactions 4-25
  between transactions with exclusive concurrency 4-26
  between transactions with optimistic concurrency 4-27
  between transactions with ReadOnly concurrency 4-27
  combined caching 4-24
  flushing the CMP cache 5-30
  relationship caching 5-51
calling
  multiple EJBs 4-42
cascade delete
  database cascade delete method 5-54
  method 5-54
  removing objects 5-53

class name
    possible generated collisions 6-13
class requirements
    message-driven beans 3-7
classloading 6-14
clients
    accessing local clients 2-6
    accessing remote clients 2-6
CLOB
    character large object 5-27
    DBMS column support 5-27
    specifying with deployment descriptors 5-28
cluster
    distributing transactions across EJBS 4-43
    entity EJBs 4-35
    read-wrie entity EJBs 4-36
    stateful session EJBs 4-33
    stateless session EJBs 4-31
    using EJBs 4-29
clustered
    EJBObjrcts 4-30
CMP 5-1
    EJB persistence services 5-3
    flushing the cache 5-30
    groups 5-58
    Java data types for CMP fields 5-59
    relationships 5-2
    using SQL with 1.1 finder queries 5-10
combined caching
    with entity beans 4-24
combined caching support 1-10
compiling
    EJBs 6-12
components
    bean class 1-2
    EJBs 1-2
    home interface 1-2
    remote interface 1-2
concurrency strateg
    optimistic 4-17
concurrency strategy
    database 4-17
    exclusive 4-16
    for EJBs 4-14
    ReadOnly 4-20
    ReadOnly restrictions 4-21
    read-write EJBs 4-15
    specifying 4-16
concurrent processing
    topics and queues 3-3
configure
    application level cache 4-24
container
    EJB 4-1
container-managed
    restrictions for EJBs 4-40
    setting transaction isolation levels 4-41
    transactions 4-39
container-managed persistence 5-1
    groups 5-58
    relationships 5-2
context interface 3-8
correlated subqueries 5-20
customer support contact information xxiii

D

data source factories 4-46
data types
    java data types
        for CMP fields 5-59
database
    concurrency strategy 4-17
database insert
    bulk insert 4-45
database insert support 4-44
    delay-database-insert-until 4-44
datastore
    managing transactions 2-9
DDConverter 8-23

conversion options 8-24
examples 8-27
options 8-27
syntax 8-26
defining the methods 3-7
delay-database-insert-until 4-44
delay-updates-until-end-of-tx
ejbStore 4-14
transactions 4-14
delete
cascade 5-53
deploy
arguments 8-29
depercated tool 8-28
invoking deployed EJBs 2-6
options 8-29
syntax 8-28
Deployer
weblogic.Deployer tool 8-28
deploying
message-driven beans 3-11
deployment
descriptors
EJB 9-2
descriptors (DOCTYPE header information 9-2
EJB files 6-3
EJB roles and responsibilities 7-1
EJBs at server startup 7-2
EJBs in different applications 7-3
names 7-4
new EJBS in a running environment 7-4
on a running server 7-3
packaging EJBs 6-10
relationship among deployment files 6-4
uncompiled EJBs 7-9
undeploying EJBs 7-6
updating EJBs 7-7
weblogic-cmp-rdbms-jar.xml file structure 10-5
deployment descriptor
creating deployment files 6-6
editing using Administration Console 1-12
ejb-jar.xml 6-4
weblogic-cmp-rdbms.xml 6-4
weblogic-ejb-jar.xml 6-4
Deployment Descriptor Editor 1-12
deployment descriptors
editing in the Console 6-7
manually editing 6-6
design tips 2-1
designing
message-driven beans 2-4
session beans 2-1
developer tools
ANT 1-11
EJB 1-11
EJB DD Editor 1-12
EJBGen 1-12
WebLogic Builder 1-11
weblogic.Deployer 1-12
XML Editor 1-13
DISTINCT clause
with subqueries 5-20
distributing
transactions accross multiple EJBs 4-42
documentation, where to find it xxi
DTD
elements 9-7
valid definitions (weblogic-cmp-rdbms-jar.xml) 10-4
weblogic-cmp-rdbms-jar.xml 10-1
dynamic queries
enabling 5-25
executing 5-25
queries
dynamic
EJB QL
dynamic queries 5-24
dynamic query support 1-8

