BEA WebLogic Server

Programming WebLogic Enterprise JavaBeans

WebLogic Server 6.0
Document Date March 3, 2001

Programming WebLogic Enterprise JavaBeans

	Document Date	Software Version
	March 3, 2001	WebLogic Server 6.0

About This Document xvii
Audience xviii
e-docs Web Site xviii
How to Print the Document xviii
Related Information xix
Contact Us! xix
Documentation Conventions xx
EJB Features and Changes in WebLogic Server 1-1
Implementation of Nonfinal Specification 1-1
EJB 2.0 Upgrade for WebLogic Server 1-2
EJB 2.0 Features 1-2
Supported Features 1-2
Features in Development 1-3
Major EJB Changes in This Release 1-3
Message-Driven Bean Support 1-4
Container-Managed Persistence 1-4
EJB QL 1-5
Application Assembly Support in Deploying EJBs 1-5
.jar, .ear, and Directory Deployment Units 1-5
Unsupported Deployment Utilities 1-5
DDConverter Upgrade Utility 1-6
WebLogic Server EJB Design and Development 2-1
Invoking Deployed EJBs 2-1
Accessing EJBs from either Local or Remote Clients 2-2
Restrictions on Accessing EJB Instances 2-3
Storing EJB References in Home Handles 2-3
Using Home Handles Across a Firewall 2-3
EJB Design Tips 2-4
Preserve Transaction Resources 2-4
Allow the Datastore to Manage Transactions 2-5
Using Container-Managed Transactions Instead of Bean-Managed Transactions 2-5
Never Demarcate Transactions from Application 2-5
Use Correct Modeling for Entity EJBs 2-6
Entity EJBs Should Be Coarse-Grained 2-6
Entity EJBs Should Contain Business Logic 2-6
Optimize Entity EJB Data Access 2-6
Use isModified() Where Appropriate 2-7
Using Inheritance with EJBs 2-7
Using Session Beans 2-8
Using Message-Driven Beans 3-1

What Are Message-Driven Beans? 3-1
Differences Between Message-Driven Beans and Standard JMS Consumers 3-2
Differences Between Message-Driven Beans and Stateless Session EJBs 3-3
Concurrent Support for Message-Driven Beans 3-3
Invoking a Message-Driven Bean 3-4
Developing Message-Driven Beans 3-4
Bean Class Requirements 3-4
Creating and Removing Bean Instances 3-5
Using the Message-Driven Bean Context 3-6
Implementing Business Logic with onMessage() 3-6
Handling Exceptions 3-7
Transaction Services for Message-Driven Beans 3-7
Message Receipts 3-8
Message Acknowledgment 3-8
Deploying Message-Driven Beans in WebLogic Server 3-9
Deployment Descriptors 3-9
Deployment Elements 3-9
The WebLogic Server EJB Container 4-1
EJB Life Cycle in WebLogic Server 4-2
Stateless Session EJB Life Cycle 4-2
Initializing EJB Instances 4-2
Activating and Pooling EJBs 4-3
Stateful EJB Life Cycle 4-3
Initializing and Using EJB Instances 4-4
Passivating Stateful EJBs 4-4
Removing Stateful Session EJB Instances 4-5
Using max-beans-in-free-pool 4-6
Stateful Session EJB Requirements 4-7
Special Use of max-beans-in-free pool 4-7
Locking Model for Entity EJBs 4-8
ejbLoad() and ejbStore() Behavior for Entity EJBs 4-8
Using db-is-shared to Limit Calls to ejbLoad() 4-9
Restrictions and Warnings for db-is-shared 4-10
Using is-modified-method-name to Limit Calls to ejbStore() 4-10
Warning for is-modified-method-name 4-11
Using delay-updates-until-end-of-tx to Change ejbStore() Behavior 4-11
Setting the Entity EJBs to Read-Only 4-12
Read-Write Cache Strategy 4-12
Read-Only-Cache-Strategy 4-12
Restrictions for Read-Only EJBs 4-12

Read-Mostly Pattern 4-13
EJBs in WebLogic Server Clusters 4-14
Overview 4-14
Clustered EJBHome Objects 4-15
Clustered EJBObjects 4-15
Session EJBs in a Cluster 4-16
Stateless Session EJBs 4-16
Stateful Session EJBs 4-17
In-Memory Replication for Stateful Session EJBs 4-18
Requirements and Configuration 4-18
Limitations of In-Memory Replication 4-19
Entity EJBs in a Cluster 4-19
Read-Write Entity EJBs 4-19
Non-Transactional Datastores 4-21
Transaction Management 4-21
Transaction Management Responsibilities 4-21
Using javax.transaction.UserTransaction 4-22
Restriction for Container-Managed EJBs 4-23
Distributing Transactions Across Multiple EJBs 4-23
Calling Multiple EJBs from a Single Transaction Context 4-23
Encapsulating a Multi-Operation Transaction 4-24
Distributing Transactions Across EJBs in a WebLogic Server Cluster 4-24
Transaction Isolation Level 4-25
Limitations of TRANSACTION_SERIALIZABLE 4-25
Special Note for Oracle Databases 4-25
Resource Factories 4-26
Setting Up JDBC Datasource Factories 4-26
Setting up URL Connection Factories 4-27
Persistence Services 4-28
Using WebLogic Server RDBMS Persistence 4-29
Writing Finders for RDBMS Persistence 4-29
Finder Signature 4-30
finder-list Stanza 4-30
finder-query Element 4-31
Using WebLogic Query Language (WLQL) 4-31
Syntax 4-31
Operators 4-31
Operands 4-32
Examples of WLQL Expressions 4-33
Using Java Expressions in WLQL 4-34

Example of Finder Method Signatures 4-36
Restrictions 4-37
Locking and Caching Services for Entity EJBs 4-38
Pessimistic Locking Services 4-38
Database Locking Services 4-38
Setting Up Database Locking 4-39
Home Method Support for Entity EJBs 4-40
WebLogic Server Container-Managed Persistence Services 5-1
EJB 2.0 Persistence Features and Changes 5-2
“get” and “set” Method Restrictions 5-2
BLOB and CLOB DBMS Column Restrictions 5-2
EJB QL Requirement for EJB 2.0 Beans 5-3
isModified() Not Required for CMP Beans 5-3
Using EJB QL 5-3
Basic EJB QL Syntax 5-4
EJB QL String Literals 5-4
EJB QL Operators 5-4
Finder Methods 5-4
Finder Parameter Placeholders 5-5
Select Methods 5-5
EJB QL Conditional Expressions 5-5
EJB QL Examples 5-6
Migrating from (EJB 1.1) WLQL to (EJB 2.0) EJB QL 5-7
Using WebLogic Query Language Extension 5-8
ORDERBY 5-8
Container-Managed Relationships 5-8
One-to-One Relationships 5-9
One-to-Many Relationships 5-10
Many-to-Many Relationships 5-10
Unidirectional Relationships 5-10
Bidirectional Relationships 5-10
Primary Keys 5-11
Foreign Keys 5-11
Groups 5-11
Specifying Field Groups 5-12
Using Groups 5-12
Supported Data Types 5-13
Deploying EJBs to WebLogic Server 6-1
Required Steps for Deploying EJBs 6-1
Setting Deployment Properties 6-2

Generating EJB Container Classes 6-4
Loading EJB Classes into WebLogic Server 6-5
Deploying EJBs at WebLogic Server Startup 6-5
Deploying EJBs in Different Applications 6-6
Deploying EJBs in a Running WebLogic Server (Dynamic Deployment) 6-6
EJB Deployment Names 6-7
Viewing Deployed EJBs 6-7
Deploying New EJBs into a Running Environment 6-8
Undeploying Deployed EJBs 6-8
Undeploying EJBs 6-9
Updating Deployed EJBs 6-9
The Update Process 6-10
Updating the EJB 6-10
Deploying EJBs in the EJB Container 7-1
Roles and Responsibilities 7-1
WebLogic Server Deployment Files 7-2
Automatic Deployment Directory 7-4
Deploying Compiled EJB .jar Files 7-4
Deploying Uncompiled EJB .jar Files 7-5
Deploying from an EJB .jar Directory 7-6
WebLogic Server EJB Utilities 8-1
ejbc 8-1
Syntax 8-2
Arguments 8-2
Options 8-2
Examples 8-3
DDConverter 8-4
Converting EJBs for Use in WebLogic Server 6.0 8-4
Converting EJB CMP 1.1 Beans to EJB CMP 2.0 Beans 8-4
Converting CMP Beans between WebLogic Server Versions 8-5
Syntax 8-5
Arguments 8-6
Options 8-6
Examples 8-6
deploy 8-7
Syntax 8-7
Arguments 8-7
Options 8-8
WebLogic Server 6.0 EJB Deployment Properties 9-1
Manually Editing XML Deployment Files 9-1

DOCTYPE Header Information 9-2
Document Type Definitions (DTDs) for Validation 9-3
weblogic-ejb-jar.xml Deployment Descriptor File 9-4
Index of weblogic-ejb-jar Deployment Elements 9-5
allow-concurrent-calls 9-8
Function 9-8
Example 9-8
concurrency-strategy 9-9
Function 9-9
Example 9-9
db-is-shared 9-10
Function 9-10
Example 9-10
delay-updates-until-end-of-tx 9-11
Function 9-11
Example 9-11
description 9-12
Function 9-12
Example 9-12
destination-jndi-name 9-13
Function 9-13
Example 9-13
ejb-name 9-14
Function 9-14
Example 9-14
ejb-reference-description 9-15
Function 9-15
Example 9-15
ejb-ref-name 9-16
Function 9-16
Example 9-16
enable-call-by-reference 9-17
Function 9-17
Example 9-17
entity-cache 9-18
Function 9-18
Example 9-18
entity-clustering 9-19
Function 9-19
Example 9-19

entity-descriptor 9-20
Function 9-20
Example 9-20
finders-load-bean 9-21
Function 9-21
Example 9-21
home-call-router-class-name 9-22
Function 9-22
Example 9-22
home-is-clusterable 9-23
Function 9-23
Example 9-23
home-load-algorithm 9-24
Function 9-24
Example 9-24
idle-timeout-seconds 9-25
Function 9-25
Example 9-25
initial-beans-in-free-pool 9-26
Function 9-26
Example 9-26
is-modified-method-name 9-27
Function 9-27
Example 9-27
isolation-level 9-28
Function 9-28
Example 9-29
jndi-name 9-29
Function 9-29
Example 9-29
lifecycle 9-30
Function 9-30
Example 9-30
max-beans-in-cache 9-31
Function 9-31
Example 9-31
max-beans-in-free-pool 9-32
Function 9-32
Example 9-32
message-driven-descriptor 9-33

Function 9-33
Example 9-33
method 9-34
Function 9-34
Example 9-34
method-intf 9-35
Function 9-35
Example 9-35
method-name 9-36
Function 9-36
Example 9-36
method-param 9-37
Function 9-37
Example 9-37
method-params 9-38
Function 9-38
Example 9-38
passivation-strategy 9-39
Function 9-39
Example 9-39
persistence 9-40
Function 9-40
Example 9-40
persistence-type 9-41
Function 9-41
Example 9-41
persistence-use 9-42
Function 9-42
Example 9-42
persistent-store-dir 9-43
Function 9-43
Example 9-43
pool 9-44
Function 9-44
Example 9-44
principal-name 9-45
Function 9-45
Example 9-45
read-timeout-seconds 9-46
Function 9-46

Example 9-46
reference-descriptor 9-47
Function 9-47
Example 9-47
replication-type 9-48
Function 9-48
Example 9-48
res-env-ref-name 9-49
Function 9-49
Example 9-49
res-ref-name 9-50
Function 9-50
Example 9-50
resource-env-description 9-51
Function 9-51
Example 9-51
resource-description 9-52
Function 9-52
Example 9-52
role-name 9-53
Function 9-53
Example 9-53
run-as-identity-principal 9-54
Function 9-54
Example 9-54
security-role-assignment 9-55
Function 9-55
Example 9-55
stateful-session-cache 9-56
Function 9-56
Example 9-56
stateful-session-clustering 9-57
Function 9-57
Example 9-57
stateful-session-descriptor 9-58
Function 9-58
Example 9-58
stateless-bean-call-router-class-name 9-59
Function 9-59
Example 9-59

stateless-bean-is-clusterable 9-60
Function 9-60
Example 9-60
stateless-bean-load-algorithm 9-61
Function 9-61
Example 9-61
stateless-bean-methods-are-idempotent 9-62
Function 9-62
Example 9-62
stateless-clustering 9-63
Function 9-63
Example 9-63
stateless-session-descriptor 9-64
Function 9-64
Example 9-64
transaction-descriptor 9-65
Function 9-65
Example 9-65
transaction-isolation 9-66
Function 9-66
Example 9-66
trans-timeout-seconds 9-67
Function 9-67
Example 9-67
type-identifier 9-68
Function 9-68
Example 9-68
type-storage 9-69
Function 9-69
Example 9-69
type-version 9-70
Function 9-70
Example 9-70
weblogic-cmp-rdbms-jar.xml Deployment Descriptor File 9-71
Index of weblogic-cmp-rdbms-jar.xml Deployment Elements 9-71
cmp-field 9-73
Function 9-73
Example 9-73
cmr-field 9-74
Function 9-74

Example 9-74
column-map 9-75
Function 9-75
Example 9-75
data-source-name 9-76
Function 9-76
Example 9-76
dbms-column 9-77
Function 9-77
Example 9-77
ejb-name 9-78
Function 9-78
Example 9-78
field-group 9-79
Function 9-79
Example 9-79
field-map 9-80
Function 9-80
Example 9-80
foreign-key-column 9-81
Function 9-81
Example 9-81
group-name 9-82
Function 9-82
Example 9-82
key-column 9-83
Function 9-83
Example 9-83
max-elements 9-84
Function 9-84
Example 9-84
method-name 9-85
Function 9-85
Example 9-85
method-param 9-86
Function 9-86
Example 9-86
method-params 9-87
Function 9-87
Example 9-87

query-method 9-88
Function 9-88
Example 9-88
relation-name 9-89
Function 9-89
Example 9-89
relationship-role-name 9-90
Function 9-90
Example 9-90
table-name 9-91
Function 9-91
Example 9-91
weblogic-ql 9-92
Function 9-92
Example 9-92
weblogic-query 9-93
Function 9-93
Example 9-93
weblogic-relationship-role 9-94
Function 9-94
Example 9-94
WebLogic Server 5.1 EJB Deployment Properties 10-1
Manually Editing XML Deployment Files 10-1
Basic Conventions 10-2
DOCTYPE Header Information 10-2
Document Type Definitions (DTDs) for Validation 10-3
weblogic-ejb-jar.xml Deployment Descriptor File 10-4
Caching Properties 10-4
caching-descriptor 10-5
Persistence Properties 10-7
persistence-descriptor 10-7
Clustering Properties 10-10
clustering-descriptor 10-10
Transaction Properties 10-12
transaction-descriptor 10-13
EJB References 10-13
reference-descriptor 10-13
Isolation Level Settings 10-14
transaction-isolation 10-14
isolation-level 10-15

method 10-15
Security Role Assignments 10-17
security-role-assignment 10-17
enable-call-by-reference 10-17
weblogic-cmp-rdbms-jar.xml Deployment Descriptor File 10-17
RDBMS Definition Elements 10-19
pool-name 10-19
schema-name 10-19
table-name 10-19
EJB Field-Mapping Elements 10-20
attribute-map 10-20
object-link 10-20
bean-field 10-20
dbms-column 10-20
Finder Elements 10-21
finder-list 10-21
finder 10-21
method-name 10-21
method-params 10-21
method-param 10-22
finder-query 10-22
finder-expression 10-22

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, “EJB Features and Changes in WebLogic Server,” is an overview of EJB features supported in WebLogic Server.
- Chapter 2, “WebLogic Server EJB Design and Development,” is an overview of design techniques developers can use to create EJBs.
- Chapter 3, “Using Message-Driven Beans,” explains how to develop and deploy message-driven beans in the WebLogic Server container.
- Chapter 4, “The WebLogic Server EJB Container,” describes the services available to the EJB with the WebLogic Services container.
- Chapter 5, “WebLogic Server Container-Managed Persistence Services,” describes the EJB 2.0 container-managed persistence services available for the WebLogic Server container.
- Chapter 6, “Deploying EJBs to WebLogic Server,” describes the steps necessary to deploy EJB to WebLogic Server.
- Chapter 7, “Deploying EJBs in the EJB Container,” 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 Server 6.0 EJB Deployment Properties,” describes the WebLogic-specific deployment descriptors for the EJB 2.0 container that are supplied in WebLogic Server 6.0.

- Chapter 10, "WebLogic Server 5.1 EJB Deployment Properties," describes the WebLogic-specific deployment descriptors shipped with WebLogic Server 5.1. These properties are provided for reference purposes.

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 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 main topic 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

Aditional documentation related to EJBs includes:

Java Remote Method Invocation (RMI)

Extensible Markup Language (XML)

Java Naming and Directory Interface (JNDI)

Contact Us!

Your feedback on BEA 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 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	Usage
Ctrl+Tab	Keys you press simultaneously.
italics	Emphasis and book titles.

`monospace text`	Code samples, commands and their options, Java classes, data types, directories, and file names and their extensions. Monospace text also indicates text that you enter from the keyboard.
	Examples:
	`import java.util.Enumeration;`
	`chmod u+w *`
	`config/examples/applications`
	`.java`
	`config.xml`
	`float`

`monospace italic text`	Variables in code. *Example*: `String` *`CustomerName`*`;`
UPPERCASE TEXT	Device names, environment variables, and logical operators. *Example*s: LPT1 BEA_HOME OR
`{ }`	A set of choices in a syntax line.

`[ ]`	Optional items in a syntax line. *Example*:
	`java utils.MulticastTest -n` *`name`* `-a` *`address`*
	`[-p` *`portnumber`*`] [-t` *`timeout`*`] [-s` *`send`*`]`

Convention	Usage

`|`	Separates mutually exclusive choices in a syntax line. *Example*:
	`java weblogic.deploy [list|deploy|undeploy|update]`
	`     password {application} {source}`

`...`	Indicates one of the following in a command line: ■ An argument can be repeated several times in the command line. ■ The statement omits additional optional arguments. ■ You can enter additional parameters, values, or other information
`.` `.` `.`	Indicates the omission of items from a code example or from a syntax line.

1 EJB Features and Changes in WebLogic Server

WebLogic Server Version 6.0 includes an implementation of Sun Microsystems Enterprise JavaBeans 1.1 and 2.0 architecture. This architecture defines a standard way of creating server components that are part of distributed object-oriented applications. Enterprise JavaBeans are the standard used to define server-side components.

The following sections provide an overview of the EJB features and the changes introduced in the WebLogic Server Version 6.0 Enterprise JavaBeans implementation:

- Implementation of Nonfinal Specification
- EJB 2.0 Upgrade for WebLogic Server
- EJB 2.0 Features
- Major EJB Changes in This Release

Implementation of Nonfinal Specification

The Enterprise JavaBeans 2.0 implementation in WebLogic Server Version 6.0 will be fully supported and can be used in production. However, be advised that the Sun Microsystems EJB 2.0 specification is not yet finalized, and the WebLogic Server implementation of the EJB 2.0 architecture is based on the most current public draft of this specification. Consequently once the specification is finalized, there could be changes to the Enterprise JavaBeans 2.0 implementation in future versions of WebLogic Server. These changes may cause application code developed for WebLogic Server Version 6.0 to be incompatible with EJB 2.0 implementations supported in future releases.

EJB 2.0 Upgrade for WebLogic Server

Before you can use the EJB 2.0 features with WebLogic Server Version 6.0, you must download the `EJB20.jar` file and the `EJB 2.0 README` file after you have installed WebLogic Server Version 6.0. The EJB 2.0.jar file is an upgrade to WebLogic Server Version 6.0 that enables the EJB 2.0 features. To download the `.jar` file:

1. Select the product download option from the BEA web site, http://www. bea.com.
2. Copy the zip file for the EJB 2.0 upgrade to your machine. This zip file contains the `EJB20.jar` and the `EJB 2.0 README` files.
3. Use the instructions in the `EJB 2.0 README` file to install the JAR file.

EJB 2.0 Features

This section lists the EJB 2.0 features that are new to WebLogic Server Version 6.0 and the EJB 2.0 features that are in development.

