Oracle® Fusion Middleware

Developing Enterprise JavaBeans, Version 3.2, for Oracle WebLogic Server

14c (14.1.2.0.0)
F61466-02
December 2024

ORACLE

Oracle Fusion Middleware Developing Enterprise JavaBeans, Version 3.2, for Oracle WebLogic Server, 14c (14.1.2.0.0)

F61466-02

Contents

2 Designing EJBs

3 Implementing EJBs

C weblogic-cmp-jar.xml Deployment Descriptor Reference

E Important Information for EJB 1.1 Users

F EJB Query Language (EJB-QL) and WebLogic Server

Preface

This document is a resource for software developers who develop applications that include WebLogic Server EJBs, version 3.2 or earlier.

Audience

This document is a resource for software developers who develop applications that include WebLogic Server EJBs, version 3.2. It also contains information that is useful for business analysts and system architects who are evaluating WebLogic Server or considering the use of WebLogic Server EJBs for a particular application.

The topics in this document are relevant during the design and development phases of a software project. The document also includes topics that are useful in solving application problems that are discovered during test and pre-production phases of a project. This document does not address production phase administration, monitoring, or performance tuning.

It is assumed that the reader is familiar with Jakarta EE and EJB 3.2 concepts. This document emphasizes the value-added features provided by WebLogic Server EJBs and key information about how to use WebLogic Server features and facilities to get an EJB application up and running.

Documentation Accessibility

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

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners,

we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Documentation

This document contains EJB-specific design and development information. For information on programming and packaging 3.2 EJBs, see *Developing Enterprise JavaBeans for Oracle WebLogic Server*.

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

- *Developing Message-Driven Beans for Oracle WebLogic Server* is a resource for developing applications that use message-driven beans (MDBs).
- *Developing Applications for Oracle WebLogic Server* is a guide to developing WebLogic Server applications.
- *Deploying Applications to Oracle WebLogic Server* is the primary source of information about deploying WebLogic Server applications in development and production environments.
- *Tuning Performance of Oracle WebLogic Server* provides information on how to monitor performance and tune the components in a WebLogic Server.

Samples and Tutorials

Oracle provides a variety of code examples and tutorials that show WebLogic Server configuration and API use, and provide practical instructions on how to perform key development tasks. For more information, see Sample Applications and Code Examples in *Understanding Oracle WebLogic Server*.

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

New and Changed WebLogic Server Features

For a comprehensive listing of the new WebLogic Server features introduced in this release, see What's New in WebLogic Server in *What's New in Oracle WebLogic Server*.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Understanding Jakarta Enterprise Beans

Review the different Jakarta Enterprise Bean (EJB) types and the functions they can serve in an application, and how they work with other application objects and WebLogic Server.
It is assumed the reader is familiar with Java programming and EJB 3.x concepts and features.

This chapter includes the following topics:

- How Do Applications Use EJBs?
  Examine the purpose and capabilities of each bean type.
- EJB Anatomy and Environment
  Examine the classes required for each bean type, the EJB run-time environment, and the deployment descriptor files that govern a bean's run-time behavior.
- EJBs, Clients, and Application Objects
  Understand how EJBs typically relate to other components of a WebLogic Server application and to clients.
- EJBs and Message Destination References
  Learn how to use logical message destinations to map a logical message destination, defined in `ejb-jar.xml`, to an actual message destination, defined in `weblogic-ejb-jar.xml`.
- WebLogic Server Value-Added EJB Features
  Examine the key features in WebLogic Server that ease the process of EJB development, and enhance the performance, reliability, and availability of EJB applications.
- Securing EJBs
  You can use the WebLogic Server security features to control both access to your EJBs (authorization) and verification of an EJB's identity when it interacts with other application components and other EJBs (authentication).

How Do Applications Use EJBs?

Examine the purpose and capabilities of each bean type.

This section includes the following topics:

- Session EJBs Implement Business Logic
- Entity EJBs Maintain Persistent Data
- Message-Driven Beans Implement Loosely Coupled Business Logic

Session EJBs Implement Business Logic

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

- Stateless Session Beans
- Stateful Session Beans

Stateless Session Beans

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

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

Stateful Session Beans

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

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

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

Entity EJBs Maintain Persistent Data

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

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

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

Message-Driven Beans Implement Loosely Coupled Business Logic

A message-driven bean implements loosely coupled or asynchronous business logic in which the response to a request need not be immediate. A message-driven bean receives messages

from a JMS Queue or Topic, and performs business logic based on the message contents. It is an asynchronous interface between EJBs and JMS.

Throughout its life cycle, an MDB instance can process messages from multiple clients, although not simultaneously. It does not retain state for a specific client. All instances of a message-driven bean are equivalent—the EJB container can assign a message to any MDB instance. The container can pool these instances to allow streams of messages to be processed concurrently.

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

See *Developing Message-Driven Beans for Oracle WebLogic Server*.

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

EJB Anatomy and Environment

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

This section includes the following topics:

EJB Components

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

Table 1-1 Components of an EJB

EJB Component	Description	Stateless Session	Stateful Session	Entity	MDB
Remote interface	The remote interface exposes business logic to remote clients—clients running in a separate application from the EJB. It defines the business methods a remote client can call.	Yes	Yes	Yes	No
Local interface	The local interface exposes business logic to local clients—those running in the same application as the EJB. It defines the business methods a local client can call. **Note**: Not available for 1.1 EJBs.	Yes	Yes	Yes	No

Table 1-1 (Cont.) Components of an EJB

EJB Component	Description	Stateless Session	Stateful Session	Entity	MDB
Local home interface	The local home interface, also referred to as an EJB factory or life-cycle interface, provides methods that local clients—those running in the same application as the EJB—can use to create, remove, and in the case of an entity bean, find instances of the bean. The local home interface also has "home methods"—business logic that is not specific to a particular bean instance.	Yes	Yes	Yes	No
Remote home interface	The remote home interface, also referred to as an EJB factory, or life-cycle interface, provides methods that remote clients—those running in a separate application from the EJB—can use to create, remove, and find instances of the bean.	Yes	Yes	Yes	No
Bean class	The bean class implements business logic.	Yes	Yes	Yes	Yes
Primary key class	Only entity beans have a primary key class. The primary key class maps to one or more fields in a database—identifying the persistent data to which the entity bean corresponds.	No	No	Yes	No

The EJB Container

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

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

EJB Deployment Descriptors

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

WebLogic Server EJBs have three deployment descriptors:

- `ejb-jar.xml`—The standard Jakarta EE deployment descriptor. All beans must be specified in an `ejb-jar.xml`. An `ejb-jar.xml` can specify multiple beans that will be deployed together.
- `weblogic-ejb-jar.xml`—WebLogic Server-specific deployment descriptor that contains elements related to WebLogic Server features such as clustering, caching, and

transactions. This file is required if your beans take advantage of WebLogic Server-specific features. Like `ejb-jar.xml`, `weblogic-ejb-jar.xml` can specify multiple beans that will be deployed together. See weblogic-ejb-jar.xml Deployment Descriptor Reference.

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

Key Deployment Element Mappings

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

Table 1-2 lists the elements whose values should match in each descriptor file. The elements listed in the table are defined in Bean and Resource References and Security Roles.

Table 1-2 Element Mapping Among Descriptor Files

Map this element...	in this element of ejb-jar.xml...	to the same element in this element of weblogic-ejb-jar.xml...	and to this element in weblogic-cmp-jar.xml...
`role-name`	`security-role`	`security-role-assignment`	N/A
`ejb-name`	`message-driven`, `entity`, or `session`	`weblogic-enterprise-bean`	`weblogic-rdbms-bean`
`ejb-ref-name`	`assembly-descriptor`	`ejb-reference-description` if the referenced bean runs in a different container than the current bean.	N/A
`res-ref-name`	`resource-ref`	`resource-description`	N/A

- Bean and Resource References
- Security Roles

Bean and Resource References

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

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

A given bean or resource factory is identified by the same value in each descriptor file that contains it. Table 1-1 lists the bean and resource references elements, and their location in each descriptor file.

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

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

would appear in the:

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

Security Roles

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

For information on:

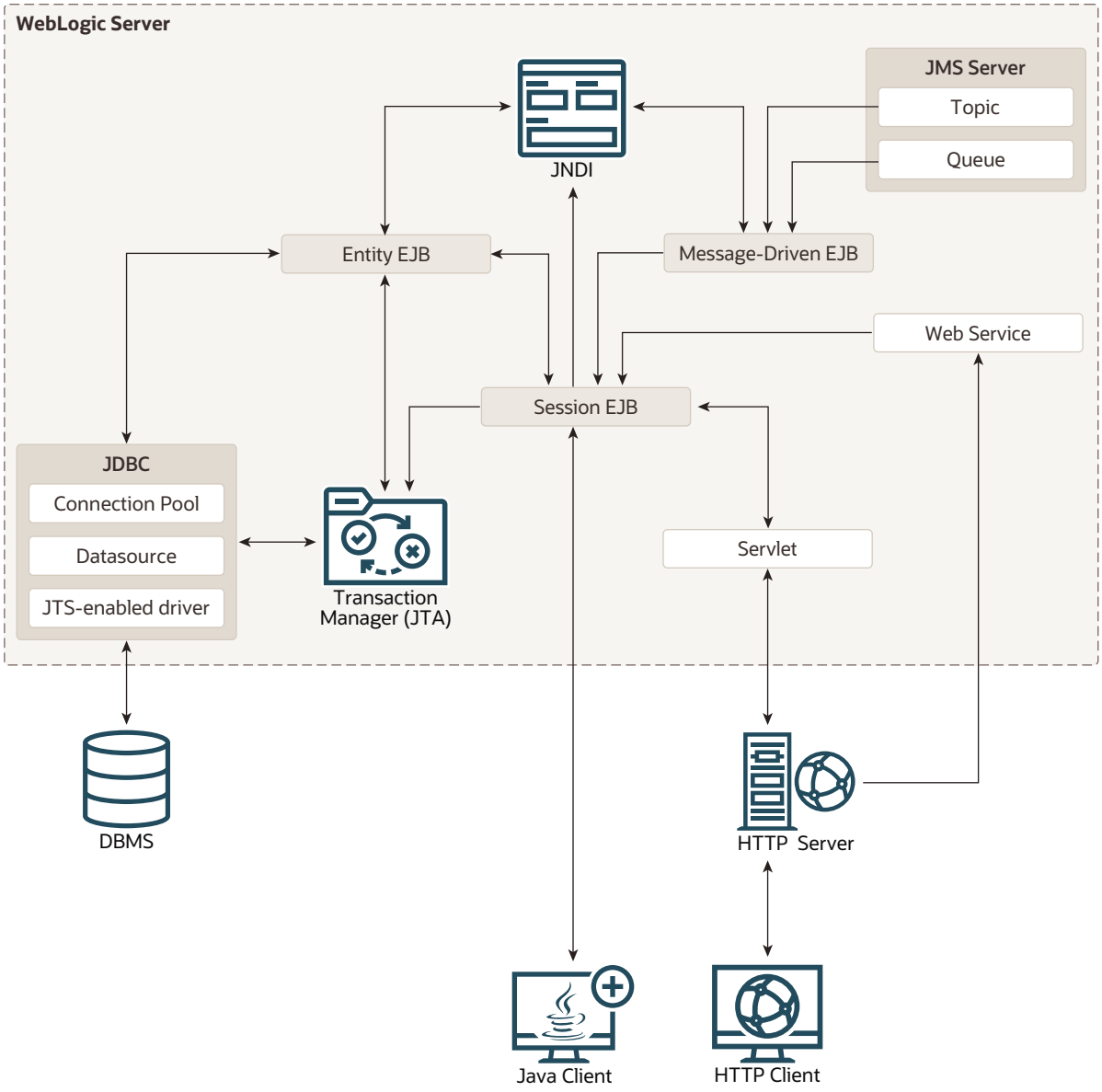
- Programming security features for an EJB, see Securing Enterprise JavaBeans (EJBs) in *Developing Applications with the WebLogic Security Service*.
- Editing deployment descriptor files, see Edit Deployment Descriptors.
- Elements in `ejb-jar.xml`, see `http://xmlns.jcp.org/xml/ns/javaee/ejb-jar_3_2.xsd`.
- Elements in `weblogic-ejb-jar.xml`, see weblogic-ejb-jar.xml Deployment Descriptor Reference.
- Elements in `weblogic-cmp-jar.xml`, see weblogic-cmp-jar.xml Deployment Descriptor Reference.

EJBs, Clients, and Application Objects

Understand how EJBs typically relate to other components of a WebLogic Server application and to clients.

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

Figure 1-1 EJBs and Other Application Components



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

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

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

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

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

See Programming Client Access to EJBs for information on programming client access to an EJB.

- EJB Communications

EJB Communications

WebLogic Server EJBs use:

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

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

You can specify the attributes of the network connection an EJB uses by binding the EJB to a WebLogic Server custom network channel. See Configuring Network Communications for an EJB.

EJBs and Message Destination References

Learn how to use logical message destinations to map a logical message destination, defined in `ejb-jar.xml`, to an actual message destination, defined in `weblogic-ejb-jar.xml`.

You can use logical message destinations to map a logical message destination, to an actual message destination, as described in Configuring EJBs to Use Logical Message Destinations in *Developing Message-Driven Beans for Oracle WebLogic Server*.

If a message destination reference cannot be resolved during deployment, a warning is issued but the deployment will succeed. MDBs linked to unavailable message destinations periodically attempt to connect to the message destination. Until the message destination is available, attempts to look up `message-destination-references` declared in `ejb-jar.xml` fail with a `javax.naming.NamingException`. When the message destination becomes available, the MDBs will connect to it and service messages from it.

WebLogic Server Value-Added EJB Features

Examine the key features in WebLogic Server that ease the process of EJB development, and enhance the performance, reliability, and availability of EJB applications.

This section describes the key WebLogic Server features in detail.

- Performance-Enhancing Features for WebLogic Server EJBs
- Pooling Improves EJB Response Time

- Caching Improves EJB Performance
- Additional Caching Capabilities for CMP Entities
- Field Groups for Efficient Queries (CMP Entities)
- Configurable Write Behaviors
- Operation Ordering and Batching (CMP Entities)
- Optimized Database Updates (CMP Entities)
- Read-Only Pattern and Read-Only Invalidation (CMP Entities)
- CMP Beans Increase Developer Productivity
- Automatic Primary Key Generation (CMP Entities)
- Automatic Table Creation (CMP Entities)
- Dynamic Queries (CMP Entities)
- Reliability and Availability Features
- Load Balancing Among Clustered EJBs Increases Scalability
- Failover for Clustered EJBs Increases Reliability

Performance-Enhancing Features for WebLogic Server EJBs

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

Pooling Improves EJB Response Time

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

- Understanding Entity Pooling
- Message-Driven EJB Life Cycle and the Free Pool in *Developing Message-Driven Beans for Oracle WebLogic Server*
- Pooling for Stateless Session EJBs

Caching Improves EJB Performance

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

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

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

For more information, see:

- Understanding Entity Caching

- Caching and Passivating Stateful Session EJBs

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

Additional Caching Capabilities for CMP Entities

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

- Relationship caching—Relationship caching improves the performance of entity beans by loading related beans into the cache and avoiding multiple queries by issuing a join query for the related beans. For more information, see Relationship Caching.
- Application-level caching—Application-level caching, also known as "combined caching," allows multiple entity beans that are part of the same Jakarta EE application to share a single runtime cache. For more information, see Configuring Application-Level Caching .
- Caching between transactions—Use caching between transactions or long term caching to enable the EJB container to cache an entity bean's persistent data between transactions. For more information, see Limiting Database Reads with cache-between-transactions.

Field Groups for Efficient Queries (CMP Entities)

A group specifies a set of persistent attributes of an entity bean. A field-group represents a subset of the container-managed persistence (CMP) and container-managed relation (CMR) fields of a bean. You can put related fields in a bean into groups that are faulted into memory together as a unit. You can associate a group with a query or relationship, so that when a bean is loaded as the result of executing a query or following a relationship, only the fields mentioned in the group are loaded. For more information, see Specifying Field Groups.

Configurable Write Behaviors

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

Operation Ordering and Batching (CMP Entities)

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

Optimized Database Updates (CMP Entities)

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

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

For EJB data that is only occasionally updated, you can create a "read-mostly pattern" by implementing a combination of read-only and read-write EJBs. If you use the read-mostly

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

CMP Beans Increase Developer Productivity

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

Automatic Primary Key Generation (CMP Entities)

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

Automatic Table Creation (CMP Entities)

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

Dynamic Queries (CMP Entities)

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

Reliability and Availability Features

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

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

Failover and load balancing for EJBs are handled by replica-aware stubs, which can locate instances of the object throughout the cluster. Replica-aware stubs are created for EJBs as a result of the object compilation process. EJBs can have two different replica-aware stubs: one for the `EJBHome` interface and one for the `EJBObject` interface. This allows some bean types to take advantage of load balancing and failover features at the home level when a client looks up an EJB object using the `EJBHome` stub and at the method level when a client makes method calls against the EJB using the EJBObject stub. Table 1-1 summarizes the load balancing and failover support (method level and home level) for each EJB type.

Load Balancing Among Clustered EJBs Increases Scalability

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

WebLogic Server clusters support multiple algorithms for load balancing clustered EJBs; the default is the round-robin method. See Load Balancing in a Cluster in *Administering Clusters for Oracle WebLogic Server*.

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

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

Failover for Clustered EJBs Increases Reliability

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

EJB failover requires that bean methods must be idempotent, and configured as such in `weblogic-ejb-jar.xml`. See Replication and Failover for EJBs and RMI Objects in *Administering Clusters for Oracle WebLogic Server*.

Table 1-3 summarizes failover and load balancing features for clustered EJBs.

Table 1-3 Failover and Load Balancing for Clustered EJBs

EJB Type	Home Level Failover	Method Level Failover	Notes
Stateless Session	Supported	Supported	Stateless session EJB clustering behaviors are configured in `weblogic-ejb-jar.xml`. See WebLogic-Specific Configurable Behaviors for Session Beans.
Stateful session	Supported	Supported	The `EJBObject` stub maintains the location of the EJB's primary and secondary states. Secondary server instances are selected using the same rules defined in Using Replication Groups in *Administering Clusters for Oracle WebLogic Server*. Stateful session EJB clustering behaviors are configured in `weblogic-ejb-jar.xml`. See WebLogic-Specific Configurable Behaviors for Session Beans.
Read-Write Entity	Supported	Not supported	`EJBHome` stubs do not fail over in the event of a recoverable call failure. Failover is not supported during method execution, only after method completion, or if the method fails to connect to a server instance. A read-write bean's home obtains a local instance of the bean and returns a stub pinned to the local server instance. Entity clustering behaviors are configured in `weblogic-ejb-jar.xml`. See Clustering Elements.

Table 1-3 (Cont.) Failover and Load Balancing for Clustered EJBs

EJB Type	Home Level Failover	Method Level Failover	Notes
Read-Only Entity	Supported	Supported	`EJBHome` stubs do not failover in the event of a recoverable call failure. Entity clustering behaviors are configured in `weblogic-ejb-jar.xml`. See Clustering Elements.
Message-Driven	N/A	N/A	WebLogic Jakarta Messaging Service (JMS) supports clustering of JMS servers. See JMS and Clustering in *Administering Clusters for Oracle WebLogic Server*.

See Replication and Failover for EJBs and RMI Objects and Load Balancing for EJBs and RMI Objects in *Administering Clusters for Oracle WebLogic Server*.

Securing EJBs

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

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

For more information about security and EJBs:

- Security Fundamentals in *Understanding Security for Oracle WebLogic Server* has introductory information about authentication, authorization and other security topics.
- Securing Enterprise JavaBeans (EJBs) in *Developing Applications with the WebLogic Security Service* provides instructions for configuring authentication and authorization for EJBs.
- *Securing Resources Using Roles and Policies for Oracle WebLogic Server* contains instructions for on configuring authentication and authorization for your EJBs.

2

Designing EJBs

Understand the design options for WebLogic Server EJBs, bean behaviors to consider during the design process, and recommended design patterns.
This chapter includes the following topics:

It is assumed the reader is familiar with Java programming, EJB 3.2 and earlier, and the features described in WebLogic Server Value-Added EJB Features.

After finalizing your bean design, refer to programming and other implementation topics in Implementing EJBs.

- Choosing the Right Bean Type
  Examine the different natures and capabilities of the various bean types, when you choose the bean type for a processing task.
- Persistence Management Alternatives
  Persistence management strategy determines how an entity bean's database access is performed.
- Transaction Design and Management Options
  A transaction is a unit of work that changes application state—whether on disk, in memory or in a database—that, once started, is completed entirely, or not at all.
- Satisfying Application Requirements with WebLogic Server EJBs
  Understand the variety of value-added features for enterprise beans that you can configure to meet the requirements of your application in WebLogic Server.

Choosing the Right Bean Type

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

Bean types vary in terms of the bean's relationship with its client. Some bean types stick with a client throughout a series of processes, serving as a sort of general contractor—acquiring and orchestrating services for the client. There are other bean types that act like subcontractors, they deliver the same single function to multiple client-oriented general contractor beans. A client-oriented bean keeps track of client interactions and other information associated with the client process, throughout a client session—a capability referred to as *maintaining state*. Beans that deliver commodity services to multiple client-oriented beans do not maintain state.

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

> **Note:**
>
> For a basic overview of each bean type, including an example of a business process to which each bean type is well suited, see How Do Applications Use EJBs?.

- Session Bean Features

Session Bean Features

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

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

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

Stateful Session Beans

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

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

Stateless Session Beans

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

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

Stateless Beans Offer Performance and Scalability Advantages

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

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

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

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

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

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

Exposing Stateless Session Beans as Web Services

In this release of WebLogic Server, you can expose stateless session beans as Web Services through the Web service endpoint interface. See *Developing JAX-WS Web Services for Oracle WebLogic Server*.

Entity Bean Features

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

- Key Features of Entity Beans
- Read-Write versus Read-Only Entity Beans
- Entity Bean Performance and Data Consistency Characteristics

Key Features of Entity Beans

These are the key features of entity beans:

- Persistence—Entity bean persistence can be managed by the EJB container, or the bean itself. If a bean uses container-managed persistence, the EJB container automatically generates the necessary database access calls. The code that you write for the entity bean does not include these calls. With bean-managed persistence, you must write the database access code and include it in the bean.
- Shared Access—Throughout its life cycle, an entity bean instance can support multiple clients, although not at the same time. Because the clients might want to change the same data, it is important that entity beans work within transactions. Typically, the EJB container provides transaction management. In this case, you specify the transaction attributes in the bean's `ejb-jar.xml` file that control how transactions are managed. You do not have to code the transaction boundaries in the bean—the container marks the boundaries for you. For information about transaction management, see Transaction Design and Management Options.
- Primary Key—Each entity bean has a unique object identifier. A customer entity bean, for example, might be identified by a customer number. The unique identifier, or primary key,

enables the client to locate a particular entity bean. For more information, see Using Container-Managed Relationships (CMRs).

- Relationships—Like a table in a relational database, an entity bean may be related to other entity beans. You implement relationships differently for entity beans with bean-managed persistence and for those with container-managed persistence. With bean-managed persistence, the code that you write implements the relationships. But with container-managed persistence, the EJB container takes care of the relationships for you. For this reason, relationships in entity beans with container-managed persistence are often referred to as container-managed relationships. For more information, see Using Cascade Delete for Entities in CMRs.

Read-Write versus Read-Only Entity Beans

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

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

Some applications require the use of read-write entity beans—the choice depends on frequency of updates and data consistency requirements. Table 2-1 provide key guidelines.

Table 2-1 Comparing Read-Write and Read-Only Entity Beans

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

Entity Bean Performance and Data Consistency Characteristics

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

- Use Read-Only Beans to Improve Performance If Stale Data Is Tolerable
- Use Read-Write Beans for Higher Data Consistency
- Combine Read-Only and Read-Write Beans to Optimize Performance
- Use Session Facades to Optimize Performance for Remote Entity Beans
- Avoid the Use of Transfer Objects

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

Read-only entity beans are recommended whenever stale data is tolerable—they are suitable for product catalogs and the majority of content within many applications. Primary key-based reads are performed against a local entity cache that is invalided on a timer basis. Other

queries are made against the database. Read-only entity beans perform three to four times faster than transactional entities.

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

Use Read-Write Beans for Higher Data Consistency

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

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

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

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

Use Session Facades to Optimize Performance for Remote Entity Beans

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

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

Avoid the Use of Transfer Objects

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

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

Message-Driven Beans

A message-driven bean (MDB) is an enterprise bean that allows Jakarta EE applications to process messages asynchronously. An MDB acts as a JMS or JCA message listener, which is

similar to an event listener except that it receives messages instead of events. The messages may be sent by any Jakarta EE component—an application client, another enterprise bean, or a Web component—or by non-Jakarta EE applications.

See *Developing Message-Driven Beans for Oracle WebLogic Server*.

Persistence Management Alternatives

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

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

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

- Use Container-Managed Persistence (CMP) for Productivity and Portability
- Use Bean-Managed Persistence (BMP) Only When Necessary

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

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

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

Container-managed persistence offers these advantages:

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

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

For more information on features supported by CMP entities, see Entity EJBs.

Use Bean-Managed Persistence (BMP) Only When Necessary

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

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

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

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

Transaction Design and Management Options

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

This section describes the transaction design and management options in detail.

- Understanding Transaction Demarcation Strategies and Performance
- Demarcating Transactions at the Server Level is Most Efficient
- Container-Managed Transactions Are Simpler to Develop and Perform Well
- Bean-Level Transaction Management
- Client-Level Transaction Management is Costly
- Transaction Isolation: A Performance vs. Data Consistency Choice

Understanding Transaction Demarcation Strategies and Performance

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

Demarcating Transactions at the Server Level is Most Efficient

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

Container-Managed Transactions Are Simpler to Develop and Perform Well

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

Each method in a CMT bean can be associated with a single transaction, but does not have to be. In a container-managed transaction, the EJB container manages the transaction, including

start, stop, commit, and rollback. Usually, the container starts a transaction just before a bean method starts, and commits it just before the method exits.

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

- Rollback
- Transaction Boundaries
- Distributing Transactions Across Beans
- Costly Option: Distributing Transactions Across Databases

Rollback

If an exception is thrown during a transaction, the container will automatically roll back the transaction. You can configure the EJB container to automatically retry container-managed transactions that have rolled back, provided the transactions did not roll back as the result of system exception-based errors. You can also explicitly program rollbacks in your bean. For more information see Implementing EJBs.

Transaction Boundaries

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

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

Distributing Transactions Across Beans

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

Costly Option: Distributing Transactions Across Databases

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

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

Bean-Level Transaction Management

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

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

When to Use Bean-Managed Transactions

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

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

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

Keep Bean-Managed Transactions Short

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

For information about implementing bean-managed transactions, see Programming Bean-Managed Transactions.

Client-Level Transaction Management is Costly

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

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

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

Transaction Isolation: A Performance vs. Data Consistency Choice

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

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

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

For more information on setting container-managed transaction isolation levels, see *Developing JTA Applications for Oracle WebLogic Server*.

Satisfying Application Requirements with WebLogic Server EJBs

Understand the variety of value-added features for enterprise beans that you can configure to meet the requirements of your application in WebLogic Server.

These features are described in WebLogic Server Value-Added EJB Features.

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

Table 2-2 Features and Design Patterns

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

3
Implementing EJBs

This chapter describes the EJB implementation process, and provides guidance for how to get an EJB up and running in WebLogic Server.
It is assumed that you understand WebLogic Server's value-added EJB features, have selected a design pattern for your application, and have made key design decisions.

For a review of WebLogic Server EJB features, see WebLogic Server Value-Added EJB Features.

For discussion of design options for EJBs, factors to consider during the design process, and recommended design patterns see Designing EJBs.

This chapter includes the following topics:

- Overview of the EJB Development Process
  Learn about the key implementation tasks and associated results in the process of developing EJB.
- Create a Source Directory
  Learn how to create a source directory where you can assemble the EJB.
- Create EJB Classes and Interfaces
- Programming the EJB Timer Service
- Declare Web Service References
  Web Service references, declared in an EJB's deployment descriptor, maps a logical name for a Web Service to an actual Web Service interface, which allows you to refer to the Web Service using a logical name. The bean code then performs a JNDI lookup using the Web Service reference name.
- Compile Java Source
  Learn about the tools that support compilation and the compilation process.
- Edit Deployment Descriptors
  Elements in `ejb-jar.xml`, `weblogic-ejb-jar.xml`, and for container-managed persistence entity beans, `weblogic-cmp-jar.xml`, control the run-time characteristics of your application. You can modify these deployment descriptors using an XML editing tool.
- Generate EJB Wrapper Classes, and Stub and Skeleton Files
  Container classes include the internal representation of the EJB that WebLogic Server uses and the implementation of the external interfaces (home, local, and/or remote) that clients use. You can use Oracle Workshop for WebLogic Platform or `appc` to generate container classes.
- Package
  You can package EJBs as part of an enterprise application.
- Deploy
  Deploying an EJB enables WebLogic Server to serve the components of an EJB to clients. You can deploy an EJB using one of several procedures, depending on your environment and whether or not your EJB is in production.
- Solving Problems During Development
  Learn about the WebLogic Server features that are useful for checking out and debugging deployed EJBs.

- WebLogic Server Tools for Developing EJBs
  Examine the Oracle tools that support the EJB development process.

Overview of the EJB Development Process

Learn about the key implementation tasks and associated results in the process of developing EJB.

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

Figure 3-1 illustrates the process of developing an EJB. The steps in the process, and the results of each are described in Table 3-1. Subsequent sections detail each step in the process.

Figure 3-1 EJB Development Process Overview

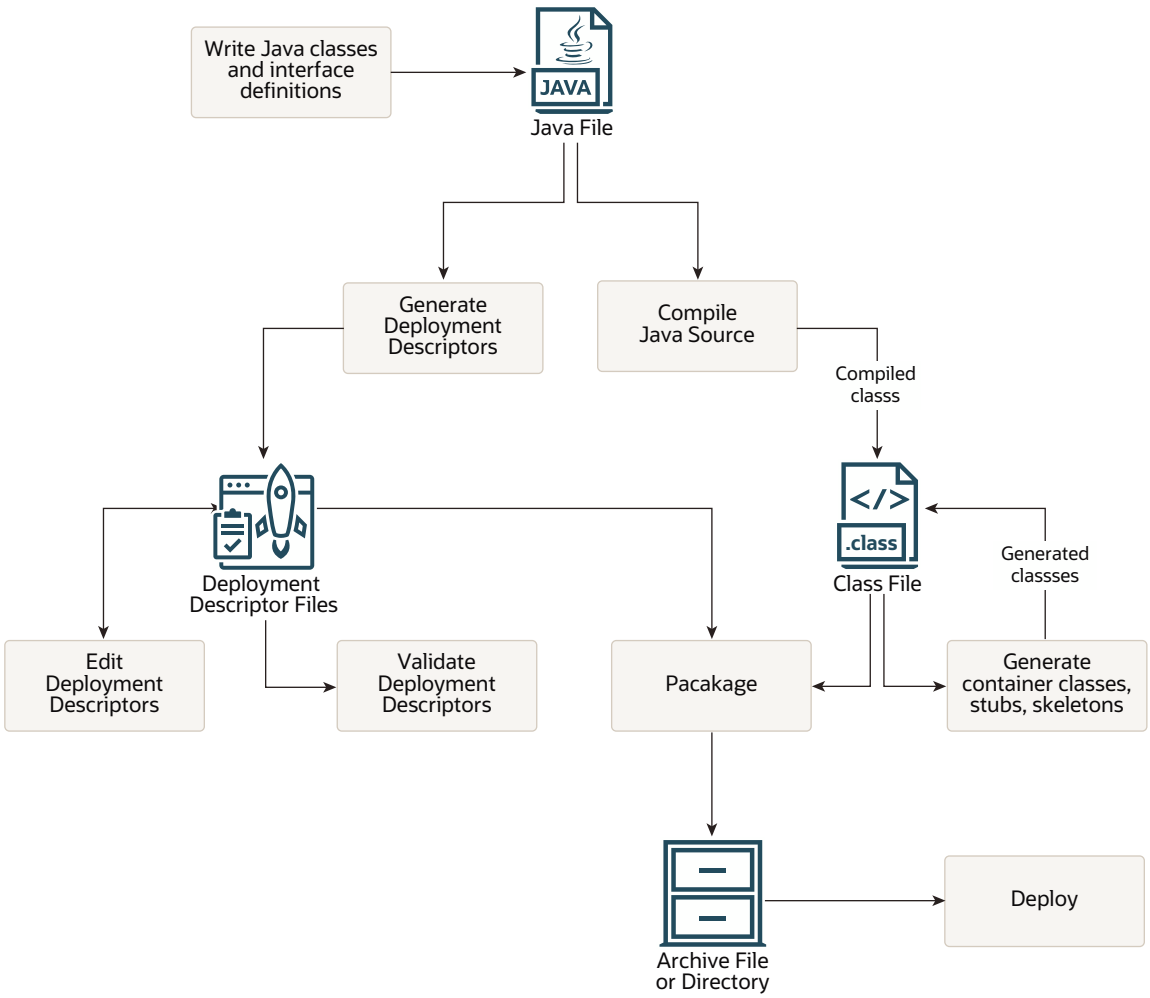


Table 3-1 EJB Development Tasks and Results

Step	Description	Result
Create a Source Directory	Create the directory structure for your source files, deployment descriptors, and files that are generated during the implementation process.	A directory structure on your local drive.
Create EJB Classes and Interfaces	Create the classes that make up your bean. Insert appropriate tags in your source code to enable automatic generation of deployment descriptor elements later in the implementation process.	`.java` file for each class
Compile Java Source	Compile source code.	`.class` file for each class
Edit Deployment Descriptors	You may need to edit deployment descriptors to ensure they correctly reflect all desired runtime behaviors for your bean.	• `ejb-jar.xml`, • `weblogic-ejb-jar.xml`, which contains elements that control WebLogic Server-specific features, and • `weblogic-cmp-jar.xml`, if the bean is a container-managed persistence entity bean.
Generate EJB Wrapper Classes, and Stub and Skeleton Files	Generate the container classes used to access the deployment unit, including classes for home and remote interfaces.	Generated classes are added to archive or directory.
Package	Package compiled files, generated files, and deployment descriptors for deployment. If appropriate, you can leave your files unarchived in an exploded directory.	Archive, either a JAR or an EAR
Deploy	Target the archive or application directory to desired Managed Server, or a WebLogic Server cluster, in accordance with selected staging mode.	The deployment settings for the bean are written to `EJBComponent` element in `config.xml`.

Create a Source Directory

Learn how to create a source directory where you can assemble the EJB.

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

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

Example 3-1 Directory Structure for Packaging JAR

```
myEJB/
    META-INF/
        ejb-jar.xml
```

```
        weblogic-ejb-jar.xml
        weblogic-cmp-jar.xml
    foo.class
    fooHome.class
    fooBean.class
```

Create EJB Classes and Interfaces

The classes required depend on the type of EJB you are developing, as described in EJB Components.