E

eager
    relationship caching 5-51
editing
    deployment descriptors in the Console 6-7
    EJB deployment descriptors 6-5
    manually editing deployment descriptors 6-6
EJB 6-5
    accessing local clients 2-6
    activating EJB instance from free pool 4-7
    ANT tasks 1-11
    building applications 2-1
    changes in this release 1-1
    coarse-grained entity EJBs 2-3
    compiling 6-12
    components 1-2
    concurrency strategy 4-14
    container 4-1
    container context 3-8
    conversion options 8-24
    converting to latest version of WLS 8-26
    creating bean instances 3-10
    creating deployment files 6-6
    creating message-driven beans 3-4
    deploying at server startup 7-2
    deploying in a running environment 7-4
    deploying in different applications 7-3
    deploying on a running server 7-3
    deploying uncompiled EJBs 7-9
    Deployment Descriptor Editor 1-12, 6-7
    deployment descriptors 9-2
    deployment descriptors (weblogic-cmp-rdbms-jar.xml) 10-2
    deployment names 7-4
    design and development 2-1
    developer tools 1-11
    document type definitions 9-4
    editing deployment descriptors 6-5
    ejb-client.jar 6-14
    EJBGen 1-12
    ejb-jar 6-12
    ejb-jar.xml file 6-4
    enhancements for this release 1-7
    enity bean home interface 2-2
    features 1-1
    generated class name collisions 6-13
    generating 6-12
    handling beans 3-9
    initializing EJB instances 4-6
    invoking deployed EJBs 2-6
    lifecycle of EJB instances 4-2
    lifecycle of stateless session EJBs 4-6
    links 5-59
    links support 1-10
    loading into WebLogic Server 6-14
    manually editing deployment descriptors 6-6
    message-driven beans 3-1
    modeling entity EJBs 2-2
    multiple table mapping for 2.0 CMP 5-38
    optimizing data access 2-3
    overview 1-2
    packaging for use in the container 6-1
    packaging in deployment directory 6-10
    packaging steps 6-2
    persistence services 5-3
    pooling EJB instances 4-7
    QL for 2.0 beans 5-11
    referencing applicaion-scoped EJBs 6-9
    referencing external EJBs 6-8
    removing bean instances 3-10
    restriction for accessing EJBs 2-7
    source file components 6-2
    storing references in home handles 2-7
    transaction resources 2-8
    tuned 1.1 CMP updates 5-29
    types of 1-2

undeploying 7-6
updating 7-7
using inheritance 2-5
utilities 8-1
viewing deployed EJBs
EJB
viewing deployed 7-6
WebLogic Builder 1-11
weblogic.Deployer 1-12
weblogic-cmp-rdbms.xml file 6-4
weblogic-ejb-jar.xml file 6-4
writing RDBMS persistence for 1.1 CMP beans 5-5
XML Editor 1-13
EJB 2.0 support 1-4
EJB container
description 4-2
resource factoires 4-46
supported services 4-1
EJB deployment files 6-3
EJB life cycle
stateful session 4-7
EJB manifest class-path 6-16
EJB QL
migration from WLQL 5-12
requirements for 2.0 EJBs 5-12
WebLogic QL extension for EJB 2.0 5-13
EJB roles 1-5
application roles 1-5
deployment roles 1-7
infrastructure roles 1-6
management roles 1-7
EJB Support
combined caching 1-10
EJB support
bulk insert 1-10, 4-45
database insert 4-44
dynamic query 1-8
EJB links 1-10
EJB WebLogic QL enhancements 1-8
message-driven bean migratable service 1-8
multiple table mapping 1-8
optimistic concurrency 1-9
ReadOnly entity concurrency 1-9
relationship caching 1-10
EJB WebLogic QL
aggregate functions 1-9
ResultSets 1-9
subqueries 1-8
UPDATE NO WAIT 1-9
EJB WebLogic QL enhancements support 1-8
ejbc 8-19
arguments 8-21
examples 8-23
options 8-21
syntax 8-20
ejbCreate() 3-10
EJBGen 1-12, 8-1
example 8-5
syntax 8-2
tags 8-7
ejbLoad
entity beans 4-12
EJBObjects
clustered 4-30
ejbRemove() 3-10
EJBs
distributing transactions across multiple EJBs 4-42
in clusters 4-29
restrictions for container-managed 4-40
ejbStore
delay-updates-until-end-of-tx 4-14
entity beans 4-12
enabling
caching between transactions 4-27
encapsulating
multi-operational transactions 4-43

entity bean
  home interface 2-2
entity beans
  combined caching 4-24
  relationship caching 5-51
entity EJBs 1-3
  behavior with ejbLoad entity EJBs
    behavior with ejbStore 4-12
  in a cluster 4-35
examples
  DDConverter 8-27
  ejbc 8-23
exceptions
  for message-driven beans 3-9
exclusive
  concurrency strategy 4-16
EXISTS
  comparison operator 5-18