Supported Features

Support for the following features is provided in WebLogic Server 6.0.

- Support for message-driven beans
- EJB 2.0 container-managed persistence services, including support for EJB-QL

- Support for entity EJB home methods
- EJB 2.0 XML deployment properties
- In-memory replication for stateful session EJBs in a cluster
- Database locking option for entity EJBs
- Improved EJB Deployment and redeployment capabilities
- `isModified()` no longer required for EJB 2.0 CMP entity beans
- Compliance with EJB 1.1 specification

Features in Development

Because the container is based on pre-release versions of the EJB 2.0 specification, certain EJB 2.0 features are not yet available. Known differences between the container and the publicly available EJB 2.0 specification are acknowledged in this document where applicable.

The following EJB 2.0 container and WebLogic Server features will not be supported until a future release of the WebLogic Server EJB 2.0 container:

- Support for the final EJB 2.0 specification
- Dependent objects
- Run-as-specified-identity

Major EJB Changes in This Release

This release of WebLogic Server Version 6.0 contains major changes to the supported EJB features. The following sections describe some of those changes:

Message-Driven Bean Support

EJBs are integrated with the Java Message Service (JMS) to provide the ability for a message-driven bean to act as a standard JMS message consumer. The message-driven bean is a stateless component that is invoked by the EJB container as a result of receiving messages from a JMS Queue or Topic. The message-driven bean then performs business logic based on the message contents. Using the message-driven bean model allows EJB developers to work with a familiar framework and set of tools and also provides access to the additional support provided by the container.

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. WebLogic Server automatically creates and removes message-driven bean instances as needed to process incoming messages.

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. The goal of the message-driven bean model is to assure that developing an EJB, that is asynchronously invoked to handle the processing of incoming JMS messages, is as easy as developing the same functionality in any other JMS `MessageListener`. For more information on message-driven beans, see "Developing Message-Driven Beans" on page 3-4.

Container-Managed Persistence

This release provides added support for container-managed relationships among entity beans. This container-managed persistence model is an improvement over the limitations of the field-based approach to container-managed persistence in earlier versions.

With container-managed persistence, database access calls are not written in the entity bean. Instead, persistence is handled by the EJB container that is available at run time. The persistent fields and relationships for which the container must generate data access calls are specified in the deployment descriptors. When the entity bean is deployed, the container is used to generate the necessary database access calls. For more information on container-managed persistence, see "EJB 2.0 Persistence Features and Changes" on page 5-2.

EJB QL

This release supports for Enterprise JavaBeans Query Language (EJB QL). EJB QL is a syntax for the definition of finder methods or queries for entity beans with container-managed persistence. This syntax allows the Persistence Manager to provide for the implementation of the finder methods. EJB QL defines finder methods so that they are portable across containers and persistence managers. EJB QL is a declarative, SQL-like language that is meant to be compiled to the target language of the persistent datastore used by a Persistence Manager. For more information on EJB QL, see "Using EJB QL" on page 5-3.

Application Assembly Support in Deploying EJBs

WebLogic Server simplifies the process of deploying multiple EJB deployment units to one or more WebLogic Servers. When WebLogic Server is started, the EJBs are automatically assembled and deployed to the appropriate WebLogic Server.

.jar, .ear, and Directory Deployment Units

You can deploy an Enterprise Application Archive (EAR) file or a Java Archive (JAR) file by either copying the file to the `config/examples/applications` directory or by deploying the file from the WebLogic Administration Console. The files and subdirectories contained in the deployment directory must observe the same restrictions as files and subdirectories stored in a `.jar` file.

Unsupported Deployment Utilities

The following EJB deployment utilities no longer exist with WebLogic Server Version 6.0:

- `DDCreator`
- `EJB Deployment Wizard`

WebLogic Server does not provide a `DDCreator` utility to generate new `ejb-jar.xml` deployment files. However, you can use `DDConverter` with an earlier WebLogic Server text deployment description to generate a valid `ejb-jar.xml` file.

Note: If you are an EJB provider and you need to create a new `ejb-jar.xml` file from scratch, see the JavaSoft EJB 1.1 or 2.0 specification for instructions.

DDConverter Upgrade Utility

The new `DDConverter` utility provides a quick and easy way to upgrade earlier WebLogic Server EJB deployment descriptors, such as those in WebLogic Server 5.1, to WebLogic Server Version 6.0. `DDConverter` takes an existing WebLogic Server text description file and generates the `.xml` and `weblogic-ejb-jar.xml` files required for deploying to Version 6.0. `DDConverter` also automatically generates a `rdbms-jar.xml` deployment file for entity EJBs that use WebLogic RDBMS-based persistence services.

See “DDConverter” on page 8-4 for instructions on using the `DDConverter` utility.

CHAPTER

2 WebLogic Server EJB Design and Development

The following sections provide a collection of design tips and debugging points to keep in mind when building applications with Enterprise JavaBeans. A number of these tips apply to remote object models, Remote Method Invocation (RMI) and Common Object Request Broker Architecture (CORBA), as much as they do to EJB.

Invoking Deployed EJBs

EJB Design Tips

Invoking 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 in the same server as the EJB, or on a remote computer — can access deployed EJBs in a similar fashion.

With the EJB 1.1 and 2.0 specification, all EJBs must specify their environment properties using Java Naming and Directory Interface (JNDI). EJB clients can configure their JNDI name spaces 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. Due to 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.

Accessing EJBs from either Local or Remote Clients

One difference between accessing EJBs from local clients or from remote clients is in obtaining an `InitialContext` for the bean. Remote clients obtain an `6InitialContext` from the WebLogic Server `InitialContext` factory. WebLogic Server local clients generally use a `getInitialContext` method to perform this lookup, similar to the following excerpt:

```
...

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);

   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 Accessing EJB Instances

Although database concurrency is the default and recommended process for concurrency access, it is possible to use entity EJBs that can be accessed by multiple clients in a serial fashion. However, this process is available for use with legacy applications that depend on this exclusive option. Using this exclusive option means that if two clients 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 “Locking Model for Entity EJBs” on page 4-8.

Simultaneous access to a stateful session EJB results in a `RemoteException`, as required by the EJB 1.1 specification. However, you can set the `allow-concurrent-calls` option to specify that a stateful session bean instance will allow concurrent method calls. This access restriction on stateful session EJBs applies whether the EJB client is remote or internal to WebLogic Server.

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/servelet can use a stateful session bean in request scope.

Storing EJB References in Home Handles

Once a client has obtained the `EJBHome` object for an EJB instance, it can create a reference 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 instance 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 are unable to locate `EJBHome` objects using 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-hand pane, expand the Servers node and select a server.
4. In the right-hand 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.

EJB Design Tips

The following sections describe design tips for developing and deploying EJBs on your system.

Preserve Transaction Resources

Database transactions are typically one of the most valuable resources in an online transaction-processing system. When using EJBs with WebLogic Server, transaction resources are even more valuable due to 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.

Allow the Datastore to Manage Transactions

Many RDBMS systems provide high-performance locking systems for 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. You should never prevent the RDBMS from automatically delimiting transactions.

Using Container-Managed Transactions Instead of Bean-Managed Transactions

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 may need bean-managed transactions are:

- You must define multiple transactions from within a single method call. WebLogic Server demarcates transactions on a per-method basis; if you require multiple transactions in a single method call, you must use bean-managed transactions.

  Note: However, keep in mind that if your EJBs use multiple transactions in a single method call, it is still better to break the transactions out across multiple methods and use container-managed transactions with the revised bean.

- A single transaction must “span” multiple EJB method calls. For example, one method begins a transaction, and another method commits or rolls back the transaction. In general, this practice should be avoided where possible since it requires detailed information about the workings of the EJB object. If it 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.

Use Correct Modeling for Entity EJBs

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 EJBs Should Be Coarse-Grained

You should 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, as the trade-off in network resources is unacceptable.

For example, line items in an invoice or 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 may be modeled as an entity EJB, if additional business logic is required for the data.

Entity EJBs Should Contain Business Logic

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.

Use isModified() Where Appropriate

Use the `isModified()` method in entity EJBs to eliminate unnecessary database writes for read-only operations. However, this is no longer required for EJB 2.0 container-managed persistence (CMP) entity beans. See "Using is-modified-method-name to Limit Calls to ejbStore()" on page 4-10 for more information.

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 that apply to EJB implementations.

For bean-managed 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. 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()` methods.

Additional restrictions exist for EJBs where inheriting another EJB implementation changes the interface. For example, the following table describes a situation where a derived bean adds a new method that is meant to be accessible remotely:

Bean	Method	Interface	Method
`ABean`	`afoo()`	`ARemote`	`afoo()`
`BBean` (extends `ABean`)	`bfoo()`	`BRemote`	`bfoo()`

However if you use this example, you cannot have the `BHome` interface inherit from the `AHome` interface, because `AHome.create()` and `BHome.create()` return different remote interfaces.

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 example Enterprise JavaBean subclass Child example packages and classes in the WebLogic Server distribution.

Using Session Beans

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

The model should 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.) With this design 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 which 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 of the references, the client must go back to the datastore to retrieve additional data, which consumes considerable network bandwidth and may yield unacceptable performance.

CHAPTER

3 Using Message-Driven Beans

The following sections describe how to develop and deploy message-driven beans in the EJB 2.0 for BEA WebLogic Server container. Because message-driven beans utilize parts of the standard JMS API, you should first become familiar with the WebLogic JMS messaging system before attempting to implement message-driven beans. See the Programming WebLogic JMS document for more information.

- What Are Message-Driven Beans?
- Developing Message-Driven Beans
- Transaction Services for Message-Driven Beans
- Deploying Message-Driven Beans in WebLogic Server

What Are Message-Driven Beans?

A message-driven bean is a special kind of 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 utilizes 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 4-21 and in “Transaction Services for Message-Driven Beans” on page 3-7.

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 the particular WebLogic Server instance.

Note: One limitation of message-driven beans compared to standard JMS listeners is that a given message bean deployment can be associated with only one Queue or Topic, as described in Deploying Message-Driven Beans in WebLogic Server. 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

In several ways, the dynamic creation and allocation of message-driven bean instances mimics the behavior of stateless session EJB instances. However, message-driven beans are different 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 Support for Message-Driven Beans

Message-Driven Beans support concurrent processing for both topics and queues. Previously, only concurrent processing for Queues was supported.

To ensure concurrency, change the `weblogic-ejb-jar.xml` deployment descriptor `max-beans-in-free-pool` setting to >1. If this element is set to more than one, the container will spawn as many threads as specified. For more information on this element see, “max-beans-in-free-pool” on page 9-32.

Invoking a Message-Driven Bean

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

1. Obtain a new bean instance.

   Obtain a new bean instance from the connection pool if one already exists, or create a new one. See "Creating and Removing Bean Instances" on page 3-5.

2. If the bean cannot be located in the pool and a new one must be created, call the bean's `setMessageDrivenContext()` to associate the instance with a container context. The bean can utilize elements of this context as described in "Using the Message-Driven Bean Context" on page 3-6.

3. Call the bean's `onMessage()` method to perform business logic. See "Implementing Business Logic with onMessage()" on page 3-6.

Note: These instances can be pooled.

Developing Message-Driven Beans

To create message-driven EJBs, you must follow certain conventions described in the JavaSoft EJB 2.0 specification, as well as observe several general practices that result in proper bean behavior.

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 in **bold** are required as part of the EJB 2.0 specification:

```
public class MessageTraderBean implements
javax.ejb.MessageDrivenBean {

       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 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.

       finalize{};

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

}
```

Creating and Removing Bean Instances

The WebLogic Server container calls the message-driven bean’s `ejbCreate()` and `ejbRemove()` methods when creating or removing an instance of the bean class. 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).

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 instance by using the following methods from the `MessageDrivenContext` interface:

- `getCallerPrincipal()`
- `isCallerInRole()`
- `setRollbackOnly()`– The EJB can use this method only if it utilizes container-managed transaction demarcation.
- `getRollbackOnly()` – The EJB can use this method only if it utilizes container-managed transaction demarcation.
- `getUserTransaction()`– The EJB can use this method only if it utilizes bean-managed transaction demarcation.

Note: Although `getEJBHome()` is also inherited as part of the `MessageDrivenContext` interface, message-driven EJBs 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 performs all of 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 EJB’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 instance that WebLogic Server creates.

Transaction Services for Message-Driven Beans

As with other EJB types, 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, as required by the EJB 2.0 specification.

Because no client provides a transaction context for calls to a message-driven bean, beans that use container-managed transactions must be deployed using the `Required` or `NotSupported` transaction attribute in `ejb-jar.xml`. Transaction attributes are defined in `ejb-jar.xml` as follows:

```
<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. For EJBs that use bean-managed transactions, the message receipt is always outside the scope of the bean's transaction, as described in the EJB 2.0 specification.

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 attribute 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 of the EJB transaction context. WebLogic Server automatically acknowledges messages for EJBs with bean-managed transactions, but the deployer can configure acknowledgment semantics using the jms-acknowledge-mode deployment parameter.

Deploying Message-Driven Beans in WebLogic Server

To deploy a message-driven bean on WebLogic Server, you edit the XML file to create the deployment descriptors that associate the EJB with a configured JMS destination.

Deployment Descriptors

The deployment descriptor for a message-driven bean also specifies:

- Whether the EJB is associated with a JMS Topic or Queue
- Whether an associated Topic is durable or non-durable
- Transaction attributes for the EJB
- JMS acknowledgment semantics to use for beans that demarcate their own transactions

Deployment Elements

The EJB 2.0 specification adds the following new XML deployment elements for deploying message-driven beans.

- `message-driven-destination` specifies whether the EJB should be associated with a JMS Queue or Topic destination.
- `subscription-durability` specifies whether or not an associated Topic should be durable.
- `jms-acknowledge-mode` specifies the JMS acknowledgment semantics to use for beans that demarcate their own transaction boundaries. This element has two possible values: `AUTO_ACKNOWLEDGE` (the default) or `DUPS_OK_ACKNOWLEDGE`.

These elements are defined in the `ejb-jar.xml` deployment file, as described in the EJB 2.0 specification. The following excerpt shows a sample XML stanza for defining a message-driven bean:

```
<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>

                     <jms-destination-type>

                            javax.jms.Topic

                     </jms-destination-type>

              </message-driven-destination>

              ...

       </message-driven>

       ...

</enterprise-beans>
```

In addition to the new `ejb-jar.xml` elements, the `weblogic-ejb-jar.xml` file includes a new message-driven-descriptor stanza to associate the message-driven bean with an actual destination in WebLogic Server.

CHAPTER

4 The WebLogic Server EJB Container

The following sections describe the services that are available to EJBs using the WebLogic Server container:

- EJB Life Cycle in WebLogic Server
- Locking Model for Entity EJBs
- ejbLoad() and ejbStore() Behavior for Entity EJBs
- EJBs in WebLogic Server Clusters
- Transaction Management
- Resource Factories
- Persistence Services
- Locking and Caching Services for Entity EJBs
- Home Method Support for Entity EJBs

See “EJB Features and Changes in WebLogic Server” on page 1-1 for a basic introduction to EJB in WebLogic Server.

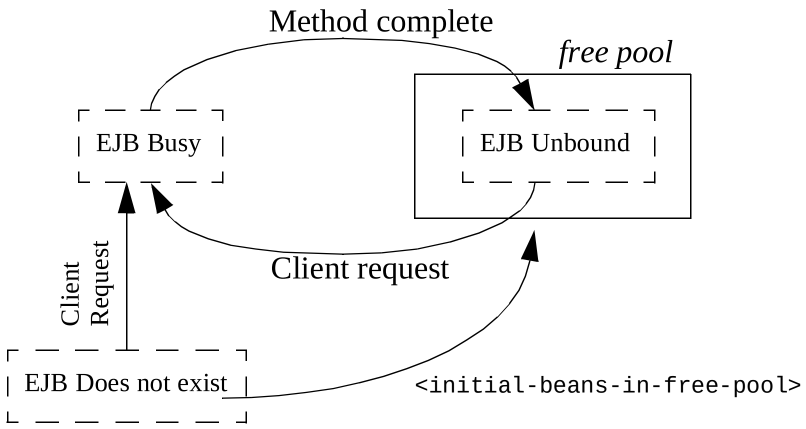
EJB Life Cycle in WebLogic Server

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

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 EJBs are instances of a stateless session EJB class that are not processing a method call.

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



Initializing EJB Instances

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

You can optionally set the initial-beans-in-free-pool property in `weblogic-ejb-jar.xml` to automatically create unbound EJBs in the free pool during startup. This can improve response time when accessing 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 either by available memory, or the value of the max-beans-in-free-pool deployment element.

Activating and Pooling EJBs

When a client calls a method on a stateless 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 is returned to the free pool. Because WebLogic Server unbinds stateless session beans from clients after each method call, the actual bean class instance that a client uses may be different from invocation to invocation.

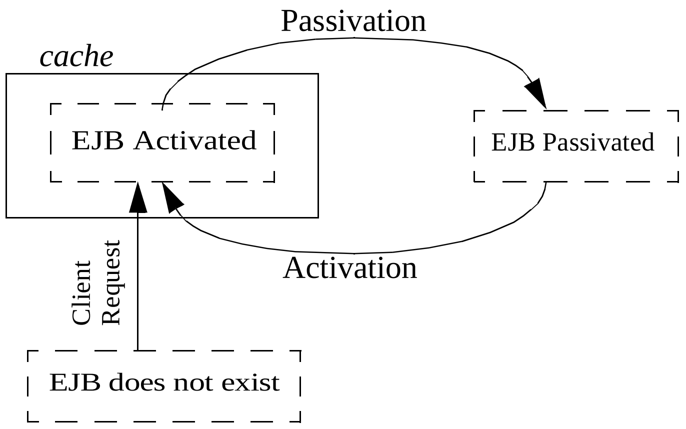
If all instances of an EJB class are active and max-beans-in-free-pool has been reached, new clients requesting the EJB class will be blocked until an active EJB completes a method call. If the transaction times out (or, for non-transactional calls, if five minutes elapse), WebLogic Server throws a `RemoteException`.

Stateful EJB Life Cycle

Note: This section describes the life cycle of stateful session EJBs in WebLogic Server.

WebLogic Server uses a cache of bean instances to improve the performance of stateful EJBs. The cache stores active EJB instances in memory so that they are immediately available for client requests. Active EJBs consist of instances that are currently in use by a client, as well as instances that were recently in use, as described in the following sections. The cache is unlike the free pool insofar as beans in the cache are bound to a particular client (as with stateful session beans).

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



Initializing and Using EJB Instances

No stateful EJB instances exist in WebLogic Server at startup time. (Entity EJBs logically exist in a datastore, but they do not yet exist from the WebLogic Server perspective.) As clients look up and obtain references to individual beans, WebLogic Server initializes new instances of the EJB class and stores them in the cache.

While in cache, the EJBs can be quickly accessed by clients. WebLogic Server locks a cached instance of an entity EJB only for the duration of a transaction. If the EJB is not involved in a transaction, the instance is locked only for the duration of each method invoke. This means that multiple clients can access the same entity EJB in a serial fashion, but only if the bean is not involved in a transaction. See “Locking Model for Entity EJBs” on page 4-8 for more information.

Passivating Stateful EJBs

To achieve high performance, WebLogic Server reserves the cache for EJBs that clients are currently using and EJBs that were recently in use. When EJBs no longer meet these criteria, they become eligible for *passivation*. Passivation is the process

WebLogic Server uses to remove an EJB from cache while preserving the EJB's state on disk. While passivated, EJBs use minimal WebLogic Server resources, but they are not immediately available for client requests (as they are while in the cache).

Note: Stateful session EJBs must abide by certain rules to ensure that bean fields can be serialized to persistent storage. See "Stateful Session EJB Requirements" on page 4-7 for more information.

WebLogic Server provides the max-beans-in-cache deployment element, which provides some control over when EJBs are passivated.

If `max-beans-in-cache` is reached and there are EJBs in cache that are not being used, WebLogic Server passivates some of those beans. This occurs even if the unused beans have not reached their `idle-timeout-seconds` limit. If `max-beans-in-cache` is reached and all EJBs in the cache are being used by clients, WebLogic Server throws a `CacheFullException`.

Note: When an EJB becomes eligible for passivation, it does not mean that WebLogic Server passivates the bean immediately. In fact, the bean may not be passivated at all. Passivation occurs only when the EJB is eligible for passivation and there is pressure on server resources, or when WebLogic Server performs regular cache maintenance.