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

- Using WebLogic Server Generic Bean Templates
- Programming Client Access to EJBs
- Programming Client to Obtain Initial Context
- Programming Client to Look Up a Home Interface
- Configuring EJBs to Send Requests to a URL
- Specifying an HTTP Resource by URL
- Specifying an HTTP Resource by Its JNDI Name
- Accessing HTTP Resources from Bean Code
- Configuring Network Communications for an EJB
- Programming and Configuring Transactions
- Programming Container-Managed Transactions
- Configuring Automatic Retry of Container-Managed Transactions
- Programming Bean-Managed Transactions
- Programming Transactions That Are Distributed Across EJBs

Using WebLogic Server Generic Bean Templates

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

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

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

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

Programming Client Access to EJBs

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

Programming Client to Obtain Initial Context

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

Example 3-2 Local Client Performing a Lookup

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

    return new InitialContext(h);
}
```

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

Programming Client to Look Up a Home Interface

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

- By following an EJB reference. This approach offers better performance than the alternative, looking up the home interface directly from the Java Naming and Directory Interface, and is a Oracle best practice. For instructions on using EJB references, see the following section, Using EJB Links.
- Directly from the Java Naming and Directory Interface (JNDI). The container binds the entity bean's home interface in the global, server-side JNDI name space. See *Developing JNDI Applications for Oracle WebLogic Server*.
- Using EJB Links

Using EJB Links

Using EJB links is a Oracle best practice and WebLogic Server fully supports EJB links as defined in the EJB 2.1 Specification. You can link an EJB reference that is declared in one application component to an enterprise bean that is declared in the same Jakarta EE application.

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

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

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

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

Configuring EJBs to Send Requests to a URL

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

Specifying an HTTP Resource by URL

To specify the URL to which an EJB sends requests:

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

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

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

Specifying an HTTP Resource by Its JNDI Name

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

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

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

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

Accessing HTTP Resources from Bean Code

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

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

Configuring Network Communications for an EJB

You can control the attributes of the network connection an EJB uses for communications by configuring a custom network channel and assigning it to the EJB. For information about WebLogic Server network channels and associated configuration instructions see Configure Network Resources in *Administering Server Environments for Oracle WebLogic Server*. After you configure a custom channel, assign it to an EJB using the `network-access-point` element in `weblogic-ejb-jar.xml`.

Programming and Configuring Transactions

Transaction design decisions are discussed in Transaction Design and Management Options. The following sections contain guidelines for programming transactions.

For information using transactions with entity beans, see Understanding ejbLoad() and ejbStore() Behavior.

Programming Container-Managed Transactions

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

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

Key programming guidelines for container-managed transactions include:

- Preserve transaction boundaries—Do not invoke methods that interfere with the transaction boundaries set by the container. Do not use:
  - The `commit`, `setAutoCommit`, and `rollback` methods of `java.sql.Connection`
  - The `getUserTransaction` method of `javax.ejb.EJBContext`
  - Any method of `javax.transaction.UserTransaction`
- Roll back transactions explicitly—To cause the container to roll back a container-managed transaction explicitly, invoke the `setRollbackOnly` method of the `EJBContext` interface. (If the bean throws an application exception, typically an `EJBException`, the rollback is automatic.)
- Avoid serialization problems—Many data stores provide limited support for detecting serialization problems, even for a single user connection. In such cases, even with transaction-isolation in `weblogic-ejb-jar.xml` set to `TransactionSerializable`, exceptions or rollbacks in the EJB client might occur if contention occurs between clients for the same rows. To avoid such exceptions, you can:
  - Include code in your client application to catch SQL exceptions, and resolve them appropriately; for example, by restarting the transaction.
  - For Oracle databases, use the transaction isolation settings described in isolation-level.

Configuring Automatic Retry of Container-Managed Transactions

In this release of WebLogic Server, you can specify that, if a business method that has started a transaction fails because of a transaction rollback that is not related to a system exception, the EJB container will start a new transaction and retry the failed method up to a specified number of times. If the method fails for the specified number of retry attempts, the EJB container throws an exception.

> **Note:**
>
> The EJB container does not retry any transactions that fail because of system exception-based errors.

To configure automatic retry of container-managed transactions:

1. Make sure your bean is a container-managed session or entity bean.

   You can configure automatic retry of container-managed transactions for container-managed session and entity beans only. You cannot configure automatic retry of container-managed transactions for message-driven beans because MDBs do not acknowledge receipt of a message they are processing when the transaction that brackets the receipt of the message is rolled back; messages are automatically retried until they are acknowledged. You also cannot configure automatic retry of container-managed transactions for timer beans because, when a timer bean's ejbTimeout method starts and is rolled back, the timeout is always retried.

2. Make sure the business methods for which you want to configure automatic retry of transactions are defined in the bean's remote or local interface or as home methods (local home business logic that is not specific to a particular bean instance) in the home interface; the methods must have one of the following container-managed transaction attributes:

   - `RequiresNew`. If a method's transaction attribute (`trans-attribute` element in `ejb-jar.xml`) is `RequiresNew`, a new transaction is always started prior to the invocation of the method and, if configured, automatic retry of transactions occurs if the transaction fails.
   - `Required`. If a method's transaction attribute (`trans-attribute` element in `ejb-jar.xml`) is `Required`, the method is retried with a new transaction only if the failed transaction was begun on behalf of the method.

   For more information on:

   - Programming interfaces, see Create EJB Classes and Interfaces.
   - The `trans-attribute` element in `ejb-jar.xml`, see `trans-attribute` in Container-Managed Transactions Elements.

3. Make sure the methods for which you want to enable automatic retry of transactions are safe to be re-invoked. A retry of a failed method must yield results that are identical to the results the previous attempt, had it been successful, would have yielded. In particular:

   - If invoking a method initiates a call chain, it must be safe to reinvoke the entire call chain when the method is retried.
   - All of the method's parameters must be safe for reuse; when a method is retried, it is retried with the same parameters that were used to invoke the failed attempt. In

general, parameters that are primitives, immutable objects, or are references to read-only objects are safe for reuse. If a parameter is a reference to an object that is to be modified by the method, reinvoking the method must not negatively affect the result of the method call.

- If the bean that contains the method that is being retried is a stateful session bean, the bean's conversational state must be safe to re-invoke. Since a stateful session bean's state is not transactional and is not restored during a transaction rollback, in order to use the automatic retry of transactions feature, you must first be sure the bean's state is still valid after a rollback.

4. Specify the methods for which you want the EJB container to automatically retry transactions and the number of retry attempts you want the EJB container to make in the `retry-methods-on-rollback` element in `weblogic-ejb-jar.xml`.

Programming Bean-Managed Transactions

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

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

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

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

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

  Once you create a database connection within a transaction context, that connection is reserved until the transaction commits or rolls back. To optimize performance and throughput, ensure that transactions complete quickly, so that the database connection can be released and made available to other client requests.

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

- Setting transaction isolation level—For bean-managed transactions, you define isolation level in the bean code. Allowable isolation levels are defined on the java.sql.Connection interface. For information on isolation level behaviors, see isolation-level.

  See Example 3-3 for a code sample.

- Avoid restricted methods—Do not invoke the `getRollbackOnly` and `setRollbackOnly` methods of the `EJBContext` interface in bean-managed transactions. These methods should be used only in container-managed transactions. For bean-managed transactions, invoke the `getStatus` and rollback methods of the `UserTransaction` interface.

- Use one connection per active transaction context—You can associate only a single database connection with an active transaction context.

Example 3-3 Setting Transaction Isolation Level in BMT

```
import javax.transaction.Transaction;
import java.sql.Connection
import weblogic.transaction.TxHelper:
import weblogic.transaction.Transaction;
import weblogic.transaction.TxConstants;
User Transaction tx = (UserTransaction)
ctx.lookup("javax.transaction.UserTransaction");
//Begin user transaction
    tx.begin();
//Set transaction isolation level to TransactionReadCommitted
Transaction tx = TxHelper.getTransaction();
    tx.setProperty (TxConstants.ISOLATION_LEVEL, new Integer
    (Connection.TransactionReadCommitted));
//perform transaction work
    tx.commit();
```

Programming Transactions That Are Distributed Across EJBs

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

- Calling multiple EJBs from a client's transaction context
- Using an EJB "Wrapper" to Encapsulate a Cross-EJB Transaction

Calling multiple EJBs from a client's transaction context

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

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

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

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

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

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

The wrapper EJB can explicitly obtain a transaction context before invoking other EJBs, or WebLogic Server can automatically create a new transaction context, if the wrapper's `trans-attribute` element in `ejb-jar.xml` is set to `Required` or `RequiresNew`.

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

Programming the EJB Timer Service

WebLogic Server supports the EJB timer service defined in the EJB 2.1 Specification and EJB 3.2 Specification. The EJB timer service is an EJB-container-provided service that allows you to create timers that schedule callbacks to occur when a timer object expires.
For more information about EJB timer services, see Chapter 13 - Timer Service in the Jakarta Enterprise Bean 3.2 specifications document. You can download this document from the Community Development of Java Technology Specifications page.

Timer objects can be created for entity beans, message-driven beans, and stateless session beans. Timer objects expire at a specified time, after an elapsed period of time, or at specified intervals. For instance, you can use the timer service to send out notification when an EJB remains in a certain state for an elapsed period of time.

The WebLogic EJB timer service is intended to be used as a coarse-grained timer service. Rather than having a large number of timer objects performing the same task on a unique set of data, Oracle recommends using a small number of timers that perform bulk tasks on the data. For example, assume you have an EJB that represents an employee's expense report. Each expense report must be approved by a manager before it can be processed. You could use one EJB timer to periodically inspect all pending expense reports and send an email to the corresponding manager to remind them to either approve or reject the reports that are waiting for their approval.

Clustered Versus Local EJB Timer Services

You can configure two types of EJB timer services: clustered or local.

Clustered EJB Timer Services

Clustered EJB timer services provide the following advantages:

- Better visibility.

  Timers are accessible from any node in a cluster. For example, the `javax.ejb.TimerService.getTimers()` method returns a complete list of all stateless session or message-driven bean timers in a cluster that were created for the EJB. If you

pass the primary key of the entity bean to the `getTimers()` method, a list of timers for that entity bean are returned.

- Automatic load balancing and failover.

  Clustered EJB timer services take advantage of the load balancing and failover capabilities of the Job Scheduler.

For information about the configuring a clustered EJB timer service, see Configuring Clustered EJB Timers.

Local EJB Timer Services

Local EJB timer services execute only on the server on which they are created and are visible only to the beans on that server. With a local EJB timer service, you do not have to configure a cluster, database, JDBC data source, or leasing service, as you do for clustered EJB timer services.

You cannot migrate a local EJB timer object from one server to another; timer objects can only be migrated as part of an entire server. If a server that contains EJB timers goes down for any reason, you must restart the server or migrate the entire server in order for the timers to execute.

Enterprise bean timers are of two types: programmatic timers and automatic timers.

Using Java Programming Interfaces to Program Timer Objects

This section summarizes the Java programming interfaces defined in the EJB 2.1 Specification that you can use to program timers. For detailed information on these interfaces, refer to the EJB 2.1 Specification. This section also provides details about the WebLogic Server-specific timer-related interfaces.

EJB 2.1 Timer-related Programming Interfaces

EJB 2.1 interfaces you can use to program timers are described in the following table.

Table 3-2 EJB 2.1 Timer-related Programming Interfaces

Programming Interface	Description
`javax.ejb.TimedObject`	Implement for the enterprise bean class of a bean that will be registered with the timer service for timer callbacks. This interface has a single method, `ejbTimeout`.
`EJBContext`	Access the timer service using the `getTimerService` method.
`javax.ejb.TimerService`	Create new EJB timers or access existing EJB timers for the EJB.
`javax.ejb.Timer`	Access information about a particular EJB timer.
`javax.ejb.TimerHandle`	Define a serializable timer handle that can be persisted. Since timers are local objects, a `TimerHandle` must not be passed through a bean's remote interface or Web service interface.

For more information on EJB 2.1 timer-related programming interfaces, see the EJB 2.1 Specification.

WebLogic Server-specific Timer-related Programming Interfaces

WebLogic Server-specific interfaces you can use to program timers include:

- `weblogic.management.runtime.EJBTimerRuntimeMBean`—provides runtime information and administrative functionality for timers from a particular `EJBHome`. The `weblogic.management.runtime.EJBTimerRuntimeMBean` interface is shown in Example 3-4.
- `weblogic.ejb.WLTimerService` interface—extends the `javax.ejb.TimerService` interface to allow users to specify WebLogic Server-specific configuration information for a timer. The `weblogic.ejb.WLTimerService` interface is shown in Example 3-5; for information on the `javax.ejb.TimerService`, see the EJB 2.1 Specification.

> **Note:**
>
> The `weblogic.ejb.WLTimerService` interface is not supported by the clustered EJB timer service, as described in Configuring Clustered EJB Timers.

- `weblogic.ejb.WLTimerInfo` interface—used in the `weblogic.ejb.WLTimerService` interface to pass WebLogic Server-specific configuration information for a timer. The `weblogic.ejb.WLTimerInfo` method is shown in Example 3-6.

> **Note:**
>
> The `weblogic.ejb.WLTimerService` interface is not supported by the clustered EJB timer service, as described in Configuring Clustered EJB Timers.

- `weblogic.ejb.WLTimer` interface—extends the `javax.ejb.Timer` interface to provide additional information about the current state of the timer. The `weblogic.ejb.WLTimer` interface is shown in Example 3-7.

> **Note:**
>
> The `weblogic.ejb.WLTimerService` interface is not supported by the clustered EJB timer service, as described in Configuring Clustered EJB Timers.

Example 3-4 weblogic.management.runtime.EJBTimerRuntimeMBean Interface

```
public interface weblogic.management.runtime.EJBTimerRuntimeMBean {
  public int getTimeoutCount(); // get the number of successful timeout notifications
that have been made
  public int getActiveTimerCount(); // get the number of active timers for this EJBHome
  public int getCancelledTimerCount(); // get the number of timers that have been
cancelled for this EJBHome
  public int getDisabledTimerCount(); // get the number of timers temporarily disabled
for this EJBHome
  public void activateDisabledTimers(); // activate any temporarily disabled timers
}
```

Example 3-5 weblogic.ejb.WLTimerService Interface

```
public interface WLTimerService extends TimerService {
  public Timer createTimer(Date initial, long duration, Serializable info,
    WLTimerInfo wlTimerInfo)
    throws IllegalArgumentException, IllegalStateException, EJBException;
  public Timer createTimer(Date expiration, Serializable info,
    WLTimerInfo wlTimerInfo)
    throws IllegalArgumentException, IllegalStateException, EJBException;
  public Timer createTimer(long initial, long duration, Serializable info
    WLTimerInfo wlTimerInfo)
    throws IllegalArgumentException, IllegalStateException, EJBException;
  public Timer createTimer(long duration, Serializable info,
    WLTimerInfo wlTimerInfo)
    throws IllegalArgumentException, IllegalStateException, EJBException;
}
```

Example 3-6 weblogic.ejb.WLTimerInfo Interface

```
public final interface WLTimerInfo {
  public static int REMOVE_TIMER_ACTION  = 1;
  public static int DISABLE_TIMER_ACTION = 2;
  public static int SKIP_TIMEOUT_ACTION  = 3;
  /**
   * Sets the maximum number of retry attempts that will be
   * performed for this timer.  If all retry attempts
   * are unsuccesful, the timeout failure action will
   * be executed.
   */
  public void setMaxRetryAttempts(int retries);
  public int getMaxRetryAttempts();
  /**
   * Sets the number of milliseconds that should elapse
   * before any retry attempts are made.
   */
  public void setRetryDelay(long millis);
  public long getRetryDelay();
  /**
   * Sets the maximum number of timeouts that can occur
   * for this timer.  After the specified number of
   * timeouts have occurred successfully, the timer
   * will be removed.
   */
  public void setMaxTimeouts(int max);
  public int getMaxTimeouts();
/**
   * Sets the action the container will take when ejbTimeout
   * and all retry attempts fail.  The REMOVE_TIMER_ACTION,
   * DISABLE_TIMER_ACTION, and SKIP_TIMEOUT_ACTION fields
   * of this interface define the possible values.
   */
  public void setTimeoutFailureAction(int action);
  public int getTimeoutFailureAction();
}
```

Example 3-7 weblogic.ejb.WLTimer Interface

```
public interface WLTimer extends Timer {
  public int getRetryAttemptCount();
  public int getMaximumRetryAttempts();
  public int getCompletedTimeoutCount();
}
```

Timer Deployment Descriptors

The following deployment descriptor elements pertain to timers.

Table 3-3 Timer Deployment Descriptors

Element	Description
timer-descriptor	EJB timer object.
timer-implementation	Whether the EJB timer service is clustered or non-clustered. For information about the clustered EJB timer service, see Configuring Clustered EJB Timers.
persistent-store-logical-name	Name of a persistent store on the server's file system where WebLogic Server stores timer objects.

For more information on these elements, see weblogic-ejb-jar.xml Deployment Descriptor Reference.

Configuring Clustered EJB Timers

> **Note:**
>
> To review the advantages of using clustered EJB timers, see Clustered Versus Local EJB Timer Services.

To configure the clustering of EJB timers, perform the following steps:

1. Ensure that you have configured the following:
   - A clustered domain. For more information, see Setting up WebLogic Clusters in *Administering Clusters for Oracle WebLogic Server*.
   - Features of the Job Scheduler, including:
     - HA database, such as Oracle, DB2, Informix, MySQL, Sybase, or MSSQL.
     - JDBC data source that is mapped to the HA database using the `<data-source-for-job-scheduler>` element in the `config.xml` file.
     - Leasing service. By default, database leasing will be used and the JDBC data source defined by the `<data-source-for-job-scheduler>` element in the `config.xml` file will be used.

   For more information about configuring the Job Scheduler, see "The Timer and Work Manager API" in *Developing CommonJ Applications for Oracle WebLogic Server*.
2. To enable the clustered EJB timer service, set the `timer-implementation` element in the `weblogic-ejb-jar.xml` deployment descriptor to `Clustered`:

   ```
   <timer-implementation>Clustered</timer-implementation>
   ```

   For more information, see timer-implementation.

Please note the following changes in the behavior of the clustered EJB timer service:

- The `weblogic.ejb.WLTimer*` interfaces are not supported with clustered EJB timer services.
- When creating a new clustered EJB timer using the `createTimer()` method, you may notice a delay in timeout execution during the initial setup of the timer.
- The Job Scheduler provides an "at least once" execution guarantee. When a clustered EJB timer expires, the database is not updated until the timer listener callback method completes. If the server were to crash before the database is updated, the timer expiration would be executed twice.
- Timer configuration options related to the actions to take in the event of failure are not valid for the clustered EJB timer service. These configuration options include: retry delay, maximum number of retry attempts, maximum number of timeouts, and timeout failure actions.
- The Job Scheduler queries the database every 30 seconds to identify timers that are due to expire. Execution may be delayed for timers with an interval duration less than 30 seconds.
- Only transactional timers will be retried in the event of failure.
- Fixed rate scheduling of timer execution is not supported.

Declare Web Service References

Web Service references, declared in an EJB's deployment descriptor, maps a logical name for a Web Service to an actual Web Service interface, which allows you to refer to the Web Service using a logical name. The bean code then performs a JNDI lookup using the Web Service reference name.

This release of WebLogic Server complies with the EJB 2.1 requirements related to declaring and accessing external Web Services.

See *Developing JAX-WS Web Services for Oracle WebLogic Server*.

Compile Java Source

Learn about the tools that support compilation and the compilation process.

For a list of tools that support the compilation process, see Table 3-1.

For information on the compilation process, see *Developing Applications for Oracle WebLogic Server*.

Edit Deployment Descriptors

Elements in `ejb-jar.xml`, `weblogic-ejb-jar.xml`, and for container-managed persistence entity beans, `weblogic-cmp-jar.xml`, control the run-time characteristics of your application. You can modify these deployment descriptors using an XML editing tool.

If you need to modify a descriptor element, you can edit the descriptor file with any plain text editor. However, to avoid introducing errors, use a tool designed for XML editing.

The following sections are a quick reference to WebLogic Server-specific deployment elements. Each section contains the elements related to a type of feature or behavior. The table in each section defines relevant elements in terms of the behavior it controls, the bean type it relates to (if bean type-specific), the parent element in `weblogic-ejb-jar.xml` that

contains the element, and the behavior you can expect if you do not explicitly specify the element in `weblogic-ejb-jar.xml`.

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

- weblogic-ejb-jar.xml Deployment Descriptor Reference
- weblogic-cmp-jar.xml Deployment Descriptor Reference
- Your Sun documentation for elements in `ejb-jar.xml`.

> **Note:**
>
> In the sections that follow, click the element name in the "Element" column to view detailed documentation on the element.

- Security Elements
- Resource Mapping Elements
- Persistence Elements
- Clustering Elements
- Data Consistency Elements
- Container-Managed Transactions Elements
- Performance Elements
- Network Communications Elements

Security Elements

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

Table 3-4 Security Elements in weblogic-ejb-jar.xml

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

Resource Mapping Elements

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

Table 3-5 Resource Mapping Elements in weblogic-ejb-jar.xml

Element	Bean Type	Description	Default
jndi-name	All	JNDI name of a resource or reference available in WebLogic Server. **Note**: Assigning a JNDI name to a bean is not recommended. Global JNDI names generate heavy multicast traffic during clustered server startup. See Using EJB Links for the better practice.	none
local-jndi-name	All	JNDI name for a bean's local home. If a bean has both a remote and a local home, then it must have two JNDI names; one for each home.	none
concurrency-strategy	MDB	JNDI name of the JMS connection factory that the bean uses to create queues and topics.	`weblogic.jms.Message.DrivenBeanConnectionFactory`
destination-jndi-name	MDB	JNDI name that associates a message-driven bean with a queue or topic in the JNDI tree.	none
initial-context-factory	MDB	Initial context factory that the EJB container uses to create connection factories.	`weblogic.jndi.WLInitial.Context.Factory`
jms-client-id	MDB	Client ID for the message-driven bean associated with a durable subscriber topic.	Value of ejb-name
message-destination-descriptor	MDB	`Maps a message destination reference in the ejb-jar.xml` file to an actual message destination, such as a JMS Queue or Topic, in WebLogic Server.	n/a
provider-url	MDB	Specifies the URL provider to be used by the `InitialContext`.	`t3://localhost:7001`

Persistence Elements

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

Table 3-6 Persistence Elements in weblogic-ejb-jar.xml

Element	Bean Type	Description	Default
type-identifier	Entity	Specifies EJB persistence type. WebLogic Server RDBMS-based persistence uses the identifier, `WebLogic_CMP_RDBMS`	n/a
type-storage	Entity	Defines path, relative to the top level of the EJB's JAR deployment file or deployment directory, of the file that stores data for this persistence type. WebLogic Server RDBMS-based persistence generally uses an XML file named `weblogic-cmp-jar.xml` to store persistence data for a bean. This file is stored in the `META-INF` subdirectory of the JAR file.	n/a

Table 3-6 (Cont.) Persistence Elements in weblogic-ejb-jar.xml

Element	Bean Type	Description	Default
type-version	Entity	Version of the persistence type specified by type-identifier. For WebLogic 2.0 CMP persistence, use the value `2.0`. For WebLogic 1.1 CMP persistence, use the value 1.1.	n/a
delay-updates-until-end-of-tx	Entity	If true, the EJB container attempts to delay writing updates to a bean's state to the database until the end of a transaction. However, the container still flushes updates to the database before executing an EJB finder or select query if the `include-updates` element (in the `weblogic-query` element of `weblogic-cmp-jar.xml`) for the query is true. Applicable to both container-managed persistence and bean-managed persistence beans.	True
finders-load-bean	Entity	Causes beans returned by a `finder` or `ejbSelect` method to be loaded immediately into the cache before the method returns. **Note**: Applicable to container-managed persistence beans only.	True
persistent-store-dir	Stateful Session	Directory where state of passivated stateful session bean instances is stored.	`pstore`
is-modified-method-name	Entity	The method called by the container to determine whether or not the bean has been modified and needs to have its changes written to the database. Applies to bean-managed persistence or EJB 1.1 container-managed persistence beans.	If not specified, bean state is persisted after each method completes.

Clustering Elements

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

Table 3-7 Clustering Elements in weblogic-ejb-jar.xml

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

Table 3-7 (Cont.) Clustering Elements in weblogic-ejb-jar.xml

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

Data Consistency Elements

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

> **Note:**
>
> For elements related to container-managed persistence, see Managing Entity Bean Pooling and Caching .

Table 3-8 Data Consistency Elements in weblogic-ejb-jar.xml

Element	Bean Type	Description	Default
concurrency-strategy	Entity	How concurrent access to an entity bean is managed.	Database
invalidation-target	Entity	The read-only entity bean to invalidate when this container-managed persistence entity bean is modified. **Note**: Only applicable to EJB 2.x CMP beans.	None
delay-updates-until-end-of-tx	Entity	If true, the EJB container attempts to delay writing updates to a bean's state to the database until the end of a transaction. However, the container still flushes updates to the database before executing an EJB finder or select query if the `include-updates` element (in the `weblogic-query` element of `weblogic-cmp-jar.xml)` for the query is `true`. Applicable to both container-managed persistence and bean-managed persistence beans.	True

Container-Managed Transactions Elements

Table 3-9 lists the elements in `ejb-jar.xml` related to container-managed transactions.

Table 3-9 Container-Managed Transaction Elements in ejb-jar.xml

Element	Description	Default
`transaction-type`	Allowable values are `Bean` or `Container`.	None, EJB 2.x requires this attribute to be specified.

Table 3-9 (Cont.) Container-Managed Transaction Elements in ejb-jar.xml

Element	Description	Default
`trans-attribute`	Specifies how the container manages the transaction boundaries when delegating a method invocation to an enterprise bean's business method. Allowable values are: • `NotSupported` With the `NotSupported` value, when an entity bean runs in an unspecified transaction, if a transaction exists, the EJB container suspends the transaction; when an entity bean runs in an unspecified transaction, and no transaction exists, the EJB container takes no action. • `Supports` With the `Supports` value, when an entity bean runs in an unspecified transaction, if a transaction exists, the EJB container uses the current transaction; when an entity bean runs in an unspecified transaction, and no transaction exists, the EJB container takes no action. • `Required` • `RequiresNew` • `Mandatory` • `Never` With the `Never` value, when an entity bean runs in an unspecified transaction, if a transaction exists, the EJB container throws an exception; when an entity bean runs in an unspecified transaction, and no transaction exists, the EJB container takes no action. **Note**: In pre-9.0 releases of WebLogic Server, the EJB container would start a new transaction when no transaction existed and the value of `trans-attribute` was `NotSupported`, `Supports`, and `Never`. Set `entity-always-uses-transaction` in `weblogic-ejb-jar.xml` to `True` if you want the EJB container to behave as it did in pre-9.0 releases of WebLogic Server and create a new transaction. Because clients do not provide a transaction context for calls to an MDB, MDBs that use `container-managed transactions must have trans-attribute of Required`.	If not specified, the EJB container issues a warning, and uses `NotSupported` for MDBs and `Supports` for other types of EJBs.
`transaction-scope`	This optional element specifies whether an enterprise bean requires distributed transactions for its methods or whether the local transaction optimization may be used. Allowable values are `Local` and `Distributed`.	If not specified, the container assumes that distributed transactions must be used.

Table 3-10 lists the elements in `weblogic-ejb-jar.xml` related to container-managed transactions.

Table 3-10 Container-Managed Transaction Elements in weblogic-ejb-jar.xml

Element	Description	Default
retry-methods-on-rollback	The methods for which you want the EJB container to automatically retry container-managed transactions that have rolled back. **Note**: Regardless of the methods specified in this element, the EJB container does not retry any transactions that fail because of system exception-based errors.	None

Table 3-10 (Cont.) Container-Managed Transaction Elements in weblogic-ejb-jar.xml

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

Performance Elements

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

Table 3-11 Performance Elements in weblogic-ejb-jar.xml

Element	Bean Type	Description	Default
allow-concurrent-calls	Stateful Session	Whether multiple clients can simultaneously access a bean without triggering a `Remote Exception`. The server throws a `RemoteException` when a stateful session bean instance is currently handling a method call and another (concurrent) method call arrives on the server.	False
cache-between-transactions	Entity	Causes the container to cache the persistent data of an entity bean between transactions.	False
cache-type	Stateful Session	Order in which stateful session beans are removed from the cache.	NRU (not recently used)
clients-on-same-server	All	Indicates that all clients of the bean are collocated with the bean on the same server instance. This element is only used if the EJB has a global JNDI name; setting it to `true` prevents the JNDI name from being replicated. A value of `true` can reduce cluster startup time in large clusters.	False
delay-updates-until-end-of-tx	Entity	If `true`, the EJB container attempts to delay writing updates to a bean's state to the database until the end of a transaction. However, the container still flushes updates to the database before executing an EJB finder or select query if the `include-updates` element (in the `weblogic-query` element of `weblogic-cmp-jar.xml`) for the query is `true`. Applicable to both container-managed persistence and bean-managed persistence beans.	True
dispatch-policy	All	Specifies the thread pool used to handle requests to the bean.	None

Table 3-11 (Cont.) Performance Elements in weblogic-ejb-jar.xml

Element	Bean Type	Description	Default
entity-cache-name	Entity	The application-level entity cache, which can cache instances of multiple entity beans that are part of the same application. **Note**: Application level caches are declared in the `weblogic-application.xml`.	None
estimated-bean-size	Entity	Estimated average size, in bytes, of an entity bean instance.	None
finders-load-bean	Entity	Causes beans returned by a `finder` or `ejbSelect` method to be loaded immediately into the cache before the method returns. **Note**: Applicable to container-managed persistence beans only.	True
idle-timeout-seconds	Entity	Number of seconds of inactivity after which a bean is passivated. **Note**: This element is not currently used.	600
idle-timeout-seconds	Stateful Session	Number of seconds of inactivity after which a bean is passivated.	600
initial-beans-in-free-pool	Entity Message-Driven Stateless Session	Number of instances of an EJB instantiated by the container at startup.	0
is-modified-method-name	Entity	The method that changes the state of bean. Specifying this method causes WebLogic server to persist the bean state when the method completes. **Note**: Applies to bean-managed persistence or EJB 1.1 container-managed persistence beans.	If not specified, bean state is persisted after each method completes.
jms-polling-interval-seconds	Message-driven	The number of seconds between attempts by the EJB container to reconnect to a JMS destination that has become unavailable.	10
max-beans-in-cache	Entity Stateful Session	Maximum number of instances in the cache.	1000
max-beans-in-free-pool	Entity Stateless Session Message-Driven	Maximum number of instances in the free pool.	1000
read-timeout-seconds	Entity	The number of seconds between `ejbLoad` calls on a **`read-only`** entity bean. If `read-timeout-seconds` is 0, `ejbLoad` is only called when the bean is brought into the cache.	600

Network Communications Elements

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

Table 3-12 Communications Elements in weblogic-ejb-jar.xml

Element	Bean Type	Description	Default
network-access-point	all	Assigns a custom network channel to an EJB.	n/a

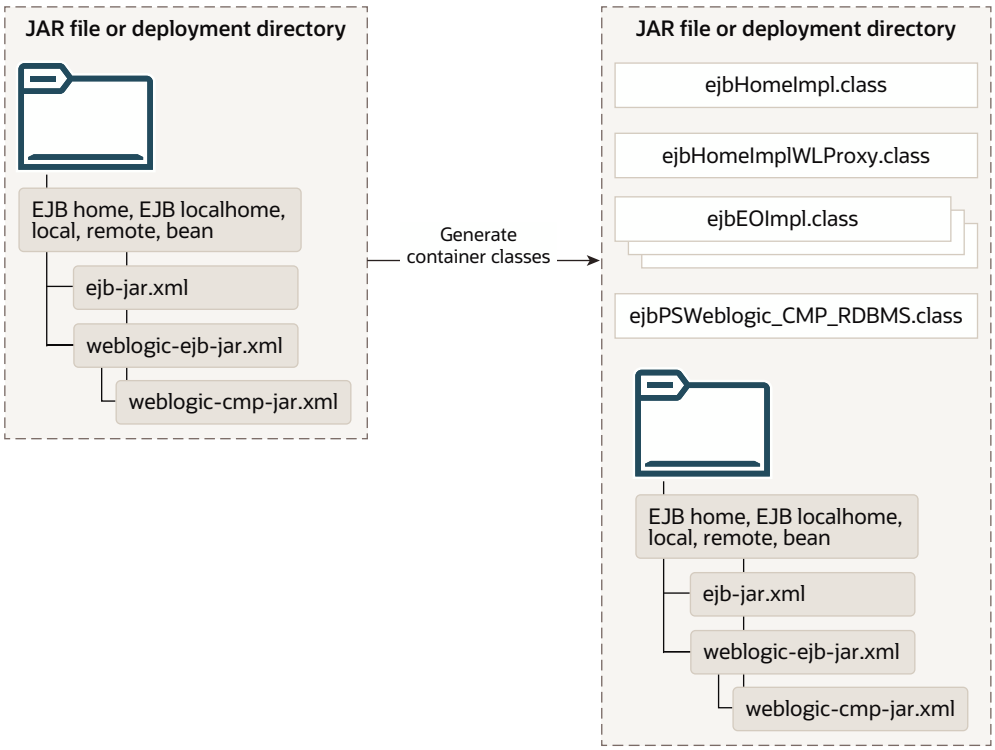
Generate EJB Wrapper Classes, and Stub and Skeleton Files

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

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

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

Figure 3-2 Generating EJB Container Classes



- appc and Generated Class Name Collisions

appc and Generated Class Name Collisions

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

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

Package

You can package EJBs as part of an enterprise application.

Oracle recommends that you package EJBs as part of an enterprise application. See Deploying and Packaging from a Split Development Directory in *Developing Applications for Oracle WebLogic Server*.

Packaging Considerations for EJBs with Clients in Other Applications

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

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

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

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

Deploy

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

For general instructions on deploying WebLogic Server applications and modules, including EJBs, see *Deploying Applications to Oracle WebLogic Server*. For EJB-specific deployment issues and procedures, see Deployment Guidelines for EJBs in this book — *Developing Enterprise JavaBeans, Version 3.2, for Oracle WebLogic Server*.

Solving Problems During Development

Learn about the WebLogic Server features that are useful for checking out and debugging deployed EJBs.

The following sections describe the debugging features in WebLogic Server.

- Adding Line Numbers to Class Files
- Creating Debug Messages

Adding Line Numbers to Class Files

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

Creating Debug Messages

For instructions on how to create messages in your application to help you troubleshoot and solve bugs and problems, see *Configuring Log Files and Filtering Log Messages for Oracle WebLogic Server*.

WebLogic Server Tools for Developing EJBs

Examine the Oracle tools that support the EJB development process.

For a comparison of the features available in each tool, see Table 3-13.

- Oracle JDeveloper
- Oracle Enterprise Pack for Eclipse
- javac
- DDInit
- WebLogic Server Ant Utilities
- weblogic.Deployer
- appc
- DDConverter
- Comparison of EJB Tool Features

Oracle JDeveloper

Oracle JDeveloper is a full-featured Java IDE that can be used for end-to-end development of EJBs. For more information, see the Oracle JDeveloper online help. For information about installing JDeveloper, see *Installing Oracle JDeveloper*.

Oracle Enterprise Pack for Eclipse

Oracle Enterprise Eclipse (OEPE) provides a collection of plug-ins to the Eclipse IDE platform that facilitate development of WebLogic Web services. For more information, see the Eclipse IDE platform online help.

javac

The `javac` compiler provided with the Java SE JDK provides java compilation capabilities. For information on `javac`, see `http://www.oracle.com/technetwork/java/index-jsp-142903.html#documentation`.

DDInit

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

See DDInit in *Command Reference for Oracle WebLogic Server*.

WebLogic Server Ant Utilities

WebLogic Server includes Ant utilities to create skeleton deployment descriptors.

The Ant task examines a directory containing an EJB and creates deployment descriptors based on the directory contents. Because the Ant utility does not have information about all desired configurations and mappings for your EJB, the skeleton deployment descriptors the utility creates are incomplete. After the utility creates the skeleton deployment descriptors, you can use a text editor or an XML editor to edit the deployment descriptors and complete the configuration of your EJB.

See Deploying Applications Using wldeploy in *Developing Applications for Oracle WebLogic Server*.

weblogic.Deployer

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

See weblogic.Deployer Command-Line Reference in *Deploying Applications to Oracle WebLogic Server*.

appc

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

> **Note:**
>
> The `appc` compiler replaces the deprecated `ejbc` utility. Therefore, Oracle recommends that you use `appc` instead of the deprecated `ejbc`.

See appc Reference.

DDConverter

`DDConverter` is a command line tool that upgrades deployment descriptors from earlier releases of WebLogic Server. Oracle recommends that you always upgrade your deployment descriptors in order to take advantage of the features in the current Jakarta EE specification and release of WebLogic Server.

You can use weblogic.DDConverter to upgrade your deployment descriptors. For information on using weblogic.DDConverter, see *Developing Applications for Oracle WebLogic Server*.

> **Note:**
>
> With this release of WebLogic Server, the EJB-specific DDConverter, `weblogic.ejb20.utils.DDConverter`, is deprecated. Instead, use the new application-level DDConverter, `weblogic.DDConverter`, to convert your application's deployment descriptors, including the EJB-specific deployment descriptors.

Comparison of EJB Tool Features

The following table lists Oracle tools for EJB development, and the features provided by each. **Yes** indicates the tool contains the corresponding feature.

Table 3-13 EJB Tools and Features

EJB Tool	Generate Interfaces and Home Interfaces	Compile Java Code	Generate Deployment Descriptors	View and Edit Deployment Descriptors	Deploy
WebLogic Workshop	**Yes**	**Yes**	**Yes**	No	**Yes**
appc	No	**Yes**	No	No	No
javac	No	**Yes**	No	No	No
DDinit	No	No	**Yes**	No	No
Deployer	No	No	No	No	**Yes**
DDConverter	No	No	**Yes**	No	No

4

Session EJBs

Learn how session beans work within the EJB container, and provides design and development guidelines that are specific to session beans.
For a description of the overall bean development process, see Implementing EJBs.

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

This chapter includes the following topics:

- Comparing Stateless and Stateful Session Beans
  Examine the differences between stateless and stateful session beans.
- Pooling for Stateless Session EJBs
  Learn how WebLogic Server initializes new instances of the EJB.
- Caching and Passivating Stateful Session EJBs
  WebLogic Server uses a cache of bean instances to improve the performance of stateful session EJBs. The cache stores active EJB instances in memory so that they are immediately available for client requests. The cache contains EJBs that are currently in use by a client and instances that were recently in use. Stateful session beans in cache are bound to a particular client.
- Design Decisions for Session Beans
  Examine some design decisions relevant to session beans.
- Implementing Session Beans
  You can configure WebLogic Server-specific session bean behavior by setting bean-specific deployment descriptor elements.

Comparing Stateless and Stateful Session Beans

Examine the differences between stateless and stateful session beans.

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

Table 4-1 Comparing Stateless and Stateful Session Beans

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

Table 4-1 (Cont.) Comparing Stateless and Stateful Session Beans

Stateless Session Beans	Stateful Sessions Beans
Do not persist. The bean has no state between calls.	Persist. A stateful session bean's state is preserved for the duration of a session.

See Choosing Between Stateless and Stateful Beans for a discussion of when to use which type of session bean.

Pooling for Stateless Session EJBs

Learn how WebLogic Server initializes new instances of the EJB.

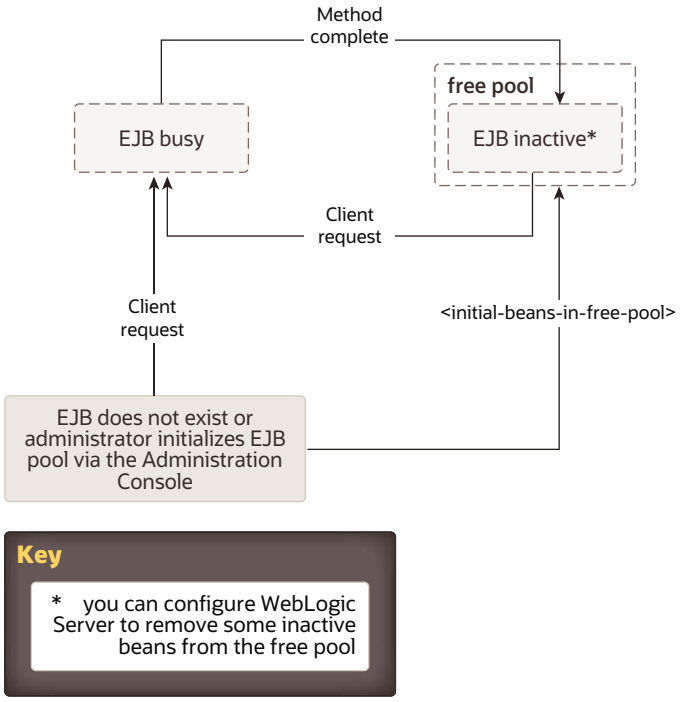
By default, no stateless session EJB instances exist in WebLogic Server at startup time. As individual beans are invoked, WebLogic Server initializes new instances of the EJB.

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

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

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

Figure 4-1 WebLogic Server Free Pool Showing Stateless Session EJB Life Cycle



If you configure a pool, WebLogic Server will service method calls with an EJB instance from the free pool, if one is available. The EJB remains active for the duration of the client's method call. After the method completes, the EJB instance is returned to the free pool. Because WebLogic Server unbinds stateless session beans from clients after each method call, the actual bean class instance that a client uses may be different from invocation to invocation.

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

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

You can configure the WebLogic Server to remove bean instances that have remained unused in the pool for a period of time specified in `idle-timeout-seconds` element in the `pool` element. When beans have been in the pool unused for the amount of time you specify in `idle-timeout-seconds`, they are removed from the pool until the number of beans in the pool reaches the number specified in `initial-beans-in-free-pool`; the number of beans in the pool will never fall below the number specified in `initial-beans-in-free-pool`.

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

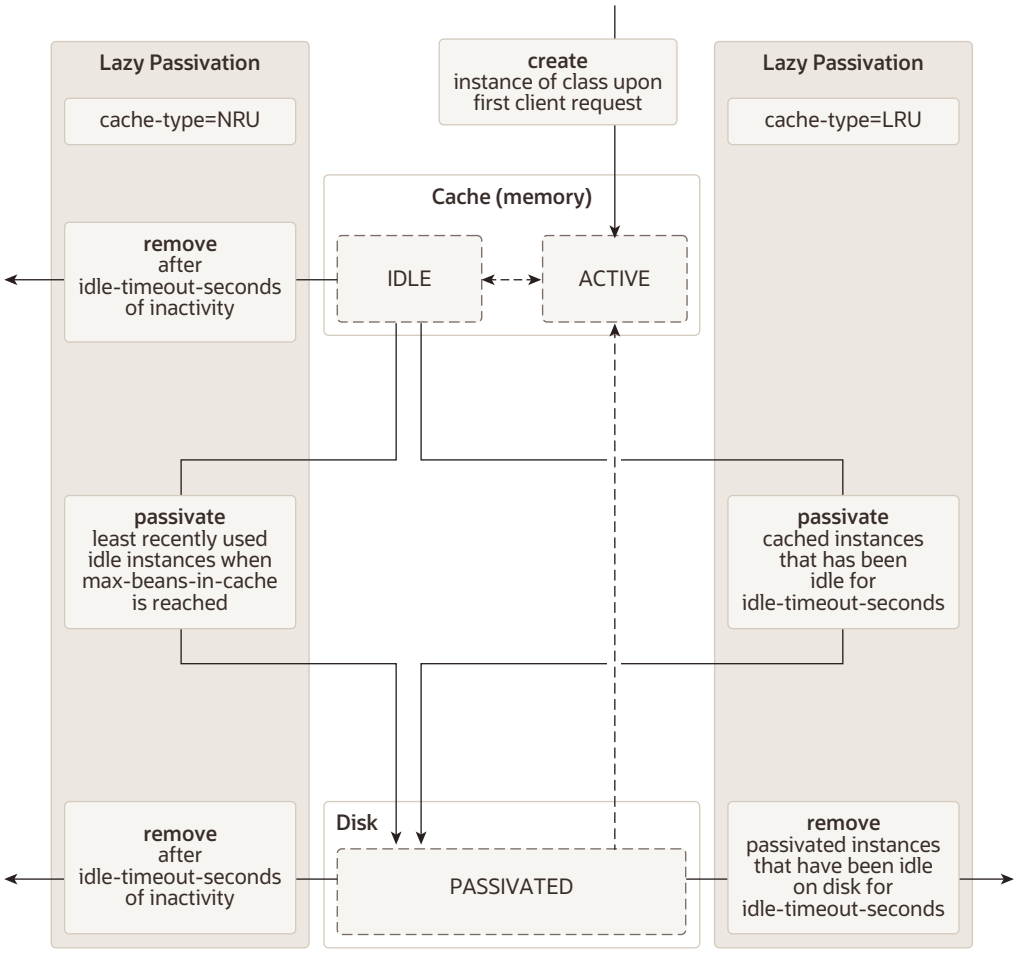
- An instance is available in the pool. WebLogic Server makes that instance available and your application proceeds with processing.
- No instance is available in the pool, but the number of instances in use is less than `max-beans-in-free-pool`. WebLogic Server allocates a new bean instance and gives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

Caching and Passivating Stateful Session EJBs

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

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

Figure 4-2 Stateful Session EJB Life Cycle



Stateful Session EJB Creation

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

Stateful Session EJB Passivation

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

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

Controlling Passivation

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

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

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

Eager Passivation (LRU)

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

- As soon as an instance has been inactive for `idle-timeout-seconds`, regardless of the value of `max-beans-in-cache`.
- When `max-beans-in-cache` is reached, even though `idle-timeout-seconds` has not expired.

The container removes a passivated instance from disk after it has been inactive for `idle-timeout-seconds` after passivation. This is referred to as a *lazy remove*.

> **Note:**
>
> `ejbRemove` may not be called on bean instances that are deleted after `idle-timeout-seconds` is reached.

Lazy Passivation (NRU)

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

When the cache is full and a bean instance must be removed from the cache to make room for another instance, the container:

> **Note:**
>
> `ejbRemove` may not be called on bean instances that are deleted after `idle-timeout-seconds` is reached.

- Removes the bean instance from the cache without passivating it to disk if `idle-timeout-seconds` has expired. This is referred to as a *eager remove*. An eager remove ensures that an inactive instance does not consume memory or disk resources.
- Passivates the bean instance to disk if `idle-timeout-seconds` has not expired.

Specifying the Persistent Store Directory for Passivated Beans

When a stateful session bean is passivated, its state is stored in a file system directory. Each server instance has its own directory for storing the state of passivated stateful session beans, known as the *persistent store directory*. The persistent store directory contains one subdirectory for each passivated bean.

The persistent store directory is created by default in the server instance directory, for example:

```
D:\releases\<version>\bea\user_domains\mydomain\myserver\tmp\pstore\
```

The path to the persistence store is:

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

where:

- `RootDirectory`—the directory where WebLogic Server runs.

  `RootDirectory` can be specified at server startup with the `-Dweblogic.RootDirectory` property.
- `ServerName`—the name of the server instance.
- `persistent-store-dir`—the value of the of the `persistent-store-dir` element in the `<stateful-session-descriptor>` element of `weblogic-ejb-jar.xml`. If no value is specified for `<persistent-store-dir>`, the directory is named `pstore` by default.

The persistent store directory contains a subdirectory, named with a hash code, for each passivated bean. For example, the subdirectory for a passivated bean in the example above might be:

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

Configuring Concurrent Access to Stateful Session Beans

In accordance with the EJB 2.x specification, simultaneous access to a stateful session EJB results in a `RemoteException`. This access restriction on stateful session EJBs applies whether the EJB client is remote or internal to WebLogic Server. To override this restriction and configure a stateful session bean to allow concurrent calls, set the `allow-concurrent-calls` deployment element.

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

Design Decisions for Session Beans

Examine some design decisions relevant to session beans.

This section discusses some of these designs in detail:

- Choosing Between Stateless and Stateful Beans
- Choosing the Optimal Free Pool Setting for Stateless Session Beans

Choosing Between Stateless and Stateful Beans

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

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

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

For examples of applications of stateless and stateful session beans, see Stateless Session Beans and Stateful Session Beans .

Choosing the Optimal Free Pool Setting for Stateless Session Beans

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

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

Implementing Session Beans

You can configure WebLogic Server-specific session bean behavior by setting bean-specific deployment descriptor elements.

Implementing EJBs takes you through session bean implementation step-by-step.

- WebLogic-Specific Configurable Behaviors for Session Beans

WebLogic-Specific Configurable Behaviors for Session Beans

Table 4-2 summarizes the deployment descriptor elements you set to configure the behavior of a *stateless* session bean and how the bean behaves if you do not configure the element. All of the elements listed are sub-elements of the `stateless-session-descriptor` element in `weblogic-ejb-jar.xml`.

Table 4-3 summarizes the deployment descriptor elements you set to configure the behavior of a *stateful* session bean and how the bean behaves if you do not configure the element. All of the elements listed are sub-elements of the `stateful-session-descriptor` element in `weblogic-ejb-jar.xml`.

Table 4-2 WebLogic-Specific Features for Stateless Session EJBs

To control	Set the following weblogic-ejb-jar.xml element	Default behavior
The number of inactive instances of a stateless session bean that exist in WebLogic Server when it is started. See Pooling for Stateless Session EJBs.	initial-beans-in-free-pool	WebLogic Server creates 0 beans in the free pool.
The number of seconds a bean can be idle in the pool before WebLogic Server can remove it. **Note**: Idle beans can only be removed from the pool until the number of beans in the pool reaches `initial-beans-in-free-pool`.	idle-timeout-seconds	WebLogic Server removes beans from the free pool when they have been idle for 600 seconds.
The maximum size of the free pool of inactive stateless session beans.	max-beans-in-free-pool	WebLogic Server limits the maximum number of beans in the free pool to 1000.
How WebLogic Server replicates stateless session EJB instances in a cluster. See Reliability and Availability Features.	• stateless-clustering • home-is-clusterable • home-load-algorithm • home-call-router-class-name • stateless-bean-is-clusterable • stateless-bean-load-algorithm • stateless-bean-call-router-class-name	The EJB can be deployed to multiple servers in a cluster.

Table 4-3 WebLogic-Specific Features for Stateful Session EJBs

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

5

Entity EJBs

Understand the value-added features of WebLogic Server for programming and using entity beans in applications, and associated design and development guidelines.
It is assumed that the reader is familiar with Java programming and entity bean features and capabilities. For an introduction to entity beans and how they are typically used in applications, see Entity EJBs Maintain Persistent Data and Entity Bean Features.

For a description of the overall bean development process, see Implementing EJBs.

This chapter includes the following topics:

- Managing Entity Bean Pooling and Caching
  Learn how to improve performance and throughput for entity EJBs.
- Using Primary Keys
  Learn how each entity bean instance can define a different class for its primary key.
- Configuring Entity EJBs for Database Operations
  You can map entity EJBs to database tables and control database access behaviors.
- Using SQL in Entity Beans
  You can use EJB-QL or standard or database-specific SQL for entity beans that use container-managed persistence (CMP). Oracle recommends that you use EJB-QL with or without WebLogic extensions for most queries and use SQL only when needed for instance, to make use of vendor-specific features that cannot be accessed without using vendor-specific SQL.
- Using Container-Managed Relationships (CMRs)
  Container-managed relationships (CMRs) are relationships that you define between two entity EJBs, analogous to the relationships between the tables in a database.
- Choosing a Concurrency Strategy
  An entity bean's concurrency strategy specifies how the EJB container should manage concurrent access to the bean; it determines how and when WebLogic Server synchronizes its cached copy of the entity with the database.
- CMP Entity Bean Descriptors Element by Feature
  Examine the WebLogic Server-specific deployment for CMP entity beans.

Managing Entity Bean Pooling and Caching

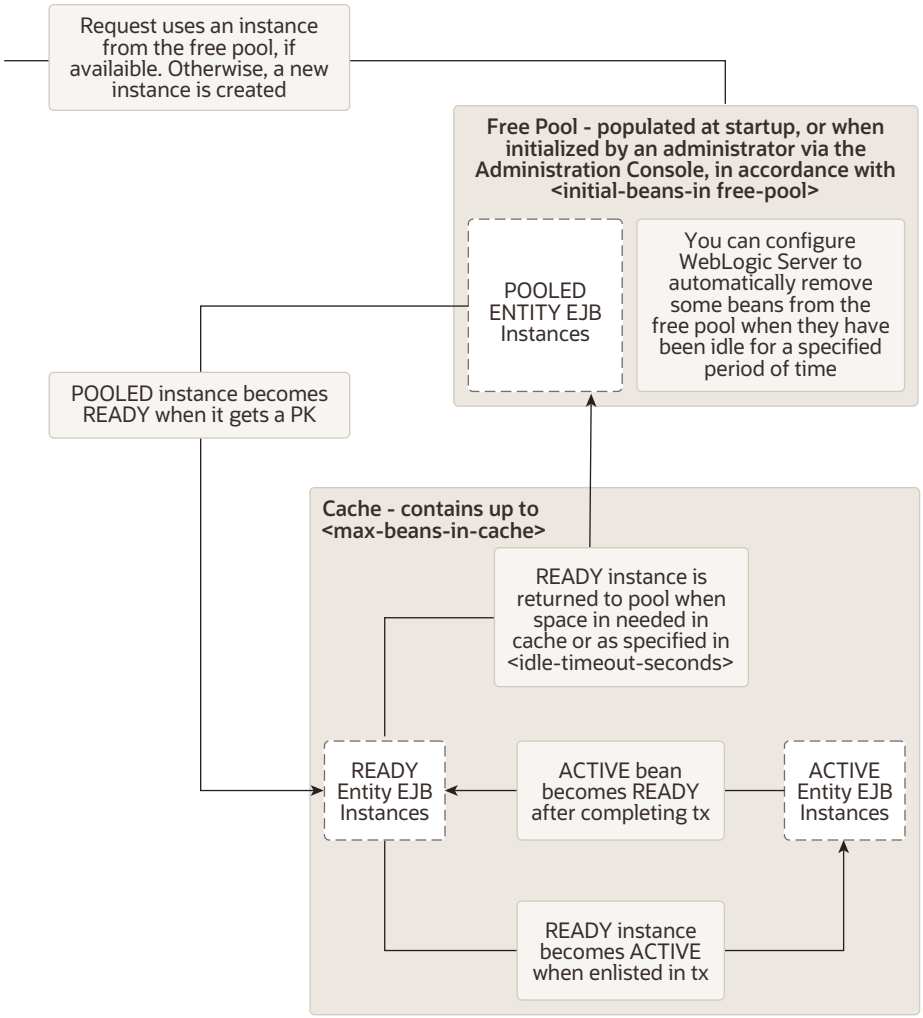
Learn how to improve performance and throughput for entity EJBs.

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

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

Figure 5-1 illustrates the life cycle of an entity bean instance. The sections that follow describe pooling and how the container populates and manages the free pool and the cache.

Figure 5-1 Entity Bean Life Cycle



- Understanding Entity Pooling
- Understanding Entity Caching
- Understanding Passivation of Entity Beans
- Understanding ejbLoad() and ejbStore() Behavior
- Controlling the Behavior of ejbLoad() and ejbStore()
- Disabling Cache Flushing
- Configuring Application-Level Caching

Understanding Entity Pooling

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

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

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

Pooled beans are anonymous instances, and are used for finders and home methods. The maximum number of instances the pool can contain is specified by the `max-beans-in-free-pool` element, in `weblogic-ejb-jar.xml` which set to 1,000 by default.

You can configure WebLogic Server to remove entity beans that have remained in the pool unused for a period of time specified in `idle-timeout-seconds` element in the `pool` element. When beans have been in the pool unused for the amount of time you specify in `idle-timeout-seconds`, they are removed from the pool until the number of beans in the pool reaches the number specified in `initial-beans-in-free-pool`; the number of beans in the pool will never fall below the number specified in `initial-beans-in-free-pool`.

Understanding Entity Caching

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

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

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

The effect of the `max-beans-in-cache` element, and the quantity of instances with the same primary key allowed in the cache vary by concurrency strategy. Figure 5-1 lists, for each concurrency strategy, how the value of the `max-beans-in-cache` element in `weblogic-ejb-jar.xml` limits the number of entity bean instances in the cache, and how many entity bean instances with the same primary key are allowed in the cache.

Table 5-1 Entity EJB Caching Behavior by Concurrency Type

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

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

You can configure WebLogic Server to periodically remove entity instances from cache when they have been idle—not participated in a transaction—for a period of time specified in `idle-timeout-seconds`. For a dedicated cache, the fixed periodicity of the cleaning task is equal to the value of `idle-timeout-seconds`. For an application level cache, multiple types of entity beans may share the cache each of which is allowed to have its own value of `idle-timeout-seconds`. In this case, the fixed periodicity of the cleaning task is equal to `MIN(Set of idle-`

`timeout-seconds for all beans in cache)`, and the cleaning task may run with a variable periodicity.

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

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

Understanding Passivation of Entity Beans

Entity beans that are involved in transactions can be passivated when necessary in order to attempt to prevent a CacheFullException when an attempt is made to insert an entity bean into a cache that is full. Passivation is handled automatically by the EJB container and you do not need to change the way you program EJBs in order to take advantage of this feature. However, you can optionally program your EJBs to communicate to the cache that they are done performing all operations in the current transaction. The cache can then make use of this information when evaluating beans for the possibility of passivation.

To optionally program EJBs to notify their cache that they are done performing operations in the current transaction, you can use the `operationsComplete` Jakarta API as follows:

```
weblogic.ejb.interfaces.EJBLocalObject
public.void.operationsComplete()
weblogic.ejb.EJBObject
public.void.operationsComplete()
```

Understanding ejbLoad() and ejbStore() Behavior

This section describes how and when the persistent data for a CMP 2.1 entity bean is loaded to cache and written back to persistent storage.

- `findXXX()`—By default, calling a finder method on a CMP bean results in immediate load of the bean's persistent data to cache. This behavior is controlled by the `finders-load-bean` element in the `persistence` element of `weblogic-ejb-jar.xml`.
- `ejbLoad()`—For CMP 2.1 entity beans, `ejbLoad()` causes a "lazy" load of a bean's persistent data to the entity cache when the next `getXXX()` for the bean's data is called. That is, when a transaction is initiated for a CMP 2.0 entity bean instance, WebLogic Server reads the bean's data from the entity cache, rather than the database, unless `ejbLoad()` has been called since the bean was last loaded to cache.

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

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

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

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

Controlling the Behavior of ejbLoad() and ejbStore()

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

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

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

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

Disabling Cache Flushing

According to the EJB specification, updates made by a transaction must be reflected in the results of query-finders and `ejbSelect`s issued during the transaction. This requirement can slow performance. If you prefer not to flush the cache before the query is executed, you can change the value of the `include-updates` element in `weblogic-cmp-jar.xml` from its default value of `True` to `False`.

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

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

Configuring Application-Level Caching

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

Application-level caching offers the following advantages:

- Reduces the number of entity bean caches, and hence the effort to configure the cache.
- Better utilization of memory and heap space, because of reduced fragmentation. For example, if a particular EJB home experiences a burst of activity, it can make use of all memory available to the combined cache, while other EJBs that use the cache are paged

out. If two EJBs use different caches, when one bean's cache becomes full, the container cannot page out EJBs in the other bean's cache, resulting in wasted memory.

- Simplifies management; combined caching enables a system administrator to tune a single cache, instead of many caches.
- Provides better scalability

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

To configure an application-level cache:

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

   Note that:

   - `entity-cache-name` should be the name of the application-level cache, as specified in `weblogic-application.xml`.
   - The `concurrency-strategy` you specify for the bean must be compatible with the `caching-strategy` specified in `weblogic-application.xml`. A read-only entity can only use a multiversion application-level cache. See "caching-strategy" in Enterprise Application Deployment Descriptor Elements in *Deploying Applications to Oracle WebLogic Server*.

The `weblogic-application.xml` deployment descriptor is documented in Enterprise Application Deployment Descriptor Elements in *Deploying Applications to Oracle WebLogic Server*.

Using Primary Keys

Learn how each entity bean instance can define a different class for its primary key.

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

If two entity bean instances have the same home and the same primary key, they are considered identical. A client can invoke the `getPrimaryKey()` method on the reference to an entity bean instance's remote interface to determine the instance's identity within its home.

The instance identity associated with a reference does not change during the lifetime of the reference. Therefore, the `getPrimaryKey()` method always returns the same value when called on the same entity object reference. A client that knows the primary key of an entity object can obtain a reference to the entity object by invoking the `findByPrimaryKey(key)` method on the bean's home interface.

Specifying Primary Keys and Primary Key Classes

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

- Mapping a Primary Key to a Single CMP Field

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

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

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

- Anonymous Primary Key Class

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

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

> **Note:**
>
> If you use the original EJB (instead of the subclass) with an anonymous primary key class, WebLogic Server displays the following error message during deployment:
>
> ```
> In EJB ejb_name, an 'Unknown Primary Key Class' ( <prim-key-class> ==
> java.lang.Object ) MUST be specified at Deployment time (as something
> other than java.lang.Object).
> ```

Guidelines for Primary Keys

Follow these suggestions when using primary keys with WebLogic Server:

- Do not construct a new primary key class with an `ejbCreate`. Instead, allow the container to create the primary key class internally, as described in Automatically Generating Primary Keys.
- In an application that manages its own primary key values, for a simple primary key—one composed of a single atomic value such as a `String` or an `Integer`—make the primary key class a container-managed field. Set the value of the primary key `cmp-field` using the `setXXX` method within the `ejbCreate` method.
- Do not use a `cmp-field` of the type `BigDecimal` as a primary key field for CMP beans. The `boolean BigDecimal.equals (object x)` method considers two `BigDecimal` equal only if they are equal in value and scale. This is because there are differences in precision between the Java language and different databases. For example, the method does not consider `7.1` and `7.10` to be equal. Consequently, this method will most likely return False or cause the CMP bean to fail.

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

  1. Implement a primary key class.
  2. In the primary key class, implement the `boolean equal (Object x)` method.
  3. In the equal method, use `boolean BigDecimal.compareTo(BigDecimal val)`.
- If you are mapping a database column to a `cmp-field` and a `cmr-field` concurrently and the `cmp-field` is a primary key field, set the value when the `ejbCreate()` method is invoked by using the `setXXX` method for the `cmp-field`. In this case, the `cmr-field` is initialized automatically, and the `setXXX` method for the `cmr-field` cannot be used. Conversely, if the `cmp-field` is *not* a primary key field, the `cmp-field` is read-only. The column is updated using the `cmr-field`, and the `cmp-field` provides a read-only view of the foreign key.

Automatically Generating Primary Keys

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

WebLogic Server supports two methods of automatic primary key generation:

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

Whichever method of generated primary keys you use, see the instructions in Declaring Primary Key Field Type.

Specifying Automatic Key Generation for Oracle Databases

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

When using the Oracle database `SEQUENCE` object for generating primary keys:

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

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

  Instead, use the AutoKey option with Oracle databases.

- In this release, WebLogic Server does not support Oracle database's synonym feature for the `SEQUENCE` schema object. If you migrate an application that uses synonyms for `SEQUENCE`s from a release prior to WebLogic Server 8.1 to this release, the following errors result:

  ```
  [EJB:011066]During EJB deployment, error(s) were encountered while setting up The
  ORACLE SEQUENCE named 'XXX' with INCREMENT value '1'
  [EJB:011064]The ORACLE SEQUENCE named 'XXX' with INCREMENT '1' was not found in the
  database'
  ```

Specifying Automatic Key Generation for Microsoft SQL Server

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

> **Note:**
>
> For instructions on creating a SQL Server table that contains an `IDENTITY` column, see Microsoft documentation.

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

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

Generating Primary Keys with a Named Sequence Table

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

Create a table named `SEQUENCE` to hold the current primary key value. The table consists of a single row with a single column, as defined by the following statement:

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

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

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

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

Specify the name of the sequence table in the `generator-name` element.

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

Oracle recommends that you define one `NAMED SEQUENCE` table per bean type. Beans of different types should not share a common `NAMED SEQUENCE` table. This reduces contention for the key table.

Declaring Primary Key Field Type

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

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

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

Support for Oracle Database SEQUENCE

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

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

String-Valued CMP Field Trimming

In this release of WebLogic Server, by default, the EJB container trims string-valued CMP fields of their trailing spaces when they are retrieved from a database. All string-valued CMP fields are also trimmed of their trailing spaces in their set method.

Benefits of String Trimming

Untrimmed primary key fields can cause comparison operators to fail and can result in non-portable behavior. Automatic string trimming is beneficial because it causes a string retrieved from a database to be identical to the string that was inserted into the database. For instance, suppose:

- a primary key field is mapped to the `char(10)` datatype in database
- you insert the value "`smith`" into the database column
- since "`smith`" is five characters in length and the char datatype is a fixed-length—of ten characters, in this case—the database pads the value by appending five blank spaces, resulting in "`smith `" being inserted into the database
- you issue a `SELECT` statement to retrieve "`smith`" from the database, only, due to the database-appended characters, "`smith `" is retrieved

A comparison of the retrieved "`smith `" value with the original "`smith`" string would fail unless the retrieved value was first trimmed of its trailing spaces. With this release of WebLogic Server, the trailing spaces are trimmed automatically and, therefore, the comparison would not fail.

Disabling String Trimming

Automatic string trimming is enabled by default in this release. When you use DDConverter to convert deployment descriptors from prior releases of WebLogic Server, DDConverter automatically disables string trimming in the new deployment descriptors it creates.

If you want to disable string trimming for deployment descriptors that are newly created in this release of WebLogic Server, you can set `disable-string-trimming` in `weblogic-cmp-jar.xml` to `True`. For more information on the `disable-string-trimming` element, see disable-string-trimming.

Configuring Entity EJBs for Database Operations

You can map entity EJBs to database tables and control database access behaviors.

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

Configuring Table Mapping

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

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

Example 5-1 and Example 5-2 contain `table-map` elements for a bean that maps to a single and a bean that maps to multiple tables, respectively.

Example 5-1 Mapping a CMP Entity to One Database Table

```
<table-map>
    <table-name>TableName</table-name>
    <field-map>
```

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

Example 5-2 Mapping a CMP Entity to Two DBMS Tables

```
<table-map>
    <table-name>TableName_1</table-name>
        <field-map>
        <!--Note 'name'is the primary key field of this EJB -->
            <cmp-field>name</cmp-field>
            <dbms-column>name_in_tablename_1</dbms-column>
        </field-map>
        <field-map>
            <cmp-field>street_address</cmp-field>
            <dbms-column>street_address_in_tablename_1</dbms-column>
        </field-map>
</table-map>
<table-map>
    <table-name>TableName_2</table-name>
         <field-map>
         <!--Note 'name'is the primary key field of this EJB -->
             <cmp-field>name</cmp-field>
             <dbms-column>name_in_tablename_2</dbms-column>
        </field-map>
        <field-map>
             <cmp-field>phone</cmp-field>
             <dbms-column>phone_in_tablename_2</dbms-column>
         </field-map>
</table-map>
```

Automatic Table Creation (Development Only)

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

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

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

Table 5-2 Controlling Automatic Table Creation Behavior With <create-default-dbms-tables>

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

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

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

Delaying Database Inserts

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

Why Delay Database Inserts?

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

- Problem 1: Non-null foreign key constraint

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

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

- Problem 2: Creating beans during `ejbPostCreate`

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

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

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

- Problem 3: Both Problem 1 and Problem 2 exist

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

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

Configuring Delayed Database Inserts

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

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

Limiting Database Reads with cache-between-transactions

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

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

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

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

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

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

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

Table 5-3 lists the allowable values for the `cache-between-transactions` element by entity bean type and concurrency strategy.

Table 5-3 Permitted cache-between-transactions values, by Concurrency Strategy and Entity Type

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

Updating the Database Before Transaction Ends

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

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

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

Dynamic Queries

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

Using dynamic queries provides the following benefits:

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

Enabling Dynamic Queries

To enable dynamic queries:

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

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

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

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

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

Executing Dynamic Queries

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

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

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

Enabling BLOB and CLOB Column Support for Oracle or DB2 Databases

WebLogic Server supports Oracle and DB2 databases Binary Large Object (BLOB) and Character Large Object (CLOB) DBMS columns for CMP entity beans.

WebLogic Server maps BLOBs to a `cmp-field` that has a byte array or serializable type. At this time, no support is available for mapping `char` arrays to a CLOB column.

To enable BLOB/CLOB support:

1. In the bean class, declare the variable.
2. Edit the XML by declaring the `dbms-default-value` and `dbms-column-type` deployment descriptors in the `weblogic-cmp-jar.xml` file.
3. Create the BLOB or CLOB in the Oracle or DB2 database.

Specifying a BLOB Column Using the Deployment Descriptor

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

```
<field-map>
    <cmp-field>photo</cmp-field>
    <dbms-column>PICTURE</dbms-column>
    <dbms-column-type>Blob</dbms-column-type>
    <dbms-default-value>DB2</dbms-default-value>
</field-map>
```

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

In this release of WebLogic Server, `cmp-field`s of type `byte[]` mapped to a Blob are not serialized by default; the EJB container persists the `byte[]` directly and does not serialize it.

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

Specifying a CLOB Column Using the Deployment Descriptor

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