F

field groups 5-58
file components
  EJBs 6-2
finder
  EJB QL for 2.0 beans 5-11
  signature 5-5
finder-list
  stanza 5-6
finder-query
  element 5-6
  using for 1.1 CMP with SQL 5-10
firewall
  using with home handles 2-8

G

generating
  EJBs 6-12
get method
  restrictions 5-27
groups 5-58
  field groups 5-58

H

home handles
  EJB references 2-7
  using across a firewall 2-8
home interface
  entity bean 2-2

I

IN 5-17
inheritance
  restrictions 2-5
  using with EJBs 2-5
initial-bean-free-pool property 4-6
initializing
  EJB instances 4-6
in-memory replication
  limitations 4-35
  requirements 4-34
installing
  EJBs 7-4
is-modified-method-name
  EJB 1.1 only 4-13
isolation levels
  setting 4-40
  setting for container-managed
      transactions 4-41

J

Java specification
  EJB 2.0 1-3
  J2EE 1-3
java.transaction.UserTransaction 4-39
JDBC data source 4-46

L

limitations
  in-memory replication 4-35
  relationship caching 5-53
  TRANSACTION_SERIALIZABLE 4-41
links
  EJB links 5-59
local client 5-56
local interfaces 5-55
  local client 5-56

M

manifest class-path 6-16
message acknowledgement 3-12
message receipts 3-12
message-driven bean migratable service 1-8
message-driven beans 3-7
  basic components 3-7
  basic invocation procedure 3-9
  container context 3-8
  deploying 3-11
  description 3-2
  designing 2-4
  develop and deploy 3-1
  developing 3-4
  differences from JMS 3-2
  differences from stateless session 3-3
  EJB services 3-2
  ejbCreate() 3-10
  ejbRemove() 3-10
  handling exceptions 3-9
  implementing business logic with onMessage() 3-9
  message acknowledgement 3-12
  message receipts 3-12
  migratable service 3-13
  migrating 3-14
  onMessage() 3-9
  transaction services 3-11
message-driven EJBs 1-3
migratable service
  enabling 3-13
  for message-driven beans 3-13
migrating
  from WLQL to EJB QL 5-12
  message-driven beans 3-14
modeling
  coarse-entity EJBs 2-3
  entity EJBs 2-2
  entity EJBs with business logic 2-3
multicast invalidation
  Read-Only beans 4-21
multiple table mapping
  for cmp-fields 5-38
  for EJB 2.0 CMP 5-38
multiple table mapping support 1-8

N

comparison operands
  5-17, 5-18

O

optimistic
  concurrency strategy 4-17
  concurrency support 1-9
optimizing
  entity EJB data access 2-3
options
  DDConverter 8-27
  ejbc 8-21
  weblogic.deploy (deprecated) 8-29
ORDERBY 5-13, 5-14
overview
  EJBs 1-2

P

packaging

EJBs 6-1
EJBs in deployment directory 6-10
persistence
finder signature 5-5
finder-list stanza 5-6
finder-query element 5-6
using this service 5-3
writing for EJB 1.1 CMP 5-5
persistence services 5-3
primary key 5-31
anonymous class 5-32
automatic generation for EJB 2.0 CMP 5-33
automatic generation support for named sequence table 5-36
automatic generation support for Oracle 5-35, 5-36
mapped to single CMP field 5-31
mapping to a database column 5-33
usage hints 5-32
wraps single or multiple CMP fields 5-32
printing product documentation xxi

Q

queries
that return ResultSets 5-22
query language
for EJB 2.0 5-11
queues and topics
concurrent processing 3-3

R

Read-Mostly pattern 4-23
Read-Only
multicast invalidation 4-21
ReadOnly
concurrency strategy 4-20
concurrency strategy restrictions 4-21
entity concurrency support 1-9
read-write
EJBs
in a cluster 4-36
read-write EJBs
concurrency strategy 4-15
referencing
application-scoped EJBs 6-9
external EJBs 6-8
relationship caching
limations 5-53
support for "eager" relationship caching 1-10
with entity beans 5-51
relationships
among deployment files 6-4
bidirectional 5-44
contianer-managed persistence 5-2
relatoinship caching
specifying 5-51
removing
cascade delete 5-53
requirements
for in-memory replications 4-34
resource factories 4-46
JDBC data source factories 4-46
URL connection factories 4-48
restrictions
accessing EJB instances 2-7
container-managed EJBs 4-40
get method 5-27
set method 5-27
ResultSets
EJB WebLogic QL 1-9
using with queries 5-22