Removing Stateful Session EJB Instances

The max-beans-in-cache and idle-timeout-seconds deployment elements also provide some control over when stateful session EJBs are removed from the cache or from disk:

- *For cached EJB instances*: When resources become scarce and there is a need for memory in the cache, WebLogic Server examines EJB classes that are approaching their `max-beans-in-cache` limit. Of those beans, WebLogic Server takes EJB instances that have not been used for `idle-timeout-seconds` and removes them from the cache (rather than passivating them to disk). Removing, rather than passivating, the instance ensures that "inactive" EJBs do not consume cache or disk resources in WebLogic Server.

  If a stateful bean has been idle for *longer* than `idle-timeout-seconds`, WebLogic Server may remove the instance from memory as a result of regular cache maintenance, even if EJB's `max-beans-in-cache` limit has not been reached.

Note: Setting `idle-timeout-seconds` to 0 stops WebLogic Server from removing EJBs as part of regular cache maintenance. However, EJBs may still be passivated if cache resources become scarce.

- *For passivated EJB instances*: After a stateful session EJB instance has been passivated, a client must use the EJB instance before `idle-timeout-seconds` is reached. Otherwise, WebLogic Server removes the passivated instance from disk.

Using max-beans-in-free-pool

In general, you should not set the `max-beans-in-free-pool` element. When you ask the free pool for a bean instance, there are three possible options that you can encounter. They are as follows:

- **Option 1:** An instance is available in the pool. You are given that instance and can proceed with processing.
- **Option 2:** No instance is available in the pool, but the number of instance in use is `max-beans-in-free-pool`. In this case, WebLogic Server allocates a new bean instance and gives it to you.
- **Option 3:** No instances are available in the pool and the number of instances in use is already `max-beans-in-free-pool`. In this case, you sleep until either your transaction times out or a bean instance becomes available.

By default, `max-beans-in-free-pool` is the Int.max. That does not mean that you will be able to use 2 billion instances. Essentially, it means that Option 3 never happens. If a pooled instance does not exist, you will always just allocate a new one. In reality, you are limited by the number of executable threads. In most cases, each thread will need, at most, a single bean instance.

The only reason to set max-beans-in-free-pool is to limit access to an underlying resource. For example, if you use stateless session EJBs to implement a legacy connection pool, you do not want to allocate more bean instance than the number of connections that can be supported by your legacy system.

Stateful Session EJB Requirements

The EJB developer must ensure that a call to the `ejbPassivate()` method leaves a stateful session bean in a condition where WebLogic Server can serialize its data and passivate the bean's instance. During passivation, WebLogic Server attempts to serialize any fields that are not declared `transient`. This means that you must ensure that all non-`transient` fields represent serializable objects, such as the bean's remote or home interface. With the EJB 1.1 specification, an EJB's non-`transient` fields can also include:

- A reference to the EJB's JNDI environment context
- A reference to the `UserTransaction` object

As of the EJB 1.1 specification, references to the `javax.ejb.SessionContext` object *cannot* be declared `transient`.

Special Use of max-beans-in-free pool

The following options describe special cases when `max-beans-in-free-pool can` be set to `0`:

- **Entity Beans:** Never pool instances for entity beans. Instead, always allow a new instance to be created.
- **Stateless Session Beans**: Always create a new instance for stateless session beans.
- **Stateful Session Beans:** Not applicable for stateful session beans. These beans are not pooled.
- **Message-Driven Beans:** Illegal instances of message-driven beans are created and registered as JMS listeners during deployment. WebLogic Server never creates new instances at runtime. So, this value must be set to `> 0`.

Locking Model for Entity EJBs

Database concurrency is the default for EJB 1.1 and 2.0. It must be set in the deployment descriptors.

Exclusive locking was the default in WLS 5.1 and 4.5.1. This method of locking provides reliable access to EJB data, and avoids unnecessary calls to `ejbLoad()` to refresh the EJB instance's persistent fields. However, in certain circumstances pessimistic locking may 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.

To improve concurrent access for entity EJBs, the WebLogic Server EJB 2.0 container enables you to defer locking services to the underlying database. In most cases, the underlying data store can provide finer granularity for locking EJB data, and improve throughput for concurrent access to the bean's data. See"EJB Features and Changes in WebLogic Server" on page 1-1 for more information.

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, this is called `in demand`, or WebLogic Server may initiate a new transaction in accordance with the bean's method transaction attributes, this is called lazy loading .
2. WebLogic Server calls `ejbLoad()` to read the most current version of the bean's persistent data from the underlying data store. `ejbLoad()` is used with BMP and CMP 1.1.

BMP and CMP 2.0 can use `in demand` or `lazy loading`, which resets the bean and then when needed reads the most current version of the beans's persistent data from the underlying store the next time the bean is loaded.

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 data store 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 parameters that enable you to configure `ejbLoad()` and `ejbStore()` behavior.

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

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

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

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

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

Restrictions and Warnings for db-is-shared

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

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

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

Note: This method is used for 1.1 CMP beans only. WebLogic Server 6.0's 2.0 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 BMP as you would need to create both the `is-modified-method-name` method. and the `ejbstore`.

By default, WebLogic Server calls `ejbStore()` at the successful completion (commit) of each transaction. `ejbStore()` is called at commit time regardless of whether the EJB's persistent fields were actually updated. WebLogic Server provides the `is-modified-method-name` deployment parameter 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 using the is-modified-method-name element in `weblogic-ejb-jar.xml`. WebLogic Server calls the specified method name when a transaction commits, and calls `ejbStore()` only if the method returns "true."

Warning for is-modified-method-name

`is-modified-method-name` can improve performance by avoiding unnecessary calls to `ejbStore()`. However, it places a greater burden on the EJB developer to correctly identify 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 to become "lost" in your system, start by ensuring that all `is-modified-method-name` methods 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 element. When you set this element to "false," WebLogic Server calls `ejbStore()` after each method call, rather than at the conclusion of the transaction.

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

Setting the Entity EJBs to Read-Only

Entity EJBs can also use the `read-only` cache strategy to modify basic `ejbLoad()` and `ejbStore()` behavior:

You can set the cache strategy by directly editing the cache-strategy element in the `weblogic-ejb-jar.xml` deployment file.

Read-Write Cache Strategy

The `read-write` strategy defines the default caching behavior for entity EJBs in WebLogic Server. With `read-write` entity EJBs, multiple clients can use the same bean instance in transactions, and data integrity is ensured.

For `read-write` EJBs, WebLogic Server loads EJB data into the cache at the beginning of each transaction, or as described in "Using db-is-shared to Limit Calls to ejbLoad()" on page 4-9. 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()" on page 4-10.

Read-Only-Cache-Strategy

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

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

Restrictions for Read-Only EJBs

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

- They cannot require updates to the EJB data, because WebLogic Server never calls `ejbStore()` for read-only entity EJBs.

- Their transaction attributes must be set to `NotSupported` (the beans cannot rely on a transaction).
- The EJB's method calls must be idempoten. See "EJBs in WebLogic Server Clusters" on page 4-14 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-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.

In a read-mostly pattern, a `read-only` entity EJB retrieves bean data at intervals specified by `read-timeout-seconds`. A separate `read-write` entity EJB models the same data as the `read-only` EJB, and updates the data at required intervals. See the read-mostly EJB example in the WebLogic Server distribution for more information.

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

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

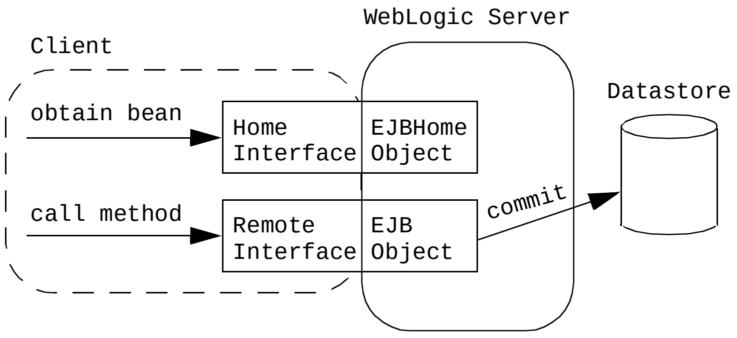
Note that in a WebLogic Server cluster, clients of the `read-only` EJB benefit from using cached EJB data. Clients of the `read-write` EJB benefit from true transactional behavior, since the `read-write` EJB's state always matches the state of its data in the underlying datastore. See "Entity EJBs in a Cluster" on page 4-19 for more information.

EJBs in WebLogic Server Clusters

This section describes the behavior of EJBs and their associated transactions in a WebLogic Server cluster, and explains key deployment descriptors that affect EJB behavior in a cluster.

Overview

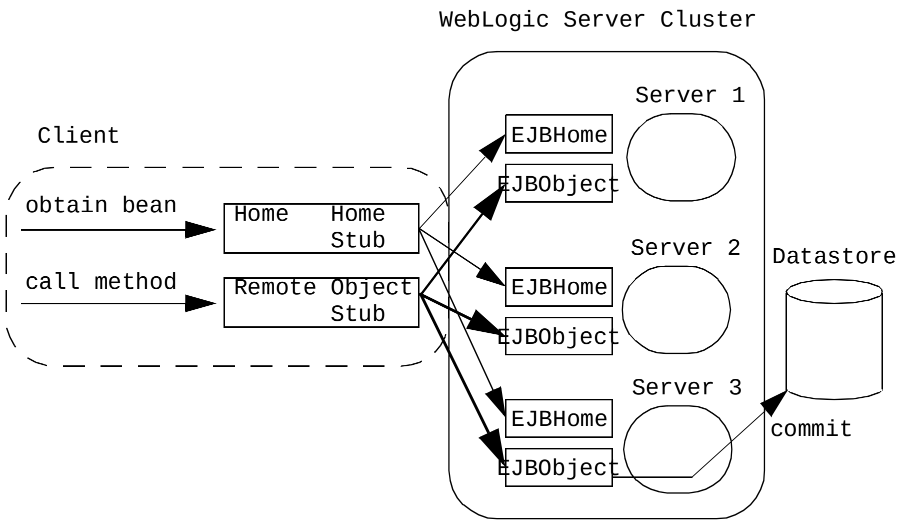
EJBs in a WebLogic Server cluster operate using modified versions of two key structures: the Home object and the EJB object. In a single server (unclustered) environment, a client looks up an EJB via the EJB's home interface, which is backed on the server by a corresponding Home object. After referencing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



Clustered EJBHome Objects

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

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



Clustered EJBObjects

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

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

Session EJBs in a Cluster

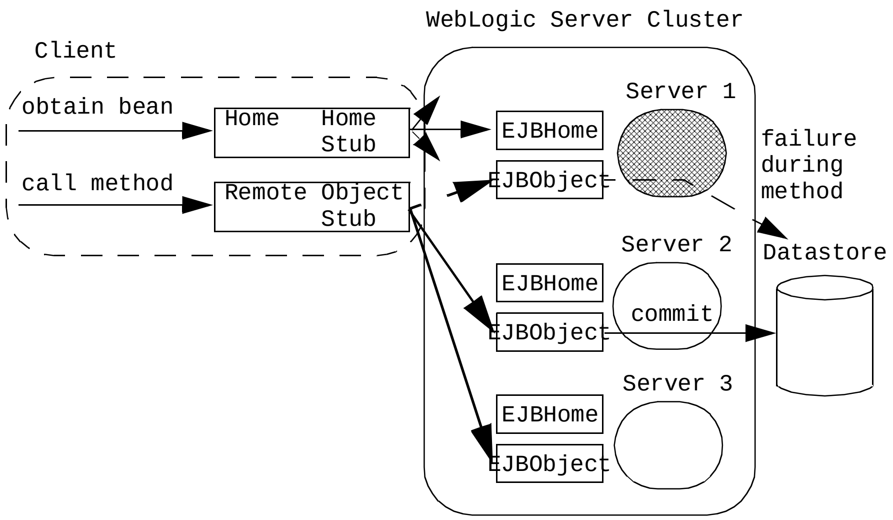
The sections that follow describe cluster capabilities and limitations for different EJB types.

Stateless Session EJBs

Stateless session EJBs can have both a cluster-aware home stub and a replica-aware EJBObject stub. By default, WebLogic Server provides failover services for EJB method calls, but only if a failure occurs *between* method calls. For example, failover is automatically supported if there is a failure 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 failover from one server to another.

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

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



Stateful Session EJBs

Stateful session EJBs can utilize cluster-aware home stubs by setting `home-is-clusterable` to "true." This provides failover and load balancing for stateful EJB lookups. Stateful session EJBs cannot use replica-aware EJBObject stubs, and WebLogic Server does not provide failover services for method calls to stateful session EJBs.

If you require cluster failover services for stateful objects, consider implementing the stateful session EJB as a servlet. Servlets can maintain state through failover in a cluster using either JDBC, an operating system file, or directly in memory. For more information on in-memory replication for stateful session EJBs, see "In-Memory Replication for Stateful Session EJBs" on page 4-18.

In-Memory Replication for Stateful Session EJBs

The WebLogic Server EJB container introduces new clustering support for stateful session EJBs. Whereas in WebLogic Server 5.1 only the `EJBHome` 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 instance 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 is used to account for the possibility of 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 4-19.

Requirements and Configuration

To replicate the state of a stateful session EJB in a WebLogic Server cluster, ensure 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 descriptors. 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. To enable replication, set the replication-type deployment parameter to `InMemory` in the `weblogic-ejb-jar.xml` deployment file. For example:

```
<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 certain rare failover scenarios, the last committed state may not be available. This can happen when:

- A client commits a transaction involving a stateful EJB, but the primary WebLogic Server fails before the EJB's state is replicated. In this scenario, the client's next method invocation will work against the previous committed state, if available.
- 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. In this scenario the client's next method invocation will fail to locate the bean instance, because the initial state could not be replicated. The client would need to recreate the EJB instance using the clustered `EJBHome` stub, and restart the transaction.
- Both the primary and secondary servers fail. In this scenario the client would need to recreate the EJB instance and restart the transaction.

Entity EJBs in a Cluster

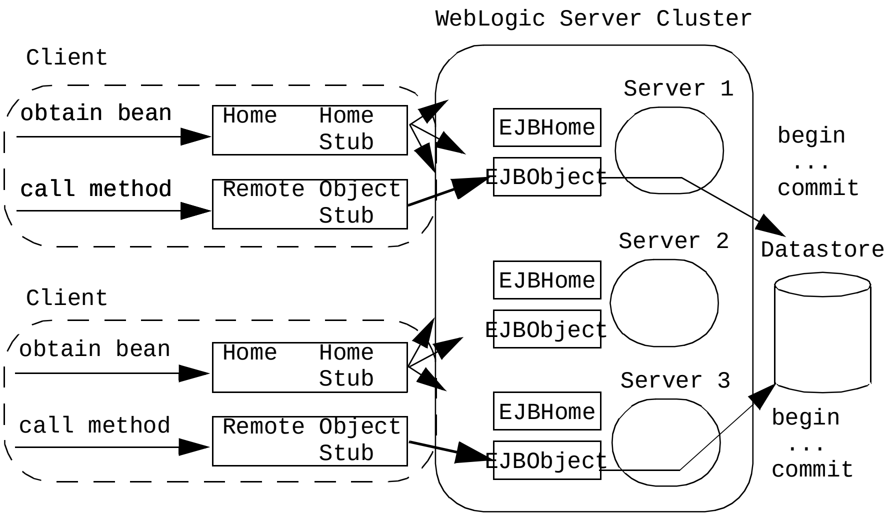
As with all EJB types, entity EJBs can utilize cluster-aware home stubs by setting `home-is-clusterable` to "true." The behavior of the EJBObject stub depends on the cache-strategy deployment property in `weblogic-ejb-jar.xml`.

Read-Write Entity EJBs

`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 (since the `db-is-shared` deployment parameter cannot be set to "false" in a cluster).
- `ejbStore()` behavior is governed by the same rules described in ejbLoad() and ejbStore() Behavior for Entity EJBs.



`read-write` entity EJBs do not use a clustered `EJBObject` stub; a client's method calls to a particular EJB always go to a single WebLogic Server instance. If the server that a client is using fails, the client must relocate the entity EJB using the cluster-aware home stub.

`read-write` entity EJBs are cached on individual WebLogic Server instances. As clients of a given EJB use the bean in transactions, the WebLogic Server instance passes the transaction to the underlying datastore. This approach preserves data integrity for clustered entity EJBs by allowing the backing datastore to manage all transactional locking.

Non-Transactional Datastores

If your backing store does not support transactional locking, additional design may be required to preserve data integrity for entity EJBs in a cluster. One approach is to ensure that all updates to a given entity EJB take place on the same WebLogic Server instance. This approach forces multiple clients to access the EJB in a serial fashion, because the WebLogic Server instance locks the EJB during transactions.

To implement this solution, you create a custom "dictionary" object to keep track of which WebLogic Server instance is currently hosting a given entity EJB instance. Clients to the EJB need to look up EJBs using the dictionary object, rather than the EJB's primary key class.

To support failover, the dictionary object also needs to keep track of which WebLogic Server instances are currently available. This process can be accomplished by having each participating WebLogic Server instance populate the JNDI tree with a unique name. The dictionary object then polls the local JNDI tree at regular intervals to determine whether a participating server's JNDI "signature" available.

Transaction Management

The following sections describe EJBs in several transaction scenarios. EJBs that engage in distributed transactions (transactions that make updates in multiple datastores) cannot guarantee that all branches of the transaction commit or roll back as a logical unit.

The current version of WebLogic Server supports Java Messaging Service (JMS), which can be used for implementing distributed transactional applications.

Transaction Management Responsibilities

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

If bean- or client-managed transactions are required, the managing code must use the `javax.transaction.UserTransaction` interface. The EJB or client can then access a `UserTransaction` object via JNDI and specify transaction boundaries with explicit calls to `tx.begin()`, `tx.commit()`, `tx.rollback()`, and so forth. See "Using javax.transaction.UserTransaction" on page 4-22 for more information on defining transaction boundaries.

For EJBs that use container-managed transactions (or EJBs that mix container and bean-managed transactions) the EJB 1.1 specification defines several deployment elements to control the transactional requirements for individual EJB methods.

You can use two-phase commmit your EJBs. The **two-phase commit protocol** is a method of coordinating a single transaction across two or more resource managers. However, when usingtwo-phase commit with EJB 1.1 beans, you need to set up a `txdatasource` for those beans. To set up the `txdatasource`, see "Setting Up JDBC Datasource Factories" on page 4-26.

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.

Using javax.transaction.UserTransaction

To define transaction boundaries in EJB or client code, you must obtain a `UserTransaction` object and begin a transaction *before* you obtain a Java Transaction Service (JTS) or JDBC database connection. If you start a transaction after obtaining a database connection, the connection has no relationship to the new transaction, and there are no semantics to "enlist" the connection in a subsequent transaction context. If a JTS connection is not associated with a transaction context, it operates similarly to a standard JDBC connection, 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 "Preserve Transaction Resources" on page 2-4 for more information.

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

Restriction for Container-Managed EJBs

For container-managed entity EJBs, you can use the `javax.transaction.UserTransaction` interface or access a transactional JDBC connection from within an EJB method. However, you must ensure that the transaction does not access the bean's container-managed database fields. Also note that the nested transaction in the EJB method ultimately commits or rolls back depending on the fate of the entity EJB transaction.

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. This can be accomplished explicitly 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 has been set to `Required`:

```
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.

This type of transaction operates similarly to the scenario described in “Calling Multiple EJBs from a Single Transaction Context” on page 4-23. The “wrapper” EJB may explicitly obtain a transaction context before invoking other EJBs, or WebLogic Server may automatically create a new transaction context if the EJB’s `trans-attribute` element is set to `Required` or `RequiresNew`. 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 utilize 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.

Transaction Isolation Level

The isolation level for transactions is set in the transaction-isolation element of the `weblogic-ejb-jar.xml` deployment file. WebLogic Server passes this value to the underlying database. The behavior of the transaction depends both on the EJB's isolation level setting and the concurrency control of the underlying persistent store.

To mirror the transaction behavior in earlier versions of WebLogic Server, set transaction-isolation to a value that is consistent with the default isolation level for your data store.