```
<field-map>
    <cmp-field>description</cmp-field>
    <dbms-column>product_description</dbms-column>
    <dbms_column-type>Clob</dbms-column-type>
    <dbms-default-value>Oracle</dbms-default-value>
</field-map>
```

Optimized CLOB Column Insertion on Oracle 10g

The Oracle 9i and 10g drivers have different requirements for successful insertion of CLOB column values into database rows. The Oracle 9i driver requires that a database row is locked

before a CLOB value can be inserted into it. As a result, on Oracle 9i, WebLogic Server does the following to insert a row that contains a CLOB column value into a table:

1. Inserts a row with all values other than the CLOB column into the table.
2. Issues a `SELECT FOR UPDATE` statement on the row created in step 1, above.
3. Inserts the CLOB value into the row.

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

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

See "Handling CLOBs - Made easy with Oracle JDBC 10g" at `http://www.oracle.com/technetwork/java/jms/index.html#DUPS_OK_ACKNOWLEDGE`.

Specifying Field Groups

A field group represents a subset of a bean's container-managed persistence (CMP) and container-managed relationship (CMR) fields. To improve performance, you can put related fields in a bean into groups that are faulted into memory together as a unit.

You can associate a group with a query or relationship, so that when a bean is loaded as the result of executing a query or following a relationship, only the fields mentioned in the group are loaded.

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

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

You specify field groups in the `field-group` element of `weblogic-cmp-jar.xml` file, as shown in Example 5-3.

Example 5-3 Specifying Field Groups

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

Ordering and Batching Operations

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

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

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

Operation Ordering

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

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

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

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

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

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

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

Batch Operations Guidelines and Limitations

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

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

There are several limitations on using batch operations:

- You cannot use the Automatically Generating Primary Keys feature with batch operations if the `generator-type` is set to `sybase` or `sqlServer`. If you have enabled automatic primary key generation with either of these types, WebLogic Server automatically disables batch operations and issues a warning.
- The total number of entries created in a single batch operation cannot exceed the `max-beans-in-cache` setting, which is specified in the `weblogic-ejb-jar.xml` file.

- If you set the `dbms-column-type` element in `weblogic-cmp-jar.xml` to either `Blob` or `Clob`, batch operation automatically turns off because you will not save much time if a Blob or Clob column exists in the database table. In this case, WebLogic Server performs one insert per bean, which is the default behavior.

Using Query Caching (Read-Only Entity Beans)

You can choose to cache read-only entity EJBs at the query level. Caching read-only entity EJBs at the query level improves performance because it enables the EJBs to be accessed in cache by any finder, thereby avoiding the need to access the database. For information, see enable-query-caching.

Using SQL in Entity Beans

You can use EJB-QL or standard or database-specific SQL for entity beans that use container-managed persistence (CMP). Oracle recommends that you use EJB-QL with or without WebLogic extensions for most queries and use SQL only when needed for instance, to make use of vendor-specific features that cannot be accessed without using vendor-specific SQL.

You can use SQL in queries that cache multiple related objects, to implement finder and select methods, or with stored procedures.

To use EJB-QL in queries in this release of WebLogic Server, you do not need to change any mapping information in `weblogic-cmp-jar.xml`. You simply continue to map each CMP and CMR field to database columns as you did in pre-9.0 releases of WebLogic Server. For information, see, Configuring Entity EJBs for Database Operations and Using Container-Managed Relationships (CMRs). To use SQL in queries, you must describe the shape of the SQL result with the `sql-shape` element. SQL shape primarily consists of the SQL tables and columns that are returned. The EJB containers uses the SQL shape together with the CMP and CMP field mappings to return objects from the query. For more information, see:

- sql-shape
- sql-query

Using Container-Managed Relationships (CMRs)

Container-managed relationships (CMRs) are relationships that you define between two entity EJBs, analogous to the relationships between the tables in a database.

If you define a CMR between two EJBs that are involved in the same processing task, your application can benefit from these features:

- Related beans can be cached together, reducing the number of queries necessary to accomplish a processing task.
- Batched database operations can be ordered correctly at the end of a transaction, avoiding data consistency problems.
- Related beans can be deleted automatically, using the cascade delete feature.

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

- CMR Requirements and Limitations
- CMR Cardinality
- CMR Direction

CMR Requirements and Limitations

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

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

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

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

CMR Cardinality

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

CMR Direction

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

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

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

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

Removing CMRs

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

Defining Container-Managed Relationships (CMRs)

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

- Specifying Relationships in ejb-jar.xml
- Specifying Relationships in weblogic-cmp-jar.xml

Specifying Relationships in ejb-jar.xml

Container-managed relationships are defined in the `ejb-relation` element of `ejb-jar.xml`. Example 5-4 shows the `ejb-relation` element for a relationship between two entity EJBs: `TeacherEJB` and `StudentEJB`.

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

Example 5-4 One-to-Many, Bidirectional CMR in ejb-jar.xml

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

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


Specifying Relationship Cardinality

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

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

The cardinality of the CMR in Example 5-5, is one-to-one—the `multiplicity` is set to `one` in both role elements for the relationship.

Table 5-4 Cardinality and cmr-field-type

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

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

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

If a side of a relationship has a `<multiplicity>` of `Many`, its `<cmr-field>` is a collection, and you must specify its `<cmr-field-type>` as `java.util.Collection`, as shown in the `StudentEJB` side of the relationship in Example 5-4. It is not necessary to specify the `cmr-field-type` when the `cmr-field` is a single valued object.

Table 5-4 lists the contents of `cmr-field` for each bean in a relationship, based on the cardinality of the relationship.

Specifying Relationship Directionality

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

A bidirectional CMR has a `cmr-field` element in the `ejb-relationship-role` element for both sides of the relationship, as shown in Example 5-4.

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

Specifying Relationships in weblogic-cmp-jar.xml

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

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

- One-to-One and One-to-Many Relationships
- Many-to-Many Relationships
- Specifying CMRs for EJBs that Map to Multiple Tables

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

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

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

Example 5-6 is the `weblogic-rdbms-relation` element for a the one-to-one relationship between `MentorEJB` and `StudentEJB`, whose `<ejb-relation>` is shown in Example 5-5.

Example 5-6 One-to-One CMR weblogic-cmp-jar.xml

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

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

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

Example 5-7 is the `weblogic-rdbms-relation` element for a the one-to-many relationship between `TeacherEJB` and `StudentEJB`, whose `<ejb-relation>` is shown in Example 5-4.

Example 5-7 weblogic-rdbms-relation for a One-to-Many CMR

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

Many-to-Many Relationships

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

Example 5-8 shows the `weblogic-rdbms-relation` element for the `friends` relationship between two employees.

The `FRIENDS` join table has two columns, `first-friend-id` and `second-friend-id`. Each column contains a foreign key that designates a particular employee who is a friend of another employee. The primary key column of the employee table is `id`. The example assumes that the employee bean is mapped to a single table. If employee bean is mapped to multiple tables, then the table containing the primary key column must be specified in the `relation-role-map`. For an example, see Specifying CMRs for EJBs that Map to Multiple Tables.

Example 5-8 weblogic-rdbms-relation for a Many-to-Many CMR

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

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

Specifying CMRs for EJBs that Map to Multiple Tables

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

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

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

Example 5-9 contains a `relationship-role-map` for a CMR in which the bean on the foreign-key side of a one-to-one relationship, `Fk_Bean`, is mapped to two tables: `Fk_BeanTable_1` and `Fk_BeanTable_2`.

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

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

Example 5-9 One-to-One CMR, One Bean Maps to Multiple Tables

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

About CMR Fields and CMR Field Accessor Methods

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

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

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

Using Cascade Delete for Entities in CMRs

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

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

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

WebLogic Server supports two methods of cascade delete:

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

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

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

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

In a high volume transaction environment, transactions that use exclusive concurrency can encounter deadlocks when a transaction that performs a cascade delete needs access to the same entity bean as a transaction that does not perform a cascade delete. For information on how to avoid such deadlocks, see Deadlock Prevention for Exclusive Concurrency and Cascade Deletes.

Relationship Caching

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

Suppose your application contains entity beans with the following relationships:

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

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

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

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

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

The relationship caching feature has the following limitations:

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

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

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

Enabling Relationship Caching

To enable relationship caching:

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

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

2. Make sure that the `finders-load-bean` element (in `weblogic-ejb-jar.xml`) for the bean that contains the finder is not set to `False` or relationship caching will not be enabled. The default value of `finder-load-bean` is `True`.
3. Specify the `database-type` element in the `weblogic-cmp-jar.xml` file. Relationship caching uses outer joins for queries and outer join syntax can vary by database type.

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

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

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

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

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

Choosing a Concurrency Strategy

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

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

Exclusive Concurrency

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

Multiple clients can use the exclusive concurrency option to access entity EJBs in a serial fashion. Using this exclusive option means that if two clients simultaneously attempt to access the same entity EJB instance (an instance having the same primary key), the second client is blocked until the EJB is available.

Database Concurrency

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

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

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

Optimistic Concurrency

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

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

Preventing Stale Optimistic Bean Data

To prevent optimistic data from going stale, the container activates a new bean instance for each transaction so that requests proceed in parallel. WebLogic Server calls `ejbLoad()` for entity beans based on the value specified in `read-timeout-seconds`, provided the value of `cache-between-transactions` is `True`.

Explicit Invalidation of Optimistic Beans

You can invalidate optimistic beans explicitly when the underlying data is updated outside the EJB container (this is also known as a "backdoor update"). For information, see Invalidating Entity EJBs Explicitly.

Invalidation Options for Optimistic Concurrency in Clusters

By default, when a bean that has a `concurrency-strategy` of `Optimistic` is deployed in a cluster and a member of the cluster updates the bean, the EJB container attempts to invalidate all copies of the bean in all servers in the cluster. This invalidation allows you to avoid optimistic concurrency failures, but can be a drain on performance because it is a resource-intensive operation. If you prefer, you can set `cluster-invalidation-disabled` in weblogic-cmp-jar.xml to `True` to prevent the EJB container from attempting to invalidate copies of the bean across the cluster.

Check Data for Validity with Optimistic Concurrency

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

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

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

The `verify-rows` element specifies the rows in a table that the EJB container should check when optimistic concurrency is used; the `verify-columns` element specifies how columns in a table are checked for validity when you use the optimistic concurrency strategy.

1. Set the `verify-rows` element to:
   - `Read`—To check all rows in the table that have been read during the transaction. This includes both rows that are simply read and rows that are read and then updated or deleted by the transaction. Checking all rows entails additional overhead because it generally increases the amount of optimistic checking that must be performed by the EJB container. With the `Read` option, committed transactions read a set of rows that are guaranteed not to be modified by another transaction until after the transaction commits. This results in a high level of consistency which is very close to the ANSI definition of SERIALIZABLE consistency.
   - `Modified`—To check only the rows that have been updated or deleted by the current transaction. This setting ensures that two transactions will not update the same row concurrently, resulting in a lost update, but it allows reads and updates of different transactions to be interleaved. This results in a level of consistency that falls between the ANSI READ_COMMITTED and REPEATABLE_READ levels of consistency.
2. Set the value of the `verify-columns` element to:
   - `Read`—To check all columns in the table that have been read during the transaction. This includes both rows that are simply read and rows that are read and then updated or deleted by the transaction.
   - `Modified`—To check only the columns that have been updated or deleted by the current transaction.

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

   - `Version`—To check that a version column exists in the table and that this column is used to implement optimistic concurrency.

     A version column must be created with an initial value of 0 or any positive integer, and must increment by 1 whenever the row is modified. For more information, see version-column-initial-value.

     Additionally, if you use database triggers to update version columns in the database and you set `trigger-updates-optimistic-column` to `True`, the database triggers *must* initialize the version column in the database when the bean is created.
   - `Timestamp`—To check that a timestamp column exists in the table and that this column is used to implement optimistic concurrency. Timestamp-based optimistic concurrency requires a 1 second granularity for the database column.

   By default, the EJB container manages the version and timestamp columns and ensures that these columns are kept up to date. If you choose to instead let database triggers maintain version and timestamp columns, set the value of `trigger-updates-optimistic-column` to `True`.

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

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

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

   b. Specify the initial value of the version column using the `version-column-initial-value` element in the `weblogic-cmp-jar.xml` file.

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

Optimistic Concurrency and Oracle Databases

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

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

- Set the Oracle database column type to `Timestamp`, a higher precision type introduced in Oracle9i.
- Include logic in your application to zero out the milliseconds of a `java.util.Date` value. To accomplish this, prepare a date field for an entity bean `java.util.Date` field in this way:

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

Read Only Concurrency

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

To allow generation of more efficient code for read-only beans, create and remove operations are not allowed for EJBs that use ReadOnly concurrency in this release of WebLogic Server.

To allow read-only beans to use create and remove operations—for instance, to support legacy behavior—set the `allow-readonly-create-and-remove` element in `weblogic-cmp-jar.xml` to `True`.

Concurrency Strategy Trade-Offs

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

Table 5-5 Concurrency Strategy Trade-offs

Concurrency Strategy	Benefits	Risks and Limitations	When to Choose It
`Database`	Deferring concurrency control to the database improves throughput, compared to exclusive concurrency, for concurrent access to data and provides deadlock detection.	Risk of deadlock, as each transaction must obtain an update lock from the database. Mitigate deadlock risk by setting `use-select-for-update` in `weblogic-cmp-jar.xml`. This causes the database to take out an exclusive lock when the read is done, avoiding the deadlock that can occur when a read lock is upgraded to an exclusive lock. Makes the bean more dependent on the database's lock policies, which might reduce the bean's portability.	If the database concurrency control is sufficient for your application and you do not require additional features provided by the container. **Note**: Use the `transaction-isolation` element in combination with Database concurrency to achieve the desired concurrency behavior.
`Optimistic`	Provides highest level of concurrent access, as it holds no locks in the EJB container or database during a transaction.	Multiple transactions can access the same application data at the same time.	If multiple transactions are unlikely to attempt to modify the same application data at the same time.
`Exclusive`	Serializes access to EJB data in a single server (non-clustered environment) for a high level of consistency. Avoids deadlocks due to lock upgrades, and prevents unnecessary calls to `ejbLoad()` to refresh the bean instance's persistent fields.	Performance degradation can result. Once a client has locked an EJB instance, other clients are blocked from the EJB's data, even those who require only read-access.	To provide backwards compatibility for applications that rely on this strategy. For applications in which a high level of concurrency is essential, and more important than performance.
`Read Only`	N/A	N/A	N/A

Configuring Concurrency Strategy

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

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

Deadlock Prevention for Exclusive Concurrency and Cascade Deletes

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

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

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

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

Using the Read-Mostly Pattern

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

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

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

Configuring Entity Beans for Read-Mostly Pattern

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

- Configuring the read-only beans' `read-timeout-seconds` element in `weblogic-ejb-jar.xml`:
  - To the same value in all beans that can be updated in the same transaction.
  - To the shortest period that yields acceptable performance levels.
- Design read-write beans:
  - To update the minimum data set necessary; avoid implementing beans that write numerous, unchanged fields to the datastore at each `ejbStore()`.

- To update their data on a timely basis; do not use a read-write bean in lengthy transactions that might run longer than the `read-timeout-seconds` setting for its read-only counterparts.
- To invalidate the read-only counterpart of a read-write bean when the read-write bean updates bean data.

If you are running EJB 2.x, you can approximate the read-mostly pattern using a single bean that uses optimistic concurrency. An optimistic bean acts like a read-only bean when performing a read—it reads from the cache and can return stale data. However, when an optimistic bean performs a write, the container ensures that the data being updated has not changed—providing the same level of consistency for writes as a bean that uses Database concurrency. See Choosing a Concurrency Strategy.

Invalidating Read-Only Entity EJBs Implicitly

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

`invalidation-target` may only be specified for an EJB 2.x CMP entity bean. The target `ejb-name` must be a read-only entity EJB.

Invalidating Entity EJBs Explicitly

In this release of WebLogic Server, you can invalidate any optimistic entity bean that has `cache-between-transactions` enabled, by calling the following `invalidate()` method on either the `CachingHome` or `CachingLocalHome` interface.

Example 5-10 CachingHome and CachingLocalHome interfaces

```
package weblogic.ejb;

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

public interface CachingLocalHome {
    public void invalidate(Object pk) throws RemoteException;
    public void invalidate (Collection pks) throws RemoteException;
    public void invalidateAll() throws RemoteException
}
```

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

Example 5-11 Casting the Home and Calling the Method

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

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

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

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

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

CMP Entity Bean Descriptors Element by Feature

Examine the WebLogic Server-specific deployment for CMP entity beans.

Each of the following section contains the elements related to a particular type of feature or behavior. The table in each section defines relevant elements terms of: the behavior it controls, the parent element in `weblogic-cmp-jar.xml` that contains the element, and the behavior you can expect if you do not explicitly specify the element in `weblogic-cmp-jar.xml`.

- Container-Managed Relationship Elements
- Primary Key Elements

Container-Managed Relationship Elements

The following lists the container-managed relationship elements in `weblogic-cmp-jar.xml`.

Table 5-6 Container-managed Relationship Elements in weblogic-cmp-jar.xml

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

Primary Key Elements

The following lists the primary key elements in `weblogic-cmp-jar.xml`.

Table 5-7 Primary Key Elements in weblogic-cmp-jar.xml

Element	Description	Default
generator-type	Identifies the facility used to generate primary keys. Values include `Oracle`, `SQLServer`, or `SQLServer2000`, `NamedSequenceTable`.	n/a
generator-name	Defines the `Oracle SEQUENCE`, or the name of a `SEQUENCE` table used.	n/a
key-cache-size	Specifies the size of the key cache.	`1`
create-default-dbms-table	Determines behavior related to if and how the EJB container will create database tables.	Disabled

6
Message-Driven EJBs

Learn about the message-driven bean (MDB) life cycle, design considerations, and instructions for key implementation tasks.

Developing Message-Driven Beans for Oracle WebLogic Server describes the message-driven bean (MDB) life cycle, design considerations, and instructions for key implementation tasks. For a description of the overall EJB development process, see Implementing EJBs.

7

Deployment Guidelines for EJBs

Learn about the EJB-specific deployment guidelines.
For deployment topics that are common to all deployable application units, you will see cross-references to topics in *Deploying Applications to Oracle WebLogic Server*, a comprehensive guide to deploying WebLogic Server applications and modules.

This chapter includes the following topics:

- Before You Deploy an EJB
  Examine the tasks that need to be completed before you begin with the EJB deployment.
- Understanding and Performing Deployment Tasks
  Examine some of the deployment strategies to perform the deployment tasks.
- Deployment Guidelines for EJBs
  Understand the guidelines to deploy EJBs on WebLogic Server.

Before You Deploy an EJB

Examine the tasks that need to be completed before you begin with the EJB deployment.

Before starting the deployment process you should have:

- Functional, tested bean code, in an exploded directory format or packaged in an archive file—a JAR for a stand-alone EJB or an EAR if the EJB is part of an enterprise application—along with the deployment descriptors. For production environments, Oracle recommends that you package your application as an EAR.

  For an overview of the steps required to create and package an EJB, see Overview of the EJB Development Process.
- Configured the required deployment descriptors—`ejb-jar.xml` and `weblogic-ejb-jar.xml`, and, for entity EJBs that use container-managed persistence, `weblogic-cmp-jar.xml`.

Understanding and Performing Deployment Tasks

Examine some of the deployment strategies to perform the deployment tasks.

Table 7-1 is a guide to WebLogic Server documentation topics that help you make decisions about deployment strategies and provide instructions for performing deployment tasks. For EJB-specific deployment topics, see Deployment Guidelines for EJBs.

Table 7-1 Deployment Tasks and Topics

If You Want To....	See This Topic
Deploy in a development environment	Deploying and Packaging from a Split Development Directory in *Developing Applications for Oracle WebLogic Server*.
Select a deployment tool	Deployment Tools in *Deploying Applications to Oracle WebLogic Server*

Table 7-1 (Cont.) Deployment Tasks and Topics

If You Want To....	See This Topic
Determine appropriate packaging for a deployment	Preparing Applications and Modules for Deployment in *Deploying Applications to Oracle WebLogic Server*.
Organizing EJB components in a split directory structure.	EJBs in *Developing Applications for Oracle WebLogic Server*.
Select staging mode	Controlling Deployment File Copying with Staging Modes in *Deploying Applications to Oracle WebLogic Server*.
Perform specific deployment tasks	Overview of the Deployment Process in *Deploying Applications to Oracle WebLogic Server*.

Deployment Guidelines for EJBs

Understand the guidelines to deploy EJBs on WebLogic Server.

The following sections provide guidelines for deploying EJBs.

- Deploy EJBs as Part of an Enterprise Application
- Deploy EJBs That Call Each Other in the Same Application
- Deploying EJBs that Use Dependency Injection
- Deploy Homogeneously to a Cluster
- Deploying Pinned EJBs to a Cluster
- Redeploying an EJB
- Using FastSwap Deployment to Minimize Deployment
- Understanding Warning Messages
- Disabling EJB Deployment Warning Messages

Deploy EJBs as Part of an Enterprise Application

Oracle recommends that you package and deploy your stand-alone EJB applications as part of an Enterprise application. An Enterprise application is a Jakarta EE deployment unit that bundles together Web applications, EJBs, and Resource Adapters into a single deployable unit.

This is a Oracle best practice, which allows for easier application migration, additions, and changes. Also, packaging your applications as part of an Enterprise application allows you to take advantage of the split development directory structure, which provides a number of benefits over the traditional single directory structure. See Overview of the Split Development Directory Environment in *Developing Applications for Oracle WebLogic Server*.

Deploy EJBs That Call Each Other in the Same Application

When an EJB in one application calls an EJB in another application, WebLogic Server passes method arguments by value, due to classloading requirements. When EJBs are in the same application, WebLogic Server can pass method arguments by reference; this improves the performance of method invocation because parameters are not copied.

For best performance, package components that call each other in the same application, and set `enable-call-by-reference` in `weblogic-ejb-jar.xml` to `True`. (By default, `enable-call-by-reference` is `False`.)

Switching Protocol Limitation

If an application client request has multiple hops, and QOS is configured differently between servers, then you must switch the protocol.

For example, when a client sends an SSL request to a JMS front-end cluster, the JMS front-end cluster then forwards the request to the JMS back-end cluster using clear text. In this case, you may need to switch from the *t3s* protocol to the *t3* protocol.

> **Note:**
>
> You can switch the protocol only in a default channel. Custom channels do not support protocol switching.

Deploying EJBs that Use Dependency Injection

When an EJB uses dependency injection, the resource name defined in the class and the superclass must be unique. For example:

```
public class ClientServlet extends HttpServlet {
    @EJB(name = 'DateServiceBean', beanInterface = DateService.class)
    private DateService bean; .... }
public class DerivedClientServlet extends ClientServlet {
    @EJB(name = MyDateServiceBean', beanInterface = DateService.class)
    private DateService bean; .... }
```

For more information about dependency injection, see Using Jakarta Annotations and Dependency Injection in *Developing Applications for Oracle WebLogic Server*.

Deploy Homogeneously to a Cluster

If your EJBs will run on a WebLogic Server cluster, Oracle recommends that you deploy them homogeneously—to each Managed Server in the cluster. Alternatively, you can deploy an EJB to only to a single server in the cluster (that is, "pin" a module to a server). This type of deployment is less common, and should be used only in special circumstances where pinned services are required. See, Understanding Cluster Configuration in *Administering Clusters for Oracle WebLogic Server*.

Deploying Pinned EJBs to a Cluster

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

During deployment, the uncompiled EJB is copied to each server instance in the cluster, but it is compiled only on the server instance to which it has been deployed. As a result, the server instances in the cluster to which the EJB was not targeted lack the classes generated during

compilation that are necessary to invoke the EJB. When a client on another server instance tries to invoke the pinned EJB, it fails, and an Assertion error is thrown in the RMI layer.

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

For more information on pinned deployments, see Deploying to a Single Server Instance (Pinned Deployment) in *Administering Clusters for Oracle WebLogic Server*.

Redeploying an EJB

When you make changes to a deployed EJB's classes, you must redeploy the EJB. If you use automatic deployment, deployment occurs automatically when you restart WebLogic Server. Otherwise, you must explicitly redeploy the EJB.

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

When you redeploy, the classes currently loaded for the EJB are immediately marked as unavailable in the server, and the EJB's classloader and associated classes are removed. At the same time, a new EJB classloader is created, which loads and maintains the revised EJB classes.

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

You can redeploy an EJB that is standalone or part of an application using any of the administration tools listed in Summary of System Administration Tools and APIs in *Understanding Oracle WebLogic Server*. See Redeploying Applications in a Production Environment in *Deploying Applications to Oracle WebLogic Server*.

Production redeployment is not supported for:

- applications that use JTS drivers.
- applications that include EJB 1.1 container-managed persistence (CMP) EJBs. To use production redeployment with applications that include CMP EJBs, use EJB 2.x CMP instead of EJB 1.1 CMP.

For more information on production redeployment limitations, see Requirements and Restrictions for Production Redeployment in *Deploying Applications to Oracle WebLogic Server*.

Using FastSwap Deployment to Minimize Deployment

During iterative development of an EJB application, you make many modifications to the EJB implementation class file, typically redeploying an EJB module multiple times during its development.

Java EE 5 introduces the ability to redefine a class at runtime without dropping its ClassLoader or abandoning existing instances. This allows containers to reload altered classes without disturbing running applications, vastly speeding up iterative development cycles and improving the overall development and testing experiences.

With FastSwap, Java classes are redefined in-place without reloading the ClassLoader, thereby having the decided advantage of fast turnaround times. This means that you do not have to wait for an application to redeploy for your changes to take affect. Instead, you can make your changes, auto compile, and then see the effects immediately.

For more information about FastSwap, see Using FastSwap Deployment to Minimize Redeployment in *Deploying Applications to Oracle WebLogic Server*.

Understanding Warning Messages

To get information about a particular warning, use the `weblogic.GetMessage` tool. For example:

```
java weblogic.GetMessage -detail -id BEA-010202
```

Disabling EJB Deployment Warning Messages

You can disable certain WebLogic Server warning messages that occur during deployment. You may find this useful if the messages provide information of which you are already aware.

For example, if the methods in your EJB makes calls by reference rather than by value, WebLogic Server generates this warning during deployment: `"Call-by-reference not enabled."`

You can use the `disable-warning` element in `weblogic-ejb-jar.xml` to disable certain messages. For a list of messages you can disable, and instructions for disabling the messages, see disable-warning.

A

Deployment Descriptor Schema and Document Type Definitions Reference

Examine the EJB XSD-based deployment descriptors, namespace declarations for WebLogic Server, EJB DTD-based deployment descriptors, and DOCTYPE headers for deployment descriptors from pre-9.0 releases of WebLogic Server.
WebLogic Server deployment descriptors are XML Schema Definition-based (XSD). In pre-9.0 releases of WebLogic Server, deployment descriptors were Document Type Definition-based (DTD). For backward compatibility, WebLogic Server supports XSD- or DTD-based deployment descriptors; you can deploy applications that use DTD-based descriptors without modifying them.

> **Note:**
>
> If you are using metadata annotations in your EJB 3.x implementation, refer to EJB Metadata Annotations Reference in *Developing Enterprise JavaBeans for Oracle WebLogic Server*.

For information on the elements in WebLogic Server EJB deployment descriptors, see the following sections:

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

For information on the EJB 1.1 deployment descriptor elements, see Important Information for EJB 1.1 Users.

This appendix includes the following topics:

- XML Schema Definitions and Namespace Declarations
  An XSD deployment descriptor file requires a namespace declaration in the root element of the file. Namespace declarations in the root element of a deployment descriptor file apply to all elements in the descriptor unless a specific element includes another namespace declaration that overrides the root namespace declaration.
- Document Type Definitions and DOCTYPE Header Information
  WebLogic Server ignores the DTD locations 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.

XML Schema Definitions and Namespace Declarations

An XSD deployment descriptor file requires a namespace declaration in the root element of the file. Namespace declarations in the root element of a deployment descriptor file apply to all elements in the descriptor unless a specific element includes another namespace declaration that overrides the root namespace declaration.

The contents and arrangement of elements in your deployment descriptor files must conform to the appropriate XSD.

> **Note:**
>
> If you use DDConverter to convert your DTD-based deployment descriptors to XSD-based, the correct namespace declaration is automatically written to your descriptor files.

Oracle recommends that you always include the schema location URL along with the namespace declaration in your XML deployment descriptor files; if you do not include the schema location in your XML deployment descriptor files, you may not be able to edit the descriptor files with a third-party tool.

weblogic-ejb-jar.xml Namespace Declaration and Schema Location

The correct text for the namespace declaration and schema location for the WebLogic Server `weblogic-ejb-jar.xml` file is as follows.

```
<weblogic-ejb-jar xmlns="http://xmlns.oracle.com/weblogic/weblogic-ejb-jar"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.oracle.com/weblogic/weblogic-ejb-jar  http://
xmlns.oracle.com/weblogic/weblogic-ejb-jar/1.6/weblogic-ejb-jar.xsd">
...
</weblogic-ejb-jar>
```

weblogic-cmp-jar.xml Namespace Declaration and Schema Location

The correct text for the namespace declaration and schema location for the WebLogic Server weblogic-cmp-jar.xml file is as follows.

```
<weblogic-rdbms-jar xmlns="http://xmlns.oracle.com/weblogic/weblogic-rdbms-jar"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.oracle.com/weblogic/weblogic-rdbms-jar/1.2/weblogic-
rdbms-jar.xsd">
...
</weblogic-rdbms-jar>
```

ejb-jar.xml Namespace Declaration and Schema Location

The correct text for the namespace declaration and schema location for the Enterprise JavaBeans 2.1 ejb-jar.xml file is as follows.

```
<ejb-jar xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/ejb-
jar_2_1.xsd">
...
</ejb-jar>
```

Document Type Definitions and DOCTYPE Header Information

WebLogic Server ignores the DTD locations 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.

In prior releases of WebLogic Server, the contents and arrangement of elements in your deployment descriptor files must have conformed to the appropriate DTD.

When editing or creating XML deployment files, it is critical to include the correct DOCTYPE header for each deployment file. In particular, using an incorrect `PUBLIC` element within the DOCTYPE header can result in parser errors that may be difficult to diagnose.

The header refers to the location and version of the Document Type Definition (DTD) file for the deployment descriptor. Although this header references an external URL at java.sun.com, WebLogic Server contains its own copy of the DTD file, so your host server need not have access to the Internet. However, you must still include this `<!DOCTYPE...>` element in your `weblogic-ejb-jar.xml` and `weblogic-cmp-jar.xml` files, and have them reference the external URL because the version of the DTD contained in this element is used to identify the version of this deployment descriptor.

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

```
SAXException: This document may not have the identifier `identifier_name'
```

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

The correct text for the `PUBLIC` elements for the WebLogic-Server-specific `weblogic-ejb-jar.xml` file is listed, by WebLogic Server release, in Table A-1.

Table A-1 PUBLIC Elements of weblogic-ejb-jar.xml

WebLogic Server Release	XML File	PUBLIC Element String
8.1.x	`weblogic-ejb-jar.xml`	`'-//BEA Systems, Inc.//DTD WebLogic 8.1.0 EJB//EN' 'http://www.bea.com/servers/wls810/dtd/weblogic-ejb-jar.dtd'`
7.0.x	`weblogic-ejb-jar.xml`	`'-//BEA Systems, Inc.//DTD WebLogic 7.0.0 EJB//EN' 'http://www.bea.com/servers/wls700/dtd/weblogic-ejb-jar.dtd'`
6.1.x and 6.0.x	`weblogic-ejb-jar.xml`	`'-//BEA Systems, Inc.//DTD WebLogic 6.0.0 EJB//EN' 'http://www.bea.com/servers/wls600/dtd/weblogic-ejb-jar.dtd'`
5.1.0	`weblogic-ejb-jar.xml`	`'-//BEA Systems, Inc.//DTD WebLogic 5.1.0 EJB//EN'` `'http://www.bea.com/servers/wls510/dtd/weblogic-ejb-jar.dtd'`

The correct text for the `PUBLIC` elements for the WebLogic-Server-specific `weblogic-cmp-jar.xml` file is listed, by WebLogic Server release, in Table A-2.

Table A-2 PUBLIC Elements of weblogic-cmp-jar.xml

WebLogic Server Release	XML File	PUBLIC Element String
8.1.x	`weblogic-cmp-jar.xml`	`'-// BEA Systems, Inc.//DTD WebLogic 8.1.0 EJB RDBMS Persistence//EN' 'http://www.bea.com/servers/wls810/dtd/weblogic-rdbms20-persistence-810.dtd'`
7.0.x	`weblogic-cmp-jar.xml`	`'-// BEA Systems, Inc.//DTD WebLogic 7.0.0 EJB RDBMS Persistence//EN' 'http://www.bea.com/servers/wls700/dtd/weblogic-rdbms20-persistence-700.dtd'`
6.1.x and 6.0.x	`weblogic-cmp-jar.xml`	`'-// BEA Systems, Inc.//DTD WebLogic 6.0.0 EJB RDBMS Persistence//EN' 'http://www.bea.com/servers/wls600/dtd/weblogic-rdbms20-persistence-600.dtd'`

See Deployment Descriptor Schema and Document Type Definitions Reference for more information on the `weblogic-cmp-jar.xml` file.

The correct text for the `PUBLIC` elements for the Sun-Microsystems-specific `ejb-jar.xml` file is listed, by Enterprise JavaBeans version, in Table A-3.

Table A-3 PUBLIC Elements of ejb-jar.xml

EJB Version	XML File	PUBLIC Element String
2.0	`ejb-jar.xml`	`'-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN'` `'http://java.sun.com/dtd/ejb-jar_2_0.dtd'`
1.1	`ejb-jar.xml`	`'-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN'` `'http://www.java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'`

B

weblogic-ejb-jar.xml Deployment Descriptor Reference

The EJB 2.1 elements in the `weblogic-ejb-jar.xml` file, the WebLogic-specific XML Schema-based (XSD) deployment descriptor file. Use these definitions to create the WebLogic-specific `weblogic-ejb-jar.xml` file that is part of your EJB deployment.
In pre-9.0 releases of WebLogic Server, `weblogic-ejb-jar.xml` was Document Type Definition-based (DTD). For backward compatibility, WebLogic Server still supports XSD-based or DTD-based EJB descriptors; you can deploy applications that use DTD-based descriptors in WebLogic Server without modifying the descriptors.

> **Note:**
>
> If you are using metadata annotations in your EJB 3.x implementation, refer to EJB Metadata Annotations Reference in *Developing Enterprise JavaBeans for Oracle WebLogic Server*.

For information on:

- XML Schema Definitions and the namespace declaration required in `weblogic-ejb-jar.xml`, as well as Document Type Definitions and DOCTYPE headers, see Deployment Descriptor Schema and Document Type Definitions Reference.
- the `weblogic-cmp-jar.xml` file, see weblogic-cmp-jar.xml Deployment Descriptor Reference.
- EJB 1.1 deployment descriptor elements, see Important Information for EJB 1.1 Users.

See the complete `weblogic-ejb-jar.xsd` schema at `http://www.oracle.com/webfolder/technetwork/weblogic/weblogic-ejb-jar/index.html`.

This appendix includes the following topics:

- confidentiality
- connection-factory-jndi-name
- connection-factory-resource-link
- create-as-principal-name
- delay-updates-until-end-of-tx
- description
- destination-jndi-name
- destination-resource-link
- disable-warning
- dispatch-policy
- distributed-destination-connection
- durable-subscription-deletion
- ejb-name
- ejb-reference-description
- ejb-ref-name
- enable-bean-class-redeploy
- enable-call-by-reference
- enable-dynamic-queries
- entity-always-uses-transaction
- entity-cache
- entity-cache-name
- entity-cache-ref
- entity-clustering
- entity-descriptor
- estimated-bean-size
- externally-defined
- finders-load-bean
- generate-unique-jms-client-id
- global-role
- home-call-router-class-name
- home-is-clusterable
- home-load-algorithm
- idempotent-methods
- identity-assertion
- idle-timeout-seconds
- iiop-security-descriptor
- init-suspend-seconds
- initial-beans-in-free-pool

- initial-context-factory
- integrity
- invalidation-target
- is-modified-method-name
- isolation-level
- jms-client-id
- jms-polling-interval-seconds
- jndi-binding
- jndi-name
- local-jndi-name
- max-beans-in-cache
- max-beans-in-free-pool
- max-messages-in-transaction
- max-queries-in-cache
- max-suspend-seconds
- message-destination-descriptor
- message-destination-name
- message-driven-descriptor
- method
- method-intf
- method-name
- method-param
- method-params
- network-access-point
- passivate-as-principal-name
- persistence
- persistence-use
- persistent-store-dir
- persistent-store-logical-name
- pool
- principal-name
- provider-url
- read-timeout-seconds
- remote-client-timeout
- remove-as-principal-name
- replication-type
- resource-env-ref-name
- res-ref-name

- resource-adapter-jndi-name
- resource-description
- resource-env-description
- resource-link
- retry-count
- retry-methods-on-rollback
- role-name
- run-as-identity-principal
- run-as-principal-name
- run-as-role-assignment
- security-permission
- security-permission-spec
- security-role-assignment
- service-reference-description
- session-timeout-seconds
- singleton-bean-call-router-class-name
- singleton-bean-is-clusterable
- singleton-bean-load-algorithm
- singleton-clustering
- singleton-session-descriptor
- stateful-session-cache
- stateful-session-clustering
- stateful-session-descriptor
- stateless-bean-call-router-class-name
- stateless-bean-is-clusterable
- stateless-bean-load-algorithm
- stateless-clustering
- stateless-session-descriptor
- stick-to-first-server
- timer-descriptor
- timer-implementation
- transaction-descriptor
- transaction-isolation
- transport-requirements
- trans-timeout-seconds
- type-identifier
- type-storage
- type-version

- use-serverside-stubs
- use81-style-polling
- weblogic-compatibility
- weblogic-ejb-jar
- weblogic-enterprise-bean
- work-manager

2.1 weblogic-ejb-jar.xml File Structure

The WebLogic Server `weblogic-ejb-jar.xml` deployment descriptor file describes the elements that are unique to WebLogic Server.

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

- `description`
- `weblogic-enterprise-bean`
  - `ejb-name`
  - `entity-descriptor | stateless-session-descriptor | stateful-session-descriptor | message-driven-descriptor`
  - `transaction-descriptor`
  - `iiop-security-descriptor`
  - `enable-call-by-reference`
  - `network-access-point`
  - `clients-on-same-server`
  - `run-as-principal-name`
  - `create-as-principal-name`
  - `remove-as-principal-name`
  - `passivate-as-principal-name`
  - `jndi-name`
  - `local-jndi-name`
  - `dispatch-policy`
  - `remote-client-timeout`
  - `stick-to-first-server`
- `security-role-assignment`
- `run-as-role-assignment`
- `security-permission`
- `transaction-isolation`
- `message-destination-descriptor`
- `idempotent-methods`

- `retry-methods-on-rollback`
- `enable-bean-class-redeploy`
- `timer-implementation`
- `disable-warning`
- `work-manager`
- `component-factory-class-name`
- `weblogic-compatibility`

2.1 weblogic-ejb-jar.xml Elements

Examine the elements in `weblogic-ejb-jar.xml` that are supported in this release of WebLogic Server.

The following list of the elements in `weblogic-ejb-jar.xml` :

- allow-concurrent-calls
- allow-remove-during-transaction
- cache-between-transactions
- cache-type
- client-authentication
- client-cert-authentication
- clients-on-same-server
- component-factory-class-name
- concurrency-strategy
- confidentiality
- connection-factory-jndi-name
- connection-factory-resource-link
- create-as-principal-name
- delay-updates-until-end-of-tx
- description
- destination-jndi-name
- destination-resource-link
- disable-warning
- dispatch-policy
- distributed-destination-connection
- durable-subscription-deletion
- ejb-name
- ejb-reference-description
- ejb-ref-name
- enable-bean-class-redeploy

- enable-call-by-reference
- enable-dynamic-queries
- entity-always-uses-transaction
- entity-cache
- entity-cache-name
- entity-cache-ref
- entity-clustering
- entity-descriptor
- estimated-bean-size
- externally-defined
- finders-load-bean
- generate-unique-jms-client-id
- global-role
- home-call-router-class-name
- home-is-clusterable
- home-load-algorithm
- idempotent-methods
- identity-assertion
- idle-timeout-seconds
- iiop-security-descriptor
- init-suspend-seconds
- initial-beans-in-free-pool
- initial-context-factory
- integrity
- invalidation-target
- is-modified-method-name
- isolation-level
- jms-client-id
- jms-polling-interval-seconds
- jndi-binding
- jndi-name
- local-jndi-name
- max-beans-in-cache
- max-beans-in-free-pool
- max-messages-in-transaction
- max-queries-in-cache
- max-suspend-seconds
- message-destination-descriptor

- message-destination-name
- message-driven-descriptor
- method
- method-intf
- method-name
- method-param
- method-params
- network-access-point
- passivate-as-principal-name
- persistence
- persistence-use
- persistent-store-dir
- persistent-store-logical-name
- pool
- principal-name
- provider-url
- read-timeout-seconds
- remote-client-timeout
- remove-as-principal-name
- replication-type
- resource-env-ref-name
- res-ref-name
- resource-adapter-jndi-name
- resource-description
- resource-env-description
- resource-link
- retry-count
- retry-methods-on-rollback
- role-name
- run-as-identity-principal
- run-as-principal-name
- run-as-role-assignment
- security-permission
- security-permission-spec
- security-role-assignment
- service-reference-description
- session-timeout-seconds
- singleton-bean-call-router-class-name

- singleton-bean-is-clusterable
- singleton-bean-load-algorithm
- singleton-clustering
- singleton-session-descriptor
- stateful-session-cache
- stateful-session-clustering
- stateful-session-descriptor
- stateless-bean-call-router-class-name
- stateless-bean-is-clusterable
- stateless-bean-load-algorithm
- stateless-clustering
- stateless-session-descriptor
- stick-to-first-server
- timer-descriptor
- timer-implementation
- transaction-descriptor
- transaction-isolation
- transport-requirements
- trans-timeout-seconds
- type-identifier
- type-storage
- type-version
- use-serverside-stubs
- use81-style-polling
- weblogic-compatibility
- weblogic-ejb-jar
- weblogic-enterprise-bean
- work-manager

allow-concurrent-calls

> **Note:**
>
> The `allows-concurrent-calls` element is deprecated in the WebLogic Server EJB 3.1 container. Oracle recommends using the `javax.ejb.AccessTimeout` metadata annotation instead, which enables you to specify the amount of time that a concurrent access attempt should block before timing out. See EJB Metadata Annotations Reference in *Developing Enterprise JavaBeans for Oracle WebLogic Server*.

Range of values: `True | False`

Default value: `False`

Parent elements:

- `weblogic-enterprise-bean`
- `stateful-session-descriptor`

- Function
- Example

Function

Specifies whether a stateful session bean instance allows concurrent method calls. By default, `allows-concurrent-calls` is `False`, in accordance with the EJB specification, and WebLogic Server will throw a `RemoteException` when a stateful session bean instance is currently handling a method call and another (concurrent) method call arrives on the server.

When this value is set to `True`, the EJB container blocks the concurrent method call and allows it to proceed when the previous call has completed.

Example

See stateful-session-descriptor.

allow-remove-during-transaction

Range of values: `True | False`

Default value: `False`

Parent elements:

```
weblogic-enterprise-bean
  stateful-session-descriptor
```

- Function
- Example

Function

Specifies that the `remove` method on a stateful session bean can be invoked within a transaction context.

> **Note:**
>
> Stateful session beans implementing the `Synchronization` interface should not use this tag and then call remove before the transaction ends. If this is done the EJB container will not invoke the synchronization callbacks.

Example

See stateful-session-descriptor.

cache-between-transactions

Range of values: `True | False`

Default value: `False`

Parent elements:

```
weblogic-enterprise-bean
  entity-descriptor
    entity-cache or entity cache-ref
```

- Function
- Example

Function

Formerly the `db-is-shared` element, specifies whether the EJB container will cache the persistent data of an entity bean across (between) transactions.

Specify `True` to enable the EJB container to perform long term caching of the data. Specify `False` to enable the EJB container to perform short term caching of the data.

A Read-Only bean ignores the value of the `cache-between-transactions` element because WebLogic Server always performs long term caching of Read-Only data.

See Limiting Database Reads with cache-between-transactions for more information.

Example

See persistence.

cache-type

Range of values: `NRU | LRU`

Default value: `NRU`

Parent elements:

```
weblogic-enterprise-bean
  stateful-session-cache
```

- Function
- Example

Function

Specifies the order in which EJBs are removed from the cache. The values are:

- Least recently used (LRU)
- Not recently used (NRU)

  The minimum cache size for NRU is 8. If `max-beans-in-cache` is less than 8, WebLogic Server uses a value of 8 for `max-beans-in-cache`.

Example

```
<stateful-session-cache>
    <cache-type>NRU</cache-type>
</stateful-session-cache>
```

client-authentication

Range of values: `none | supported | required`

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  iiop-security-descriptor
```

- Function
- Example

Function

Specifies whether the EJB supports or requires client authentication.