S

SELECT DISTINCT 5-14
SELECT HINTS 5-26
serializable objects

BLOB 5-27
session beans 2-1
designing 2-1
set method
restrictions 5-27
setting
container-managed isolation levels 4-41
JDBC data source factories 4-46
transaction isolation levels 4-40
URL connection factories 4-48
specification
final EJB version 1-4
specifying
concurrency strategy 4-16
EJB deployment descriptors 6-5
ejb-client.jar 6-14
field groups 5-58
primary key support for named sequence table 5-36
primary key support for Oracle 5-35, 5-36
relationship caching 5-51
SQL
for CMP 1.1 finder queries 5-10
stateful session
activating instances 4-8
EJB life cycle 4-7
stateful session EJBs 1-3
in a cluster 4-33
stateless session
lifecycle of these EJBs 4-6
stateless session EJBs 1-3
in a cluster 4-31
storing
EJB references in home handles 2-7
string objects
CLOB 5-27
subqueries 5-15
aggregate functions 5-20
arithmetic operators 5-18
as comparison operands 5-17
correlated 5-20
EJB WebLogic QL 1-8
uncorrelated 5-19
with DISTINCT clause 5-20
subquery return types 5-16
aggregate functions 5-16
beans with simple primary key 5-17
single cmp-field type 5-16
support
DBMS column 5-27
technical xxiii
syntax
DDConverter 8-26
deploy 8-28
ejbc 8-20
EJBGen 8-2

T

table creation
automatic 5-40
tags
EJBGen 8-7
tips
allow datastore to manage transactions 2-9
business logic in entity EJBs 2-3
demarcating transactions 2-10
modeling enitty EJBs 2-2
optimizing EJB data access
data access
optimizing for EJBs 2-3
preserve transaction resources 2-8
using coarse-grained entity EJBs 2-3
using container-managed transactions 2-9
using inheritance with EJBs 2-5
using session beans 2-1
topics and queues
concurrent processing 3-3
transaction boundaries

using java.transaction.UserTransaction 4-39
transaction management 4-38
responsibilities 4-38
TRANSACTION_SERIALIZABLE
limitations 4-41
transactions
bean-managed 4-38
caching betwteen 4-25
container-managed 4-39
container-managed over bean-managed 2-9
demarcating in WebLogic Server 2-10
distributing across EJBS in a cluster 4-43
encapsulating multi-opeartional 4-43
managed by the datastore 2-9
message acknowledgements 3-12
message receipts 3-12
preserving resources 2-8
single context (calling multiple EJBs) 4-42
with message-driven beans 3-11
tuned
EJB 1.1 CMP updates 5-29
types of EJBs 1-2
entity 1-3
message-driven 1-3
stateful session 1-3
stateless session 1-3

U

uncorrelated subqueries 5-19
undeploying
EJBs 7-6
UPDATE NO WAIT
EJB WebLogic QL 1-9
updating
EJBs 7-7
URL connections 4-48
using
DDConverter 8-26
Oracle SELECT HINTS 5-26
RDBMS persistence 5-3
utilities
DDConverter 8-23
Deployer 8-28
ejbc 8-19
EJBGen 8-1
for EJBs 8-1
weblogic.deploy (deprecated) 8-28

W

WebLogic Builder 1-11
WebLogic QL
EJB QL extension 5-13
ORDERBY 5-13, 5-14
SELECT DISTINCT 5-14
subqueries 5-15
WebLogic Query Language
expressions 5-9
for EJB 1.1 CMP 5-7
operands 5-8
operators 5-7
syntax 5-7
WebLogic Server
creating bean instances 3-10
developing message-driven beans 3-4
EJB 2.0 support 1-4
EJB container 4-1
features 1-1
free pool 4-6
invocation procedure for message-driven beans 3-9
message-driven beans 3-2
removing bean instances 3-10
WebLogic Server immplementation of final EJBspecification 1-4
weblogic.Deployer 1-12
weblogic-cmp-rdbms.xml file 6-4

weblogic-cmp-rdbms-jar.xml
  definitions 10-1
  descriptor elements 10-6
weblogic-ejb-jar.xml
  2.0 file structure 9-6
  descriptor elements 9-7
weblogic-ejb-jar.xml file 6-4
weblogiic-cmp-rdbms-jar.xml
  DOCTYPE Header information 10-2
WLQL
  expressions 5-9
  for EJB 1.1 CMP 5-7
  migration to EJB QL 5-12
  operands 5-8
  operators 5-7
  syntax 5-7

X

XML Editor 1-13