Limitations of TRANSACTION_SERIALIZABLE

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

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

Special Note for Oracle Databases

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

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

Even if you use the `TRANSACTION_SERIALIZABLE` setting for an EJB, you may receive exceptions or rollbacks in the EJB client if contention occurs between clients for the same rows. To avoid these problems, you must ensure that clients catch and examine the SQL exceptions, and take appropriate action, such as restarting the transaction.

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

Resource Factories

In WebLogic Server Version 6.0, EJBs can access JDBC connection pools by directly instantiating a JDBC driver. However, it is recommended that you instead bind a JDBC 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 and URL resources to JNDI names in WebLogic Server.

Note: WebLogic Server also supports JMS connection factories.

Setting Up JDBC Datasource Factories

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

1. Set up a JDBC connection pool in the Administration Console.
2. Start the WebLogic Server.
3. Start the WebLogic Server Administration Console.
4. In the Console, click the Services node and expand JDBC.
5. Select Data Sources and choose the Create a new Data Source option.
6. *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.

For example, to set up a non-transactional connection pool for demonstration purposes, you might enter:

```
weblogic.jdbc.DataSource.weblogic.jdbc.demoPool=demoPool
```

This binds a transactional datasource for the "demoPool" pool to the JNDI name, "weblogic.jdbc.demoPool".

For transactional JDBC datasources, 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 example, to set up a non-transactional connection pool for demonstration purposes, you might enter:

```
weblogic.jdbc.TXDataSource.weblogic.jdbc.jts.demoPool=demoPool
```

This binds a transactional datasource for the "demoPool" pool to the JNDI name, "weblogic.jdbc.jts.demoPool".

7. Click Create to save the changes.
8. Bind the JNDI name of the datasource to the EJB's local JNDI environment. To do this:
   - Map an existing EJB resource factory reference to the JNDI name, or
   - Directly edit the resource-description tag in the `weblogic.ejb-jar.xml` deployment file using a text editor

Setting up URL Connection Factories

To setup 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 requred, 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 config.xml and reboot WebLogic Server.

Persistence Services

An entity EJB can save its state in any transactional or non-transactional persistent storage (“bean-managed persistence”), or it can ask the container to save its non-transient instance variables automatically (“container-managed persistence”). WebLogic Server allows both choices and a mixture of the two.

If an EJB uses container-managed persistence, the `weblogic-ejb-jar.xml` deployment file specifies the type of persistence services that an EJB uses. High-level definitions for automatic persistence services are stored in the persistence-type and persistence-use elements. `persistence-type` defines one or more automatic services that the EJB may use. `persistence-use` defines the actual service that 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 4-29.

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

Using WebLogic Server RDBMS Persistence

To use WebLogic Server RDBMS-based persistence, you must create a dedicated XML deployment file and define the persistence elements for each EJB that will use container-managed persistence. If this file is created by WebLogic Server utilities, such as DDConverter, 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 options for EJBs using WebLogic Server RDBMS-based persistence services.

Each `weblogic-cmp-rdbms-jar.xml` defines the following persistence options:

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

Writing Finders for RDBMS Persistence

For EJBs that use RDBMS persistence, WebLogic Server Version 6.0 provides an easy way to write dynamic finders. The EJB provider writes the method signature of a finder in the EJBHome interface, and defines the finder's query expressions in the `ejb-jar.xml` deployment file.

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

The key components of a finder for RDBMS persistence are:

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

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

Finder Signature

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

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

finder-list Stanza

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

```
<finder-list>

     <finder>

          <method-name>findBigAccounts</method-name>

          <method-params>

               <method-param>double</method-param>

          </method-params>

          <finder-query><![CDATA[(> balance $0)]]></finder-query>

     </finder>

</finder-list>
```

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

finder-query Element

`finder-query` defines the WebLogic Query Language (WLQL) expression used to query EJB objects from the RDBMS. WLQL uses a standard set of operators against finder parameters, EJB attributes, and Java language expressions. See "Using WebLogic Query Language (WLQL)" on page 4-31 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.

Using WebLogic Query Language (WLQL)

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.

Note: Use WLQL for EJBs and its corresponding deployment files that are based on the EJB 1.1 specification. WLQL is deprecated, and we do not recommend that you continue to use it as it is not longer supported.

Syntax

WLQL strings use the prefix notation for comparison operators:

```
(operator operand1 operand2)
```

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

Operators

The following are valid WLQL operators.

Operator	Description	Sample Syntax
`=`	Equals	`(= operand1 operand2)`
`<`	Less than	`(< operand1 operand2)`
`>`	Greater than	`(> operand1 operand2)`

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

Operands

Valid WLQL operands include:

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

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

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

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

Examples of WLQL Expressions

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

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

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

  The sample `<finder>` stanza is:

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

  Note that the `balance` field must be defined in 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[(orderBy 'id' (> ))]]></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 Java Expressions in WLQL

WebLogic Server supports embedded Java statements in WLQL for writing advanced finder capabilities. The basic WLQL functionality described in “Using WebLogic Query Language (WLQL)” on page 4-31 provides capabilities that are necessary for most finders.

However, WebLogic Server Version 6.0's EJB 2.0 container uses 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, we recommend that you create entity EJB finders *without* embedding Java expressions in WLQL.

This section describes how to embed Java expressions into WLQL by editing XML elements directly in the deployment file.

In basic WLQL expressions, there is a one-to-one mapping between the parameters in the finder method signature and the expressions designated by the `$n` notation. For example, in the method signature:

```
public Enumeration findSomeAccounts(double maxBal, String ownerID)
     throws FinderException, RemoteException
```

WLQL assigns `$0` to the value of `maxBal` and `$1` to the value of `OwnerID` by default. Using this default mapping, you can create finders such as:

```
<finder>
     <method-name>findSomeAccounts</method-name>
     <method-params>
          <method-param>double</method-param>
          <method-param>java.lang.String</method-param>
     </method-params>
     <finder-query><![CDATA[(& (< balance $0) (= owner $1))]]></finder-query>
</finder>
```

WLQL also enables EJB developers to embed Java expressions within WLQL expressions designated by `$n`. To use this feature, you must override the assignment of individual `$n` expressions using the `finder-expression` stanza in `weblogic-cmp-rdbms-jar.xml`.

The `finder-expression` stanza includes the following XML elements:

- `expression-number` identifies the number of the expression you want to override. The expression number must be unique among the expressions defined for a finder method. Expression numbers start at zero and continue in sequence up to the total number of expressions used in the finder.

- `expression-text` contains the full Java expression that will replace the `$`*n* expression number in the finder's WLQL string. Within the expression text, you can use the `@`*n* notation to refer to the *n*th parameter in the finder method signature. Expressions need not map to individual parameters, advanced finders can define more expressions than parameters.

  Note: Always define the value of `expression-text` using the XML `CDATA` attribute. Using `CDATA` ensures that the Java expression does not cause compilation errors when the finder is compiled.

- `expression-type` identifies the Java return type of the Java expression. The expression must produce a value that is compatible with the specified `expression-type`. If you are unsure of the exact type of the return value, specify an encompassing Java type (such as `long` when you expect that only an `int` is necessary).

  Note: If the generated Java expression is incompatible with the specified `expression-type`, `ejbc` displays a error when you compile the EJB code.

Example of Finder Method Signatures

The following example uses the finder method signature:

```
public Enumeration findSomeAccounts(double maxBal, String ownerID)
     throws FinderException, RemoteException
```

By embedding a simple Java expression in the WLQL string, you can convert the supplied `maxBal` value to another currency before querying the RDBMS. For example, if `maxBal` is supplied in U.S. dollars and the conversion rate to pounds is 1.6483, you can use a simple expression to multiply the value:

```
<finder>
       <method-name>findSomeAccounts</method-name>
       <method-params>
              <method-param>double</method-param>
              <method-param>java.lang.String</method-param>
```

```
        </method-params>

        <finder-query>(& (< balance $0) (= owner $1))</finder-query>

        <finder-expression>

                <expression-number>0</expression-number>

                <expression-text>@0 * 1.6483</expression-text>

                <expression-type>long</expression-type>

        </finder-expression>

</finder>
```

In the above example, `$0` is replaced by the Java expression `@0 * 1.6483`, which multiplies the value of `maxBal` by 1.6483. Because the EJB provider did not override the value of `$1`, WLQL maps `$1` to the second parameter in the finder method signature, `ownerID`.

A more advanced version of this finder could use Java to determine the conversion rate when converting `maxBal`:

```
...

<finder-expression>

     <expression-number>0</expression-number>

     <expression-text>@0 *
Double.parseDouble(System.getProperties().get(“rate.pounds.dollar
s”))</expression-text>

     <expression-type>long</expression-type>

</finder-expression>

...
```

Restrictions

WebLogic Server does not parse or in any way validate the expression supplied in `expression-text`. The EJB provider must ensure that the expression is valid Java.

Any errors in the expression text will appear as compilation errors when you compile the EJB with ejbc.

Locking and Caching Services for Entity EJBs

The WebLogic Server Version 6.0 container supports both the database locking and exclusive locking mechanisms. The default is database locking.

Pessimistic Locking Services

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, When using exclusive locking, WebLogic Server places an exclusive lock on the EJB instance or method for the duration of the transaction. Other clients requesting the same EJB or method block until the current transaction completes.

This method of locking provides reliable access to EJB data, and avoids unnecessary calls to `ejbLoad()` to refresh the EJB instance’s persistent fields. However, pessimistic 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.

To improve concurrent access for entity EJBs, the WebLogic Server container enables you to defer locking services to the underlying database. In most cases, the underlying data store can provide finer granularity for locking EJB data (as well as provide deadlock detection), and improve throughput for concurrent access to the bean’s data.

Database Locking Services

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

Setting Up Database Locking

The deployment descriptors specifies the locking mechanism to use for an EJB by setting the concurrency-strategy deployment parameter in `weblogic-ejb-jar.xml`. concurrency-strategy is set at the individual EJB level, so that you can mix locking mechanisms within the EJB container.

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

```
<entity-descriptor>

       <entity-cache>

       ...

       <concurrency-strategy>Database</concurrency-strategy>

       </entity-cache>

       ...

</entity-descriptor>
```

If you do not specify `concurrency-strategy`, WebLogic Server performs database locking for entity EJB instances, as described in "Locking Model for Entity EJBs" on page 4-8.

Home Method Support for Entity EJBs

The EJB 2.0 container provides new support for entity EJB *home methods*. Home methods provide an easy way to perform business logic that does not require access to an actual instance of an EJB. For example, using home methods you can query for supporting data immediately after obtaining the home interface for an EJB (but before using a finder to obtain EJB instances).

The EJB 2.0 specification provides few restrictions on developing and using entity EJB home methods. Each home method must throw `java.rmi.RemoteException`, and its method signature must not resemble a finder method or instance method. (For example, the method cannot begin with `find`, `create`, or `remove`, since those signatures are reserved for working with the EJB instance). Within these restrictions, you can perform any business logic that does not involve the EJB instance.

CHAPTER

5 WebLogic Server Container-Managed Persistence Services

The following sections describe the new container-managed persistence (CMP) services available with the EJB 2.0 for BEA WebLogic Server container. This information supplements the basic discussion of WebLogic RDBMS-based persistence services in Persistence Services.

- EJB 2.0 Persistence Features and Changes
- Using EJB QL
- Using WebLogic Query Language Extension
- Container-Managed Relationships
- Groups
- Supported Data Types

EJB 2.0 Persistence Features and Changes

The EJB 2.0 specification places new requirements on entity EJB finder methods and field accessor methods, and introduces a new, portable query language for creating finders. This section summarizes those features. See the complete EJB 2.0 specification for details.

Note: Container-managed persistence beans need to be configured with a connection pool with maximum connections greater than 1. This is because WebLogic Server's container-managed persistence service may sometimes need to get two connections simultaneously.

"get" and "set" Method Restrictions

The EJB 2.0 specification standardizes the "get" and "set" methods that a container uses for reading and modifying container-managed fields. These container-generated classes must begin with "get" or "set" and use the actual name of a persistent field defined in `ejb-jar.xml`. The methods are also declared as `public`, `protected`, and `abstract`.

BLOB and CLOB DBMS Column Restrictions

WebLogic Server cannot support BLOB and CLOB DBMS columns with EJB CMP at this time, until JDBC defines a standard UPDATE mechanism for BLOBs and CLOBs. The small BLOB/CLOB is accessible by JDBC1.x getX/setX methods. However, if the column exceeds a certain size, the DBMS will shift to normal BLOB/CLOB semantics which are not supported in CMP.

To work around this restriction, you can do one of the following:

- Use BMP with JDBC-specific driver code for UPDATEs.
- Use CMP with LONG RAW/LONG 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 entity EJBs that use EJB 2.0 features, or that use EJB 2.0 deployment files. If you want to use WLQL, the entity EJB and its deployment files must be based on the EJB 1.1 specification. See“Using WebLogic Server RDBMS Persistence” on page 4-29 for more information on WLQL.

isModified() Not Required for CMP Beans

The `isModified()` method is no longer required for CMP entity EJBs based on the EJB 2.0 specification. When you deploy EJB 2.0 entity beans with container-managed persistence, WebLogic Server automatically detects which EJB fields have been modified during a transaction, and writes only those fields to the underlying datastore.

You can still use `isModified()` for entity EJBs in the EJB 2.0 container that use bean-managed persistence and CMP. See is-modified-method-name for more information.

Using EJB QL

The EJB 2.0 specification introduces EJB QL as a portable query language for creating entity EJB finders. EJB QL is an SQL-like language that uses a single `WHERE` clause to select one or more entity EJB objects. 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). Because EJB QL is portable among containers, EJB QL strings are defined in `ejb-jar.xml` rather than in `weblogic-cmp-rdbms-jar.xml`.

Basic EJB QL Syntax

The basic syntax of an EJB QL query consists of a mandatory `FROM` clause followed by a `WHERE` clause.

The `WHERE` clause defines conditions that limit the result set in a similar manner to WebLogic Query Language (WLQL). The remaining sections focus on the EJB QL clauses and operators that you can specify in the `WHERE` clause.

Note: Because EJB QL queries may contain “>” and “<” characters, always specify the entire EJB QL string within the `CDATA` tag to avoid confusing the query syntax with XML elements.

EJB QL String Literals

As with WLQL, EJB QL designates literal string values by enclosing them within single quotes. To represent a single quote as part of a literal string, use two consecutive single quotes, as in:

```
WHERE accountType = ’partner’’s’
```

EJB QL Operators

EJB QL uses many of the same logical and arithmetic operators used in WLQL. See “Migrating from (EJB 1.1) WLQL to (EJB 2.0) EJB QL” on page 5-7 for a list of WLQL operators and their EJB QL counterparts. See the EJB 2.0 specification for more detailed information about all EJB QL operators.

Finder Methods

The home interface of the entity bean defines one or more finder method. One is defined for each way to find an entity object or collection of entity objects within the home. The entity bean implementation uses the arguments of the finder method to locate the requested entity objects. The finder method must be associated with a query element in the deployment descriptor. The entity bean specifies the EJB QL finder query and associates it with the finder method in the deployment descriptor. The finder method is characterized by an EJB QL query string specified in the query element.

Finder Parameter Placeholders

EJB QL enables you to represent finder method parameters using the convention ?*n* where *n* indicates the number of the parameter in the associated finder method. In EJB QL, the first finder parameter is represented by ?1.

If you are familiar with WLQL, be extra careful when converting WLQL strings that contain parameter placeholders. In WLQL, the first finder parameter is represented by 0 (as in $0), whereas in EJB QL the parameter is represented by 1 (?1).

Select Methods

Select methods are special finder methods used within the entity bean instance. Select methods are abstract methods that are defined in the entity bean's implementation class AND are defined by an EJB QL query string. Although these select methods are similar to finder methods, they can return values of any cmp-field or cmr-field.

The EJB QL string that is specified for a select methods must have a SELECT clause that designates the result type. Select methods are executed using the instance on which the query is invoked.

EJB QL Conditional Expressions

The following table summarizes conditional expressions that you can use in the `WHERE` clause of an EJB QL query. For more details on using EJB QL, see the EJB 2.0 specification.

EJB QL Expression	Function	Sample Syntax
`[NOT] BETWEEN`	Compares values that are between (or are not between) a range of values.	`WHERE accountId BETWEEN 1000 and 2000`
`[NOT] IN`	Selects EJBs that match (or do not match) those in a specified list of string literals. Note that the specified list must contain only string literals.	`WHERE accountId IN ('1000', '1020', '1025')`

EJB QL Expression	Function	Sample Syntax

`=, <>`	Selects value that are equal.
	■ Supports CMP field and local bean interfaces only.
		`WHERE accountId = 1000`
		`WHERE account <> 500`

`[NOT] LIKE ESCAPE`	Selects values based on wildcard symbol (%) in the supplied *`text_string`*	WHERE accountType LIKE ‘custom%’
`IS [NOT] NULL`	Tests for NULL values.	`WHERE accountId IS NOT NULL`

EJB QL Examples

The following excerpts show sample EJB QL strings from the `ejb-jar.xml` file.

This example returns a collection of `AccountBean` EJBs whose `balance` values are greater than the parameter passed to the `findBigAccounts` method:

```
<query>

       <query-method>

              <method-name>findBigAccounts</method-name>

              <method-params>

                     <method-param>double</method-param>

              </method-params>

       </query-method>

       <ejb-ql>

              <![CDATA[WHERE balance > ?1]]>

       </ejb-ql>

</query>
```

The following `ejb-ql` element returns an account balance:

```
<ejb-ql><![CDATA[WHERE accountType IN (’partners’,
’channels’)]]></ejb-ql>
```

The following `ejb-ql` element returns a collection of EJBs whose `accountType` field is NULL:

```
<ejb-ql><![CDATA[WHERE accountType IS NULL]]></ejb-ql>
```

Migrating from (EJB 1.1) WLQL to (EJB 2.0) EJB QL

If you have used previous versions of WebLogic Server, you may already have container-managed entity EJBs that utilize (EJB 1.1) WLQL for finder methods. This section provides a quick reference to common (EJB 1.1) WLQL operations, and explains how to express those operations using (EJB 2.0) EJB QL. See the EJB 2.0 specification for a complete discussion of WLQL syntax.

Sample (EJB 1.1)WLQL Syntax	Equivalent (EJB 2.0) 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 operand NOT value`
`(& operand)`	`WHERE expression1 AND expression2`
`(| operand)`	`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 WebLogic Query Language Extension

WebLogic Server has a SQL-like language, called WebLogic QL, that works with the Finder expressions and is used to query EJB objects from the RDBMS. The `query` is defined in the `ejb-jar.xml` deployment descriptor. The WebLogic specific query extension included in Version 6.0 is `ORDERBY`.

ORDERBY

The WebLogic Query Language extension, ORDERBY, is a keyword that works with the Finder method to allow you to specify which CMP file you want to use for your selections.

The following example shows the use of the extension, ORDERBY.

```
ORDERBY

          SELECT A from A for Account.Bean

                    ORDERBY A.id
```

Container-Managed Relationships

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

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

Entity beans can have relationships with other beans. These relationships can be either bidirectional or unidirectional. Also, support is provided for local and 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, they 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-map are specified if the role just specifies a group-name. No group-name is specified if the relationship is remote.

You specify relationships in the `ejb-jar.xml` file and `weblogic-cmp-rdbms-jar.xml`. You specify container-managed field mappings in the `weblogic-cmp-rdbms-jar.xml`.

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

- One-to-one
- One-to-many
- Many-to-many

One-to-One Relationships

A WebLogic Server one-to-one relationship involves the physical mapping from a foreign key in one bean to the primary key in another bean. See "Primary Keys" on page 5-11 for more information on primary keys.

One-to-Many Relationships

A WebLogic Server one -o-many relationship involves the physical mapping from a foreign key in one bean to the primary key of another. However, in a one-to-many relationship, the foreign key is always contained in the role that occupies the “many” side of the relationship.

Many-to-Many Relationships

A WebLogic Server many-to-many relationship involves the physical mapping of a join table. Each row in the join table contains two foreign keys that map to the primary keys of the entities involved in the relationship.

Unidirectional Relationships

Unidrectional relationships can only navigate in one direction. These types of relationships are used with remote beans, and only unidirectional relationships can be remote. A remote bean is one whose abstract persistence schema is not defined in the same ejb-jar file. For example, if entity A and entity B are in a one-to-one, unidirectional relationship and the direction is from entity A to entity B, than entity A is aware of entity B, but entity B is unaware of entity A. This type of relationship is implemented with a cmr-field on the entity bean from which navigation can take place and no related cmr-field on the target entity bean.