Example

See iiop-security-descriptor.

client-cert-authentication

Range of values: `none | supported | required`

Default value: `n/a`

Parent elements:

```
weblogic-enterprise-bean
  iiop-security-descriptor
    transport-requirements
```

- Function
- Example

Function

Specifies whether the EJB supports or requires client certificate authentication at the transport level.

Example

See transport-requirements.

clients-on-same-server

Range of values: `True | False`

Default value: `False`

Parent element:

`weblogic-enterprise-bean`

Function

Determines whether WebLogic Server sends JNDI announcements for this EJB when it is deployed. When this attribute is `False` (the default), a WebLogic Server cluster automatically updates its JNDI tree to indicate the location of this EJB on a particular server. This ensures that all clients can access the EJB, even if the client is not collocated on the same server.

You can set `clients-on-same-server` to `True` when you know that all clients that will access this EJB will do so from the same server on which the bean is deployed. In this case, a WebLogic Server cluster does not send JNDI announcements for this EJB when it is deployed. Because JNDI updates in a cluster utilize multicast traffic, setting `clients-on-same-server` to `True` can reduce the startup time for very large clusters.

See Optimization for Collocated Objects in *Administering Clusters for Oracle WebLogic Server* for more information on collocated EJBs.

Example

```
<weblogic-enterprise-bean>
    <ejb-name>AccountBean</ejb-name>
    ...
    <clients-on-same-server>True</clients-on-same-server>
</weblogic-enterprise-bean>
```

component-factory-class-name

Range of values: N/A

Default value: null

Parent element:

`weblogic-ejb-jar`

Function

Enable the Spring extension by setting this element to org.springframework.jee.interfaces.SpringComponentFactory. This element exists in EJB, Web, and application descriptors. A module-level descriptor overwrites an application-level descriptor. If the tag is set to null (default), the Spring extension is disabled.

concurrency-strategy

Range of values: `Exclusive | Database | ReadOnly | Optimistic`

Default value: `Database`

Parent elements:

```
weblogic-enterprise-bean
  entity-descriptor
    entity-cache
```

Or

```
weblogic-enterprise-bean
  entity-descriptor
    entity-cache-ref
```


Function

Specifies how the container should manage concurrent access to an entity bean. Set this element to one of four values:

- `Exclusive` causes WebLogic Server to place an exclusive lock on cached entity EJB instances when the bean is associated with a transaction. Other requests for the EJB instance are blocked until the transaction completes. This option was the default locking behavior for WebLogic Server 3.1 through 5.1.
- `Database` causes WebLogic Server to defer locking requests for an entity EJB to the underlying datastore. With the `Database` concurrency strategy, WebLogic Server allocates a separate entity bean instance and allows locking and caching to be handled by the database. This is the default option.
- `ReadOnly` is used for read-only entity beans. Activates a new instance for each transaction so that requests proceed in parallel. WebLogic Server calls `ejbLoad()` for `ReadOnly` beans are based on the `read-timeout-seconds` parameter.
- `Optimistic` holds no locks in the EJB container or database during a transaction. The EJB container verifies that none of the data updated by a transaction has changed before committing the transaction. If any updated data changed, the EJB container rolls back the transaction.

> **Note:**
>
> When a cluster member updates a bean with a `concurrency-strategy` of `Optimistic` that is deployed to a cluster, the EJB container attempts to invalidate all copies of the bean in all servers in the cluster. You can disable this behavior by setting `cluster-invalidation-disabled` in `weblogic-cmp-jar.xml` to `True`. For more information, see Invalidation Options for Optimistic Concurrency in Clusters.

See Choosing a Concurrency Strategy for more information on the `Exclusive` and `Database` locking behaviors. See Read-Write versus Read-Only Entity Beans for more information about `read-only` entity EJBs.

Example

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

confidentiality

Range of values: `none | supported | required`

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  iiop-security-descriptor
    transport-requirements
```

- Function
- Example

Function

Specifies the transport confidentiality requirements for the EJB. Using the `confidentiality` element ensures that the data is sent between the client and server in such a way as to prevent other entities from observing the contents.

Example

See transport-requirements.

connection-factory-jndi-name

Range of values: Valid JNDI name

Default value: If not specified, the default is `weblogic.jms.MessageDrivenBeanConnectionFactory`, which must have been declared in the `JMSConnectionFactory` element in `config.xml`.

Parent elements:

```
weblogic-enterprise-bean
  message-driven-descriptor
```


- Function
- Example

Function

Specifies the JNDI name of the JMS Connection Factory that a message-driven EJB looks up to create its queues and topics. See "Configuring MDBs for Destinations" and "How to Set connection-factory-jndi-name" in

Example

```
<message-driven-descriptor>
    <connection-factory-jndi-name>
    java:comp/env/jms/MyConnectionFactory
    </connection-factory-jndi-name>
</message-driven-descriptor>
```

connection-factory-resource-link

Range of values: Valid resource within a JMS module

Default value: n/a.

Parent elements:

```
weblogic-enterprise-bean
  message-destination-descriptor
```


- Function

Function

Maps to a resource within a JMS module defined in `ejb-jar.xml` to an actual JMS Module Reference in WebLogic Server.

create-as-principal-name

Range of values: Valid principal name

Default value: n/a

Parent element:

```
weblogic-enterprise-bean
```


- Function

Function

Introduced in WebLogic Server 8.1 SP01, specifies the principal to be used in situations where `ejbCreate` would otherwise run with an anonymous principal. Under such conditions, the choice of which principal to run as is governed by the following rule:

if `create-as-principal-name` is set

then use that principal

else

If a `run-as` role has been specified for the bean in `ejb-jar.xml`

then use a principal according to the rules for setting the `run-as-role-assignment`

else

run `ejbCreate` as an anonymous principal.

The `create-as-principal-name` element only needs to be specified if operations within `ejbCreate` require more permissions than the anonymous principal would have.

This element effects the `ejbCreate` methods of stateless session beans and message-driven beans.

See also remove-as-principal-name, passivate-as-principal-name, and principal-name.

delay-updates-until-end-of-tx

Range of values: `True | False`

Default value: `True`

Parent elements:

```
weblogic-enterprise-bean
  entity-descriptor
    persistence
```


Function

Set the `delay-updates-until-end-of-tx` element to `True` (the default) to update the persistent store of all beans in a transaction at the completion of the transaction. This setting generally improves performance by avoiding unnecessary updates. However, it does not preserve the ordering of database updates within a database transaction.

If your datastore uses an isolation level of `TransactionReadUncommitted`, you may want to allow other database users to view the intermediate results of in-progress transactions. In this case, set `delay-updates-to-end-of-tx` to `False` to update the bean's persistent store at the conclusion of each method invoke. See Understanding ejbLoad() and ejbStore() Behavior for more information.

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

Example

```
<entity-descriptor>
    <persistence>
    ...
    ...
    <delay-updates-until-end-of-tx>False</delay-updates-until-end-of-tx>
    </persistence>
</entity-descriptor>
```

description

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-ejb-jar
```

and

```
weblogic-ejb-jar
  transaction-isolation
    method
```

and

```
weblogic-ejb-jar
  idempotent-methods
    method
```

and

```
weblogic-ejb-jar
  retry-methods-on-rollback
```


Function

Describes the parent element.

Example

```
<description>Contains a description of parent element</description>
```

destination-jndi-name

Range of values: Valid JNDI name

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  message-destination-descriptor
```

and

```
weblogic-enterprise-bean
  message-driven-descriptor
```


Function

Specifies the JNDI name used to associate a message-driven bean with an actual JMS Queue or Topic deployed in the WebLogic Server JNDI tree.

Example

See message-destination-descriptor and message-driven-descriptor.

destination-resource-link

Range of values: Valid resource within a JMS module

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  message-destination-descriptor
```


Function

Maps to a resource within a JMS module defined in `ejb-jar.xml` to an actual JMS Module Reference in WebLogic Server.

disable-warning

Range of values: `BEA-010001` | `BEA-010054` | `BEA-010200` | `BEA-010202`

Default value: n/a

Parent element:

```
weblogic-ejb-jar
```

Function

Specifies that WebLogic Server should disable the warning message whose ID is specified. Set this element to one of four values:

- `BEA-010001`—Disables this warning message: "EJB class loaded from system classpath during deployment."
- `BEA-010054`—Disables this warning message: "EJB class loaded from system classpath during compilation."
- `BEA-010200`—Disables this warning message: "EJB impl class contains a public static field, method or class."
- `BEA-010202`—Disables this warning message: "Call-by-reference not enabled."

Example

To disable the warning message: "Call-by-reference not enabled", set `<disable-warning>`, as shown below.

```
<disable-warning>BEA-010202</disable-warning>
```

dispatch-policy

Range of values: Valid execute queue name

Default value: n/a

Parent element:

`weblogic-enterprise-bean`

Function

Designates which server execute thread pool the EJB should run in. Dispatch polices are supported for all types of beans, including entity, session, and message-driven.

If no `dispatch-policy` is specified, or the specified `dispatch-policy` refers to a nonexistent server execute thread pool, then the server's default execute thread pool is used instead.

WebLogic Server ignores `dispatch-policy` if the host server instance does not have an execute thread queue bearing a corresponding name.

If a message-driven bean (MDB) is driven by a foreign (non-WebLogic) destination source, WebLogic Server might ignore `dispatch-policy`, as the MDB may instead run in the foreign provider's threading mechanism. For example, for the IBM WebSphere MQSeries messaging software, `dispatch-policy` is not honored for non-transactional queues; instead the application code runs in an MQSeries thread. For MQSeries transactional queues, and both non-transactional and transactional topics, `dispatch-policy` is honored.

The maximum number of concurrently running MDB instances is designated by a combination of `max-beans-in-free-pool` and `dispatch-policy` values. See MDB Thread Management in *Tuning Performance of Oracle WebLogic Server*.

Example

```
<dispatch-policy>queue_name</dispatch-policy>
```

distributed-destination-connection

Range of values: `Local0nly` | `EveryMember`

Default value: `Local0nly`

Parent elements:

```
weblogic-ejb-jar
  weblogic-enterprise-bean
    message-driven-descriptor
```


Function

> **Note:**
>
> This element is valid for WebLogic JMS 9.0 or later.

Specifies whether an MDB that accesses a WebLogic JMS distributed queue in the same cluster consumes from all distributed destination members or only those members local to the current Weblogic Server.

Valid values include:

- `Local0nly`—Deployment descriptor and message-driven bean are in the same cluster.
- `EveryMember`—Deployment descriptor is on a remote server.

If set to `EveryMember`, the total number of connections will be equal to: `(the number of servers where message-driven bean is deployed) x (the number of destinations)`. For larger deployments, the number of connections may consume a considerable amount of resources.

The `EveryMember` setting incurs additional network and CPU overhead transferring messages from remote servers to the local MDB; it is normally only recommended for limited use cases (such as MDBs with JMS selector filters that are unique to the current server).

Example

```
<distributed-destination-connection>EveryMember</distributed-destination-connection>
```

durable-subscription-deletion

Range of values: `True | False`

Default value: `False`

Parent elements:

```
weblogic-ejb-jar
  weblogic-enterprise-bean
    message-driven-descriptor
```


Function

Indicates whether you want durable topic subscriptions to be automatically deleted when an MDB is undeployed or removed.

Example

```
<durable-subscription-deletion>True</durable-subscription-deletion>
```

ejb-name

Range of values: Name, which conforms to the lexical rules for an NMTOKEN, of an EJB that is defined in `ejb-jar.xml`

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
```

and

```
weblogic-enterprise-bean
  method
```


Function

Specifies an enterprise bean's name, using the same name for the bean that is specified in `ejb-jar.xml`. The enterprise bean code does not depend on the name; therefore the name can be changed during the application assembly process without breaking the enterprise bean's function. There is no architected relationship between the `ejb-name` in the deployment descriptor and the JNDI name that the Deployer will assign to the enterprise bean's home.

> **Note:**
>
> `Not recommended in weblogic-enterprise-bean`. For more information, see Using EJB Links.

Example

See method.

ejb-reference-description

Range of values: n/a

Default value: n/a

Parent element:

`weblogic-enterprise-bean`

- Function
- Example

Function

Maps the JNDI name of an EJB in WebLogic Server to the name by which it is specified in the `ejb-ref-name` element in `ejb-jar.xml`.

Example

```
<ejb-reference-description>
    <ejb-ref-name>AdminBean</ejb-ref-name>
    <jndi-name>payroll.AdminBean</jndi-name>
</ejb-reference-description>
```

ejb-ref-name

Range of values: Valid `ejb-ref-name` specified in the associated `ejb-jar.xml` file

Default value: n/a

Parent elements:

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

- Function
- Example

Function

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

Example

```
<ejb-reference-description>
    <ejb-ref-name>AdminBean</ejb-ref-name>
    <jndi-name>payroll.AdminBean</jndi-name>
</ejb-reference-description>
```

enable-bean-class-redeploy

Range of values: `True | False`

Default value: `False`

Parent element:

`weblogic-enterprise-jar`

Function

By default, the EJB implementation class is loaded in the same classloader as the rest of the EJB classes. When the `enable-bean-class-redeploy` element is enabled, the implementation class, along with its super classes, gets loaded in a child classloader of the EJB module classloader. This allows the EJB implementation class to be redeployed without redeploying the entire EJB module.

There are some potential problems with loading the bean class in a child classloader. First, the bean class will no longer be visible to any classes loaded in the parent classloader, so those classes cannot refer to the bean class or errors will occur. Also, the bean class will not be able to invoke any package protected methods on any classes loaded in a different classloader. So, if your bean class invokes a helper class in the same package, the helper class methods must be declared public or `IllegalAccessErrors` will result.

> **Note:**
>
> This element is deprecated for EJB 3.0. Starting with WebLogic Server 10.3.0, you can replace this feature with FastSwap. See Using FastSwap Deployment to Minimize Redeployment in *Deploying Applications to Oracle WebLogic Server*.

Example

The following XML element enables redeployment of an individual bean class:

```
<enable-bean-class-redeploy>True</enable-bean-class-redeploy>
```

enable-call-by-reference

Range of values: `True | False`

Default value: `False`

Parent element:

`weblogic-enterprise-bean`

Function

When `enable-call-by-reference` is `False`, parameters to the EJB methods are copied—or passed by value—regardless of whether the EJB is called remotely or from within the same EAR.

When `enable-call-by-reference` is `True`, EJB methods called from within the same EAR file or standalone JAR file will pass arguments by reference. This improves the performance of method invocation since parameters are not copied.

> **Note:**
>
> When an EJB is called remotely, method parameters are *always* passed by value. Remote calls may be between applications on different JVMs or between applications on the same JVM.
>
> For example, suppose two applications named `EJBApp1.ear`, and `EJBApp2.ear` are deployed on the same server. `EJBApp1.ear` includes `EJB1`, and `EJBApp2.ear` includes `EJB2`. In this situation, the calls between `EJB1` and `EJB2` are considered remote calls even though they are on the same JVM.

Example

```
<weblogic-enterprise-bean>
        <ejb-name>AccountBean</ejb-name>

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

</weblogic-enterprise-bean>
```

enable-dynamic-queries

Range of values: `True | False`

Default value: `True`

Parent elements:

```
weblogic-enterprise-bean
  entity-descriptor
```

Function

Set to `True` to enable dynamic queries. Dynamic queries are only available for use with EJB 2.x CMP beans.

Example

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

entity-always-uses-transaction

Range of values: `True | False`

Default value: `False`

Parent elements:

```
weblogic-ejb-jar
  weblogic-compatibility
```


Function

This element, introduced in WebLogic Server 9.0, allows you to specify whether an entity bean must always use a transaction. Before WebLogic Server 9.0, when an entity bean ran in an unspecified transaction, the EJB container would create a transaction for the entity bean. Now, the EJB container no longer creates a transaction when an entity bean runs in an unspecified transaction. To disable this behavior and cause the EJB container to create a transaction for entity beans that run in unspecified transaction, set the value of this element to `True`.

entity-cache

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  entity-descriptor
```


Function

Defines the following options used to cache entity EJB instances within WebLogic Server:

- `max-beans-in-cache`
- `idle-timeout-seconds`
- `read-timeout-seconds`

- `concurrency-strategy`

See Understanding Entity Caching for more information.

Example

```
<entity-descriptor>
    <entity-cache>
        <max-beans-in-cache>...</max-beans-in-cache>
        <idle-timeout-seconds>...</idle-timeout-seconds>
        <read-timeout-seconds>...<read-timeout-seconds>
        <concurrency-strategy>...</concurrency-strategy>
    </entity-cache>
    <persistence>...</persistence>
    <entity-clustering>...</entity-clustering>
</entity-descriptor>
```

entity-cache-name

Range of values: Name assigned to an application level entity cache in the `weblogic-application.xml` file

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  entity-descriptor
    entity-cache-ref
```


Function

Refers to an application level entity cache that the entity bean uses. An application level cache is a cache that may be shared by multiple entity beans in the same application. The value you specify for `entity-cache-name` must match the name assigned to an application level entity cache in the `weblogic-application.xml` file.

For more information about the `weblogic-application.xml` file, see Enterprise Application Deployment Descriptor Elements in *Developing Applications for Oracle WebLogic Server*.

Example

See entity-cache-ref.

entity-cache-ref

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  entity-descriptor
```

Function

Refers to an application level entity cache which can cache instances of multiple entity beans that are part of the same application. Application level entity caches are declared in the `weblogic-application.xml` file.

Use concurrency-strategy to define the type of concurrency you want the bean to use. The `concurrency-strategy` must be compatible with the application level cache's caching strategy. For example, an `Exclusive` cache only supports beans with a `concurrency-strategy` of `Exclusive`. A `MultiVersion` cache supports the `Database`, `ReadOnly`, and `Optimistic` concurrency strategies.

Example

```
<entity-cache-ref>
    <entity-cache-name>AllEntityCache</entity-cache-name>
    <read-timeout-seconds>600</read-timeout-seconds>
    <cache-between-transactions>true</cache-between-transactions>
    <concurrency-strategy>ReadOnly</concurrency-strategy>
    <estimated-bean-size>20</estimated-bean-size>
</entity-cache-ref>
```

entity-clustering

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  entity-descriptor
```


Function

Specifies how an entity bean will be replicated in a WebLogic cluster:

- `home-is-clusterable`
- `home-load-algorithm`
- `home-call-router-class-name`
- `use-serverside-stubs`

Example

```
<entity-clustering>
    <home-is-clusterable>True</home-is-clusterable>
    <home-load-algorithm>random</home-load-algorithm>
    <home-call-router-class-name>beanRouter</home-call-router-class-name>
```

```
    <use-serverside-stubs>True</use-serverside-stubs>
</entity-clustering>
```

entity-descriptor

Range of values: n/a

Default value: n/a

Parent element:

`weblogic-enterprise-bean`

- Function
- Example

Function

Specifies the following deployment parameters that are applicable to an entity bean:

- `pool`
- `timer-descriptor`
- `entity-cache or entity-cache-ref`
- `persistence`
- `entity-clustering`
- `invalidation-target`
- `enable-dynamic-queries`

Example

```
<entity-descriptor>
    <pool>...</pool>
    <timer-descriptor>...</timer-descriptor>
    <entity-cache>...</entity-cache>
    <persistence>...</persistence>
    <entity-clustering>...</entity-clustering>
    <invalidation-target>...</invalidation-target>
    <enable-dynamic-queries>...</enable-dynamic-queries>
</entity-descriptor>
```

estimated-bean-size

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  entity-descriptor
```


- Function
- Example

Function

Specifies the estimated average size of the instances of an entity bean in bytes. This is the average number of bytes of memory that is consumed by each instance.

Use the `estimated-bean-size` element when the application level cache you use to cache beans is also specified in terms of bytes and megabytes.

Although you may not know the exact number of bytes consumed by the entity bean instances, specifying a size allows you to give some relative weight to the beans that share a cache at one time.

For example, suppose bean A and bean B share a cache, called AB-cache, that has a size of 1000 bytes and the size of A is 10 bytes and the size of B is 20 bytes, then the cache can hold at most 100 instances of A and 50 instances of B. If 100 instances of A are cached, this implies that 0 instances of B are cached.

Example

See entity-cache-ref.

externally-defined

Range of values: `True | False`

Default value: none

Parent elements:

```
weblogic-ejb-jar
  security-role-assignment
```

- Function

Function

Indicates that a particular security role is defined externally in a security realm, outside of the deployment descriptor. Because the security role and its principal-name mapping is defined elsewhere, principal-names are not to be specified in the deployment descriptor. This tag is used as an indicative placeholder instead of a set of `principal-name` elements. Use this element instead of global-role, which has been deprecated and was removed from WebLogic Server in release 9.0.

finders-load-bean

Range of values: `True | False`

Default value: `True`

Parent elements:

```
weblogic-enterprise-bean
  entity-descriptor
    persistence
```

- Function

Function

Valid only for CMP entity EJBs. The `finders-load-bean` element determines whether WebLogic Server loads the EJB into the cache after a call to a finder method returns a reference to the bean. If you set this element to `True`, WebLogic Server immediately loads the bean into the cache if a reference to a bean is returned by the finder. If you set this element to `False`, WebLogic Server does not automatically load the bean into the cache until the first method invocation; this behavior is consistent with the EJB 1.1 specification.

Example

```
<entity-descriptor>
    <persistence>
        <finders-load-bean>True</finders-load-bean>
    </persistence>
</entity-descriptor>
```

generate-unique-jms-client-id

Range of values: `True | False`

Default value: `False`

Parent elements:

```
weblogic-ejb-jar
  weblogic-enterprise-bean
    message-driven-descriptor
```


Function

Indicates whether or not you want the EJB container to generate a unique client-id for every instance of an MDB. Enabling this flag makes it easier to deploy durable MDBs to multiple server instances in a WebLogic Server cluster.

global-role

The `global-role` element is deprecated and was removed from WebLogic Server in release 9.0. Use the `externally-defined` element instead.

home-call-router-class-name

Range of values: Valid name of a custom class

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  entity-descriptor
    entity-clustering
```

and

```
weblogic-enterprise-bean
  stateful-session-descriptor
    stateful-session-clustering
```

and

```
weblogic-enterprise-bean
  stateless-session-descriptor
    stateless-session-clustering
```


Function

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

Example

See entity-clustering and stateful-session-clustering.

home-is-clusterable

Range of values: `True | False`

Default value: `True`

Parent elements:

```
weblogic-enterprise-bean
  entity-descriptor
    entity-clustering
```

and

```
weblogic-enterprise-bean
  stateful-session-descriptor
    stateful-session-clustering
```

and

```
weblogic-enterprise-bean
  stateful-session-descriptor
    stateless-clustering
```

Function

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

Example

See entity-clustering.

home-load-algorithm

Range of values: `round-robin | random | weight-based | RoundRobinAffinity | RandomAffinity | WeightBasedAffinity`

Default value: Value of `weblogic.cluster.defaultLoadAlgorithm`

Parent elements:

```
weblogic-enterprise-bean
  entity-descriptor
    entity-clustering
```

and

```
weblogic-enterprise-bean
  stateful-session-descriptor
    stateful-session-clustering
```

and

```
weblogic-enterprise-bean
  entity-descriptor
    entity-clustering
```


Function

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

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

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

- `round-robin-affinity`—server affinity governs connections between external Java clients and server instances; round robin load balancing is used for connections between server instances.
- `weight-based-affinity`—server affinity governs connections between external Java clients and server instances; weight-based load balancing is used for connections between server instances.
- `random-affinity`—server affinity governs connections between external Java clients and server instances; random load balancing is used for connections between server instances.

See Load Balancing for EJBs and RMI Objects in *Administering Clusters for Oracle WebLogic Server*.

Example

See entity-clustering and stateful-session-clustering.

idempotent-methods

Range of values: n/a

Default value: n/a

Parent element:

`weblogic-ejb-jar`

Function

Defines list of methods of a clustered EJB which are written in such a way that repeated calls to the same method with the same arguments has exactly the same effect as a single call. This allows the failover handler to retry a failed call without knowing whether the call actually compiled on the failed server. When you enable idempotent-methods for a method, the EJB stub can automatically recover from any failure as long as it can reach another server hosting the EJB.

Clustering must be enabled for the EJB. To enable clustering, see entity-clustering, stateful-session-clustering, and stateless-clustering.

The methods on stateless session bean homes and read-only entity beans are automatically set to be idempotent. It is not necessary to explicitly specify them as idempotent.

Example

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

identity-assertion

Range of values: `none | supported | required`

Default value: none

Parent elements:

```
weblogic-enterprise-bean
  iiop-security-descriptor
```


- Function
- Example

Function

Specifies whether the EJB supports or requires identity assertion.

Example

See iiop-security-descriptor.

idle-timeout-seconds

Range of values: `0` to `maxSeconds`, where `0` denotes unlimited and `maxSeconds` is the maximum value of an int

Default value: `600`

Parent elements:

```
weblogic-enterprise-bean
  entity-descriptor
    entity-cache
```

and

```
weblogic-enterprise-bean
  entity-descriptor
    entity-cache-ref
```

and

```
weblogic-enterprise-bean
  stateful-session-descriptor
    stateful-session-cache
```

and

```
weblogic-enterprise-bean
  stateless-session-descriptor or message-driven-descriptor or entity-descriptor
    pool
```


- Function
- Example

Function

Defines the maximum length of time an EJB should remain in the cache. After this time has elapsed, WebLogic Server removes the bean instance if the number of beans in cache approaches the limit of `max-beans-in-cache`. The removed bean instances are passivated. See Caching and Passivating Stateful Session EJBs and Managing Entity Bean Pooling and Caching for more information.

Also defines the maximum length of time an EJB should remain idle in the free pool before it is removed. After this time has elapsed, WebLogic Server removes the bean instance from the free pool so long as doing so will not cause the number of beans in the pool to fall below the number specified in `initial-beans-in-free-pool`.

> **Note:**
>
> Although `idle-timeout-seconds` appears in the `entity-cache` element, WebLogic Server 8.1 SP1 and SP2 do not use its value in managing the life cycle of entity EJBs—in those service packs, `idle-timeout-seconds` has no effect on when entity beans are removed from cache.

Example

The following entry indicates that the stateful session EJB, `AccountBean`, should become eligible for removal if `max-beans-in-cache` is reached and the bean has been in cache for 20 minutes:

```
<weblogic-enterprise-bean>
    <ejb-name>AccountBean</ejb-name>
    <stateful-session-descriptor>
        <stateful-session-cache>
            <max-beans-in-cache>200</max-beans-in-cache>
            <idle-timeout-seconds>1200</idle-timeout-seconds>
        </stateful-session-cache>
    </stateful-session-descriptor>
</weblogic-enterprise-bean>
```

iiop-security-descriptor

Range of values: n/a

Default value: n/a

Parent element:

`weblogic-enterprise-bean`

Function

Specifies security configuration parameters at the bean level. These parameters determine the IIOP security information contained in the IOR.

Example

```
<iiop-security-descriptor>
    <transport-requirements>...</transport-requirements>
    <client-authentication>supported<client-authentication>
    <identity-assertion>supported</identity-assertion>
</iiop-security-descriptor>
```

init-suspend-seconds

Range of values: any integer

Default value: `5`

Parent elements:

```
weblogic-enterprise-bean
  message-driven-descriptor
```


Function

Specifies the initial number of seconds to suspend an MDB's JMS connection when the EJB container detects a JMS resource outage. See Configuring Suspension of Message Delivery During JMS Resource Outages in *Developing Message-Driven Beans for Oracle WebLogic Server*.

initial-beans-in-free-pool

Range of values: `0` to `maxBeans`

Default value: `0`

Parent elements:

```
weblogic-enterprise-bean
  stateless-session-descriptor or message-driven-descriptor or entity-descriptor
    pool
```


Function

If you specify a value for `initial-beans-in-free-pool`, you set the initial size of the pool. WebLogic Server populates the free pool with the specified number of bean instances for every bean class at startup. Populating the free pool in this way improves initial response time for the EJB, because initial requests for the bean can be satisfied without generating a new instance.

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

Example

See pool.

initial-context-factory

Range of values: Valid name of an initial context factory

Default value: `weblogic.jndi.WLInitialContextFactory`

Parent elements:

```
weblogic-enterprise-bean
  message-destination-descriptor
```

and

```
weblogic-enterprise-bean
  message-driven-descriptor
```


Function

Specifies the initial context factory used by the JMS provider to create initial context. See Configuring MDBs for Destinations and How to Set initial-context-factory in *Developing Message-Driven Beans for Oracle WebLogic Server*.

Example

```
<message-driven-descriptor>
    <initial-context-factory>fiorano.jms.rtl.FioranoInitialContextFactory
    </initial-context-factory>
</message-driven-descriptor>
```

See also message-destination-descriptor.

integrity

Range of values: `none | supported | required`

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  iiop-security-descriptor
    transport-requirements
```


Function

Specifies the transport integrity requirements for the EJB. Using the `integrity` element ensures that the data is sent between the client and server in such a way that it cannot be changed in transit.

Example

See transport-requirements.

invalidation-target

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  entity-descriptor
```


Function

Specifies a Read-Only entity EJB that should be invalidated when this container-managed persistence entity EJB has been modified.

The target `ejb-name` must be a Read-Only entity EJB and this element can only be specified for an EJB 2.x container-managed persistence entity EJB.

Example

```
<invalidation-target>
    <ejb-name>StockReaderEJB</ejb-name>
</invalidation-target>
```

is-modified-method-name

Range of values: Valid entity EJB method name

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  entity-descriptor
    persistence
```


Function

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

Providing a method and setting it as appropriate can improve performance for EJB 1.1-compliant beans, and for beans that use bean-managed persistence. However, any errors in the method's return value can cause data inconsistency problems.

> **Note:**
>
> `isModified()` is no longer required for 2.x CMP entity EJBs based on the EJB 2.x specification. However, it still applies to BMP and 1.1 CMP EJBs. When you deploy EJB 2.x entity beans with container-managed persistence, WebLogic Server automatically detects which EJB fields have been modified, and writes only those fields to the underlying datastore.

Example

```
<entity-descriptor>
    <persistence>
        <is-modified-method-name>semidivine</is-modified-method-name>
    </persistence>
</entity-descriptor>
```

isolation-level

Range of values: `TransactionSerializable | TransactionReadCommitted | TransactionReadUncommitted | TransactionRepeatableRead | TransactionReadCommittedForUpdate | TransactionReadCommittedForUpdateNoWait`

Default value: Default value of the underlying database

Parent elements:

```
weblogic-ejb-jar
  transaction-isolation
```

Function

Defines method-level transaction isolation settings for an EJB. Allowable values include:

- `TransactionSerializable`—Simultaneously executing this transaction multiple times has the same effect as executing the transaction multiple times in a serial fashion.

  > **Note:**
  >
  > For Oracle databases, Oracle recommends that you use the `TransactionReadCommittedForUpdate` isolation level instead of the `TransactionSerializable` isolation level. This is because Oracle databases do not lock read data at the `TransactionSerializable` isolation level. Additionally, at the `TransactionSerializable` isolation level, it is possible for concurrent transactions on Oracle databases to proceed without throwing the Oracle exception ORA-08177 "`can't serialize access for this transaction`"). For more information on the `TransactionReadCommittedForUpdate` isolation level, see Oracle Database-Only Isolation Levels.

- `TransactionReadCommitted`—The transaction can view only committed updates from other transactions.
- `TransactionReadUncommitted`—The transaction can view uncommitted updates from other transactions.
- `TransactionRepeatableRead`—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.