Bidirectional Relationships

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

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. The primary key is specified in the deployment descriptor. 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.

A cmp field of the type `BigDecimal` should not be used 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:

1. Implement a primary key class.
2. In this primary key class, implement the `boolean equal (Object x)` method.
3. In the equal method, use `boolean BigDecimal.compareTo(BigDecimal val)`.

Foreign Keys

Foreign key columns in the database that are mapped to cmr fields in a bean must allow null values. If not, an error will be thrown by JDBC when a bean instance is created.

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 out 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.

Field groups are specified in the `weblogic-rdbms-cmp-jar.xml` file. You use field groups when you want to access a subset of fields.

Specifying Field Groups

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

```
<group-name>financial-data</group-name>
<group-name>medical-data</group-name>
       <cmr-field>patient</cmr-field>
       <cmr-field>doctors</cmr-fields>
       <cmr-field>insurance-providers</cmr-fields>
```

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

Using Groups

The field group is an optimizing element that should be used with care because it is possible to corrupt the database.

For example,

You have the following cmp fields; A, B, and C.

A and B belong to the same group.

You set up the following scenario:

```
getA() // loads A and B
modify A
// then an external process modifies the row getC()
```

Because C is not in the group, there are two possibilities:

- The container will load C and all the other fields as well. In this case, the modification that was made to A will be lost.
- The container will load C and only C. When the transaction commits, the new value for A that was assigned during the transaction might overwrite the newer value in the database.

In both cases, the database will be corrupted. The reason is that you told the container that within this transaction, that only A and B would be read and instead C also was read. The correct step to take would have been to add C to the group or to specify no groups at all.

Supported Data Types

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.

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

The SQL CHAR data type should not be used 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/hashCode.

CHAPTER

6 Deploying EJBs to WebLogic Server

The following sections provide an overview of deploying EJBs to WebLogic Server, and explain how to set the deployment properties used to EJBs. They also describe the two different ways you can deploy EJBs to WebLogic Server: *static deployment* and *dynamic deployment*.

- Required Steps for Deploying EJBs
- Deploying EJBs at WebLogic Server Startup
- Deploying EJBs in a Running WebLogic Server (Dynamic Deployment)

Required Steps for Deploying EJBs

Deploying EJBs to WebLogic Server involves the following steps:

1. Setting the EJB deployment descriptors
2. Generating EJB container classes
3. Loading EJB classes in
4. Deploying EJBs at WebLogic Server startup or dynamically

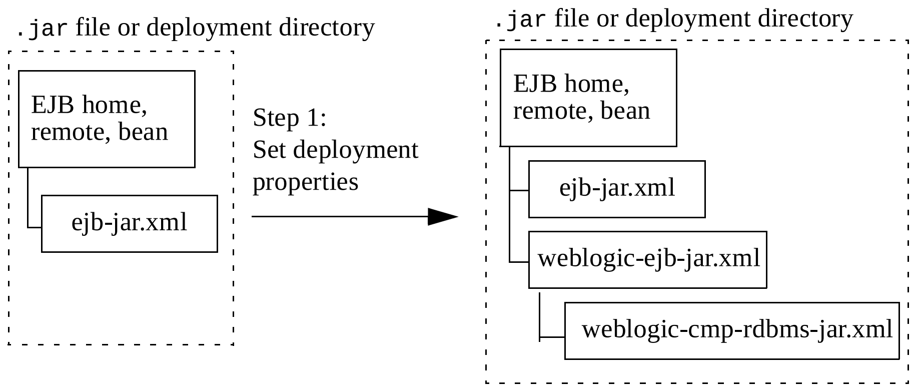
The following sections described these steps in detail.

Setting Deployment Properties

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.

Note: The deployment descriptors do not need to include a `MANIFEST` file in the `.jar` file, as was required with the EJB 1.0 specification. See the JavaSoft EJB 1.1 or 2.0 specification for more information.


.jar file or deployment directory
EJB home, remote, bean
ejb-jar.xml
Step 1: Set deployment properties
.jar file or deployment directory
EJB home, 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 WebLogic Server-specific deployment descriptions in the `weblogic-ejb-jar.xml` file, and add that file to the deployment. `weblogic-ejb-jar.xml` defines caching, clustering, and performance behavior, and is required for all EJBs. See "weblogic-ejb-jar.xml Deployment Descriptor File" on page 10-4 for a complete list of properties available in the file.

If you are deploying an entity EJB that uses container-managed persistence, you must also include an additional deployment file for the bean's persistence type. For WebLogic Server RDBMS-based persistence services the file is generally named `weblogic-cmp-rdbms-jar.xml`, and you require a separate file for each bean that uses RDBMS persistence. 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.

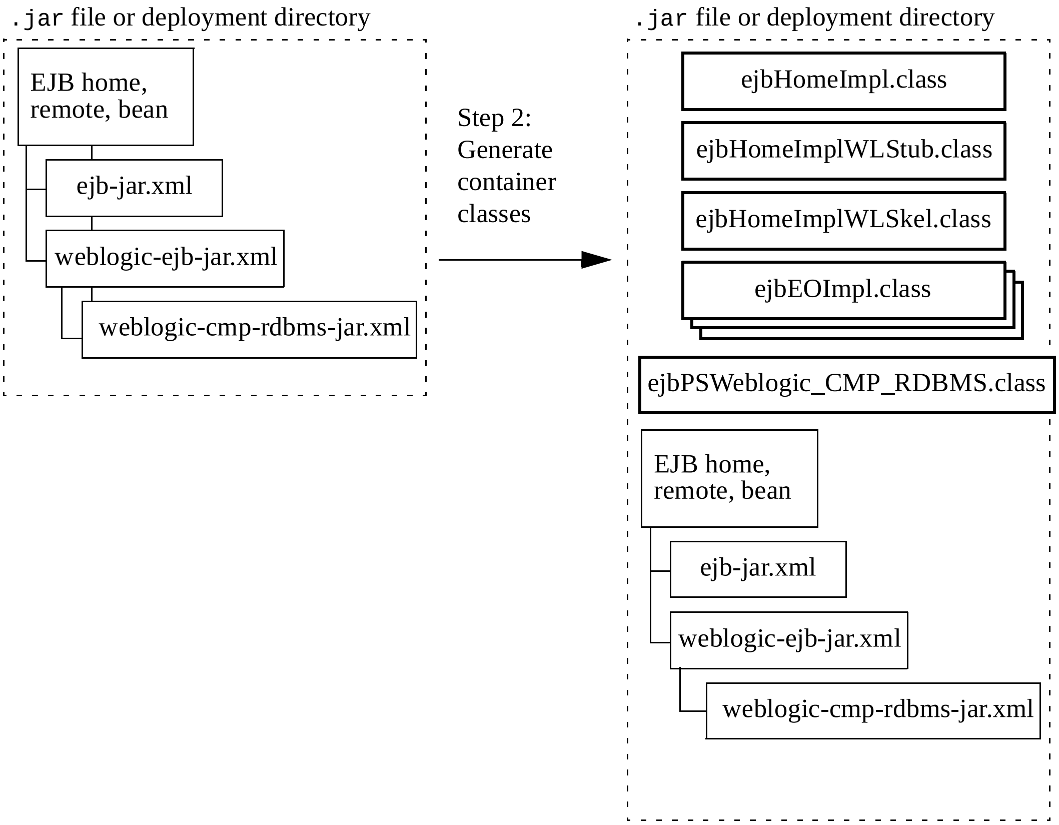
If you do not have an `ejb-jar.xml` file, you must manually create or edit an existing one to set the necessary deployment properties for the EJB. You can use a text editor to edit the properties.

See "Manually Editing XML Deployment Files" on page 9-1 for a description of the WebLogic Server Version 6.0 specific properties and files, and guidelines for editing those files by hand.

See "Manually Editing XML Deployment Files" on page 10-1 for a description of the WebLogic Server 5.1 specific properties and files, and guidelines for editing those files by hand. These properties are provided for reference purposes, in case you need to edit the deployment files for 1.0 or 1.1 EJBs. To deploy an EJB to a WebLogic Server 2.0 EJB container, you can either edit the deployment properties of existing EJB 1.0 or 1.1 beans, or use the `DDConverter` utility, provided with WebLogic Server Version 6.0, to convert the deployment descriptors.

Generating EJB Container Classes

After you compile the EJB classes and add the required WebLogic Server XML deployment files identified in “Setting Deployment Properties” on page 6-2 to your deployment unit, you must generate the container classes that are used to access the EJB. Container classes include both the internal representation of the EJB that WebLogic Server uses, as well as implementation of the external interfaces (home and remote) that clients use.



The `ejbc` compiler generates container classes according to the deployment properties you have specified in WebLogic Server-specific XML deployment 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 8-1 for more information.

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 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*.

Deploying EJBs at WebLogic Server Startup

To deploy EJBs automatically when WebLogic Server starts:

1. Follow the instructions in "Setting Deployment Properties" on page 6-2 to ensure that your deployable EJB `.jar` file or deployment directory contains the required WebLogic Server XML deployment files.

2. Use a text editor to edit the XML deployment properties, as necessary.

3. Follow the instructions in "Generating EJB Container Classes" on page 6-4 to compile implementation classes required for WebLogic Server.

   Compiling the container places the `.jar` file in the `weblogic/config/examples/applications` directory, where it is automatically deployed when WebLogic Server starts. If your EJB `.jar` file is located in a different directory, make sure that you copy it to this directory if you want to deploy it at startup.

4. Start WebLogic Server.

   When you boot WebLogic Server, it automatically attempts to deploy the specified EJB `.jar` file or deployment directory.

5. Launch the Administration Console.
6. In the right pane, click EJB Deployments.

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

Deploying EJBs in Different Applications

When EJBs are not depoyed in the same application, call by reference cannot be used to invoke on the EJBs. Instead, pass by value would be used. Components that commonly interact with each other should be located in the same application where they can do call by reference. 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 the EJB is called remotely (not from within the server

Deploying EJBs in a Running WebLogic Server (Dynamic Deployment)

Although placing the EJB `.jar` file or deployment directory in the `weblogic/config/examples/applications` directory allows you to automatically deploy the EJB, it requires that you start or reboot the server before the bean can be used. Once a bean has been deployed, another edit and reboot is required to remove the bean.

Dynamic deployment is provided for situations where rebooting WebLogic Server is not feasible. Using dynamic deployment features, you can:

- Deploy a newly developed EJB to a running, production 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 dynamic deployment features. The following sections describe dynamic deployment concepts and procedures.

To work with beans in a dynamic deployment environment (running server), you first follow the steps described in "Required Steps for Deploying EJBs" on page 6-1.

EJB Deployment Names

When you deploy an EJB `.jar` file or deployment directory, you must specify a name for the deployment unit. This name provides 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.

Viewing Deployed EJBs

To view EJBs that are deployed on a local WebLogic Server, use the following steps:

From the command line:

1. Enter the following:

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

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

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

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

From the WebLogic Server Administration Console:

1. Choose EJB Deployments from J2EE node in the right-hand pane of the Console.
2. View a list of deployed EJBs.

Deploying New EJBs into a Running Environment

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

Use the command:

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

where:

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

For example:

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

Undeploying Deployed EJBs

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

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

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

Undeploying EJBs

To undeploy a deployed EJB, use the following steps:

From the command line:

you need only reference the assigned deployment unit name, as in:

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

From the WebLogic Server Administration Console:

1. Choose EJB Deployments from J2EE node in the right-hand pane of the Console.
2. Click the EJB you want to undeploy from the list.
3. Choose the Target tab and undeploy the EJB by toggling the server to the Available column.

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

Updating Deployed EJBs

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

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

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

The Update Process

When you update the currently-loaded implementation, classes for the EJB are immediately marked as unavailable in the server, and the EJB's classloader and associated classes are removed. WebLogic Server automatically propagates a `RedeploymentException` to all clients that were using the bean. At the same time, a new EJB classloader is created, which loads and maintains the revised EJB implementation classes.

When clients next acquire a reference to the EJB, their EJB method calls use the updated EJB implementation classes.

Note: You can update only the EJB implementation classes, as described in "Loading EJB Classes into WebLogic Server" on page 6-5. You cannot update the EJB's public interfaces, or any support classes that are used by the public interfaces. If you make any changes to the EJB's public classes and attempt to update the EJB, WebLogic Server displays an incompatible class change error when a client next uses the EJB instance.

Updating the EJB

To update or redeploy the EJB implementation class, use the following steps:

From the command line:

Use the `update` argument and specify the active EJB deployment name:

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

From the WebLogic Server Administration Console:

1. Choose EJB Deployments from J2EE node in the right-hand pane of the Console.
2. Click the EJB you want to update from the list.
3. Choose the Target tab and update the EJB by toggling the server to the Chosen column.

You can update only the EJB implementation class, not the public interfaces or public support classes

.

7 Deploying EJBs in the EJB Container

The BEA WebLogic Server container provides simplified methods for deploying and redeploying EJBs. The following sections describe how to deploy EJBs into the EJB 2.0 container.

You can continue to deploy EJB 1.1 beans into the EJB 1.1 container by using the instructions in "Deploying EJBs in a Running WebLogic Server (Dynamic Deployment)" on page 6-6. You should become familiar with those EJB 1.1 instructions before attempting to deploy EJB 2.0 beans.

- Roles and Responsibilities
- WebLogic Server Deployment Files
- Automatic Deployment Directory
- Deploying Compiled EJB .jar Files
- Deploying Uncompiled EJB .jar Files
- Deploying from an EJB .jar Directory

Roles and Responsibilities

The following sections are written primarily for:

- Deployers who are configuring EJBs to run in the WebLogic Server container

- Application Assemblers who are linking multiple EJBs and EJB resources to create larger web application systems
- EJB developers who are creating and configuring new EJB `.jar` files

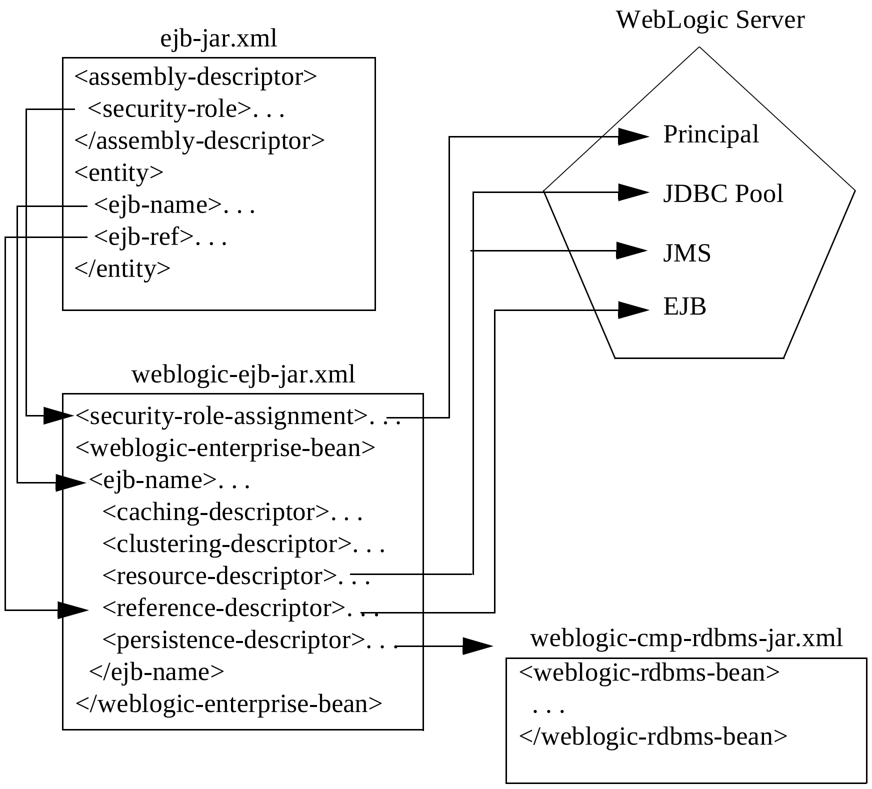
With WebLogic Server, you can create, modify, and deploy EJBs in one or more WebLogic Servers. You can set up your EJB deployments, and map EJB references to actual resource factories, roles, and other EJBs available on a server by editing the XML.

WebLogic Server Deployment Files

To modify the deployment properties, you need to update the XML deployment descriptor dtds in the XML files.

The `weblogic-ejb-jar.xml` contains descriptors that define the caching, clustering, and performance behavior of EJBs. It also contains descriptors that map available WebLogic Server resources to EJBs. WebLogic Server resources include security role names, data sources such as JDBC pools and JMS connection factories, and other deployed EJBs. Descriptors for container-managed persistence services are available in `weblogic-cmp-rdbms.xml`.

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



See "Manually Editing XML Deployment Files" on page 10-1 to manually edit the XML deployment files.

Automatic Deployment Directory

The `weblogic/config/domain/applications` directory acts as an automatic deployment directory for EJB `.jar` files and EJB `.jar` deployment directories. When you start WebLogic Server, it automatically deploys any valid EJB `.jar` files or `.jar` directories that reside in the `applications` directory.

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

See the sections below for more information on deploying and redeploying compiled `.jar` files, uncompiled `.jar` files, and `.jar` directories using the `applications` directory.

Note: WebLogic Server Version 6.0 also uses a new `$wl_temp_do_not_delete` directory to copy EJB `.jar` files before deploying them into the EJB container. You should not add, remove, or modify files in this directory, nor should you delete this directory while the server is running.

The home and remote interfaces for an EJB are required in the classpath of any calling conponent. When the EJB classes are compiled, they can be placed in the a directory in the classpath, such as .../config/<domain>/clientclasses.

Deploying Compiled EJB .jar Files

You create compiled EJB 2.0 `.jar` files by:

1. Compiling your EJB classes and interfaces using `javac`.
2. Packaging the EJB classes and interfaces into a valid `.jar` file.
3. Using `weblogic.ejbc` on the `.jar` file to generate WebLogic Server container classes.

These steps are similar to the steps required for compiling and deploying EJB 1.1 beans.

You may already have compiled `.jar` files that you used for deploying EJBs to the container in WebLogic Server Version 6.0 EJB 1.1 container. To deploy these files to the EJB 2.0 container:

1. Run `weblogic.ejbc` against the `.jar` file to generate EJB 2.0 container-classes.
2. Copy the compiled `.jar` files into `weblogic/config/domain/applications.`

Note: The container does not support deploying EJB 1.1-compliant CMP entity beans into the EJB 2.0 container. If you have EJB 1.1-compliant CMP beans, you must deploy them into the EJB 1.1 container.

If you change the contents of a compiled EJB `.jar` file in `applications` (by repackaging, recompiling, or copying over the existing `.jar` file), WebLogic Server automatically attempts to redeploy the `.jar` using the dynamic deployment feature.

Note: Because the automatic redeployment feature uses dynamic deployment, WebLogic Server can only redeploy an EJB's implementation classes. You cannot redeploy an EJB's public interfaces.

Deploying Uncompiled EJB .jar Files

The WebLogic Server container also enables you to automatically deploy `.jar` files that contain uncompiled EJB classes and interfaces. An uncompiled EJB `.jar` 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.

Once you have packaged the uncompiled classes, simply copy the `.jar` into the `%BEA_HOME%\wlserver6.0\config\<your domain>\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 `weblogic/config/examples/applications`, and deployed to the EJB container.

Should you ever modify an uncompiled `.jar` in `applications` (either by repackaging or copying over the `.jar` file), WebLogic Server automatically recompiles and redeploys the `.jar` using the same steps.

Note: Because the automatic redeployment feature uses dynamic deployment, WebLogic Server can only redeploy an EJB's implementation classes. You cannot redeploy an EJB's public interfaces.

Deploying from an EJB .jar Directory

The EJB 2.0 container also enables you to deploy EJBs using an EJB `.jar` subdirectory in the `applications` directory. Using this method is similar to using an uncompiled `.jar` file, in that you do not need to compile EJB classes and interfaces, and you do not need to run `weblogic.ejbc`. Using a `.jar` subdirectory also removes the requirement for packaging the EJB deployment into a `.jar` file. You simply create a subdirectory in `applications` that has the same structure of an EJB `.jar` file.