Oracle Database-Only Isolation Levels

These addition values are supported only for Oracle databases, and only for container-managed persistence (CMP) EJBs:

- `TransactionReadCommittedForUpdate`— Supported only for Oracle databases, for container-managed persistence (CMP) EJBs only. This value sets the isolation level to `TransactionReadCommitted`, and for the duration of the transaction, all SQL `SELECT` statements executed in any method are executed with `FOR UPDATE` appended to them. This causes the secluded rows to be locked for update. If the Oracle database cannot lock the rows affected by the query immediately, then it waits until the rows are free. This condition remains in effect until the transaction does a `COMMIT` or `ROLLBACK`.

  This isolation level can be used to avoid the error:

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

  which can (but does not always) occur when using the `TransactionSerializable` isolation level with Oracle databases.

> **Note:**
>
> For Oracle databases, Oracle recommends that you use this isolation level (`TransactionReadCommittedForUpdate`) instead of the `TransactionSerializable` isolation level. This is because Oracle databases do not lock read data at the `TransactionSerializable` isolation level.

- `TransactionReadCommittedForUpdateNoWait`—Supported only for Oracle databases, for container-managed persistence (CMP) EJBs only.

  This value sets the isolation level to `TransactionReadCommitted`, and for the duration of the transaction, all SQL `SELECT` statements executed in any method are executed with FOR UPDATE NO WAIT appended to them. This causes the selected rows to be locked for update.

  In contrast to the `TransactionReadCommittedForUpdate` setting, `TransactionReadCommittedForUpdateNoWait` causes the Oracle DBMS to `NOT WAIT` if the required locks cannot be acquired immediately—the affected `SELECT` query will fail and an exception will be thrown by the container.

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

Example

See transaction-isolation.

jms-client-id

Range of values: n/a

Default value: `ejb-name` for the EJB

Parent element:

`message-driven-descriptor`

Function

Specifies a client ID for the MDB when it connects to a JMS destination. Required for durable subscriptions to JMS topics.

If you specify the connection factory that the MDB uses in `connection-factory-jndi-name`, the client ID can be defined in the `ClientID` element of the associated `JMSConnectionFactory` element in `config.xml`.

If `JMSConnectionFactory` in `config.xml` does not specify a `ClientID`, or if you use the default connection factory, (you do not specify `connection-factory-jndi-name`) the message-driven bean uses the `jms-client-id` value as its client id.

Example

```
<jms-client-id>MyClientID</jms-client-id>
```

jms-polling-interval-seconds

Range of values: n/a

Default value: 10 seconds

Parent element:

`message-driven-descriptor`

Function

Specifies the number of seconds between each attempt to reconnect to the JMS destination. Each message-driven bean listens on an associated JMS destination. If the JMS destination is located on another WebLogic Server instance or a foreign JMS provider, then the JMS destination may become unreachable. In this case, the EJB container automatically attempts to reconnect to the JMS Server. Once the JMS Server is up again, the message-driven bean can again receive messages.

Example

```
<jms-polling-interval-seconds>5</jms-polling-interval-seconds>
```

jndi-binding

Range of values: Custom JNDI name

Default value: n/a

Parent elements:

`weblogic-enterprise-bean`

and

```
weblogic-enterprise-bean
  resource-description
```

and

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

Function

Specifies the custom JNDI name that can be applied to a bean class or business interface to indicate a JNDI name for a certain client view. When applied to a bean class to indicate the JNDI name of a no-interface view, the `className` is optional.

Example

See resource-description and ejb-reference-description.

jndi-name

> **Note:**
>
> The `jndi-name` element is deprecated in this release of WebLogic Server. Use `jndi-binding` instead.

Range of values: Valid JNDI name

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
```

and

```
weblogic-enterprise-bean
  resource-description
```

and

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


Function

Specifies the JNDI name of an actual EJB, resource, or reference available in WebLogic Server.

> **Note:**
>
> Assigning a JNDI name to a bean is not recommended. Global JNDI names generate heavy multicast traffic during clustered server startup. See Using EJB Links for the better practice. If you have an EAR library that contains EJBs, you cannot deploy multiple applications that reference the library because attempting to do so will result in a JNDI name conflict. This is because global JNDI name cannot be set for individual EJBs in EAR libraries; it can only be set for an entire library.

Example

See resource-description and ejb-reference-description.

local-jndi-name

> **Note:**
>
> The `local-jndi-name` element is deprecated in this release of WebLogic Server. Use `jndi-binding` instead.

Range of values: Valid JNDI name

Default value: n/a

Parent elements:

`weblogic-enterprise-bean`

Function

JNDI name for a bean's local home. If a bean has both a remote and a local home, then it can be assigned two JNDI names; one for each home.

> **Note:**
>
> Assigning a JNDI name to a bean is not recommended. See Using EJB Links for the better practice.

Example

```
<local-jndi-name>weblogic.jndi.WLInitialContext
</local-jndi-name>
```

max-beans-in-cache

Range of values: `1` to `maxBeans`

Default value: `1000`

Parent elements:

```
weblogic-enterprise-bean
  entity-descriptor
    entity-cache
```

and

```
weblogic-enterprise-bean
  stateful-session-descriptor
    stateful-session-cache
```


Function

Specifies the maximum number of objects of this class that are allowed in memory. When `max-bean-in-cache` is reached, WebLogic Server passivates some EJBs that have not recently been used by a client. `max-beans-in-cache` also affects when EJBs are removed from the WebLogic Server cache, as described in Caching and Passivating Stateful Session EJBs.

Example

```
<weblogic-enterprise-bean>
    <ejb-name>AccountBean</ejb-name>
    <entity-descriptor>
        <entity-cache>
            <max-beans-in-cache>200</max-beans-in-cache>
        </entity-cache>
    </entity-descriptor>
</weblogic-enterprise-bean>
```

max-beans-in-free-pool

Range of values: `0` to `maxBeans`

Default value: `1000`

Parent elements:

```
weblogic-enterprise-bean
  stateless-session-descriptor
    pool
```

and

```
weblogic-enterprise-bean
  message-driven-descriptor
    pool
```

and

```
weblogic-enterprise-bean
  entity-descriptor
    pool
```

- Function
- Example

Function

WebLogic Server maintains a free pool of EJBs for every entity bean, stateless session bean, and message-driven bean class. The `max-beans-in-free-pool` element defines the size of this pool.

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

Example

See pool.

max-messages-in-transaction

Range of values: All positive integers

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  message-driven-descriptor
```

- Function

Function

Specifies the maximum number of messages that can be in a transaction for this MDB.

max-queries-in-cache

Range of values: All positive integers

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  entity-descriptor
```

- Function

Function

This element, introduced in WebLogic Server 9.0, specifies the maximum number of read-only entity queries to cache at the bean level. For information on caching read-only entity queries at the application level, see Using Query Caching (Read-Only Entity Beans).

max-suspend-seconds

Range of values: Any integer

Default value: `60`

Parent elements:

```
weblogic-enterprise-bean
  message-driven-descriptor
```

- Function

Function

Specifies the maximum number of seconds to suspend an MDB's JMS connection when the EJB container detects a JMS resource outage. To disable JMS connection suspension when the EJB container detects a JMS resource outage, set the value of this element to `0`. See Configuring Suspension of Message Delivery During JMS Resource Outages in *Developing Message-Driven Beans for Oracle WebLogic Server*.

message-destination-descriptor

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
```

- Function
- Example

Function

Maps a message destination reference in the `ejb-jar.xml` file to an actual message destination, such as a JMS Queue or Topic, in WebLogic Server.

Example

```
<message-destination-descriptor>
    <message-destination-name>...</message-destination-name>
    <destination-jndi-name>...</destination-jndi-name>
    <resource-link>...</resource-link>
    <initial-context-factory>...</initial-context-factory>
```

```
    <provider-url>...</provider-url>
</message-destination-descriptor>
```

message-destination-name

Range of values: A valid message destination reference name from the `ejb-jar.xml` file

Default value: n/a

Requirements: This element is required if the EJB specifies messages destination references in `ejb-jar.xml`.

Parent elements:

```
weblogic-enterprise-bean
  message-destination-descriptor
```


Function

Specifies the name of a message destination reference. This is the reference that the EJB provider places within the `ejb-jar.xml` deployment file.

Example

See message-destination-descriptor.

message-driven-descriptor

Range of values: n/a

Default value: n/a

Parent element:

```
weblogic-enterprise-bean
```


Function

Associates a message-driven bean with a JMS destination in WebLogic Server.

Example

```
<message-driven-descriptor>
    <pool>...</pool>
    <timer-descriptor>...</timer-descriptor>
    <destination-jndi-name>...</destination-jndi-name>
    <initial-context-factory>...</initial-context-factory>
    <provider-url>...</provider-url>
    <connection-factory-jndi-name>...</connection-factory-jndi-name>
```

```
    <jms-polling-interval-seconds>...</jms-polling-interval-seconds>
    <jms-client-id>...</jms-client-id>
    <generate-unique-jms-client-id>...</generate-unique-jms-client-id>
    <durable-subscription-deletion>...</durable-subscription-deletion>
    <max-messages-in-transaction>...</max-messages-in-transaction>
    <distributed-destination-connection>...</distributed-destination-connection>
    <use81-style-polling>...</use81-style-polling>
    <init-suspend-seconds>...</init-suspend-seconds>
    <max-suspend-seconds>...</max-suspend-seconds>
</message-driven-descriptor>
```

method

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-ejb-jar
  transaction-isolation
```

and

```
weblogic-ejb-jar
  idempotent-methods
```

and

```
weblogic-ejb-jar
  retry-methods-on-rollback
```


Function

Defines a method or set of methods for an enterprise bean's home or remote interface.

Example

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

method-intf

Range of values: `Home | Remote | Local | Localhome`

Default value: n/a

Parent elements:

```
weblogic-ejb-jar
  transaction-isolation
    method
```

and

```
weblogic-ejb-jar
  idempotent-methods
    method
```

- Function
- Example

Function

Specifies the EJB interface to which WebLogic Server applies isolation level properties, if the method has the same signature in multiple interfaces.

Example

See method.

method-name

Range of values: Name of an EJB defined in `ejb-jar.xml | *`

Default value: n/a

Parent elements:

```
weblogic-ejb-jar
  transaction-isolation
    method
```

and

```
weblogic-ejb-jar
  idempotent-methods
    method
```

- Function
- Example

Function

Specifies the name of an individual EJB method to which WebLogic Server applies isolation level properties. Use the asterisk (*) to specify all methods in the EJB's home and remote interfaces.

If you specify a `method-name`, the method must be available in the specified `ejb-name`.

Example

See method.

method-param

Range of values: Fully-qualified Java type name of a method parameter

Default value: n/a

Parent elements:

```
weblogic-ejb-jar
  transaction-isolation
    method
      method-params
```

and

```
weblogic-ejb-jar
  idempotent-methods
    method
      method-params
```


Function

Specifies the fully-qualified Java type name of a method parameter.

Example

See method-params.

method-params

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-ejb-jar
  transaction-isolation
    method
```

and

```
weblogic-ejb-jar
  idempotent-methods
    method
```


Function

Contains one or more elements that define the Java type name of each of the method's parameters.

Example

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

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

network-access-point

Range of values: Name of a custom network access point

Default value: n/a

Parent elements:

`weblogic-enterprise-bean`

Function

Assigns a custom network channel that the EJB will use for network communications. A network channel defines a set of connection attributes. See Configuring Network Resources in *Administering Server Environments for Oracle WebLogic Server*.

Example

```
<weblogic-enterprise-bean>
    <network-access-point>SSLChannel</network-access-point>
</weblogic-enterprise-bean>
```

passivate-as-principal-name

Range of values: Valid WebLogic Server principal

Default value: n/a

Parent elements:

`weblogic-enterprise-bean`

Function

The `passivate-as-principal-name` element, introduced in WebLogic Server 8.1 SP01, specifies the principal to be used in situations where `ejbPassivate` would otherwise run with an anonymous principal. Under such conditions, the choice of which principal to run as is governed by the following rule:

If `passivate-as-principal-name` is set

then use that principal

else

if a `run-as` role has been specified for the bean in `ejb-jar.xml`

then use a principal according to the rules for setting the `run-as-role-assignment`

else

run `ejbPassivate` as an anonymous principal.

The `passivate-as-principal-name` element only needs to be specified if operations within `ejbPassivate` require more permissions than the anonymous principal would have.

This element affects the `ejbPassivate` methods of stateless session beans when passivation occurs due to a cache timeout.

See also remove-as-principal-name, create-as-principal-name, and principal-name.

persistence

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  entity-descriptor
```


Function

Required only for entity EJBs that use container-managed persistence services. The `persistence` element defines the following options that determine the persistence type, transaction commit behavior, and `ejbLoad()` and `ejbStore()` behavior for entity EJBs in WebLogic Server:

- `is-modified-method-name`
- `delay-updates-until-end-of-tx`
- `finders-load-bean`
- `persistence-use`

Example

```
<entity-descriptor>
    <persistence>
        <is-modified-method-name>...</is-modified-method-name>
        <delay-updates-until-end-of-tx>...</delay-updates-until-end-of-tx>
        <finders-load-beand>...</finders-load-bean>
        <persistence-use>...</persistence-use>
    </persistence>
</entity-descriptor>
```

persistence-use

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  entity-descriptor
    persistence
```


Function

Required only for entity EJBs that use container-managed persistence services. The `persistence-use` element stores an identifier of the persistence type to be used for this particular bean.

Example

```
<persistence-use>
    <type-identifier>WebLogic_CMP_RDBMS</type-identifier>
    <type-version>5.1.0</type-version>
    <type-storage>META-INF/weblogic-cmp-jar.xml</type-storage>
</persistence-use>
```

persistent-store-dir

Range of values: Valid file system directory

Default value: `pstore`

Parent elements:

```
weblogic-enterprise-bean
  stateful-session-descriptor
```


Function

Specifies a file system directory where WebLogic Server stores the state of passivated stateful session bean instances. For more information, see Specifying the Persistent Store Directory for Passivated Beans.

Example

```
<stateful-session-descriptor>
    <stateful-session-cache>...</stateful-session-cache>
    <allow-concurrent-calls>...</allow-concurrent-calls>
```

```
    <persistent-store-dir>MyPersistenceDir</persistent-store-dir>
    <stateful-session-clustering>...</stateful-session-clustering>
    <allow-remove-during-transaction>
</stateful-session-descriptor>
```

persistent-store-logical-name

Range of values: Valid name of a persistent store on the file system

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  entity-descriptor
    timer-descriptor
```

and

```
weblogic-enterprise-bean
  message-driven-descriptor
    timer-descriptor
```


Function

Specifies the name of a persistent store on the server's file system where WebLogic Server stores timer objects. If you do not specify a persistent store name in this element, WebLogic Server stores timer objects in the default store.

pool

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  stateless-session-descriptor or message-driven-descriptor
```

and

```
  entity-descriptor
```


Function

Configures the behavior of the WebLogic Server free pool for entity EJBs, stateless session EJBs, and message-driven EJBs. The options are:

- `max-beans-in-free-pool`
- `initial-beans-in-free-pool`
- `idle-timeout-seconds`

Example

```
<stateless-session-descriptor>
    <pool>
        <max-beans-in-free-pool>500</max-beans-in-free-pool>
        <initial-beans-in-free-pool>250</initial-beans-in-free-pool>
    </pool>
</stateless-session-descriptor>
```

principal-name

Range of values: Valid WebLogic Server principal name

Default value: n/a

Parent elements:

```
weblogic-ejb-jar
  security-role-assignment
```


Function

Specifies the name of an actual WebLogic Server principal to apply to the specified `role-name`. At least one `principal-name` is required in the `security-role-assignment` element. You may define more than one `principal-name` for each `role-name`.

Example

See security-role-assignment.

provider-url

Range of values: Valid URL

Default value: `t3://localhost:7001`

Parent elements:

```
weblogic-enterprise-bean
  message-destination-descriptor
```

and

```
weblogic-enterprise-bean
  message-driven-descriptor
```

Function

Specifies the URL to be used by the `InitialContext`. See Configuring MDBs for Destinations and How to Set provider-urlin *Developing Message-Driven Beans for Oracle WebLogic Server*.

Example

```
<message-driven-descriptor>
    <provider-url>WeblogicURL:Port</provider-url>
</message-driven-descriptor>
```

See also message-destination-descriptor.

read-timeout-seconds

Range of values: `0` to `maxSeconds`, where `maxSeconds` is the maximum value of an int

Default value: `600`

Parent elements:

```
weblogic-enterprise-bean
  entity-descriptor
    entity-cache
```

or

```
weblogic-enterprise-bean
  entity-descriptor
    entity-cache-ref
```


Function

Specifies the number of seconds between `ejbLoad()` calls on a `Read-Only` entity bean. A value of 0 causes WebLogic Server to call `ejbLoad()` only when the bean is brought into the cache.

Example

The following entry causes WebLogic Server to call `ejbLoad()` for instances of the `AccountBean` class only when the instance is first brought into the cache:

```
<weblogic-enterprise-bean>
    <ejb-name>AccountBean</ejb-name>
    <entity-descriptor>
        <entity-cache>
            <read-timeout-seconds>0</read-timeout-seconds>
        </entity-cache>
    </entity-descriptor>
</weblogic-enterprise-bean>
```

remote-client-timeout

Range of values: `0` to `maxSeconds`, where `maxSeconds` is the maximum value of an int

Default value: `0`

Parent elements:

`weblogic-enterprise-bean`

Function

Specifies the length of time that a remote RMI client will wait before it will time out. See Using a Read Timeout in *Developing RMI Applications for Oracle WebLogic Server*.

Example

The following entry causes a remote RMI client to timeout after waiting 5 seconds.

```
<weblogic-enterprise-bean>
    <ejb-name>AccountBean</ejb-name>
    ...
    <remote-client-timeout>5</remote-client-timeout>
</weblogic-enterprise-bean>
```

remove-as-principal-name

Range of values: n/a

Default value: n/a

Parent element:

`weblogic-enterprise-bean`

Function

This parameter only needs to be specified if operations within `ejbRemove` need more permissions than the anonymous principal would have.

The `remove-as-principal-name` element, introduced in WebLogic Server 8.1 SP1, specifies the principal to be used in situations where `ejbRemove` would otherwise run with an anonymous principal. Under such conditions, the choice of which principal to run as is governed by the following rule:

If `remove-as-principal-name` is set

then use that principal

else

if a `run-as` role has been specified for the bean in `ejb-jar.xml`

then use a principal according to the rules for setting the `run-as-role-assignment`

else

run `ejbRemove` as an anonymous principal

The `remove-as-principal-name` element only needs to be specified if operations within `ejbRemove` require more permissions than the anonymous principal would have.

This element effects the `ejbRemove` methods of stateless session beans and message-drive beans.

See also passivate-as-principal-name, create-as-principal-name, and principal-name.

replication-type

Range of values: `InMemory | None`

Default value: `None`

Parent elements:

```
weblogic-enterprise-bean
  stateful-session-descriptor
    stateful-session-clustering
```

- Function
- Example

Function

Determines whether WebLogic Server replicates the state of stateful session EJBs across WebLogic Server instances in a cluster. If you select `InMemory`, the state of the EJB is replicated. If you select `None`, the state is not replicated.

Example

See stateful-session-clustering.

resource-env-ref-name

Range of values: A valid resource environment reference name from the `ejb-jar.xml` file

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  resource-env-description
```

- Function
- Example

Function

Specifies the name of a resource environment reference.

Example

See resource-description.

res-ref-name

Range of values: A valid resource reference name from the `ejb-jar.xml` file

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  resource-description
```


Function

Specifies the name of a `resourcefactory` reference. This is the reference that the EJB provider places within the `ejb-jar.xml` deployment file. Required element if the EJB specifies resource references in `ejb-jar.xml`

Example

See resource-description.

resource-adapter-jndi-name

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-ejb-jar
  weblogic-enterprise-bean
    message-driven-descriptor
```


Function

Identifies the resource adapter that this MDB receives messages from.

resource-description

Range of values: n/a

Default value: n/a

Parent element:

`weblogic-enterprise-bean`

Function

Maps a resource reference defined in `ejb-jar.xml` to the JNDI name of an actual resource available in WebLogic Server.

Example

```
<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>
```

resource-env-description

Range of values: n/a

Default value: n/a

Parent element:

`weblogic-enterprise-bean`

Function

Maps a resource environment reference defined in `ejb-jar.xml` to the JNDI name of an actual resource available in WebLogic Server.

Example

```
<resource-env-description>
    <resource-env-ref-name>. . .</resource-env-ref-name>
    <jndi-name>...</jndi-name>
</reference-env-description>
```

When `jndi-name` is not a valid URL, WebLogic Server treats it as a object that maps to a URL and is already bound in the JNDI tree, and binds a `LinkRef` with that `jndi-name`.

resource-link

Range of values: Valid resource within a JMS module

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  message-destination-descriptor
```


Function

Maps to a resource within a JMS module defined in `ejb-jar.xml` to an actual JMS Module Reference in WebLogic Server.

Example

See message-destination-descriptor.

retry-count

Range of values: Any positive integer

> **Note:**
>
> While it is possible to set this value to less than or equal to 0, Oracle recommends that you do not do so because the EJB container will not retry transactions when this value is not greater than or equal to 1.

Default value: n/a

Parent elements:

```
weblogic-ejb-jar
  retry-methods-on-rollback
```


Function

Specifies the number of times you want the EJB container to automatically retry a container-managed transaction method that has rolled back.

retry-methods-on-rollback

Range of values: n/a

Default value: n/a

Parent element:

```
weblogic-ejb-jar
```


- Function

Function

Specifies the methods for which you want the EJB container to automatically retry container-managed transactions that have rolled back. Automatic transaction retry is supported for session and entity beans that use container-managed transaction demarcation. Additionally, regardless of the methods specified in this element, the EJB container does not retry transactions that fail because of system exception-based errors.

role-name

Range of values: Valid application role name

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  security-role-assignment
```


- Function
- Example

Function

Identifies an application role name that the EJB provider placed in the `ejb-jar.xml` deployment descriptor. Subsequent `principal-name` elements in the element map WebLogic Server principals to the specified `role-name`.

Example

See security-role-assignment.

run-as-identity-principal

Range of values: Valid security principal name

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
```


- Function
- Example

Function

> **Note:**
>
> The `run-as-identity-principal` element is deprecated in this release of WebLogic Server. Use `run-as-principal-name` instead.

The `run-as-identity-principal` element specifies which security principal name is to be used as the run-as principal for a bean that has specified a security identity `run-as-role-name` its `ejb-jar.xml` deployment descriptor.

For an explanation of how the mapping of run-as role-names to run-as-identity-principals or run-as-principal-names occurs, see the comments for the `run-as-role-assignment` element.

Example

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

run-as-principal-name

Range of values: Valid principal

Default value: n/a

Parent elements:

`weblogic-enterprise-bean`

Function

Specifies which security principal name is to be used as the run-as principal for a bean that has specified a `security-identity` run-as `role-name` in its `ejb-jar.xml` deployment descriptor.

For an explanation of how the mapping of run-as `role-names` to `run-as-principal-names` occurs, see the comments for the `run-as-role-assignment` element.

Example

```
<run-as-principal-name>
    Fred
</run-as-principal-name>
```

run-as-role-assignment

Range of values: n/a

Default value: n/a

Parent elements:

`weblogic-enterprise-bean`

Function

Maps a given security-identity run-as `role-name` specified in the `ejb-jar.xml` deployment descriptor file to a `run-as-principal-name`.

The value of the `run-as-principal-name` for a given role-name that is specified here is scoped to all beans in the `ejb-jar.xml` deployment descriptor; it applies to all beans that specify that role-name as their security-identity `run-as-role-name`.

The `run-as-principal-name` value specified here can be overridden at the individual bean level by specifying a run-as-principal-name under that bean's `weblogic-enterprise-bean` element.

> **Note:**
>
> For a given bean, if there is no `run-as-principal-name` specified in either a `run-as-role-assignment` or in a bean specific `run-as-principal-name` tag, then the EJB container chooses the first `principal-name` of a security user in the `weblogic-enterprise-bean security-role-assignment` for the `role-name` and uses that `principal-name` as the `run-as-principal-name`.

Example

Suppose that in the `ejb-jar.xml` deployment descriptor file:

- Beans 'A_EJB_with_runAs_role_X' and 'B_EJB_with_runAs_role_X'

  specify a security-identity run-as role-name 'runAs_role_X'.
- Bean 'C_EJB_with_runAs_role_Y'

  specifies a security-identity run-as role-name 'runAs_role_Y'.

Consider the following excerpts from the corresponding `weblogic-ejb-jar.xml` deployment descriptor file:

```
<weblogic-ejb-jar>
    <weblogic-enterprise-bean>
        <ejb-name>
            A_EJB_with_runAs_role_x
        </ejb-name>
    </weblogic-enterprise-bean>
    <weblogic-enterprise-bean>
        <ejb-name>
            B_EJB_with_runAs_role_X
        </ejb-name>
        <run-as-principal-name>
            Joe
        </run-as-principal-name>
    </weblogic-enterprise-bean>
    <weblogic-enterprise-bean>
        <ejb-name>
            C_EJB_with_runAs_role_Y
        </ejb-name>
    </weblogic-enterprise-bean>
    <security-role-assignment>
        <role-name>
            runAs_role_Y
        </role-name>
        <principal-name>
            first_principal_of_role_Y
        </principal-name>
        <principal-name>
            second_principal_of_role_Y
        </principal-name>
    </security-role-assignment>
    <run-as-role-assignment>
        <role-name>
            runAs_role_X
        </role-name>
        <run-as-principal-name>
            Fred
        </run-as-principal-name>
    </run-as-role-assignment>
</weblogic-ejb-jar>
```

Each of the beans chooses a different principal name to use as its `run-as-principal-name`.

A_EJB_with_runAs_role_X

This bean's run-as role-name is 'runAs_role_X'. The jar scoped `<run-as-role-assignment>` mapping will be used to look up the name of the principal to use.

The `<run-as-role-assignment>` mapping specifies that for `<role-name>` 'runAs_role_X' we are to use `<run-as-principal-name>` 'Fred'.

"Fred" is the principal name that will be used.

B_EJB_with_runAs_role_X

This bean's run-as role-name is also 'runAs_role_X'. This bean will not use the jar scoped `<run-as-role-assignment>` to look up the name of the principal to use because that value is overridden by this bean's `<weblogic-enterprise-bean> <run-as-principal-name>` value 'Joe'.

"Joe" is the principal name that will be used.

C_EJB_with_runAs_role_Y

This bean's run-as role-name is 'runAs_role_Y'. There is no explicit mapping of 'runAs_role_Y' to a run-as principal name; that is, there is no jar-scoped `<run-as-role-assignment>` for 'runAs_role_Y' nor is there a bean scoped `<run-as-principal-name>` specified in this bean's `weblogic-enterprise-bean`.

To determine the principal name to use, the `<security-role-assignment>` for `<role-name>` "runAs_role_Y" is examined. The first `<principal-name>` corresponding to a User (i.e. not a Group) is chosen.

"first_principal_of_role_Y" is the principal name that will be used.

security-permission

Range of values: n/a

Default value: n/a

Parent elements:

`weblogic-ejb-jar`

Function

Specifies a security permission that is associated with a Java EE Sandbox.

For more information, see the security permission specification:

http://docs.oracle.com/javase/8/docs/technotes/guides/security/PolicyFiles.html

Example

```
<security-permission>
    <description>Optional explanation goes here</description>
    <security-permission-spec>
    ...
    </security-permission-spec>
</security-permission>
```

security-permission-spec

Range of values: n/a

Default value: n/a

Parent element:

`security-permission`

Function

Specifies a single security permission based on the Security policy file syntax.

For more information, see the security permission specification:

http://docs.oracle.com/javase/8/docs/technotes/guides/security/PolicyFiles.html

Example

To grant the "read" permission to "java.vm.version," and prevent it from being overwritten:

1. Set the `security-permission-spec` as shown below:

```
<security-permission>
    <description>Optional explanation goes here</description>
    <security-permission-spec> grant { permission     java.util.PropertyPermission
"java.vm.version", "read"; };
    </security-permission-spec>
</security-permission>
```

2. Modify the `startWeblogic` script to start the server using this option:

```
JAVA_OPTIONS=-Djava.security.manager
```

3. Create a directory named `lib` in your domain directory.
4. Add this line to the `%WL_HOME%\server\lib\weblogic.policy` file:

```
add grant codeBase "file:/<Your user_projects dir>/YourDomain/lib/-" { permission
java.security.AllPermission; };
```

   This is necessary because the EJB stub's classpath is `lib`.

security-role-assignment

Range of values: n/a

Default value: n/a

Parent elements:

`weblogic-ejb-jar`

Function

Maps application roles in the `ejb-jar.xml` file to the names of security principals available in WebLogic Server.

Example

```
<security-role-assignment>
    <role-name>PayrollAdmin</role-name>
    <principal-name>Tanya</principal-name>
    <principal-name>system</principal-name>
```

```
    <externally-defined>True</externally-defined>
    ...
</security-role-assignment>
```

service-reference-description

Range of values: n/a

Default value: n/a

Parent element:

`weblogic-enterprise-bean`

Function

Maps a Web Service destination reference in the `ejb-jar.xml` file to an actual Web Service in WebLogic Server.

- `wsdl-url` — the url of the dynamic `wsdl` of the referenced web service used by the client to create stub to invoke remote web service.
- `port-info` — defines `wsdl:port` information specified in `wsdl`.
- `port-name` — defines the local name of `wsdl:port`.
- `stub-property` — property to be set on the client-side stub, it has the same effect of as `javax.xml.rpc.Stub._setProperty(prop_name, prop_value)`.

Example

```
<service-reference-description>
    <service-ref-name>service/WebServiceBroker</service-ref-name>
    <wsdl-url>
        http://@PROXY_SERVER@/webservice/BrokerServiceBean?wsdl
    </wsdl-url>
    <call-property>...</call-property>
    <port-info>
        <port-name>BrokerServiceIntfPort</port-name>
        <stub-property>
            <name>javax.xml.rpc.service.endpoint.address</name>
            <value>http://@PROXY_SERVER@/webservice/BrokerServiceBean<;/value>
        </stub-property>
    </port-info>
</service-reference-description>
```

session-timeout-seconds

Range of values: (None specified)

Default value: `idle-timeout-seconds`

Parent elements:

```
weblogic-enterprise-bean
  stateful-session-descriptor
    stateful-session-cache
```


Function

Determines how long the EJB container leaves a passivated stateful session bean on disk. The container removes a passivated EJB `session-timeout-seconds` after passivating the bean instance to disk. If `session-timeout-seconds` is not specified, the default is the value specified by `idle-timeout-seconds`.

Example

```
<stateful-session-descriptor>
    <stateful-session-cache>
        <max-beans-in-cache>4</max-beans-in-cache>
        <idle-timeout-seconds>5</idle-timeout-seconds>
        <session-timeout-seconds>120</session-timeout-seconds>
        <cache-type>LRU</cache-type>
    </stateful-session-cache>
</stateful-session-descriptor>
```

singleton-bean-call-router-class-name

Range of values: Valid custom class name

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  singleton-session-descriptor
    singleton-clustering
```


Function

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

Example

See singleton-clustering.

singleton-bean-is-clusterable

Range of values: `true | false`

Default value: `false`

Parent elements:

```
weblogic-enterprise-bean
  singleton-session-descriptor
```


Function

Marks the remote business views of the bean as clusterable and supports load balancing and failover.

When `singleton-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 singleton-clustering.

singleton-bean-load-algorithm

Range of values: `round-robin | random | weight-based | RoundRobinAffinity | RandomAffinity | WeightBasedAffinity`

Default value: Value of `weblogic.cluster.defaultLoadAlgorithm`

Parent elements:

```
weblogic-enterprise-bean
  singleton-session-descriptor
    singleton-clustering
```


Function

Specifies the algorithm to use for load balancing between replicas of the EJB home.

You can define `singleton-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`—Load balancing is performed randomly among the servers hosting the bean.

- `weight-based`—Load balancing is performed according to the servers' current workload.
- `round-robin-affinity`—Server affinity governs connections between external Java clients and server instances; round robin load balancing is used for connections between server instances.
- `weight-based-affinity`—Server affinity governs connections between external Java clients and server instances; weight-based load balancing is used for connections between server instances.
- `random-affinity`—Server affinity governs connections between external Java clients and server instances; random load balancing is used for connections between server instances.

See Load Balancing for EJBs and RMI Objects in *Administering Clusters for Oracle WebLogic Server*.

Example

See singleton-clustering.

singleton-clustering

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  singleton-session-descriptor
```


Function

Specifies options that determine how WebLogic Server replicates singleton session EJB instances in a cluster.

Example

```
<singleton-clustering>
    <singleton-bean-is-clusterable>
        True
    </singleton-bean-is-clusterable>
    <singleton-bean-load-algorithm>
        random</singleton-bean-load-algorithm>
    <singleton-bean-call-router-class-name>
        beanRouter
    </singleton-bean-call-router-class-name>
    <singleton-bean-methods-are-idempotent>
        True
    </singleton-bean-methods-are-idempotent>
</singleton-clustering>
```

singleton-session-descriptor

Range of values: n/a

Default value: n/a

Parent element:

```
weblogic-enterprise-bean
```


Function

Defines deployment behaviors, such as caching, clustering, and persistence, for singleton session EJBs in WebLogic Server.

Example

```
<singleton-session-descriptor>
    <pool>...</pool>
    <singleton-clustering>...</singleton-clustering>
</singleton-session-descriptor>
```

stateful-session-cache

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  stateful-session-descriptor
```


Function

Defines the following options used to cache stateful session EJB instances.

- `max-beans-in-cache`
- `idle-timeout-seconds`
- `session-timeout-sections`
- `cache-type`

See Caching and Passivating Stateful Session EJBs for more information about caching of stateful session beans.

Example

The following example shows how to specify the `stateful-session-cache` element

```
<stateful-session-cache>
    <max-beans-in-cache>...</max-beans-in-cache>
    <idle-timeout-seconds>...</idle-timeout-seconds>
    <session-timeout-seconds>...</session-timeout-seconds>
    <cache-type>...</cache-type>
</stateful-session-cache>
```

stateful-session-clustering

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  stateful-session-descriptor
```


Function

Specifies the following options that determine how WebLogic Server replicates stateful session EJB instances in a cluster:

- `home-is-clusterable`
- `home-load-algorithm`
- `home-call-router-class-name`
- `replication-type`

Example

```
<stateful-session-clustering>
    <home-is-clusterable>True</home-is-clusterable>
    <home-load-algorithm>random</home-load-algorithm>
    <home-call-router-class-name>beanRouter</home-call-router-class-name>
    <replication-type>InMemory</replication-type>
</stateful-session-clustering>
```

stateful-session-descriptor

Range of values: n/a

Default value: n/a

Parent element:

```
weblogic-enterprise-bean
```

Function

Defines deployment behaviors, such as caching, clustering, and persistence, for stateless session EJBs in WebLogic Server.

Example

```
<stateful-session-descriptor>
    <stateful-session-cache>...</stateful-session-cache>
    <allow-concurrent-calls>...</allow-concurrent-calls>
    <allow-remove-during-transaction>...
    </allow-remove-during-transaction>
    <persistent-store-dir>/myPersistenceStore</persistent-store-dir>
    <stateful-session-clustering>...</stateful-session-clustering>
</stateful-session-descriptor>
```

stateless-bean-call-router-class-name

Range of values: Valid custom class name

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  stateless-session-descriptor
    stateless-clustering
```


Function

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

Example

See stateless-clustering.

stateless-bean-is-clusterable

Range of values: `True | False`

Default value: `True`

Parent elements:

```
weblogic-enterprise-bean
  stateless-session-descriptor
    stateless-clustering
```


Function

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

Example

See stateless-clustering.

stateless-bean-load-algorithm

Range of values: `round-robin | random | weight-based | RoundRobinAffinity | RandomAffinity | WeightBasedAffinity`

Default value: Value of `weblogic.cluster.defaultLoadAlgorithm`

Parent elements:

```
weblogic-enterprise-bean
  stateless-session-descriptor
    stateless-clustering
```


Function

Specifies the algorithm to use for load balancing between replicas of the EJB home.

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

- `round-robin`—Load balancing is performed in a sequential fashion among the servers hosting the bean.
- `random`—Load balancing is performed randomly among the servers hosting the bean.
- `weight-based`—Load balancing is performed according to the servers' current workload.
- `round-robin-affinity`—Server affinity governs connections between external Java clients and server instances; round robin load balancing is used for connections between server instances.
- `weight-based-affinity`—Server affinity governs connections between external Java clients and server instances; weight-based load balancing is used for connections between server instances.

- `random-affinity`—Server affinity governs connections between external Java clients and server instances; random load balancing is used for connections between server instances.

See Load Balancing for EJBs and RMI Objects in *Administering Clusters for Oracle WebLogic Server*.

Example

See stateless-clustering.

stateless-clustering

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  stateless-session-descriptor
```


Function

Specifies options that determine how WebLogic Server replicates stateless session EJB instances in a cluster.