For example, the following files define a valid EJB `.jar` deployment directory for use with the WebLogic Server EJB 2.0 container:

- `/weblogic/config/examples/applications/statefulSession/examples/ejb/basic/statefulSession/ProcessingErrorException.java`
- `/weblogic/config/examples/applications/statefulSession/examples/ejb/basic/statefulSession/TradeResult.java`
- `/weblogic/config/examples/applications/statefulSession/examples/ejb/basic/statefulSession/TraderBean.java`
- `/weblogic/config/examples/applications/statefulSession/examples/ejb/basic/statefulSession/TraderHome.java`
- `/weblogic/config/examples/applications/statefulSession/examples/ejb/basic/statefulSession/Trader.java`
- `/weblogic/config/examples/applications/statefulSession/META-INF/ejb-jar.xml`

- `/weblogic/config/examples/applications/statefulSession/META-INF/weblogic-ejb-jar.xml`

When you use an EJB `.jar` directory in this manner, WebLogic Server automatically redeploys the EJB when the `ejb-jar.xml` deployment file has changed. If you are developing a new EJB using a `.jar` directory, you can trigger redeployment by changing the timestamp of `ejb-jar.xml` (on UNIX systems, use the `touch` command; on Windows NT, open and save the file in a text editor).

CHAPTER

8 WebLogic Server EJB Utilities

The following sections provide a complete reference to the utilities and support files supplied with WebLogic Server EJBs:

- ejbc (weblogic.ejbc)
- DDConverter (weblogic.ejb.utils.DDConverter)
- deploy (weblogic.deploy)

ejbc

WebLogic Server includes the `weblogic.ejbc` utility for 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 a client-side stub and a server-side skeleton.

If you specify an output `.jar` file, `ejbc` places all generated files into the `.jar`.

By default, `ejbc` uses `javac` as a compiler. For faster performance, specify a different compiler (such as Symantec's `sj`) using the `-compiler` flag.

Syntax

```
$ java weblogic.ejbc [options] <source jar file>
      <target directory or jar file>
```

Note: If you output to a `.jar` file, the output *`.jar`* name must be different from the input `.jar` name.

Arguments

Argument	Description
<*source jar file*>	Specifies the `.jar` file containing the compiled EJB classes, interfaces, and XML deployment files.
<*target directory or jar file*>	Specifies the destination `.jar` file or deployment directory in which `ejbc` places the output `.jar`. If you specify an output `.jar` file, `ejbc` places the original EJB classes, interfaces, and XML deployment files in the `.jar`, as well as the new container classes that `ejbc` generates.

Options

Option	Description
`-help`	Prints a list of all options available for the compiler.

`-version`	Prints `ejbc` version information.
`-idl`	Generates CORBA Interface Definition Language for remote interfaces.
`-idlOverwrite`	Overwrites existing IDL files.
`-idlVerbose`	Displays verbose information while generating IDL.
`-idlDirectory <dir>`	Specifies the directory where `ejbc` creates IDL files. By default, `ejbc` uses the current directory.
`-keepgenerated`	Saves the intermediate Java files generated during compilation.
`-compiler <compiler name>`	Sets the compiler for `ejbc` to use.
`-normi`	Passed through to Symantec's java compiler, `sj`, to stop generation of RMI stubs. Otherwise `sj` creates its own RMI stubs, which are unnecessary for the EJB.
`-classpath <path>`	Sets a CLASSPATH used during compilation. This overrides the system or shell CLASSPATH.

Examples

The following example uses the `javac` compiler against an input `.jar` file in `c:\weblogic\samples\examples\ejb\basic\containerManaged\build`. The output `.jar` is placed in `c:\weblogic\config\examples\applications`.

```
$ java weblogic.ejbc -compiler javac
c:\weblogic\samples\examples\ejb\basic\containerManaged\build\std
_ejb_basic_containerManaged.jar
c:\weblogic\config\examples\ejb_basic_containerManaged.jar
```

The following example checks a `.jar` file for compliance with the EJB 1.1 specification and generates WebLogic Server container classes, but does not generate RMI stubs:

```
$ java weblogic.ejbc -normi
c:\weblogic\samples\examples\ejb\basic\containerManaged\build\std
_ejb_basic_containerManaged.jar
```

DDConverter

The `DDConverter` is a command line utility that converts WebLogic Server EJB deployment descriptors to WebLogic Server 6.0. Also, it allows you to convert deployment descriptors for WebLogic Server 4.5 container-managed persistence (CMP) 1.0 beans to CMP 1.1 beans, which run under WebLogic Server 6.0. With the `DDConverter`, the output of Weblogic Server 4.5 (EJB 1.0) is a `.txt` file; the output of WebLogic Server 5.1 (EJB 1.1) is a `.jar` file.

Converting EJBs for Use in WebLogic Server 6.0

The conversion options are:

- To convert (WebLogic Server 5.1) 1.1 EJBs to (WebLogic Server 6.0) 2.0 EJBs, input the WebLogic 5.1 `.jar` file, which includes the deployment descriptor XML files and any classes into the `DDConverter`. The output goes to a `.jar` file that includes the WebLogic 6.0 deployment descriptor XML files. Any necessary classes also are copied to the `.jar` file. Now, the `.jar` file can now be deployed to WebLogic Server 6.0.
- To convert (WebLogic 4.5) 1.0 EJBs to (WebLogic 6.0) 2.0 EJBs, input the WebLogic 4.5 deployment descriptor text into the `DDConverter`. The output goes to a `.jar` file that only includes the WebLogic 6.0 deployment descriptor. If any classes need to be copied to the `.jar` file, you will need to manually add them before you can deploy the file to WebLogic Server 6.0.

Converting EJB CMP 1.1 Beans to EJB CMP 2.0 Beans

To convert EJB CMP 1.1 beans to 2.0 beans:

1. Input the deployment descriptors into the `DDConverter`.

   The output goes to the `.jar` file.

2. Extract the XML deployment descriptors.

3. Modify the source code.
4. Compile the modified java file with the extracted XML deployment descriptors, which creates the `.jar` file.
5. Deploy the bean.

   Except for CMP beans, all other beans are converted to EJB 2.0 beans as needed, with little or no source code changes. You can input the deployment descriptors, output to a `.jar` file, and then deploy the bean.

Converting CMP Beans between WebLogic Server Versions

The `DDConverter` tool supports the following conversions:

- WebLogic 4.5 CMP 1.0 DTD conversion to WebLogic 6.0 CMP 1.1 DTD. The tool cannot produce a 6.0 CMP 2.0 DTD. (In other words, `-EJBVer` 2.0 has no effect.)
- WebLogic 5.1 CMP 1.1 DTD conversion to WebLogic 6.0 CMP 2.0 DTD. The tool cannot produce a 6.0 CMP 1.1 DTD. (In other words, `-EJBVer` 1.1 has no effect.)

To make these conversions:

1. Input the WebLogic Server 4.5 or 5.1 deployment descriptor text into the `DDConverter`.

   The output goes to a `.jar` file that only includes CMP 1.1 or 2.0 WebLogic Server 6.0 deployment descriptors.
2. Manually add the class files to the `.jar` file.

Syntax

```
$ java weblogic.ejb20.utils.DDConverter [options] file1 [file2...]
```

Arguments

`DDConverter` takes one argument of `*file1 [file2...],*` which specifies a text file containing an EJB 1.0 or 1.5 compliant deployment descriptor.

`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.

Options

Option	**Description**
`-d destDir`	Specifies the destination directory for the output of the *.jar* files. This is a required option.
`-combine 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.
`-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.

Examples

The following example converts an EJB 1.0-compliant WebLogic Server `DeploymentDescriptor.txt` file into a collection of EJB 1.1-compliant XML files. The XML files are created in the *`destDir`* subdirectory:

```
$ java weblogic.ejb20.utils.DDConverter -d destDir
DeploymentDescriptor.txt
```

deploy

Given an EJB compliant `.jar` file, the `.jar`'s EJBs are deployed into a running WebLogic Server.

Syntax

```
$ java weblogic.deploy [options] [list|deploy|undeploy|update]
password {name} {source}
```

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.
`update`	Redeploys an EJB deployment unit in 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.

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`. 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`. 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 Server 6.0 EJB Deployment Properties

The following sections provide a complete reference of the WebLogic specific XML deployment properties used in the WebLogic Server 6.0 EJB 2.0 container and an explanation of how to edit the XML deployment properties manually.

- Manually Editing XML Deployment Files
- weblogic-ejb-jar.xml Deployment Descriptor File
- Index of weblogic-ejb-jar Deployment Elements
- weblogic-cmp-rdbms-jar.xml Deployment Descriptor File
- Index of weblogic-cmp-rdbms-jar.xml Deployment Elements

Manually Editing XML Deployment Files

To create and deploy message-driven beans, or use the new caching and clustering deployment elements, you must edit the following XML deployment files manually, using a text editor:

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

See also “Basic Conventions” on page 10-2 for more information on the conventions to use when modifying XML deployment files.

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.

XML File	PUBLIC Element String
`ejb-jar.xml`	`‘-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN’ ‘http://www.java.sun.com/dtd/ejb-jar_2_0.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-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‘`

For example, the entire `DOCTYPE` header for a `weblogic-ejb-jar.xml` file is as follows:

```
<!DOCTYPE weblogic-ejb-jar PUBLIC
'-//BEA Systems, Inc.//DTD WebLogic 6.0.0 EJB//EN'
'http://www.bea.com/servers/wls600/dtd/weblogic-ejb-jar.dtd'>
```

XML files with incorrect header information may yield error messages similar to the following, when used with a utility that parses the XML (such as `ejbc`):

```
SAXException: This document may not have the identifier ‘identifier_name’
```

`identifier_name` generally includes the invalid text from the `PUBLIC` element.

Document Type Definitions (DTDs) for Validation

The contents and arrangement of elements in your XML files must conform to the Document Type Definition (DTD) for each file you use. WebLogic Server ignores the DTDs embedded within the `DOCTYPE` header of XML deployment files, and instead uses the DTD locations that were installed along with the server. However, the `DOCTYPE` header information must include a valid URL syntax in order to avoid parser errors.

The following links provide the new public DTD locations for XML deployment files used with the WebLogic Server EJB 2.0 container:

- `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`.
- `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.
- `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
  ```

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 Deployment Descriptor File

`weblogic-ejb-jar.xml` defines EJB deployment descriptor DTDs which are unique to WebLogic Server. The EJB 2.0 container uses a version of `weblogic-ejb-jar.xml` that is different from the one shipped with WebLogic Server Version 5.1. The revised DTD for `weblogic-ejb-jar.xml` includes new elements for enabling stateful session EJB replication, configuring entity EJB locking behavior, and assigning JMS Queue and Topic names for message-driven beans. The new DTD also reorganizes the major stanzas into more logical sections.

You can continue to use the earlier `weblogic-ejb-jar.xml` DTD for EJB 1.1 that you will deploy into the EJB 1.1 container. However, if you want to use any of the new EJB 2.0 features or deploy beans into the EJB 2.0 container, you must use the DTD described in the sections listed in "Index of weblogic-ejb-jar Deployment Elements" on page 9-5.

The top level elements in the EJB `weblogic-ejb-jar.xml` are as follows:

- `description of the file`
- `Copyright information`
- `weblogic-enterprise-bean`
  - `ejb-name`
  - entity-descriptor | stateless-session-descriptor | stateful-session-descriptor | message-driven-descriptor
  - transaction-descriptor
  - reference-descriptor
  - enable-call-by-reference
  - `jndi-name`
- security-role-assignment
- transaction-isolation

Use the links above, or see "Manually Editing XML Deployment Files" on page 9-1 for more information about individual EJB 2.0 deployment stanzas and elements. See also "Manually Editing XML Deployment Files" on page 10-1 for complete information about the deployment elements introduced for EJB 1.1.

Index of weblogic-ejb-jar Deployment Elements

- "allow-concurrent-calls" on page 9-8
- "concurrency-strategy" on page 9-9
- "db-is-shared" on page 9-10
- "delay-updates-until-end-of-tx" on page 9-11
- "description" on page 9-12
- "destination-jndi-name" on page 9-13
- "ejb-name" on page 9-14
- "ejb-reference-description" on page 9-15
- "ejb-ref-name" on page 9-16
- "enable-call-by-reference" on page 9-17
- "entity-cache" on page 9-18
- "entity-clustering" on page 9-19
- "entity-descriptor" on page 9-20
- "finders-load-bean" on page 9-21
- "home-call-router-class-name" on page 9-22
- "home-is-clusterable" on page 9-23
- "home-load-algorithm" on page 9-24

- "idle-timeout-seconds" on page 9-25
- "initial-beans-in-free-pool" on page 9-26
- "is-modified-method-name" on page 9-27
- "isolation-level" on page 9-28
- "jndi-name" on page 9-29
- "lifecycle" on page 9-30
- "max-beans-in-cache" on page 9-31
- "max-beans-in-free-pool" on page 9-32
- "message-driven-descriptor" on page 9-33
- "method" on page 9-34
- "method-intf" on page 9-35
- "method-name" on page 9-36
- "method-param" on page 9-37
- "method-params" on page 9-38
- "passivation-strategy" on page 9-39
- "persistence" on page 9-40
- "persistence-type" on page 9-41
- "persistence-use" on page 9-42
- "persistent-store-dir" on page 9-43
- "pool" on page 9-44
- "principal-name" on page 9-45
- "read-timeout-seconds" on page 9-46
- "reference-descriptor" on page 9-47
- "replication-type" on page 9-48
- "res-env-ref-name" on page 9-49

- "res-ref-name" on page 9-50\
- "resource-env-description" on page 9-51
- "resource-description" on page 9-52
- "role-name" on page 9-53
- "run-as-identity-principal" on page 9-54
- "security-role-assignment" on page 9-55
- "stateful-session-cache" on page 9-56
- "stateful-session-clustering" on page 9-57
- "stateful-session-descriptor" on page 9-58
- "stateless-bean-call-router-class-name" on page 9-59
- "stateless-bean-is-clusterable" on page 9-60
- "stateless-bean-load-algorithm" on page 9-61
- "stateless-bean-methods-are-idempotent" on page 9-62
- "stateless-clustering" on page 9-63
- "stateless-session-descriptor" on page 9-64
- "transaction-descriptor" on page 9-65
- "transaction-isolation" on page 9-66
- "trans-timeout-seconds" on page 9-67
- "type-identifier" on page 9-68
- "type-storage" on page 9-69
- "type-version" on page 9-70

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`

Deployment file:	`weblogic-ejb-jar.xml`

Function

The `allow-concurrent-calls` element specifies whether a stateful session bean instance will allow 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 9-58.

concurrency-strategy

Range of values:	`Exclusive | Database | ReadOnly`
Default value:	`Database`
Requirements:	Optional element. Valid only for entity EJBs.
Parent elements:	`weblogic-enterprise-bean,` `entity-descriptor,` `entity-cache`
Deployment file:	`weblogic-ejb-jar.xml`

Function

This element determines `ejbLoad() and ejbStore()` behavior for entity EJB instances. You can set this element to one of three possible values:

- `Exclusive` was the default locking behavior for WebLogic Server versions 3.1 through 5.1. WebLogic Server places an exclusive lock on cached entity EJB instances when the bean is associated with a transaction. Other requests for the EJB instance block until the transaction completes.
- `Database` is the default locking behavior for Weblogic Server versions 6.x. This value causes WebLogic Server to defer locking requests for an entity EJB to the underlying datastore. With the `Database` concurrency strategy, WebLogic Server does not cache the intermediate results of entity EJBs involved in a transaction.
- `ReadOnly` designates an entity EJB that is never modified. WebLogic Server calls `ejbLoad()` for `ReadOnly` beans based on the read-timeout-seconds parameter.

See“Locking and Caching Services for Entity EJBs” on page 4-38 for more information on the `Exclusive` and `Database` locking behaviors. See“Read-Write Cache Strategy” on page 4-12 for more information about `read-only` entity EJBs.

Example

The following entry identifies the `AccountBean` class as a read-only entity EJB:

```
<weblogic-enterprise-bean>
       <ejb-name>AccountBean</ejb-name>
       <entity-descriptor>
              <entity-cache>

<concurrency-strategy>ReadOnly</concurrency-strategy>
              </entity-cache>
       </entity-descriptor>
</weblogic-enterprise-bean>
```

db-is-shared

Range of values:	`true	false`
Default value:	`true`
Requirements:	Optional element. Valid only for entity EJBs.
Parent elements:	`weblogic-enterprise-bean, entity-descriptor, persistence`
Deployment file:	`weblogic-ejb-jar.xml`

Function

The `db-is-shared` element applies only to entity beans. When set to `true` WebLogic Server assumes that EJB data could be modified between transactions and reloads data at the beginning of each transaction. When set to `false` WebLogic Server assumes that it has exclusive access to the EJB data in the persistent store. See "Using db-is-shared to Limit Calls to ejbLoad()" on page 4-9 for more information.

Example

See "persistence" on page 9-40.

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`
Deployment file:	`weblogic-ejb-jar.xml`

Function

Set this element 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 4-8 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 entry specifies that WebLogic Server call `ejbStore()` at the end of each method invocation:

```
<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-enterprise-bean,` `transaction-isolation` `method`
Deployment file:	`weblogic-ejb-jar.xml`

Function

The `description` element is used to provide text that describes the parent element.

Example

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`

Deployment file:	`weblogic-ejb-jar.xml`

Function

`destination-jndi-name` specifies the JNDI name of an actual JMS Queue or Topic available in WebLogic Server.

Example

See "message-driven-descriptor" on page 9-33.

ejb-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-enterprise-bean`
	`    transaction-isolation`
	`        method`

Deployment file:	`weblogic-ejb-jar.xml`

Function

`ejb-name` specifies the name of an EJB to which WebLogic Server applies isolation level properties.

Example

See “method” on page 9-34.

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-description`
Deployment file:	`weblogic-ejb-jar.xml`

Function

The `resource-description` stanza 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>

       <ejb-reference-description>

              <ejb-ref-name>...</ejb-ref-name>

              <jndi-name>...</jndi-name>

       </ejb-reference-description>

</reference-descriptor>
```

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`

Deployment file:	`weblogic-ejb-jar.xml`

Function

The `resource-description` stanza maps a resource reference named in `ejb-jar.xml` to the JNDI name of an actual resource available in WebLogic Server.

- `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 `resource-description` stanza can contain additional elements as shown here:

```
<reference-descriptor>

       <ejb-reference-description>

              <ejb-ref-name>. . .</ejb-ref-name>

              <jndi-name>. . .</jndi-name>

       </ejb-reference-description>

</reference-descriptor>
```

enable-call-by-reference

Range of values:	`true | false`
Default value:	`true`
Requirements:	Optional element.

Parent elements:	`weblogic-enterprise-bean`
	    `reference-description`
	        `ejb-reference-description`

Deployment file:	`weblogic-ejb-jar.xml`

Function

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.

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 server).

Example

The following example enables pass-by-value for EJB methods:

```
<weblogic-enterprise-bean>

       <ejb-name>AccountBean</ejb-name>

       ...