Example

```
<stateless-clustering>
    <stateless-bean-is-clusterable>
        True
    </stateless-bean-is-clusterable>
    <stateless-bean-load-algorithm>
        random</stateless-bean-load-algorithm>
    <stateless-bean-call-router-class-name>
        beanRouter
    </stateless-bean-call-router-class-name>
    <stateless-bean-methods-are-idempotent>
        True
    </stateless-bean-methods-are-idempotent>
</stateless-clustering>
```

stateless-session-descriptor

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
```

Function

Defines deployment parameters, such as caching, clustering, and persistence for stateless session EJBs in WebLogic Server.

Example

```
<stateless-session-descriptor>
    <pool>...</pool>
    <stateless-clustering>...</stateless-clustering>
</stateless-session-descriptor>
```

stick-to-first-server

Range of values: True or False

Default value: False

Parent element:

```
weblogic-enterprise-bean
```


Function

Defines "sticky" load balancing in a cluster. The server chosen for servicing the first request is used for all subsequent requests.

Example

```
<stick-to-first-server>
    True
</stick-to-first-server>
```

timer-descriptor

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  entity-descriptor
```

or

```
  message-driven-descriptor
```

or

```
singleton-session-descriptor
```

or

```
stateless-session-descriptor
```

- Function

Function

Specifies the persistent store that will be used to store the local timer information for the bean.

timer-implementation

Range of values:

Valid values include:

- `Clustered`—Specifies that timers are cluster aware. This option provides EJB timers with features such as automatic failover, load balancing, and improved accessibility within the cluster. This option is recommended for EJBs that will be deployed to a cluster of WebLogic servers.
- `Local`—Specifies that timers will only execute on the server on which they are created and are only visible to the beans on that server. When you set this element to `Local`, if you deploy an EJB to a cluster, and invoke the EJB to create a timer, that call could go to any server on the cluster. Another invocation (for instance, to cancel the timer) could also go to any server on the cluster and will not necessarily go to the same server in which the call to create the timer went. For instance, if the call to create a timer is directed to `server1`, and the call to cancel it is directed to `server2`, the EJB on `server2` would not see the timer on `server1` and would, therefore, fail to cancel it.

To avoid the limitation when using local timers, you can use one of the following approaches:

- Pin the EJB deployment to a single server in the cluster. This causes all calls to the EJB to go to server to which the EJB is pinned and all timers to exist on that same server. The trade-off to using this approach is that the EJB cannot take advantage of clustering benefits such as load balancing and failover.
- Ensure that calls to cancel timers go to all servers in the cluster by using a message-driven bean (MDB) that listens on a JMS topic. The message to cancel the timer can be published to the JMS topic and serviced by an MDB on each server. Then, the MDB on each server can invoke a `cancelTimer` method on the bean. The trade-off to using this approach is that it makes your application more complex and attempting to cancel timers on all servers is inefficient.

Default value: `Local` (the current file store-backed EJB timer service is used)

Parent element:

```
weblogic-ejb-jar
```

- Function
- Example

Function

Specifies whether the EJB timer service is cluster aware.

Example

```
<timer-implementation>
    Clustered
</timer-implementation>
```

transaction-descriptor

Range of values: n/a

Default value: n/a

Parent elements:

`weblogic-enterprise-bean`

Function

Specifies options that define transaction behavior in WebLogic Server. Currently, this element includes only one child element: `trans-timeout-seconds`.

Example

```
<transaction-descriptor>
<trans-timeout-seconds>20</trans-timeout-seconds>
</transaction-descriptor>
```

transaction-isolation

Range of values: n/a

Default value: n/a

Parent elements:

`weblogic-ejb-jar`

Function

Defines method-level transaction isolation settings for an EJB.

Example

```
<transaction-isolation>
    <isolation-level>...</isolation-level>
    <method>
        <description>...</description>
```

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

For more information, see isolation-level.

transport-requirements

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  iiop-security-descriptor
```

- Function
- Example

Function

Provides the transport requirements for the EJB.

Example

```
<iiop-security-descriptor>
    <transport-requirements>
        <confidentiality>supported</confidentiality>
        <integrity>supported</integrity>
        <client-cert-authentication>suppoted
        </client-cert-authentication>
    </transport-requirements>
</iiop-security-descriptor>
```

trans-timeout-seconds

Range of values: `0` to `max`

Default value: From domain-level JTA configuration.

Parent elements:

```
weblogic-enterprise-bean
  transaction-descriptor
```

- Function
- Example

Function

Specifies the maximum duration for an EJB's container-initiated transactions. If a transaction lasts longer than `trans-timeout-seconds`, WebLogic Server rolls back the transaction.

Example

See transaction-descriptor.

type-identifier

Range of values: Valid string. `WebLogic_CMP_RDBMS` specifies WebLogic Server RDBMS-based persistence.

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  entity-descriptor
    persistence
      persistence-use
```


Function

Required only for entity EJBs that use container-managed persistence services. Specifies an entity EJB persistence type. WebLogic Server RDBMS-based persistence uses the identifier, `WebLogic_CMP_RDBMS`. If you use a different persistence vendor, consult the vendor's documentation for information on the correct `type-identifier`.

Example

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

type-storage

Range of values: Valid string

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  entity-descriptor
    persistence
      persistence-use
```

Function

Required only for entity EJBs that use container-managed persistence services. Defines the full path of the file that stores data for this persistence type. The path must specify the file's location relative to the top level of the EJB's JAR deployment file or deployment directory.

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

Example

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

type-version

Range of values: Valid string

Default value: n/a

Parent elements:

```
weblogic-enterprise-bean
  entity-descriptor
    persistence
      persistence-use
```


Function

Required for entity EJBs that use container-managed persistence service, if multiple versions of the same persistence type are installed. Identifies the version of the specified persistence type. For example, for WebLogic 2.0 CMP persistence, use the value:

```
6.0
```

For WebLogic 1.1 CMP persistence, use the value:

5.1.0

This element is necessary if multiple versions of the same persistence type are installed.

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

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

Example

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

use-serverside-stubs

Range of values: `true | false`

Default value: `false`

Parent elements:

```
weblogic-enterprise-bean
  entity-descriptor
    entity-clustering
```

and

```
weblogic-enterprise-bean
  stateful-session-descriptor
    stateful-session-clustering
```

and

```
weblogic-enterprise-bean
  entity-descriptor
    entity-clustering
```


Function

Causes the bean home to use server-side stubs in the context of server.

Example

See the example for entity-clustering.

use81-style-polling

Range of values: `true | false`

Default value: `false`

Parent elements:

```
weblogic-ejb-jar
  weblogic-enterprise-bean
    message-driven-descriptor
```

Function

Enables backwards compatibility for WLS Version 8.1-style polling.

In WLS version 8.1 and earlier, transactional MDBs with batching enabled created a dedicated polling thread for each deployed MDB. This polling thread was not allocated from the pool specified by `dispatch-policy`, it was an entirely new thread in addition to the all other threads running on the system. See Backwards Compatibility for WLS 10.0 and Earlier-style Polling in *Tuning Performance of Oracle WebLogic Server*.

Example

See the example for entity-clustering.

weblogic-compatibility

Range of values: n/a

Default value: n/a

Parent elements:

`weblogic-ejb-jar`

- Function

Function

This element, introduced in WebLogic Server 9.0 contains elements that specify compatibility flags.

weblogic-ejb-jar

Range of values: n/a

Default value: n/a

Parent elements: n/a

- Function

Function

`weblogic-ejb-jar` is the root element of the WebLogic component of the EJB deployment descriptor.

weblogic-enterprise-bean

Range of values: n/a

Default value: n/a

Parent elements:

`weblogic-ejb-jar`

- Function

Function

Contains the deployment information for a bean that is available in WebLogic Server.

work-manager

Range of values: n/a

Default value: n/a

Parent elements:

`weblogic-ejb-jar`

- Function

Function

Specifies a work manager to manage work requests for EJBs.

For more information on work managers, see Using Work Managers to Optimize Scheduled Work in *Administering Server Environments for Oracle WebLogic Server*.

C

weblogic-cmp-jar.xml Deployment Descriptor Reference

Learn about the EJB 2.1 elements in the `weblogic-cmp-jar.xml` file, the WebLogic-specific XML Schema-based (XSD) deployment descriptor file. Use these definitions to create the WebLogic-specific `weblogic-cmp-jar.xml` file that is part of your EJB deployment. Use this deployment descriptor file to specify container-managed-persistence (CMP) behavior.

> **Note:**
>
> If you are using metadata annotations in your EJB 3.x implementation, refer to EJB Metadata Annotations Reference in *Developing Enterprise JavaBeans for Oracle WebLogic Server*.

The WebLogic Server `weblogic-cmp-jar.xml` file is XML Schema-based (XSD). In pre-9.0 releases of WebLogic Server, `weblogic-cmp-jar.xml` was Document Type Definition-based (DTD). For backward compatibility, WebLogic Server still supports XSD- or DTD-based EJB descriptors; you can deploy applications that use DTD-based descriptors in WebLogic Server without modifying the descriptors.

For information on the EJB 1.1 deployment descriptor elements see Important Information for EJB 1.1 Users.

For information on:

- XML Schema Definitions and the namespace declaration required in `weblogic-cmp-jar.xml`, as well as Document Type Definitions and DOCTYPE headers, see Deployment Descriptor Schema and Document Type Definitions Reference.
- the `weblogic-ejb-jar.xml` file, see weblogic-ejb-jar.xml Deployment Descriptor Reference.

EJB 1.1 deployment descriptor elements, see Important Information for EJB 1.1 Users.

This appendix includes the following topics:

- cmr-field
- column-map
- compatibility
- create-default-dbms-table
- database-specific-sql
- database-type
- data-source-jndi-name
- db-cascade-delete
- dbms-column
- dbms-column-type
- dbms-default-value
- default-dbms-tables-ddl
- delay-database-insert-until
- description
- disable-string-trimming
- ejb-name
- ejb-ql-query
- enable-batch-operations
- enable-query-caching
- field-group
- field-map
- finders-return-nulls
- foreign-key-column
- foreign-key-table
- generator-name
- generator-type
- group-name
- include-updates
- instance-lock-order
- key-cache-size
- key-column
- lock-order
- max-elements
- method-name
- method-param
- method-params
- optimistic-column
- order-database-operations

- pass-through-columns
- primary-key-table
- query-method
- relation-name
- relationship-caching
- relationship-role-map
- relationship-role-name
- serialize-byte-array-to-oracle-blob
- serialize-char-array-to-bytes
- sql
- sql-query
- sql-select-distinct
- sql-shape
- sql-shape-name
- table-map
- table-name
- trigger-updates-optimistic-column
- unknown-primary-key-field
- use-select-for-update
- validate-db-schema-with
- verify-columns
- verify-rows
- version-column-initial-value
- weblogic-ql
- weblogic-query
- weblogic-rdbms-bean
- weblogic-rdbms-jar
- weblogic-rdbms-relation
- weblogic-relationship-role

2.1 weblogic-cmp-jar.xml Deployment Descriptor File Structure

The `weblogic-cmp-jar.xml` file defines deployment descriptors for entity EJBs that use WebLogic Server RDBMS-based persistence services. The EJB container uses a version of `weblogic-cmp-jar.xml` that is different from the XML shipped with pre-9.0 releases of WebLogic Server.

You can continue to use the earlier `weblogic-cmp-jar.xml` DTD beans that you deploy to this release of WebLogic Server; likewise, you can continue to use the `weblogic-cmp-jar.xml` DTD that was supported in WebLogic Server 8.1. Though deployment descriptors are XSD-based beginning with WebLogic Server 9.0, for backward compatibility, WebLogic Server

continues to support DTD-based descriptors. However, if you want to use any of the CMP 2.1 features, you must use the XSD described below.

Oracle recommends that you run DDConverter to convert EJB deployment descriptors from pre-9.0 releases of WebLogic Server to conform to the current release of WebLogic Server. DDConverter converts your DTD-based EJB deployment descriptors from pre-9.0 releases of WebLogic Server to XSD-based descriptors supported in this release.

> **Note:**
>
> Oracle recommends that you always convert descriptors when migrating applications to a new WebLogic Server release.

The top-level element of the WebLogic Server `weblogic-cmp-jar.xml` consists of a `weblogic-rdbms-jar` element:

```
weblogic-rdbms-jar
    weblogic-rdbms-bean
    ejb-name
    data-source-jndi-name
    table-map
    field-group
    relationship-caching
    sql-shape
    weblogic-query
    delay-database-insert-until
    use-select-for-update
    lock-order
    instance-lock-order
    automatic-key-generation
    check-exists-on-method
    cluster-invalidation-disabled

weblogic-rdbms-relation
    relation-name
    table-name
     weblogic-relationship-role
order-database-operations
enable-batch-operations
create-default-dbms-tables
validate-db-schema-with
database-type
default-dbms-tables-ddl

compatibility
    serialize-byte-array-to-oracle-blob
    serialize-char-array-to-bytes
    allow-readonly-create-and-remove
    disable-string-trimming
```

2.1 weblogic-cmp-jar.xml Deployment Descriptor Elements

This list of the elements in `weblogic-cmp-jar.xml` includes all elements that are supported in WebLogic Server.

- allow-readonly-create-and-remove

- automatic-key-generation
- caching-element
- caching-name
- check-exists-on-method
- cluster-invalidation-disabled
- cmp-field
- cmr-field
- column-map
- compatibility
- create-default-dbms-table
- database-specific-sql
- database-type
- data-source-jndi-name
- db-cascade-delete
- dbms-column
- dbms-column-type
- dbms-default-value
- default-dbms-tables-ddl
- delay-database-insert-until
- description
- disable-string-trimming
- ejb-name
- ejb-ql-query
- enable-batch-operations
- enable-query-caching
- field-group
- field-map
- finders-return-nulls
- foreign-key-column
- foreign-key-table
- generator-name
- generator-type
- group-name
- include-updates
- instance-lock-order
- key-cache-size
- key-column
- lock-order

- max-elements
- method-name
- method-param
- method-params
- optimistic-column
- order-database-operations
- pass-through-columns
- primary-key-table
- query-method
- relation-name
- relationship-caching
- relationship-role-map
- relationship-role-name
- serialize-byte-array-to-oracle-blob
- serialize-char-array-to-bytes
- sql
- sql-query
- sql-select-distinct
- sql-shape
- sql-shape-name
- table-map
- table-name
- trigger-updates-optimistic-column
- unknown-primary-key-field
- use-select-for-update
- validate-db-schema-with
- verify-columns
- verify-rows
- version-column-initial-value
- weblogic-ql
- weblogic-query
- weblogic-rdbms-bean
- weblogic-rdbms-jar
- weblogic-rdbms-relation
- weblogic-relationship-role

allow-readonly-create-and-remove

Range of values: `true | false`

Default value: `false`

Parent elements:

```
weblogic-rdbms-jar
  compatibility
```


Function

This element, introduced in WebLogic Server 8.1 SP02, is a backward compatibility flag. It is used to enable create and remove operations for an EJB that uses `ReadOnly` concurrency.

Prior to WebLogic Server 8.1 SP2, these operations were allowed, although they had no transactional meaning. They have been disallowed so that more efficient code can be generated for `ReadOnly` beans, and because using them is a bad practice.

Example

```
<compatibility>
    <allow-readonly-create-and-remove>
        true
    </allow-readonly-create-and-remove>
</compatibility>
```

automatic-key-generation

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-rdbms-bean
```


Function

The `automatic-key-generation` element specifies how primary keys will be automatically generated. For more information about this feature, see Automatically Generating Primary Keys.

Example

The following code samples show the `automatic-key-generation` element for different primary key generation methods. For supported generation methods, see generator-type.

Example C-1 automatic-key-generation With generator-type=Oracle

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

Example C-2 automatic-key-generation With generator-type=SQL-SERVER

```
<automatic-key-generation>
    <generator-type>SQL-SERVER</generator-type>
</automatic-key-generation>
```

Example C-3 automatic-key-generation With generator-type=NamedSequenceTable

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

caching-element

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-rdbms-jar
  weblogic-rdbms-bean
    relationship-caching
```


Function

Specifies the `cmr-field` and the `group-name` in the related bean. If `group-name` is not specified, the default `group-name` (load all fields) is used. For more information, see group-name.

`caching-element` can contain nested caching elements, as in the example shown in relationship-caching.

For information about relationship caching, see Relationship Caching.

Example

See relationship-caching:

caching-name

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-rdbms-jar
  weblogic-rdbms-bean
    relationship-caching
```


Function

The `caching-name` element specifies the name of a relationship cache. For more information about relationship caching, see Relationship Caching.

Example

See relationship-caching.

check-exists-on-method

Range of values: `True | False`

Default value: `True`

Parent elements:

```
weblogic-rdbms-bean
```


Function

By default, the EJB container checks that a container-managed persistence (CMP) entity bean exists before any business method invoked on the bean completes. This means the container notifies an application as soon as any business method is invoked on a container-managed entity bean that has been removed.

To specify that the EJB container wait for transaction completion to perform the existence check, set `check-exists-on-method` to `False`. This results in high performance and still provides a sufficient level of checking for most applications.

Example

The following example specifies that WebLogic Server notify the application that a business method has been invoked on a CMP entity bean that has been removed.

```
<check-exists-on-method>True</check-exists-on-method>
```

cluster-invalidation-disabled

Range of values: `true | false`

Default value: `false`

Parent elements:

```
weblogic-rdbms-bean
```


Function

This flag, introduced in WebLogic Server 9.0, is used to disable or enable invalidation of an EJB that uses `Optimistic` or `ReadOnly` concurrency when the EJB is updated by a member of a cluster to which it is deployed. For more information, see Invalidation Options for Optimistic Concurrency in Clusters.

Example

```
<cluster-invalidation-disabled>true</cluster-invalidation-disabled>
```

cmp-field

Range of values: Valid name of field in the bean. Field must have matching `cmp-entry` in `ejb-jar.xml`. Field name is case-sensitive.

Default value: n/a

Parent elements:

```
weblogic-rdbms-bean
  field-map
```

and

```
weblogic-rdbms-relation
  field-group
```


Function

This name specifies the mapped field in the bean instance which should be populated with information from the database.

Example

See field-map.

cmr-field

Range of values: Valid name of field in the bean. Field must have matching `cmr-field` entry in `ejb-jar.xml`

Default value: n/a

Parent elements:

```
weblogic-rdbms-relation
  field-group
```

and

```
relationship-caching
  caching-element
```


Function

Specifies the name of a container-managed relationship field.

Example

```
<cmr-field>stock options</cmr-field>
```

column-map

Range of values: n/a

Default value: n/a

Parent elements:

```
<weblogic-rdbms-relation>
  <weblogic-relationship-role>
    <relationship-role-map>
```


Function

This element represents the mapping of a foreign key column in one table in the database to a corresponding primary key. The two columns may or may not be in the same table. The tables to which the columns belong are implicit from the context in which the `column-map` element appears in the deployment descriptor.

Example

The following is an example of the `column-map` element:

> **Note:**
>
> You do not need to specify the `key-column` element if the `foreign-key-column` refers to a remote bean.

```
<column-map>
    <foreign-key-column>account-id</foreign-key-column>
```

```
    <key-column>id</key-column>
</column-map>
```

compatibility

Range of values: n/a

Default value: n/a

Parent elements:

`weblogic-rdbms-jar`

Function

The `<compatibility>` element, introduced in WebLogic Server 8.1 SP02, contains elements that specify compatibility flags for all of the cmp beans described in the descriptor file.

Example

```
<compatibility>
    <serialize-byte-array-to-oracle-blob>...</serialize-byte-array-to-oracle-blob>
    <serialize-char-array-to-bytes>...</serialize-char-array-to-bytes>
    <allow-readonly-create-and-remove>...</allow-readonly-create-and-remove>
    <disable-string-trimming>...</disable-string-trimming>
</compatibility>
```

create-default-dbms-table

Range of values: `Disabled | CreateOnly | DropAndCreate | DropAndCreateAlways | AlterOrCreate`

Default value: `Disabled`

Parent elements:

`weblogic-rdbms-jar`

Function

The `create-default-dbms-table` element performs two functions:

- It determines whether or how WebLogic Server automatically creates a default table based on the descriptions in the deployment files and the bean class, when underlying table schema have changed.

- It determines whether or how WebLogic Server automatically creates an Oracle database SEQUENCE.

Use this element only for convenience during development phases. This is because the table schema in the `DBMS CREATE` statement used are the EJB container's best approximation of the definition. A production environment typically requires more precise schema definition.

Automatic Table Creation

The following table describes how WebLogic Server handles automatic table creation, based on the value of `create-default-dbms-tables`.

Table C-1 Automatic Table Creation

Setting <create-default-dbms-tables> to this value...	Results in this behavior with respect to automatic table creation...
`Disabled`	The EJB container takes no action with respect to automatic table creation. This is the default value.
`CreateOnly`	The EJB container automatically generates the table upon detecting changed schema. The container attempts to create the table based on information found in the deployment files and in the bean class. If table creation fails, a '`Table Not Found`' error is thrown, and the user must create the table manually.
`DropAndCreate`	The EJB container automatically generates the table upon detecting changed schema. The container drops and recreates the table during deployment only if columns have changed. The container does not save data. You must ensure that the column type and cmp-field types are compatible. The EJB container does not attempt to ensure the column type and cmp-field types are compatible.
`DropAndCreateAlways`	The EJB container automatically generates the table upon detecting changed schema. The container drops and creates the table during deployment whether or not columns have changed. The container does not save the data.
`AlterOrCreate`	The EJB container automatically generates the table upon detecting changed schema. The container creates the table if it does not yet exist. If the table does exist, the container alters the table schema. Table data is saved. **Note**: Do not choose this setting if a new column is specified as a primary key or if a column with null values is specified as the new primary key column.

If `TABLE CREATION` fails, the server throws a `Table Not Found` error and the table must be created manually.

See Automatic Table Creation (Development Only).

Automatic Oracle Database SEQUENCE Generation

> **Note:**
>
> Automatic Oracle database SEQUENCE generation works only with servers running in development mode.

The following table describes how WebLogic Server handles automatic SEQUENCE generation, based on the value of `create-default-dbms-tables`.

Table C-2 Automatic Oracle Database SEQUENCE Generation

Setting <create-default-dbms-tables> to this value...	Results in this behavior...
`Disabled`	The EJB container takes no action with respect to SEQUENCE generation. This is the default value.
`CreateOnly`	The EJB container creates a `SEQUENCE`, and constructs its name by appending "_WL" to the value of the `generator-name` element.
`DropAndCreate`	The EJB container creates a `SEQUENCE`, and constructs its name by appending "_WL" to the value of the `generator-name` element. If the `SEQUENCE's` increment value does not match the value of the `key-cache-size` element, the container alters the increment value to match the value of `key-cache-size`.
`DropAndCreateAlways`	The EJB container creates a SEQUENCE, and constructs its name by appending "_WL" to the value of the `generator-name` element. If the `SEQUENCE's` increment value does not match the value of the `key-cache-size` element, the container alters the increment value to match the value of `key-cache-size`.
`AlterOrCreate`	The EJB container creates a `SEQUENCE`, and constructs its name by appending "_WL" to the value of the `generator-name` element. If the `SEQUENCE's` increment value does not match the value of the `key-cache-size` element, the container alters the increment value to match the value of `key-cache-size`.

For more information on automatic generation of an Oracle database SEQUENCE, see Support for Oracle Database SEQUENCE .

Example

The following example specifies the `create-default-dbms-tables` element.

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

database-specific-sql

Range of values: n/a

Default value: n/a

Requirements: `database-type` must be specified.

Parent elements:

```
weblogic-rdbms-jar
  weblogic-rdbms-bean
    weblogic-query
      sql-query
```


- Function
- Example

Function

The `database-specific-sql` element specifies a database-specific SQL statement.

Example

```
<database-specific-sql>
    <database-type>SQLServer</database-type>
    <sql>SELECT name, phone, location, testid FROM medrecappPharmacyBeanTable WHERE
         testid = ?1 AND SUBSTRING(testid, 1,5) = 'local' ORDER BY name
    </sql>
</database-specific-sql>
```

database-type

Range of values: `DB2 | Informix | MySQL | Oracle | SQLServer | SQLServer2000 | Sybase`

Default value: n/a

Parent elements:

```
weblogic-rdbms-jar
```

and

```
weblogic-rdbms-jar
  weblogic-rdbms-bean
    weblogic-quer
      sql-query
        database-specific-sql
```


- Function
- Example

Function

The `database-type` element specifies the database used as the underlying WebLogic Server dbms or the dbms against which to execute a vendor-specific SQL statement. If you specify `database-type` in the `weblogic-rdbms-jar` element, the database you specify applies to the entire `weblogic-rdbms-jar` deployment descriptor file unless it is overridden in a `database-specific-sql` element by another `database-type` element.

Example

```
<database-type>Oracle</database-type>
```

data-source-jndi-name

Range of values: Valid WebLogic Server JDBC data source name

Default value: n/a

Parent elements:

```
weblogic-rdbms-bean
```


Function

Specifies the JDBC data source name to be used for database connectivity for this bean. For more information on data sources, see *Developing JDBC Applications for Oracle WebLogic Server*.

> **Note:**
>
> Prior to WebLogic Server 9.0, this element was data-source-name.

Example

See table-name.

db-cascade-delete

Range of values: n/a

Default value: By default, database cascade delete is not used. The EJB container performs cascade deletes by issuing an individual SQL `DELETE`.

Parent elements:

```
weblogic-rdbms-bean
  weblogic-relationship-role
```

Function

Allows an application to take advantage of a database's built-in support for cascade delete, and possibly improve performance. This functionality is supported only for:

- Oracle databases
- One-to-one or one-to-many relationships.

If `db-cascade-delete` is enabled in `weblogic-cmp-rdbms-jar.xml`, you must

- Enable `cascade-delete` in `ejb-jar.xml`
- Enable cascade delete in the database table definition.

> **Note:**
>
> If `db-cascade-delete` is not specified, do not enable the database's cascade delete functionality, as this will produce incorrect results.

Setting up Oracle Database for Cascade Delete

The following Oracle database table definition will cause deletion all of the `emp` rows if the owning `dept` is deleted in the database.

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

Example

```
<weblogic-relationship-role>
    <db-cascade-delete/>
</weblogic-relationship-role>
```

dbms-column

Range of values: Valid database column

Default value: n/a

Parent elements:

```
weblogic-rdbms-bean
  field-map
```

Function

The name of the database column to which the field should be mapped.

> **Note:**
>
> `dbms-column` is case maintaining, although not all database are case sensitive.

Example

See field-map.

dbms-column-type

Range of values: `Blob | Clob | LongString | SybaseBinary`

Default value: n/a

Requirements: To specify `Blob` or `Clob` in this element, you must also set `dbms-default-value` to `Oracle` or `DB2` databases.

Parent elements:

```
weblogic-rdbms-bean
  field-map
```


Function

Specifies the type of the `cmp-field`. Maps the current field to a `Blob` or `Clob` in an Oracle or DB2 database or to a `LongString` or `SybaseBinary` in a Sybase database.

- `Blob`—maps the field to an Oracle or DB2 database Blob.
- `Clob`—maps the field to an Oracle or DB2 database Clob.
- `LongString`—tells the container to use `setCharacterStream` to write String data into the database. Some JDBC drivers have problems writing more than 4K of data using `setString`.
- `SybaseBinary`—tells the container to use `setBytes` to write bytes into the binary column, because `setBinaryStream` does not work with `SybaseXADriver`.

Example

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

dbms-default-value

Range of values: `DB2 | Informix| MySQL | Oracle | SQLServer | SQLServer2000 | Sybase`

Default value:

Parent elements:

```
weblogic-rdbms-bean
  field-map
```


Function

Specifies the database used as the default underlying dbms. This value can be overridden by the `database-type` element.

Example

```
<dbms-default-value>Oracle</dbms-default-value>
```

default-dbms-tables-ddl

Range of values: Valid file name

Default value:

Parent elements:

```
weblogic-rdbms-jar
```


Function

Specifies the `DDL` file name to which the EJB container writes the table creation scripts.

delay-database-insert-until

Range of values: `ejbCreate | ejbPostCreate`

Default value: `ejbPostCreate`

Requirememts:

Parent elements:

```
weblogic-rdbms-bean
```

Function

Specifies when a new CMP bean is inserted into the database. The allowable values cause the following behavior:

- `ejbCreate` - perform database insert immediately after `ejbCreate`. This setting yields better performance than `ejbCreate` by avoiding an unnecessary store operation.
- `ejbPostCreate` - perform insert immediately after `ejbPostCreate`.

This element has an effect only when `order-database-operations` is `False`. By default, `order-database-operations` is `true`, which causes new beans to be inserted at the transaction commit time.

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

For maximum flexibility, avoid creating related beans in your `ejbPostCreate` method. If `ejbPostCreate` creates related beans, and database constraints prevent related beans from referring to a bean that has not yet been created, it is not possible to perform a database insert until after the method completion.

> **Note:**
>
> `cmr-field`s may not be set during `ejbCreate`, before the primary key of the bean is known.

Example

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

description

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-rdbms-jar
  weblogic-rdbms-bean
    weblogic-query
```

or

```
weblogic-rdbms-jar
  weblogic-rdbms-bean
    sql-shape
```

Function

The `description` element provides text that describes the parent element.

Example

```
<description>Contains a description of parent element</description>
```

disable-string-trimming

Range of values: `True | False`

Default value: `False`

Parent elements:

`compatibility`

Function

This element, introduced in WebLogic Server 9.0, is a compatibility flag. It is used to specify whether a `cmp-field` of type `string[]` should be trimmed. Set this flag to `True` to disable string trimming. For more information on string trimming, see String-Valued CMP Field Trimming and Disabling String Trimming.

Example

```
<compatibility>
    <disable-string-trimming>True</disable-string-trimming>
</compatibility>
```

ejb-name

Range of values: Must match the `ejb-name` of a cmp entity bean defined in the `ejb-jar.xml`

Default value: n/a

Parent elements:

`weblogic-rdbms-bean`

Function

The name that specifies an EJB as defined in the `ejb-cmp-rdbms.xml`. This name must match the `ejb-name` of a cmp entity bean contained in the `ejb-jar.xml`.

Example

See table-name.

ejb-ql-query

Range of values:

Default value:

Parent elements:

```
weblogic-rdbms-jar
  weblogic-rdbms-bean
    weblogic-query
```

- Function
- Example

Function

The `ejb-ql-query` element specifies an EJB-QL query. You should only specify EJB-QL queries that implement EJB finders or contain WebLogic-specific extensions in the `weblogic-cmp-jar.xml` deployment descriptor; specify queries that use only standard EJB-QL features in the `ejb-jar.xml` deployment descriptor.

Example

See weblogic-query.

enable-batch-operations

Range of values: `True | False`

Default value: `True`

Parent elements:

```
weblogic-rdbms-jar
```

- Function
- Example

Function

This element, introduced in WebLogic Server 8.1, controls whether or not the EJB container allows batch database operations, including batch inserts, batch updates, and batch deletes.

If this element is set to `True`, the EJB delays database operations in a transaction until commit time.

Example

The following XML sample demonstrates use of the `enable-batch-operations` element:

```
<enable-batch-operations>True</enable-batch-operations>
```

enable-query-caching

Range of values: `True | False`

Default value: `True`

Parent elements:

```
weblogic-rdbms-jar
  weblogic-rdbms-bean
    weblogic-query
```

or

```
weblogic-rdbms-jar
  weblogic-rdbms-relation
```


Function

This element, introduced in WebLogic Server 9.0, controls whether read-only entity EJBs can be cached at the query level. Caching read-only entity EJBs at the query level improves performance because it enables the EJBs to be accessed in cache by any finder, thereby avoiding the need to access the database. If you set this value to `True`, you can specify the maximum number of queries to cache at the application or bean level. To specify the maximum number of queries to cache, set `max-queries-in-cache` in the `weblogic-ejb-jar.xml` deployment descriptor. For information, see max-queries-in-cache.

Example

The following XML sample demonstrates use of the `enable-query-caching` element:

```
<enable-query-caching>True</enable-query-caching>
```

field-group

Range of values: n/a

Default value: A special group named `default` is used for finders and relationships that have no `field-group` specified. The default group contains all of a bean's cmp-fields, but none of its cmr-fields.

Parent elements:

```
weblogic-rdbms-relation
```

Function

The `field-group` element represents a subset of the `cmp-field`s and `cmr-field`s of a bean. Related fields in a bean can be put into groups that are faulted into memory together as a unit. A group can be associated with a finder or relationship, so that when a bean is loaded as the result of executing a finder or following a relationship, only the fields specified in the group are loaded.

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

Example

The `field-group` element can contain the elements shown here:

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

field-map

Range of values: n/a

Default value: n/a

Parent elements:

`weblogic-rdbms-bean`

Function

The `field-map` element represents a mapping between a particular column in a database and a `cmp-field` in the bean instance.

The optional `group-name` element specifies a field group that is to be loaded when the `getXXX` method for the `cmp-field` is called and the EJB container needs to read the value from the DBMS because it is not in memory. If `group-name` is omitted, the default group, which contains all `cmp-field`s, is used when the `cmp-field` is not explicitly listed in any field groups, otherwise a field group that contains the `cmp-field` is chosen. Thus, developers should specify a `group-name` if the `cmp-field` is listed in multiple field groups or the container will pick one of the groups arbitrarily.

The `dbms-column-type` element is optional.

Example

The `field-map` element can contain the elements shown here:

```
<field-map>
    <cmp-field>....</cmp-field>
    <dbms-column>...</dbms-column>
    <dbms-column-type>...</dbms-column-type>
    <group-name>...</group name>
</field-map>
```

finders-return-nulls

Range of values: `True | False`

Default value: n/a

Parent elements:

```
weblogic-rdbms-jar
  compatibility
    weblogic-query
```


Function

This element, introduced in WebLogic Server 9.0, is a compatibility flag. It is used to specify whether finders can return null results.

By default

Example

```
<compatibility>
    <finders-return-nulls>True</finders-return-value>
</compatibility>
```

foreign-key-column

Range of values: Valid foreign key database column name

Default value: n/a

Parent elements:

```
weblogic-rdbms-bean
  column-map
```

Function

The `foreign-key-column` element represents a column of a foreign key in the database.

Example

See column-map.

foreign-key-table

Range of values: Valid database table name

Default value: n/a

Parent elements:

```
weblogic-rdbms-jar
  weblogic-rdbms-relation
    weblogic-relationship-role
      relationship-role-map
```


Function

The `foreign-key-table` element specifies the name of a DBMS table that contains a foreign key.

Example

See relationship-role-map.

generator-name

Range of values:

Default value:

Parent elements:

```
weblogic-rdbms-bean
  automatic-key-generation
```


Function

The `generator-name` element is used to specify the name of the primary key generator.

- If the `generator-type` element is `Oracle`, and WebLogic Server is running in development mode, then the EJB container constructs the Oracle database SEQUENCE name by

appending "_WL" to the `generator-name` as part of the container's automatic SEQUENCE generation feature.

For more information on automatic Oracle database SEQUENCE generation, see Support for Oracle Database SEQUENCE .

- If the `generator-type` element is `Oracle`, and WebLogic Server is running in production mode, then the EJB container sets the name of the Oracle database SEQUENCE to the value of `generator-name`.
- If the `generator-type` element is `NamedSequenceTable`, then the `generator-name` element would be the name of the `SEQUENCE_TABLE` to be used.

Example

See automatic-key-generation.

generator-type

Range of values: `Identity | Sequence | SequenceTable`

Default value:

Parent elements:

```
weblogic-rdbms-bean
  automatic-key-generation
```


Function

The `generator-type` element specifies the primary key generation method to use.

> **Note:**
>
> You must set the database-type element when using automatic-key-generation.

In addition, `generator-type` is used in conjunction with automatic Oracle database SEQUENCE generation. See Support for Oracle Database SEQUENCE .

The following databases are supported for each `generator-type`:

- `Identity`: Oracle databases are not supported.
- `Sequence`: Oracle, DB2, and Informix (version 9.2 and greater) databases are supported.
- `SequenceTable`: All databases are supported.

Example

See automatic-key-generation.

group-name

Range of values: n/a

Default value:

`field-group`

and

`caching-element`

and

`weblogic-query`

and

`field-map`

and

`weblogic-relationship-role`

Parent elements:

```
weblogic-rdbms-relation
  field-group
```


Function

Specifies the name of a field group.

Example

See field-group.

include-updates

Range of values: `True | False`

Default value: `False` for beans that use optimistic concurrency. `True` for beans that use other concurrency types.

Parent elements:

```
weblogic-rdbms-bean
  weblogic-query
```

Function

Specifies whether updates made during the current transaction must be reflected in the result of a query. If this element is set to `True`, the container will flush all changes made by the current transaction to disk before executing the query. A value of `False` provides best performance.

Example

```
<weblogic-query>
    <query-method>
    <method-name>findBigAccounts</method-name>
    <method-params>
        <method-param>double</method-param>
    </method-params>
    </query-method>
    <ejb-ql-query>
        <weblogic-ql>WHERE BALANCE>10000 ORDER BY NAME</weblogic-ql>
    </ejb-ql-query>
    <include-updates>True</include-updates>
</weblogic-query>
```

instance-lock-order

Range of values: `AccessOrder | ValueOrder`

Default value: `AccessOrder`

Parent elements:

`weblogic-rdbms-bean`

Function

Specifies a locking or processing order for instances of a particular EJB. This element can be used to prevent deadlocks in an application that would otherwise experience deadlocks. `instance-lock-order` is used whenever database operations (update, for example) that apply to multiple instances of the same EJB are performed by the container. It specifies an order for operations that can cause a database lock to be acquired for a bean instance.

For example, `instance-lock-order` could be used to specify the order in which the EJB container calls `ejbStore` for instances of a particular EJB that uses database concurrency; `ejbStore` may acquire an exclusive lock when a database update is done. `instance-lock-order` also controls the order in which beans using optimistic concurrency are locked when optimistic checking is performed.

- `AccessOrder`—The container will process beans so that locks are acquired (or upgraded) in the order in which the application originally accessed the beans during the transaction. This is the recommended value when all transactions in the system access instances of the bean, and ultimately rows in a database table, in the same order.

- `ValueOrder`—Beans are processed in order based on the value of their primary key. `ValueOrder` should be specified to avoid deadlocks when concurrent transactions access instances of the same EJB in different orders.

> **Note:**
>
> The EJB's primary key class is not required to implement the `java.lang.Comparable` interface when `ValueOrder` is specified, although this will result in a total ordering. Beans are ordered partially using the hash code value of the primary key when the primary key does not implement `java.lang.Comparable`.

Example

```
<instance-lock-order>ValueOrder</instance-lock-order>
```

key-cache-size

Range of values:

Default value: `1`

Parent elements:

```
weblogic-rdbms-bean
  automatic-key-generation
```


Function

Specifies the optional size of the primary key cache available in the automatic primary key generation feature. In addition, the EJB container uses this value to calculate the increment value for an Oracle database SEQUENCE when automatic SEQUENCE generation is enabled. See Support for Oracle Database SEQUENCE .

If `generator-type` is:

- `Oracle`—`key-cache-size` must match the Oracle database SEQUENCE INCREMENT value. If there is a mismatch between this value and the Oracle database SEQUENCE INCREMENT value, then there will likely be duplicate key problems.
- `NamedSequenceTable`—`key-cache-size` specifies how many keys the container will fetch in a single DBMS call.
- SQLServer—`key-cache-size` is ignored.

Example

See automatic-key-generation.

key-column

Range of values: Valid primary key column name

Default value: n/a

Parent elements:

```
weblogic-rdbms-bean
  column-map
```


Function

The `key-column` element represents a column of a primary key in the database.

Example

See column-map.

lock-order

Range of values: All positive integers

Default value: `0`

Parent elements:

```
weblogic-rdbms-bean
```


Function

Use this flag to specify the database locking order of an entity bean when a transaction involves multiple beans and exclusive concurrency. The bean with the lowest number is locked first.

This flag should only be used to prevent a deadlock situation and, currently, only applies when a transaction performs cascade deletes.

Example

An example of the `lock-order` element is:

```
<lock-order>1</lock-order>
    <!ELEMENT lock-order (PCDATA)>
```

max-elements

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-rdbms-bean
  weblogic-query
```


Function

`max-elements` specifies the maximum number of elements that should be returned by a multi-valued query. This element is similar to the `maxRows` feature in JDBC.