       <enable-call-by-reference>false</enable-call-by-reference>

</weblogic-enterprise-bean>
```

entity-cache

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	The `entity-cache` stanza is optional, and is valid only for entity EJBs.
Parent elements:	`weblogic-enterprise-bean,` `entity-descriptor`
Deployment file:	`weblogic-ejb-jar.xml`

Function

The `entity-cache` element defines options used to cache entity EJB instances within WebLogic Server. See“EJB Life Cycle in WebLogic Server” on page 4-2 for a general discussion of the caching services available in WebLogic Server.

Example

```
<entity-descriptor>

       <entity-cache>

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

              <idle-timeout-seconds>...</idle-timeout-seconds>

              <read-timeout-seconds>...<read-timeout-seconds>

              <concurrency-strategy>...</concurrency-strategy>

       </entity-cache>

       <lifecycle>...</lifecycle>

       <persistence>...</persistence>

       <entity-clustering>...</entity-clustering>

</entity-descriptor>
```

entity-clustering

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Optional element. Valid only for entity EJBs in a cluster.
Parent elements:	`weblogic-enterprise-bean,` `entity-descriptor`
Deployment file:	`weblogic-ejb-jar.xml`

Function

The `entity-clustering` stanza contains elements that determines how WebLogic Server replicates entity EJB instances in a cluster.

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`
Deployment file:	`weblogic-ejb-jar.xml`

Function

The `entity-descriptor` stanza defines caching, clustering, and persistence properties for entity EJBs in WebLogic Server.

Example

The following example shows the structure of the `entity-descriptor` stanza:

```
<entity-descriptor>
       <entity-cache>...</entity-cache>
       <lifecycle>...</lifecycle>
       <persistence>...</persistence>
       <entity-clustering>...</entity-clustering>
</entity-descriptor>
```

finders-load-bean

Range of values:	true | false
Default value:	`true`
Requirements:	Optional element. Valid only for CMP entity EJBs.
Parent elements:	`weblogic-enterprise-bean,` `entity-descriptor,` `persistence`
Deployment file:	`weblogic-ejb-jar.xml`

Function

`finders-load-bean` 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 immediately loads the bean into the cache if a reference to a bean is returned by the finder. If you set this element to `false`, WebLogic Server does not load automatically load the bean into the cache until the first method invocation; this behavior is consistent with the EJB 1.1 specification.

Example

The following entry specifies that EJBs are loaded into the WebLogic Server cache automatically when a finder method returns a reference to the bean:

```
<entity-descriptor>

       <persistence>

              <finders-load-bean>true</finders-load-bean>

       </persistence>

</entity-descriptor>
```

home-call-router-class-name

Range of values:	Valid router class name
Default value:	n/a
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`

Deployment file:	`weblogic-ejb-jar.xml`

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.extensions.CallRouter()`. If specified, an instance of this class is called before each method call. The router class has the opportunity to choose a server to route to based on the method parameters. The class returns either a server name or null, which indicates that the current load algorithm should select the server.

Example

See "entity-clustering" on page 9-19 and "stateful-session-clustering" on page 9-57.

home-is-clusterable

Range of values:	`true | false`
Default value:	`true`
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`

Deployment file:	`weblogic-ejb-jar.xml`

Function

When `home-is-clusterable` is `true`, the EJB can be deployed from multiple WebLogic Servers in a cluster. Calls to the home stub are load-balanced between the servers on which this bean is deployed, and if a server hosting the bean is unreachable, the call automatically fails over to another server hosting the bean.

Example

See "entity-clustering" on page 9-19.

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`

Deployment file:	`weblogic-ejb-jar.xml`

Function

`home-load-algorithm` specifies the algorithm to use for load balancing between replicas of the EJB home. If this property is not defined, WebLogic Server uses the algorithm specified by the server property, `weblogic.cluster.defaultLoadAlgorithm`.

You can define `home-load-algorithm` as one of the following values:

- `round-robin`: Load balancing is performed in a sequential fashion among the servers hosting the bean.
- `random`: Replicas of the EJB home are deployed randomly among the servers hosting the bean.
- `weight-based`: Replicas of the EJB home are deployed on host servers according to the servers' current workload.

Example

See "entity-clustering" on page 9-19 and "stateful-session-clustering" on page 9-57.

idle-timeout-seconds

Range of values:	`1` to *`maxSeconds`*
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`

Deployment file:	`weblogic-ejb-jar.xml`

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 may remove the bean instance if the number of beans in cache approaches the limit of `max-beans-in-cache`. See "EJB Life Cycle in WebLogic Server" on page 4-2 for more information.

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>

              <entity-cache>

                     <max-beans-in-cache>200</max-beans-in-cache>
```

```
<idle-timeout-seconds>1200</idle-timeout-seconds>
              </entity-cache>
       </stateful-session-descriptor>
</weblogic-enterprise-bean>
```

initial-beans-in-free-pool

Range of values:	`0` to *`maxBeans`*
Default value:	`0`
Requirements:	Optional element. Valid only for stateless session EJBs.
Parent elements:	`weblogic-enterprise-bean,` `transaction-descriptor`
Deployment file:	`weblogic-ejb-jar.xml`

Function

If you specify a value for `initial-beans-in-free-pool`, you set the intital size ofthe 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 9-44.

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`
Deployment file:	`weblogic-ejb-jar.xml`

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()" on page 4-10 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>isModified</is-modified-method-name>

       </persistence>

</entity-descriptor>
```

isolation-level

Range of values:	`Serializable | ReadCommitted | ReadUncommitted | RepeatableRead`
Default value:	n/a
Requirements:	Optional element.
Parent elements:	`weblogic-enterprise-bean, transaction-isolation`
Deployment file:	`weblogic-ejb-jar.xml`

Function

`isolation-level` specifies the isolation level for all of the EJB's database operations. The following are possible values for `isolation-level`:

- `ReadUncommitted`: The transaction can view uncommitted updates from other transactions.
- `ReadCommitted`: The transaction can view only committed updates from other transactions.
- `RepeatableRead`: 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.
- `Serializable`: Simultaneously executing this transaction multiple times has the same effect as executing the transaction multiple times in a serial fashion.

Refer to your database documentation for more information on the implications and support for different isolation levels.

Example

See “transaction-isolation” on page 9-66.

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-description`
	`          resource-description`
	`and`
	`weblogic-enterprise-bean`
	`     reference-description`
	`          ejb-reference-description`

Deployment file:	`weblogic-ejb-jar.xml`

Function

`jndi-name` specifies the JNDI name of an actual EJB or resource available in WebLogic Server.

Example

See “resource-description” on page 9-52 and “ejb-reference-description” on page 9-15.

lifecycle

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	The `lifecycle` stanza is optional. `lifecycle` is valid only for entity EJBs.
Parent elements:	`weblogic-enterprise-bean,` `entity-descriptor` and `weblogic-enterprise-bean` `stateful-session-descriptor`
Deployment file:	`weblogic-ejb-jar.xml`

Function

The `lifecycle` stanza defines properties that affect the lifecycle of entity EJB instances within WebLogic Server. Currently, the `lifecycle` stanza includes only one element: `passivation-strategy`.

Example

```
<entity-descriptor>

       <lifecycle>

              <passivation-strategy>...</passivation-strategy>

       </lifecycle>

</entity-descriptor>
```

max-beans-in-cache

Range of values:	`1` to *`maxBeans`*
Default value:	`100`
Requirements:	Optional element

Parent elements:	`weblogic-enterprise-bean,`
	`entity-descriptor,`
	`entity-cache`
	`and`
	`weblogic-enterprise-bean`
	`stateful-session-descriptor`
	`stateful-session-cache`

Deployment file:	`weblogic-ejb-jar.xml`

Function

The `max-beans-in-cache` element specifies the maximum number of objects of this class that are allowed in memory. When `max-bean-in-cache` is reached, WebLogic Server passivates some EJBs that have not been recently used by a client. `max-beans-in-cache` also affects when EJBs are removed from the WebLogic Server cache, as described in“Locking and Caching Services for Entity EJBs” on page 4-38.

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:	*`max Int`*
Requirements:	Optional element. Valid only for stateless session EJBs.
Parent elements:	`weblogic-enterprise-bean, transaction-descriptor`
Deployment file:	`weblogic-ejb-jar.xml`

Function

WebLogic Server maintains a free pool of EJBs for every stateless session bean and message driven bean class. The `max-beans-in-free-pool` element defines the size of this 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 "Stateless Session EJB Life Cycle" on page 4-2 or "Differences Between Message-Driven Beans and Stateless Session EJBs" on page 3-3 for more information.

Example

See "pool" on page 9-44.

message-driven-descriptor

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	One `message-driven-descriptor` stanza is required for each message-driven bean in the *.jar*.
Parent elements:	`weblogic-enterprise-bean`
Deployment file:	`weblogic-ejb-jar.xml`

Function

The `message-driven-descriptor` stanza associates a message-driven bean with a JMS destination in WebLogic Server. This stanza currently includes only one XML element, `destination-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-enterprise-bean` `transaction-isolation`
Deployment file:	`weblogic-ejb-jar.xml`

Function

The `method` stanza defines method-level transaction isolation settings for an EJB.

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`
Default value:	n/a
Requirements:	Optional element.

Parent elements:	`weblogic-enterprise-bean`
	`    transaction-isolation`
	`        method`

Deployment file:	`weblogic-ejb-jar.xml`

Function

`method-intf` specifies the EJB interface to which WebLogic Server applies isolation level properties. Use this element only if you need to differentiate between methods having the same signature in the EJB's home and remote interface.

Example

See "method" on page 9-34.

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-enterprise-bean`
	    `transaction-isolation`
	        `method`

Deployment file:	`weblogic-ejb-jar.xml`

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 9-34.

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-enterprise-bean`
	    `transaction-isolation`
	        `method`
	            `method-params`

Deployment file:	`weblogic-ejb-jar.xml`

Function

The `method-param` stanza specifies the fully-qualified Java type of a method parameter.

Example

See "method-params" on page 9-38.

method-params

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Optional stanza.

Parent elements:	`weblogic-enterprise-bean`
	`    transaction-isolation`
	`        method`

Deployment file:	`weblogic-ejb-jar.xml`

Function

The `method-params` stanza contains one or more elements that define the Java type name of each of the method's parameters.

Example

The `method-params` stanza contains one or more `method-param` elements, as shown here:

```
<method-params>

       <method-param>java.lang.String</method-param>

       ...

</method-params>
```

passivation-strategy

Range of values:	`default | transaction`
Default value:	`default`
Requirements:	Optional element. Valid only for entity EJBs.
Parent elements:	`weblogic-enterprise-bean,` `entity-descriptor,` `lifecycle`
Deployment file:	`weblogic-ejb-jar.xml`

Function

The `passivation-strategy` element determines whether or not WebLogic Server maintains the intermediate state of entity EJBs in its cache. See "Locking and Caching Services for Entity EJBs" on page 4-38 for more information.

Example

The following entry reverts to WebLogic Server locking and caching behavior:

```
<entity-descriptor>

       <lifecycle>

              <passivation-strategy>default</passivation-strategy>

       </lifecycle>

</entity-descriptor>
```

persistence

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Required only for entity EJBs that use container-managed persistence services.
Parent elements:	`weblogic-enterprise-bean,` `entity-descriptor`
Deployment file:	`weblogic-ejb-jar.xml`

Function

The `persistence` stanza defines properties that determine the persistence type, transaction commit behavior, and `ejbLoad()` and `ejbStore()` behavior for entity EJBs in WebLogic Server.

Example

```
<entity-descriptor>

       <persistence>


<is-modified-method-name>...</is-modified-method-name>


<delay-updates-until-end-of-tx>...</delay-updates-until-end-of-tx
>

              <finders-load-beand>...</finders-load-bean>

              <persistence-type>...</persistence-type>

              <db-is-shared>false</db-is-shared>

              <persistence-use>...</persistence-use>

       </persistence>

</entity-descriptor>
```

persistence-type

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Required only for entity EJBs that use container-managed persistence services.
Parent elements:	`weblogic-enterprise-bean,` `entity-descriptor,` `persistence`
Deployment file:	`weblogic-ejb-jar.xml`

Function

The `persistence-type` stanza defines a persistence service that the entity EJB can use. You can define multiple `persistence-type` stanzas in `weblogic-ejb-jar.xml` for testing your EJB with multiple persistence services. Only the persistence type defined in `persistence-use` is actually used during deployment.

`persistence-type` includes several elements that identify the persistence types:

- Name
- Version
- Path of the file that stores data fields and configuration information

Example

The following excerpt shows a sample `persistence-type` stanza:

```
<persistence>

       <persistence-type>


<type-identifier>WebLogic_CMP_RDBMS</type-identifier>

              <type-version>5.1.0</type-version>
```

```
<type-storage>META-INF\weblogic-cmp-rdbms-jar.xml</type-storage>

       </persistence-type>

</persistence>
```

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`

Deployment file:	`weblogic-ejb-jar.xml`

Function

The `persistence-use` stanza is similar to persistence-type, but it defines the persistence service actually used during deployment. `persistence-use` uses the type-identifier and type-version elements defined in a `persistence-type` to identify the service.

Example

To deploy an EJB using the WebLogic Server RDBMS-based persistence service defined in persistence-type, use the following `persistence-use` stanza:

```
<persistence-use>

       <type-identifier>WebLogic_CMP_RDBMS</type-identifier>

       <type-version>5.1.0</type-version>

</persistence-use>
```

persistent-store-dir

Range of values:	Fully qualified filesystem path
Default value:	n/a
Requirements:	Optional element.
Parent elements:	`weblogic-enterprise-bean` `stateful-session-descriptor`
Deployment file:	`weblogic-ejb-jar.xml`

Function

The `persistent-store-dir` element specifies a file system directory where WebLogic Server stores the state of passivated stateful session bean instances.

Example

See “stateful-session-descriptor” on page 9-58.

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`
Deployment file:	`weblogic-ejb-jar.xml`

Function

The `pool` stanza configures the behavior of the WebLogic Server free pool for a stateless session and message driven 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`
Deployment file:	`weblogic-ejb-jar.xml`

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 9-55.

read-timeout-seconds

Range of values:	`0` to *`maxSeconds`*
Default value:	`0`
Requirements:	Optional element. Valid only for entity EJBs.

Parent elements:	`weblogic-enterprise-bean,`
	`entity-descriptor,`
	`entity-cache`

Deployment file:	`weblogic-ejb-jar.xml`

Function

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 0, and WebLogic Server 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`
Deployment file:	`weblogic-ejb-jar.xml`

Function

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.

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>
```

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`

Deployment file:	`weblogic-ejb-jar.xml`

Function

The `replication-type` element determines whether or not WebLogic Server replicates the state of stateful session EJBs across WebLogic Server instances in a cluster. If you select `InMemory`, the state of the EJB is replicated. If you select `None`, the state is not replicated.

Example

See "stateful-session-clustering" on page 9-57.

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`

Deployment file:	`weblogic-ejb-jar.xml`

Function

`res-env-ref-name` specifies a resource environment reference name.

Example

See "resource-env-description" on page 9-51.

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-description`
	`        resource-description`

Deployment file:	`weblogic-ejb-jar.xml`

Function

`res-ref-name` specifies a resource reference name. This is the reference that the EJB provider places within the `ejb-jar.xml` deployment file.

Example

See "resource-description" on page 9-52.

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`
Deployment file:	`weblogic-ejb-jar.xml`

Function

The `resource-env-description` stanza 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>
```

resource-description

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Optional element.
Parent elements:	`weblogic-enterprise-bean` `reference-description`
Deployment file:	`weblogic-ejb-jar.xml`

Function

The `resource-description` stanza 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>
```

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`

Deployment file:	`weblogic-ejb-jar.xml`

Function

`role-name` 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 9-55.

run-as-identity-principal

Range of values:	Principal that will be used as the identity as defined in `ejb-jar.xml`
Default value:	n/a
Requirements:	Required element in security-role-assignment.
Parent elements:	weblogic-enterprise-bean
Deployment file:	`weblogic-ejb-jar.xml`

Function

`run-as-identity-principal` specifies the principal to be used as the identity for beans that have a `security-identity.run-as-specified-identity` set in the `ejb-jar.xml`.

The principal named in this element must be one of the principals mapped to the `run-as-specified-identity` role.

Example

The `run-as-identity-principal` stanza can contain additional elements as shown here:

```
<weblogic-ejb-jar>

    <weblogic-enterprise-bean>

        <run-as-identity-principal>Fred</run-as-identity-principal>

    </weblogic-enterprise-bean>

</weblogic-ejb-jar>
```

security-role-assignment

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Required element if `ejb-jar.xml` defines application roles.
Parent elements:	`weblogic-ejb-jar`
Deployment file:	`weblogic-ejb-jar.xml`

Function

The `security-role-assignment` stanza maps application roles in the `ejb-jar.xml` file to the names of security principals available in WebLogic Server.

Example

The `security-role` stanza can contain one or more of the following elements:

```
<security-role-assignment>

       <role-name>PayrollAdmin</role-name>

       <principal-name>Tanya</principal-name>

       <principal-name>system</principal-name>

       ...

</security-role-assignment>
```

stateful-session-cache

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	The `stateful-session-cache` stanza is optional, and is valid only for stateful session EJBs.
Parent elements:	`weblogic-enterprise-bean,` `stateful-session-descriptor`
Deployment file:	`weblogic-ejb-jar.xml`

Function

The `stateful-session-cache` stanza defines properties used to cache stateful session EJB instances within WebLogic Server. See“EJB Life Cycle in WebLogic Server” on page 4-2 for a general discussion of the caching services available in WebLogic Server.

Example

```
<stateful-session-cache>

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

       <idle-timeout-seconds>...</idle-timeout-seconds>

       <read-timeout-seconds>...<read-timeout-seconds>

</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`
Deployment file:	`weblogic-ejb-jar.xml`

Function

The `stateful-session-clustering` stanza contains elements that determine how WebLogic Server replicates 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`
Deployment file:	`weblogic-ejb-jar.xml`

Function

The `stateful-session-descriptor` stanza defines caching, clustering, and persistence properties for stateful session EJBs in WebLogic Server.

Example

The following example shows the structure of the `stateful-session-descriptor` stanza:

```
<stateful-session-descriptor>

       <stateful-session-cache>...</stateful-session-cache>

       <lifecycle>...</lifecycle>

       <persistence>...</persistence>

       <allow-concurrent-calls>...</allow-concurrent-calls>

<persistent-store-dir>/weblogic/myserver</persistent-store-dir>

<stateful-session-clustering>...</stateful-session-clustering>

</stateful-session-descriptor>
```

stateless-bean-call-router-class-name

Range of values:	Valid router class name
Default value:	n/a
Requirements:	Optional element. Valid only for stateless session EJBs in a cluster.
Parent elements:	`weblogic-enterprise-bean,` `stateless-session-descriptor` `stateless-clustering`
Deployment file:	`weblogic-ejb-jar.xml`

Function

`stateless-bean-call-router-class-name` specifies the name of a custom class to use for routing bean method calls. This class must implement `weblogic.rmi.extensions.CallRouter()`. If specified, an instance of this class is called before each method call. The router class has the opportunity to choose a server to route to based on the method parameters. The class returns either a server name or null, which indicates that the current load algorithm should select the server.

Example

See "stateless-clustering" on page 9-63.

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`

Deployment file:	`weblogic-ejb-jar.xml`

Function

When `stateless-bean-is-clusterable` is `true`, the EJB can be deployed from multiple WebLogic Servers in a cluster. Calls to the home stub are load-balanced between the servers on which this bean is deployed, and if a server hosting the bean is unreachable, the call automatically fails over to another server hosting the bean.

Example

See "stateless-clustering" on page 9-63.

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`

Deployment file:	`weblogic-ejb-jar.xml`

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 9-63.

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`
Deployment file:	`weblogic-ejb-jar.xml`

Function

You can set this element to either `true` or `false`. Set `stateless-bean-methods-are-idempotent` to "true" only if the bean is written such that repeated calls to the same method with the same arguments has exactly the same effect as a single call. This allows the failover handler to retry a failed call without knowing whether the call actually completed on the failed server. Setting this property to `true` makes it possible for the bean stub to recover automatically from any failure as long as another server hosting the bean can be reached.

Example

See "stateless-clustering" on page 9-63.

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`
Deployment file:	`weblogic-ejb-jar.xml`

Function

The `stateless-clustering` stanza contains elements that determine how WebLogic Server replicates stateless session EJB instances in a cluster.

Example

The following excerpt shows the structure of a `stateless-clustering` stanza:

```
<stateless-clustering>


<stateless-bean-is-clusterable>true</stateless-bean-is-clusterabl
e>


<stateless-bean-load-algorithm>random</stateless-bean-load-algori
thm>


<stateless-bean-call-router-class-name>beanRouter</stateless-bean
-call-router-class-name>


<stateless-bean-methods-are-idempotent>true</stateless-bean-metho
ds-are-idempotent>

</stateless-clustering>
```

stateless-session-descriptor

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	One `stateless-session-descriptor` stanza is required for each stateless session EJB in the `.jar`.
Parent elements:	`weblogic-enterprise-bean`
Deployment file:	`weblogic-ejb-jar.xml`

Function

The `stateless-session-descriptor` stanza defines caching, clustering, and persistence properties 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`
Deployment file:	`weblogic-ejb-jar.xml`

Function

The `transaction-descriptor` stanza contains elements that define transaction behavior in WebLogic Server. Currently, this stanza includes only one element: `trans-timeout-seconds`.

Example

The following example shows the structure of the `transaction-descriptor` stanza:

```
<transaction-descriptor>

       <trans-timeout-seconds>20</trans-timeout-seconds>

<transaction-descriptor>
```

transaction-isolation

Range of values:	n/a (XML stanza)
Default value:	n/a (XML stanza)
Requirements:	Optional element.
Parent elements:	`transaction-description`
Deployment file:	`weblogic-ejb-jar.xml`

Function

The `transaction-isolation` defines method-level transaction isolation settings for an EJB.

Example

The `transaction-isolation` stanza can contain the elements shown here:

```
<transaction-isolation>
       <isolation-level>Serializable</isolation-level>
       <method>
              <description>...</description>
              <ejb-name>...</ejb-name>
              <method-intf>...</method-intf>
              <method-name>...</method-name>
              <method-params>...</method-params>
       </method>
</transaction-isolation>
```

trans-timeout-seconds

Range of values:	`0` to *`max`*
Default value:	`30`
Requirements:	Optional element. Valid only for all EJBs.
Parent elements:	`weblogic-enterprise-bean, transaction-descriptor`
Deployment file:	`weblogic-ejb-jar.xml`

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 9-65.

type-identifier

Range of values:	Valid string
Default value:	n/a
Requirements:	Required only for entity EJBs that use container-managed persistence services.
Parent elements:	`weblogic-enterprise-bean,` `entity-descriptor,` `persistence` `persistence-type` and `weblogic-enterprise-bean,` `entity-descriptor,` `persistence` `persistence-use`
Deployment file:	`weblogic-ejb-jar.xml`

Function

`type-identifier` contains text that identifies an entity EJB persistence type. WebLogic Server RDBMS-based persistence uses the identifier, `WebLogic_CMP_RDBMS`. If you use a different persistence vendor, consult the vendor's documentation for information on the correct `type-identifier`.

Example

See "persistence-type" on page 9-41 for an example that shows the complete `persistence-type` definition for WebLogic Server RDBMS-based persistence.

type-storage

Range of values:	Valid string
Default value:	n/a
Requirements:	Required only for entity EJBs that use container-managed persistence services.
Parent elements:	`weblogic-enterprise-bean,` `entity-descriptor,` `persistence` `persistence-type`
Deployment file:	`weblogic-ejb-jar.xml`

Function

`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.

Example

See "persistence-type" on page 9-41 for an example that shows the complete `persistence-type` definition for WebLogic Server RDBMS-based persistence.

type-version

Range of values:	Valid string
Default value:	n/a
Requirements:	Required only for entity EJBs that use container-managed persistence services.
Parent elements:	`weblogic-enterprise-bean, entity-descriptor, persistence persistence-type` and `weblogic-enterprise-bean, entity-descriptor, persistence persistence-use`
Deployment file:	`weblogic-ejb-jar.xml`

Function

`type-version` identifies the version of the specified persistence type.

Note: If you use WebLogic Server RDBMS-based persistence, the specified version must *exactly* match the RDBMS persistence version for the WebLogic Server release. Specifying an incorrect version results in the error:

```
weblogic.ejb.persistence.PersistenceSetupException: Error
initializing the CMP Persistence Type for your bean: No installed
Persistence Type matches the signature of (identifier
'Weblogic_CMP_RDBMS', version 'version_number').
```

Example

See persistence-type for an example that shows the complete `persistence-type` definition for WebLogic Server RDBMS-based persistence.

weblogic-cmp-rdbms-jar.xml Deployment Descriptor File

`weblogic-cmp-rdbms-jar.xml` defines deployment properties for a entity EJBs that uses WebLogic Server RDBMS-based persistence services. The EJB 2.0 container uses a version of `weblogic-cmp-rdbms-jar.xml` that is different from the one shipped with WebLogic Server Version 5.1. See Persistence Services for more information.

You can continue to use the earlier `weblogic-cmp-rdbms-jar.xml` DTD for EJB 1.1 beans that you will deploy on the WebLogic Server Version 6.0. However, if you want to use any of the new CMP 2.0 features, you must use the DTD described in the sections listed in "Index of weblogic-cmp-rdbms-jar.xml Deployment Elements" on page 9-71.

Index of weblogic-cmp-rdbms-jar.xml Deployment Elements

- "cmp-field" on page 9-73
- "cmr-field" on page 9-74
- "column-map" on page 9-75
- "data-source-name" on page 9-76
- "foreign-key-column" on page 9-81
- "ejb-name" on page 9-78
- "field-group" on page 9-79
- "foreign-key-column" on page 9-81
- "foreign-key-column" on page 9-81

- "group-name" on page 9-82
- "key-column" on page 9-83
- "max-elements" on page 9-84
- "method-name" on page 9-85
- "method-param" on page 9-86
- "method-params" on page 9-87
- "query-method" on page 9-88
- "relation-name" on page 9-89
- "relationship-role-name" on page 9-90
- "table-name" on page 9-91
- "weblogic-ql" on page 9-92
- "weblogic-query" on page 9-93
- "weblogic-relationship-role" on page 9-94

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 9-80.

cmr-field

Range of values:	Valid name
Default value:	n/a
Requirements:	The field referenced in `cmr-field` must have a matching `cmr-field` entry in the `ejb-jar.xml`.
Parent elements:	`weblogic-rdbms-relation` `field-group`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `cmr-field` element specifies the name of a cmr-field.

Example

The `XML` stanza can contain the elements shown here:

```
<weblogic-rdbms-jar>
     <weblogic-rdbms-relation>
        <field-group>employee</field-group>
           <cmp-field>employee stock purchases</cmp-field>
           <cmr-field>stock options</cmr-field>
     </weblogic-rdbms-relation>
</weblogic-rdbms-jar>
```

column-map

Range of values:	n/a.
Default value:	n/a
Requirements:	The value of a key-column entry that refers to a remote bean must always be empty.
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 a 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>
```

data-source-name

Range of values:	Valid name of the data source used for all data base connectivity for this bean.
Default value:	n/a
Requirements:	Must be defined as a standard WebLogic Server JDBC data source for database connectivity. See Programming WebLogic JDBC for more information.
Parent elements:	weblogic-rdbms-bean
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `data-source-name` that specifies the JDBC data source name to be used for all database connectivity for this bean.

Example

See “table-name” on page 9-91.

dbms-column

Range of values:	Valid name
Default value:	n/a
Requirements:	`dbms-column` is case maintaining, although not all databases require case sensitive entires.

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 9-80.

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 9-91.

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-jar>

     <weblogic-rdbms-relation>

             <field-group>employee</field-group>

       </weblogic-rdbms-relation>

</weblogic-rdbms-jar>
```

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 9-75.

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>
```

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 9-75.

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.

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:	This element 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 9-93.

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 9-93.

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 9-93.

relation-name

Range of values:	Valid name
Default value:	n/a
Requirements:	Must match the `ejb-relation-name` of an ejb-relation in the associated `ejb-jar.xml` descriptor file.
Parent elements:	`weblogic-rdbms-relation`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `relation-name` element specifies the name of a relation.

Example

The `XML` stanza can contain the elements shown here:

```
<weblogic-rdbms-jar>
       <weblogic-rdbms-relation>
              <relation-name>stocks-holders</relation-name>
                     <table-name>stocks</table-name>
       </weblogic-rdbms-relation>
</weblogic-rdbms-jar>
```

relationship-role-name

Range of values:	Valid name
Default value:	n/a
Requirements:	The name must match the `ejb-relationship-role-name` of an `ejb-relationship-role` in the associated `ejb-jar.xml` descriptor file.
Parent elements:	`weblogic-rdbms-relation` `weblogic-relationship-role`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `relationship-role-name` element specifies the name of a relationship role.

Example

The `XML` stanza can contain the elements shown here:

```
<weblogic-rdbms-jar>

     <weblogic-rdbms-relation>

       <weblogic-relationship-role>stockholder</weblogic-
relationship-role>

            <relationship-role-name>stockholders</relationship-
role-name>

       </weblogic-rdbms-relation>

</weblogic-rdbms-jar>
```

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>
```

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 EJB-QL. 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 9-93.

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-relationship-role

Range of values:	Valid name
Default value:	n/a
Requirements:	The mapping of a role to a table is specified in the associated `weblogic-rdbms-bean` and `ejb-relation` elements.
Parent elements:	`weblogic-rdbms-relation`
Deployment file:	`weblogic-cmp-rdbms-jar.xml`

Function

The `weblogic-relationship-role` element is used to express a mapping from a foreign key to a primary key. Only one mapping is specified for one-to-one relationships when the relationship is local. However, with a many-to-many relationship, you must specify two mappings

Multiple column mappings are specified for a single role, it the key is complex. No `column-map` is specified if the role is just specifying a group-name.

Example

The `XML` stanza can contain the elements shown here:

```
<weblogic-rdbms-jar>

       <weblogic-rdbms-relation>

              <relation-name>stocks-holders</relation-name>

                     <table-name>stocks</table-name>

                     <weblogic-relationship-role>stockholder
</weblogic-relationship-role>
```

```
          </weblogic-rdbms-relation>
</weblogic-rdbms-jar>
```

CHAPTER

10 WebLogic Server 5.1 EJB Deployment Properties

The following sections provide a complete reference for the WebLogic Server 5.1 specific XML deployment properties used in the WebLogic Server EJB 1.1 container and an explanation of how to edit the XML deployment files manually. Use these sections if you need to refer to a list of the deployment descriptors used for EJB 1.1 beans. You can either edit the XML or use the `DDConverter` to convert the EJB 1.1 deployment descriptors to EJB 2.0 XML that can be used in the EJB 2.0 container.

- Manually Editing XML Deployment Files
- weblogic-ejb-jar.xml Deployment Descriptor File
- weblogic-cmp-rdbms-jar.xml Deployment Descriptor File

Manually Editing XML Deployment Files

To define or make changes to the XML deployment descriptors used in the WebLogic Server EJB 1.1 container you must manually define or edit the XML elements in the following files:

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

Basic Conventions

To manually edit XML elements:

- Make sure that you use an ASCII text editor that does not reformat the XML or insert additional characters that could invalidate the file
- Use the correct case for file and directory names, even if your operating system ignores the case.
- To use the default value for an optional element, you can either omit the entire element definition or specify a blank value, as in:

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

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.

XML File	PUBLIC Element String
`ejb-jar.xml`	`‘-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN’` `‘http://www.java.sun.com/j2ee/dtds/ejb-jar_1_1.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’`
`weblogicmp-rdbms-jar.xml`	`‘-//BEA Systems, Inc.//DTD WebLogic 5.1.0 EJB RDBMS Persistence//EN’` `http://www.bea.com/servers/wls510/dtd/weblogic-rdbms-persistence.dtd`

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 5.1.0 EJB//EN'
'http://www.bea.com/servers/wls510/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 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.

The following links provide the public DTD locations for XML deployment files used with WebLogic Server:

- `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`.
- `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*
- `weblogic-rdbms-persistence.dtd` contains the DTD that defines container-managed persistence properties for entity EJBs. This DTD is used to create the `weblogic-rdbms-persistence.xml` file for using WebLogic Server persistence services. Third-party persistence vendors may also create XML deployment files that conform to this DTD. This file is located at *http://www.bea.com/servers/wls510/dtd/weblogic-rdbms-persistence.dtd*

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 Deployment Descriptor File

`weblogic-ejb-jar.xml` defines EJB deployment properties unique to WebLogic Server. The top level elements in `weblogic-ejb-jar.xml` are as follows:

- `description of the file`
- `copyright information`
- `weblogic-enterprise-bean`
  - `ejb-name`
  - `caching-descriptor`
  - `persistence-descriptor`
  - `clustering-descriptor`
  - `transaction-descriptor`
  - `reference-descriptor`
  - `jndi-name`
  - `transaction-isolation`
- `security-role-assignment`

Caching Properties

This section describes the `weblogic-ejb-jar.xml` caching properties.

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 stateless session bean class only. 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 "Initializing and Using EJB Instances" on page 4-4 in "The WebLogic Server EJB Container" on page 4-1 for more information.

initial-beans-in-free-pool

Note: This element is valid only for stateless session EJBs.

If you specify a value for `initial-bean-in-free-pool`, WebLogic Server populates the free pool with the specified number of stateless session 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 "Removing Stateful Session EJB Instances" on page 4-5.

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 in WebLogic Server" on page 4-2 for more information.

`idle-timeout-seconds` defaults to 600 if you do not define the element.

cache-strategy

The `cache-strategy` element can be one of the following:

- `Read-Write`
- `Read-Only`

The default value is `Read-Write`. See "Setting the Entity EJBs to Read-Only" on page 4-12 for more information.

read-timeout-seconds

The `read-timeout-seconds` element specifies the number of seconds between `ejbLoad()` calls on a `Read-Only` entity bean. By default, `read-timeout-seconds` is set to 600 seconds. If you set this value to 0, WebLogic Server calls `ejbLoad` only when the bean is brought into the cache.

Persistence Properties

This section describes the `weblogic-ejb-jar.xml` persistence properties.

persistence-descriptor

The `persistence-descriptor` stanza specifies persistence options for entity EJBs. The following shows all elements contained in the `persistence-descriptor` stanza:

```
<persistence-descriptor>
   <is-modified-method-name>. . .</is-modified-method-name>
   <delay-updates-until-end-of-tx>. . .</delay-updates-until-end-of-tx>
   <persistence-type>
      <type-identifier>. . .</type-identifier>
      <type-version>. . .</type-version>
      <type-storage>. . .</type-storage>
   </persistence-type>
   <db-is-shared>. . .</db-is-shared>
   <stateful-session-persistent-store-dir>
      . . .
   </stateful-session-persistent-store-dir>
   <persistence-use>. . .</persistence-use>
```

```
</persistence-descriptor>
```

is-modified-method-name

`is-modified-method-name` specifies a method that WebLogic Server calls when the EJB is stored. The specified method must return a `boolean` value. If no method is specified, WebLogic Server always assumes that the EJB has been modified and always saves it.

Providing a method and setting it as appropriate can improve performance. However, any errors in the method's return value can cause data inconsistency problems. See "Using is-modified-method-name to Limit Calls to ejbStore()" on page 4-10 for more information.

delay-updates-until-end-of-tx

Set this element 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 4-8 for more information.

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

persistence-type

A `persistence-type` defines a persistence service that can be used by an EJB. You can define multiple `persistence-type` entries in `weblogic-ejb-jar.xml` for testing with multiple persistence services. Only the persistence type defined in "persistence-use" on page 10-10 is used during deployment.

`persistence-type` includes several elements that define the properties of a service:

- `type-identifier` contains text that identifies the specified persistence type. For example, WebLogic Server RDBMS persistence uses the identifier, `WebLogic_CMP_RDBMS`.
- `type-version` identifies the version of the specified persistence type.

Note: The specified version must *exactly* match the RDBMS persistence version for the WebLogic Server release. Specifying an incorrect version results in the error:

```
weblogic.ejb.persistence.PersistenceSetupException: Error
initializing the CMP Persistence Type for your bean: No installed
Persistence Type matches the signature of (identifier
'Weblogic_CMP_RDBMS', version 'version_number').
```

- `type-storage` defines the full path of the file that stores data for this persistence type. The path must specify the file's location relative to the top level of the EJB's *.jar* deployment file or deployment directory.

  WebLogic Server RDBMS-based persistence generally uses an XML file named `weblogic-cmp-rdbms-jar.xml` to store persistence data for a bean. This file is stored in the `META-INF` subdirectory of the *.jar* file.

The following shows an example `persistence-type` stanza with values appropriate for WebLogic Server RDBMS persistence:

```
<persistence-type>

     <type-identifier>WebLogic_CMP_RDBMS</type-identifier>

     <type-version>5.1.0</type-version>

     <type-storage>META-INF\weblogic-cmp-rdbms-jar.xml</type-storage>

</persistence-type>
```

db-is-shared

The `db-is-shared` element applies only to entity beans. When set to `true` (the default value), WebLogic Server assumes that EJB data could be modified between transactions and reloads data at the beginning of each transaction. When set to `false`, WebLogic Server assumes that it has exclusive access to the EJB data in the persistent store. See "Using db-is-shared to Limit Calls to ejbLoad()" on page 4-9 for more information.

stateful-session-persistent-store-dir

`stateful-session-persistent-store-dir` specifies the file system directory where WebLogic Server stores the state of passivated stateful session bean instances.

persistence-use

The `persistence-use` element is similar to persistence-type, but it defines the persistence service actually used during deployment. `persistence-use` uses the `type-identifier` and `type-version` elements defined in a `persistence-type` to identify the service.

For example, to actually deploy an EJB using the WebLogic Server RDBMS-based persistence service defined in persistence-type, the `persistence-use` stanza would resemble:

```
<persistence-use>
   <type-identifier>WebLogic_CMP_RDBMS</type-identifier>
   <type-version>5.1.0</type-version>
</persistence-use>
```

Clustering Properties

This section describes the `weblogic-ejb-jar.xml` clustering properties.

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 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.

home-call-router-class-name

`home-call-router-class-name` specifies the custom class to use for routing bean method calls. This class must implement `weblogic.rmi.extensions.CallRouter()`. If specified, an instance of this class is called before each method call. The router class has the opportunity to choose a server to route to based on the method parameters. The class returns either a server name or null, which indicates that the current load algorithm should select the server.

stateless-bean-is-clusterable

This element is similar to `home-is-clusterable`, but it is applicable only to stateless session EJBs.

stateless-bean-load-algorithm

This element is similar to `home-load-algorithm`, but it is applicable only to stateless session EJBs.

stateless-bean-call-router-class-name

Thiselement 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 element 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 element is applicable only to stateless session EJBs.

Transaction Properties

This section describes the `weblogic-ejb-jar.xml` transaction properties.

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.

EJB References

This section describes the `weblogic-ejb-jar.xml` EJB references.

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`:

- `res-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.

Isolation Level Settings

This section describes the `weblogic-ejb-jar.xml` isolation level settings.

transaction-isolation

The `transaction-isolation` stanza specifies the transaction isolation level for EJB methods. The stanza consists of one or more `isolation-level` elements that apply to a range of EJB methods. For example:

```
<transaction-isolation>
```

```
        <isolation-level>Serializable</isolation-level>
        <method>
                <description>...</description>
                <ejb-name>...</ejb-name>
                <method-intf>...</method-intf>
                <method-name>...</method-name>
                <method-params>...</method-params>
        </method>
</transaction-isolation>
```

The following sections describe each element in `transaction-isolation`.

isolation-level

`isolation-level` defines a valid transaction isolation level to apply to specific EJB methods. The following are possible values for `isolation-level`:

- `TRANSACTION_READ_UNCOMMITTED`: The transaction can view uncommitted updates from other transactions.
- `TRANSACTION_READ_COMMITTED`: The transaction can view only committed updates from other transactions.
- `TRANSACTION_REPEATABLE_READ`: Once the transaction reads a subset of data, repeated reads of the same data return the same values, even if other transactions have subsequently modified the data.
- `TRANSACTION_SERIALIZABLE`: Simultaneously executing this transaction multiple times has the same effect as executing the transaction multiple times in a serial fashion.

Refer to your database documentation for more information on the implications and support for different isolation levels.

method

The `method` stanza defines the EJB methods to which an isolation level applies. `method` defines a range of methods using the following elements:

- `description` is an optional element that describes the method.
- `ejb-name` identifies the EJB to which WebLogic Server applies isolation level properties.
- `method-intf` is an optional element that 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 Assignments

This section describes the `weblogic-ejb-jar.xml` security role assignments.

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.

enable-call-by-reference

By default, EJB methods called from within the same server 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 server).

weblogic-cmp-rdbms-jar.xml Deployment Descriptor File

`weblogic-cmp-rdbms-jar.xml` defines deployment properties for a single entity EJB that uses WebLogic Server RDBMS-based persistence services. See "Persistence Services" on page 4-28 for more information.

The top-level element of `weblogic-cmp-rdbms-jar.xml` are as follows:

- description of the file
- copyright information
- weblogic-enterprise-bean stanza

```
<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>
```

RDBMS Definition Elements

This section describes the `weblogic-cmp-rdbms-jar.xml` 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 `weblogic-cmp-rdbms-jar.xml` 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 `weblogic-cmp-rdbms-jar.xml` finder elements.

finder-list

The `finder-list` stanza defines the set of all finders that are generated to locate sets of beans. See “Writing Finders for RDBMS Persistence” on page 4-29 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)" on page 4-31 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.