Example

An example of the `max-elements` element is shown here:

```
<max-elements>100</max-elements>
    <!ELEMENT max-element (PCDATA)>
```

method-name

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-rdbms-bean
  query-method
```


Function

The `method-name` element specifies the name of a finder or `ejbSelect` method.

> **Note:**
>
> The '*' character cannot be used as a wildcard.

Example

See weblogic-query.

method-param

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-rdbms-bean
  method-params
```


Function

The `method-param` element contains the fully qualified Java type name of a method parameter.

Example

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

method-params

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-rdbms-bean
  query-method
```


Function

The `method-params` element contains an ordered list of the fully-qualified Java type names of the method parameters.

Example

See weblogic-query.

optimistic-column

Range of values: Valid database column name

Default value: n/a

Parent elements:

```
weblogic-rdbms-bean
  table-map
```


Function

The `optimistic-column` element denotes a database column that contains a version or timestamp value used to implement optimistic concurrency. For more information on optimistic concurrency, see Choosing a Concurrency Strategy.

> **Note:**
>
> Although not all databases are case sensitive, this element is case maintaining.

Example

The following sample XML shows the use of the `optimistic-column` element.

```
<optimistic-column>ROW_VERSION</optimistic-column>
```

where `ROW_VERSION` is the name of a database column that contains the value used for concurrency checking.

order-database-operations

Range of values: `True | False`

Default value: `True`

Parent elements:

```
weblogic-rdbms-jar
```


Function

This element, introduced in WebLogic Server 8.1, determines whether the EJB container delays all database operations in a transaction until commit time, automatically sorts the database dependency between the operations, and sends these operations to the database in such a way to avoid any database constraint errors.

If `enable-batch-operations` is set to `True`, the container automatically sets `order-database-operations` to `True`. To turn off `order-database-operations`, set both `order-database-operations` and `enable-batch-operations` to `False`.

See also `ejb-ql-query` and `delay-database-insert-until`.

Example

```
<order-database-operations>True</order-database-operations>
```

pass-through-columns

Range of values: Any positive integer

Default value:

Parent elements:

```
weblogic-rdbms-bean
  sql-shape
```


Function

This element, introduced in WebLogic Server 9.0, specifies the number of aggregate columns that should be passed through to a SQL query result set without being mapped to anything.

Example

See sql-shape.

primary-key-table

Range of values: Valid database table name

Default value: n/a

Parent elements:

```
weblogic-rdbms-jar
  weblogic-rdbms-relation
    weblogic-relationship-role
      relationship-role-map
```


Function

The `primary-key-table` element specifies the name of a DBMS table that contains a primary key. For more information about primary keys, see Using Primary Keys.

> **Note:**
>
> Although not all databases are case sensitive, this element is case maintaining.

Example

For examples, see relationship-role-map and Mapping a Bean on Primary Key Side of a Relationship to Multiple Tables.

query-method

Range of values: n/a

Default value: n/a

Parent elements:

`weblogic-rdbms-bean`

Function

Specifies the method that is associated with a `weblogic-query`. It also uses the same format as the `ejb-jar.xml` descriptor.

Example

See weblogic-query.

relation-name

Range of values: Must match the `ejb-relation-name` of an `ejb-relation` in the associated `ejb-jar.xml` deployment descriptor file. The `ejb-relation-name` is optional, but is required for each relationship defined in the associated `ejb-jar.xml` deployment descriptor file.

Default value: n/a

Parent elements:

`weblogic-rdbms-relation`

Function

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

For more information about container-managed relationships, see Using Container-Managed Relationships (CMRs).

Example

An example of the `relationship-name` element is shown here:

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

relationship-caching

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-rdbms-jar
  weblogic-rdbms-bean
```


Function

The `relationship-caching` element specifies relationship caching. For more information about relationship caching, see Relationship Caching.

Example

The `relationship-caching` element can contain the elements shown here:

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

relationship-role-map

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-rdbms-relation
  weblogic-relationship-role
```

Function

The `relationship-role-map` element specifies foreign key column to key column mapping for beans involved in a relationship.

A CMP bean that is involved in a relationship may be mapped to multiple DBMS tables (see the `table-map` element for more details). If the bean on the foreign key side of a one-to-one or one-to-many relationship is mapped to multiple tables, then the name of the table containing the foreign-key columns must be specified using the `foreign-key-table` element.

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

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

For more information about container-managed relationships, see Using Container-Managed Relationships (CMRs).

Example

Mapping a Bean on Foreign Key Side of a Relationship to Multiple Tables

The bean on the foreign-key side of a one-to-one relationship, `Fk_Bean`, is mapped to multiple tables. The table that holds the foreign key columns must be specified in the `foreign-key-table` element.

`Fk_Bean` is mapped to two tables: `Fk_BeanTable_1` and `Fk_BeanTable_2`. The foreign key columns for the relationship are located in table `Fk_BeanTable_2`. The foreign key columns are named `Fk_column_1` and `Fk_column_2`. The bean on the primary key side, `Pk_Bean`, is mapped to a single table with primary key columns `Pk_table_pkColumn_1` and `Pk_table_pkColumn_2`:

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

The `foreign-key-table` element must be specified so that the container can know which table contains the foreign key columns.

Mapping a Bean on Primary Key Side of a Relationship to Multiple Tables

The bean on the primary key side of a one-to-one relationship, `Pk_bean`, is mapped to multiple tables, but the bean on the foreign key side of the relationship, `Fk_Bean`, is mapped to one table, `Fk_BeanTable`. The foreign key columns are named `Fk_column_1` and `Fk_column_2`.

`Pk_bean` is mapped to tables:

- `Pk_BeanTable_1` with primary key columns `Pk_table1_pkColumn_1` and `Pk_table1_pkColumn_2` and
- `Pk_BeanTable_2` with primary key columns `Pk_table2_pkColumn_1` and `Pk_table2_pkColumn_2`.

```
<relationship-role-map>
    <primary-key-table>Pk_BeanTable_1</primary-key-table>
    <column-map>
        <foreign-key-column>Fk_column_1</foreign-key-column>
        <key-column>Pk_table1_pkColumn_1</key-column>
    </column-map>
    <column-map>
        <foreign-key-column>Fk_column_2</foreign-key-column>
        <key-column>Pk_table1_pkColumn_2</key-column>
    </column-map>
</relationship-role-map>
```

relationship-role-name

Range of values: Must match the `ejb-relationship-role-name` of an `ejb-relationship-role` in the associated `ejb-jar.xml`.

Default value: n/a

Parent elements:

```
weblogic-rdbms-relation
  weblogic-relationship-role
```


Function

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

For more information about container-managed relationships, see Using Container-Managed Relationships (CMRs).

Example

See the examples for weblogic-relationship-role.

serialize-byte-array-to-oracle-blob

Range of values: `True | False`

Default value: `False`

Parent elements:

```
weblogic-rdbms-jar
  compatibility
```


Function

This element, introduced in WebLogic Server 8.1 SP02, is a compatibility flag. It is used to specify whether a `cmp-field` of type `byte[]` mapped to a Blob in an Oracle database should be serialized. By default, the value of the tag is `false`, which means that the container will persist the `byte[]` directly and not serialize it.

In versions prior to WebLogic Server 8.1 SP02, the default behavior was to serialize a `cmp-field` of type `byte[]` mapped to a Blob in an Oracle database. To revert to the old behavior, set the value of s`erialize-byte-array-to-oracle-blob` to true.

Example

```
<compatibility>
   <serialize-byte-array-to-oracle-blob>true</serialize-byte-array-to-oracle-blob>
</compatibility>
```

serialize-char-array-to-bytes

Range of values: `True` | `False`

Default value: `False`

Parent elements:

```
weblogic-rdbms-jar
  compatibility
```


Function

This element, introduced in WebLogic Server 9.0, is a compatibility flag. It is used to specify whether a `cmp-field` of type `char[]` mapped to a byte should be serialized. By default, the value of the tag is `False`, which causes the EJB container to persist the `char[]` directly and not serialize it; If you want the EJB container to serialize the `char[]`, set this value to `True`.

Example

```
<compatibility>
    <serialize-char-array-to-bytes>true</serialize-char-array-to-bytes>
</compatibility>
```

sql

Range of values: Valid SQL

Default value: n/a

Requirements: To use database-specific SQL, you must specify the database against which to execute the SQL in the in database-type element.

Parent elements:

```
weblogic-rdbms-bean
  weblogic-query
    sql-query
```

and

```
weblogic-rdbms-bean
  weblogic-query
    sql-query
      database-specific-query
```


Function

The `sql` element contains a standard or database-specific SQL query. You should specify queries that use only standard EJB-QL language features in the `ejb-jar.xml` deployment descriptor. Specify queries that contain standard SQL, database-specific SQL, or WebLogic extensions to EJB-QL in the `weblogic-cmp-jar.xml` deployment descriptor.

Example

```
...
<weblogic-rdbms-bean>
    <weblogic-query>
        <sql-query>
            <sql>SELECT date_prescribed, dosage, drug, id, frequency,
                 instructions, pat_id, issuing_phys_id, record_id,
                 refills_remaining FROM medrecappPrescription
                 WHERE testid = ?1</sql>
        </sql-query>
    </weblogic-query>
...
</weblogic-rdbms-bean>
```

sql-query

Range of values:

Default value: n/a

Requirements: To use database-specific SQL, you must specify the database against which to execute the SQL in `database-type`.

Parent elements:

```
weblogic-rdbms-bean
  weblogic-query
```


Function

The `sql-query` element allows you to specify standard and database-specific SQL queries. EJB-QL queries that do not take advantage of WebLogic extensions to EJB-QL should be specified in the `ejb-jar.xml` deployment descriptor.

Prior to WebLogic Server 9.0, only EJB-QL queries (with or without WebLogic extensions) were supported; in this release of WebLogic Server, SQL queries, EJB-QL queries (with or without WebLogic extensions), or a combination of the two are supported.

Example

```
<sql-query>
    <sql-shape-name>...</sql-shape-name>
    <sql>...</sql>
    <database-specific-sql>...</database-specific-sql>
    <database-type>...</database-type>
    <sql>...</sql>
    <unknown-primary-key-field>...</unknown-primary-key-field>
    <cmp-field>...</cmp-field>
</sql-query>
```

sql-select-distinct

Range of values: `True | False`

Default value: `False`

Parent elements:

```
weblogic-query
```

> **Note:**
>
> This element is deprecated in this release of WebLogic Server. To achieve the same functionality, use the `SELECT DISTINCT` clause directly in finder queries.

Function

The `sql-select-distinct` element controls whether the generated SQL SELECT statement will contain a a DISTINCT qualifier. Using the DISTINCT qualifier causes the database to return unique rows.

Oracle database does not allow use of a `SELECT DISTINCT` with a `FOR UPDATE` clause. Therefore, you cannot use the `sql-select-distinct` element if any bean in the calling chain has a method with `isolation-level` of `TransactionReadCommittedForUpdate`. You specify the `transaction-isolation` element in the `weblogic-ejb-jar.xml`.

Example

The `XML` example contains the element shown here:

```
<sql-select-distinct>True</sql-select-distinct>
```

sql-shape

Range of values:

Default value:

Parent elements:

```
weblogic-rdbms-jar
  weblogic-rdbms-bean
    sql-query
```


Function

The `sql-shape` element describes the data that is returned by a SQL query. Specifying sql-shape is necessary because databases do not always provide this information. Usually the sql-shape element should simply specify the database tables and columns that are being returned. For more complex queries, sql-shape should also specify the relationships that are present in the data that is returned by the database, and whether there are aggregate columns that should be passed through (should not be mapped to anything).

Example

```
sql-shape
    description
    sql-shape-name
    table
    pass-through-columns
    ejb-relation-name
```

sql-shape-name

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-rdbms-jar
  weblogic-rdbms-bean
    weblogic-query
```

```
sql-query
  sql-shape
```

- Function
- Example

Function

The `sql-shape-name` element can be used to associate a `sql-shape` with multiple queries. If you have multiple queries that use the same `sql-shape`, you can define the shape once and use it multiple times by referencing `sql-shape-name`.

Example

See sql-shape.

table-map

Range of values: n/a

Default value: n/a

Requirements: Each `table-map` element must contain a mapping for the bean's primary key fields.

Parent elements:

`weblogic-rdbms-bean`

- Function
- Example

Function

A CMP bean can be mapped to one or more DBMS tables. The `table-map` element specifies a mapping between the `cmp-field`s of a bean and the columns of a table for all of the `cmp-fields` mapped to that table. If you map a CMP bean to multiple DBMS tables, then you must specify a `table-map` element for each of the tables.

When you map a CMP bean to multiple tables, each table contains a row that maps to a particular bean instance. Consequently, all tables will contain the same number of rows at any point in time. In addition, each table contains the same set of homogeneous primary key values. Therefore, each table must have the same number of primary key columns and corresponding primary key columns in different tables must have the same type, although they may have different names.

Each `table-map` element must specify a mapping from the primary key column(s) for a particular table to the primary key field(s) of the bean. You can only map non-primary key fields to a single table.

For information about using the `verify-rows`, `verify-columns`, and `optimistic-column` elements, see Check Data for Validity with Optimistic Concurrency.

Example

The `table-map` element can contain the elements shown here:

```
<table-map>
    <table-name>DeptTable</table-name>
    <field-map>
        <cmp-field>deptId1</cmp-field>
        <dbms-column>t1_deptId1_column</dbms-column>
    </field-map>
    <field-map>
        <cmp-field>deptId2</cmp-field>
        <dbms-column>t1_deptId2_column</dbms-column>
    </field-map>
    <field-map>
        <cmp-field>location</cmp-field>
        <dbms-column>location_column</dbms-column>
    </field-map>
        <cmp-field>budget</cmp-field>
        <dbms-column>budget</dbms-column>
    </field-map>
    <verify-rows>Read</verify-rows>
    <verify-columns>Version</verify-columns>
    <optimistic-column>ROW_VERSION</optimistic-column>
    <trigger-updates-optimistic-column>False
    </trigger-updates-optimistic-column>
    <version-column-initial-value>0</version-column-initial-value>
</table-map>
```

table-name

Range of values: Valid, fully qualified SQL name of the source table in the database.

Default value:

Requirements: `table-name` must be set in all cases.

Parent elements:

```
weblogic-rdbms-bean
  weblogic-rdbms-relation
```


Function

The fully-qualified SQL name of the table. The user defined for the `data-source` for this bean must have read and write privileges for this table, but does not necessarily need schema modification privileges.

Example

```
<weblogic-rdbms-jar>
    <weblogic-rdbms-bean>
        <ejb-name>containerManaged</ejb-name>
        <data-source-jndi-name>examples-dataSource-demoPool</data-source-jndi-name>
```

```
        <table-name>ejbAccounts</table-name>
    </weblogic-rdbms-bean>
</weblogic-rdbms-jar>
```

trigger-updates-optimistic-column

Range of values: `True` | `False`

Default value: `False`

Parent elements:

```
weblogic-rdbms-bean
  table-map
```


Function

The `trigger-updates-optimistic-column` element, introduced in WebLogic Server 9.0, indicates whether you want the EJB container to automatically update the database column value specified in `optimistic-column` that is used for concurrency checking. By default, the value of `trigger-updates-optimistic-column` is `False`, and the EJB container automatically updates the database column specified in `optimistic-column` whenever it sends SQL `UPDATE` statements to JDBC. If you have legacy applications that use database triggers to update the version values whenever the legacy application updates a database row and you do not want the EJB container to automatically update version values, set the value of this element to `True`.

> **Note:**
>
> If you set trigger-updates-optimistic-column to True, you must also ensure that your database triggers initialize the version column in the database when the bean is created.

Example

See table-map.

unknown-primary-key-field

Range of values: A valid datatype

Default value: n/a

Parent elements:

```
weblogic-rdbms-bean
```

Function

The `unknown-primary-key-field` element allows you to specify which of your cmp fields should be used as the primary key when the primary key is not specified in the `ejb-jar.xml` descriptor. The specified primary key field must be mapped to a database column using the `field-map` element. If the specified primary key field was not declared as a cmp field in the `ejb-jar.xml` descriptor, automatic key generation must be enabled and the primary key type will be `java.lang.Long`.

Example

See weblogic-rdbms-bean.

use-select-for-update

Range of values: `True | False`

Default value: `False`

Parent elements:

`weblogic-rdbms-bean`

Function

Enforces pessimistic concurrency on a per-bean basis. Specifying `True` causes `SELECT ... FOR UPDATE` to be used whenever the bean is loaded from the database. This is different from the transaction isolation level of `TransactionReadCommittedForUpdate` in that this is set at the bean level rather than the transaction level.

> **Note:**
>
> When using a pessimistic locking strategy (for example, `HOLDLOCK`) with Sybase JConnect drivers, you must specify `SELECT_OPENS_CURSOR=true` to generate a cursor when the query contains a `FOR UPDATE` clause.

Example

```
<weblogic-rdbms.jar>
    <weblogic-rdbms-bean>
        <ejb-name>containerManaged</ejb-name>
        <use-select-for-update>True</use-select-for-update>
    /weblogic-rdbms-bean>
</weblogic-rdbms-jar>
```

validate-db-schema-with

Range of values: `MetaData | TableQuery`

Default value: `TableQuery`

Parent elements:

```
weblogic-rdbms-jar
```


Function

The `validate-db-schema-with` element specifies that container-managed persistence checks that beans have been mapped to a valid database schema during deployment.

If you specify `MetaData` WebLogic Server uses the JDBC metadata to validate the schema.

If you specify `TableQuery`, the default setting, WebLogic Server queries the tables directly to verify that they have the schema expected by CMP runtime.

Example

An example of the `validate-db-schema-with` element is shown here:

```
<validate-db-schema-with>TableQuery</validate-db-schema-with>
```

verify-columns

Range of values: `Read | Modified | Version | Timestamp`

Default value: `none`

Requirements: `table-name` must be set in all cases.

Parent elements:

```
weblogic-rdbms-bean
  table-map
```


Function

The `verify-columns` element specifies the columns in a table that you want WebLogic Server to check for validity when you use the `optimistic` concurrency strategy. WebLogic Server checks columns at the end of a transaction, before committing it to the database, to make sure that no other transaction has modified the data.

See Choosing a Concurrency Strategy for more information.

Example

```
<verify-columns>Modified</verify-columns>
```

verify-rows

Range of values: `Read | Modified`

Default value: `Modified`

Requirements: `table-name` must be set in all cases.

Parent elements:

```
weblogic-rdbms-bean
  table-map
```


Function

The `verify-rows` element specifies the rows in a table that the EJB container should check when optimistic concurrency is used.

- `Modified`— only rows that are updated or deleted by a transaction are checked. This value ensures that two transactions do not update the same row concurrently, resulting in a lost update, but allows reads and updates of different transactions to be interleaved. This results in a level of consistency that falls between the ANSI `READ_COMMITTED` and `REPEATABLE_READ` levels of consistency.
- `Read`—specifies that any row that is read by the transaction should be checked. This includes both rows that are simply read and rows that are read and then updated or deleted by the transaction. Specifying a value of `Read` results in additional overhead since it generally increases the amount of optimistic checking the EJB container must perform. With the `Read` option, committed transactions read a set of rows that are guaranteed not to be modified by another transaction until after the transaction commits.This results in a high level of consistency which is very close to the ANSI definition of `SERIALIZABLE` consistency.

> **Note:**
>
> If `verify-rows` is set to `Read` then the `verify-columns` element cannot have a value of `Modified`, as this combination would result in the EJB container checking only the modified rows.

See Choosing a Concurrency Strategy for more information.

Example

```
<verify-rows>Modified</verify-rows>
```

version-column-initial-value

Range of values: `0` or any positive integer

Default value: n/a

Parent elements:

```
weblogic-rdbms-bean
  table-map
```

- Function
- Example

Function

The `version-column-initial-value` element, introduced in WebLogic Server 9.0, specifies the initial value of the version column used to implement optimistic concurrency. The version column is the database column you specify in the `optimistic-column` element. For more information, see optimistic-column.

Example

See table-map.

weblogic-ql

Range of values:

Default value:

Parent elements:

```
weblogic-rdbms-bean
  weblogic-query
```

- Function
- Example

Function

The `weblogic-ql` element specifies a query that contains a WebLogic specific extension to the ejb-ql language. You should specify queries that only use standard EJB-QL language features in the `ejb-jar.xml` deployment descriptor.

Example

See weblogic-query.

weblogic-query

Range of values: n/a

Default value: n/a

Parent elements:

`weblogic-rdbms-bean`

Function

The `weblogic-query` element allows you to specify queries that use standard or database-specific SQL or WebLogic-specific extensions to EJB-QL. Queries that do not take advantage of SQL or WebLogic extensions to EJB-QL should be specified in the `ejb-jar.xml` deployment descriptor.

The `weblogic-query` element is also 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 `weblogic-query` element can contain the elements shown here:

```
<weblogic-query>
    <description>...</description>
    <query-method>
        <method-name>findBigAccounts</method-name>
        <method-params>
            <method-param>double</method-param>
        </method-params>
    </query-method>
    <ejb-ql-query>
        <weblogic-ql>WHERE BALANCE>10000 ORDER BY NAME
        </weblogic-ql>
        <group-name>...</group-name>
        <caching-name>...</caching-name>
    </ejb-ql-query>
    <sql-query>
        <sql-shape>...</sql-shape>
        <sql>SELECT date_prescribed, dosage, drug, id, frequency, instructions,
             pat_id, issuing_phys_id, record_id, refills_remaining FROM
             medrecappPrescription WHERE testid = ?1</sql>
        <database-specific-sql>
            <database-type>SQLServer</database-type>
            <sql>SELECT name, phone, location, testid FROM medrecappPharmacyBeanTable
                 WHERE testid = ?1 AND
                 SUBSTRING(testid, 1,5) = 'local' ORDER BY name</sql>
        </database-specific-sql>
    </sql-query>
    <max-elements>...</max-elements>
    <include-updates>...</include-updates>
    <sql-select-distinct>...</sql-select-distinct>
</weblogic-query>
```

weblogic-rdbms-bean

Range of values: n/a

Default value: n/a

Parent elements:

`weblogic-rdbms-jar`

Function

The `weblogic-rdbms-bean` specifies an entity bean that is managed by the WebLogic RDBMS CMP persistence type.

Example

```
weblogic-rdbms-bean
    ejb-name
    data-source-jndi-name
    unkonown-primary-key-field
    table-map
    field-group
    relationship-caching
    weblogic-query
    dalay-database-insert-until
    automatic-key-generation
    check-exists-on-method
```

weblogic-rdbms-jar

This section describes and provides an example of the `weblogic-rdbms-jar` element.

Function

The `weblogic-rdbms-jar` element is the root level element of a WebLogic RDBMS CMP deployment descriptor. This element contains the deployment information for one or more entity beans and an optional set of relations.

Example

The `XML` structure of `weblogic-rdbms-jar` is:

```
weblogic-rdbms-jar
    weblogic-rdbms-bean
    weblogic-rdbms-relation
    create-default-dbms-tables
    validate-db-schema-with
    database-type
```

weblogic-rdbms-relation

Range of values: n/a

Default value: n/a

Parent elements:

`weblogic-rdbms-jar`

Function

The `weblogic-rdbms-relation` element represents a single relationship that is managed by the WebLogic CMP persistence type. deployment descriptor. WebLogic Server supports the following three relationship mappings:

- For one-to-one relationships, the mapping is from a foreign key in one bean to the primary key of the other bean.
- For one-to-many relationships, the mapping is also from a foreign key in one bean to the primary key of another bean.
- For many-to-many relationships, the mapping involves a join table. Each row in the join table contains two foreign keys that map to the primary keys of the entities involved in the relationship.

For more information on container managed relationships, see Using Container-Managed Relationships (CMRs).

Examples

See the following sections for examples of how one-to-one, one-to-many, and many-to-many relationships are configured.

Defining a One-to-One Relationship

Example C-6 shows the `weblogic-rdbms-bean` element that defines a one-to-one relationship between the entities defined in Example C-4 and Example C-5. The `weblogic-rdbms-relation` element is in the `weblogic-cmp-jar.xml` file, after the `weblogic-rdbms-bean` elements.

> **Note:**
>
> `NAME` is the column name for the primary key located in the Capital table.
>
> <relationship-role-name> contains the relation field specified in <`cmr-field`> in the <`ejb-relationship-role`> element in `ejb-jar.xml`.

Example C-4 Bean 1

```
<weblogic-rdbms-bean>
    <ejb-name>CountryEJB</ejb-name>
    <data-source-jndi-name>wlsd21-datasource</data-source-jndi-name>
    <table-map>
```

```
        <table-name>EXAMPLE07_COUNTRY</table-name>
        <field-map>
            <cmp-field>name</cmp-field>
            <dbms-column>NAME</dbms-column>
        </field-map>
        <field-map>
            <cmp-field>continent</cmp-field>
            <dbms-column>CONTINENT</dbms-column>
        </field-map>
    </table-map>
</weblogic-rdbms-bean>
```

Example C-5 Bean 2

```
<weblogic-rdbms-bean>
    <ejb-name>CapitalEJB</ejb-name>
    <data-source-jndi-name>wlsd21-datasource</data-source-jndi-name>
    <table-map>
        <table-name>EXAMPLE07_CAPITAL</table-name>
        <field-map>
            <cmp-field>CAPITAL_NAME</cmp-field>
            <dbms-column>NAME</dbms-column>
        </field-map>
        <field-map>
            <cmp-field>continent</cmp-field>
            <dbms-column>CONTINENT</dbms-column>
        </field-map>
    </table-map>
</weblogic-rdbms-bean>
```

Example C-6 <weblogic-rdbms-relation> Element for a One-to-One Relationship

```
<weblogic-rdbms-relation>
    <relation-name>CountryCapitalRel</relation-name>
    <weblogic-relationship-role>
        <relationship-role-name>CountryRole</relationship-role-name>
        <relationship-role-map>
            <column-map>
                <foreign-key-column>CAPITAL_NAME</foreign-key-column>
                <key-column>NAME</key-column>
            </column-map>
        </relationship-role-map>
    </weblogic-relationship-role>
</weblogic-rdbms-relation>
```

Defining a One-to-Many Relationship

Example C-7 contains a sample `<weblogic-rdbms-relation>` element that defines a one-to-many relationship:

> **Note:**
>
> <relationship-role-name> contains the relation field specified in <`cmr-field`> in the <`ejb-relationship-role`> element in `ejb-jar.xml`.
>
> <foreign-key-column> must specify the column in the table on the "many" side of the relationship.

Example C-7 <weblogic-rdbms-relation> Element for a One-to-Many Relationship

```
<weblogic-rdbms-relation>
    <relation-name>OwnerDogRel</relation-name>
    <weblogic-relationship-role>
        <relationship-role-name>DogRole</relationship-role-name>
        <relationship-role-map>
            <column-map>
                <foreign-key-column>OWNER_NAME</foreign-<key-column>
                <key-column>NAME</key-column>
            </column-map>
        <relationship-role-map>
    </weblogic-relationship-role>
</weblogic-rdbms-relation>
```

Defining a Many-to-Many Relationship

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

The following example shows a many-to-many relationship between the `FRIENDS` bean and the `EMPLOYEES` bean.

Example C-8 <weblogic-rdbms-relation> Element for a Many-to-Many Relationship

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

In Example C-7, the `FRIENDS` join table has two columns, called `first-friend-id` and `second-friend-id`. Each column contains a foreign key that designates a particular employee who is a friend of another employee. The primary key column (`key-column`) of the `EMPLOYEES` table is `id`. For this example, assume that the `EMPLOYEES` bean is mapped to a single table. If the `EMPLOYEES` bean is mapped to multiple tables, then the table containing the primary key column (`key-column`) must be specified in the `relationship-role-map`. For more information, see relationship-role-map.

weblogic-relationship-role

Range of values: n/a

Default value: n/a

Parent elements:

```
weblogic-rdbms-jar
  weblogic-rdbms-relation
```


Function

The `weblogic-relationship-role` element specifies the following DBMS schema information for an `ejb-relationship-role` specified in `ejb-jar.xml`:

- The `relationship-role-map` sub-element specifies the mapping between a foreign key and a primary key, for one side of a relationship. For a 1-1 or 1-n relationship, only the role on the foreign-key side of the relationship specifies a mapping. Both roles specify a mapping for a m-m relationship. For details and examples, see relationship-role-map.
- A `group-name` can be used to indicate the `field-group` to be loaded when the bean corresponding to the role is loaded as a result of traversing the relationship, i.e. calling a cmr `getXXX` method.
- The `db-cascade-delete` tag can to used to specify that cascade deletes use the built-in cascade delete facilities of the underlying DBMS. For more information, see db-cascade-delete.

For more information about container-managed relationships, see Using Container-Managed Relationships (CMRs).

Example

```
<weblogic-relationship-role>
    <relationship-role-name>...</relationship-role-name>
    <group-name> ....</group-name>
    <relationship-role-map>...
        ....
    </relationship-role-map>
    <db-cascade-delete/>
</weblogic-relationship-role>
```

D
appc Reference

Examine the WebLogic Server `appc` tool for compiling, validating, and generating EJB code. This appendix includes the following topic:

- appc

appc

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

- Advantages of Using appc
- appc Syntax
- Designating Alternative Deployment Descriptors
- appc Options
- appc and EJBs

Advantages of Using appc

The `appc` tool offers the following benefits:

- The flexibility of compiling an entire application, rather than compiling individual modules separately and combining them into an EAR after the fact.
- Validation checks across all modules and validation of application-level deployment descriptors against the various modules, because WebLogic Server has access to all modules during EAR compilation.

  Without appc, a user wanting to compile all modules within an EAR file had to extract the individual components of an EAR and manually execute the appropriate compiler (`jspc` or `ejbc`) to prepare the module for deployment. `appc` automates this process and makes additional pre-deployment validation checks not previously possible.
- It is easy to identify and correct errors `appc` produces.

  If an error occurs while running `appc` from the command line, `appc` exits with an error message.

  By contrast, if you defer compilation to the time of deployment and a compilation error occurs, the server fails the deployment and goes on with its work. To determine why deployment failed, you must examine the server output, fix the problem and then redeploy.
- By running `appc` prior to deployment, you potentially reduce the number of time a bean is compiled.

For example, if you deploy a JAR file to a cluster of 3 servers, the JAR file is copied to each of the three servers for deployment. If the JAR file wasn't precompiled, each of the three servers will have to compile the file during deployment.

appc Syntax

Use the following syntax to run `appc`:

```
prompt>java weblogic.appc [options] <ear, jar, or war file or directory>
```

Designating Alternative Deployment Descriptors

Jakarta EE allows you to designate an alternative Jakarta EE deployment descriptor for an EJB or Web application module, using the `<alt-dd>` element in the `<module>` element of `application.xml`.

You can use `<alt-dd>` to specify an alternate deployment descriptor only for the Jakarta EE deployment descriptors, `web.xml` and `ejb-jar.xml`. As of WebLogic Server 8.1 SP01, if you specify an alternative deployment descriptor for a module in `alt-dd`, `appc` will compile the EJB using the alternative descriptor file.

For more information about the `<alt-dd>` element, see "module" in Enterprise Application Deployment Descriptor Elements in *Developing Applications for Oracle WebLogic Server*.

In WebLogic Server 8.1 SP01and later, you can use `appc` command line options to designate alternative Jakarta EE and WebLogic Server deployment descriptors for an application, as shown below:

- `-altappdd <file>`—Use this option to specify the full path and file name of an alternative Jakarta EE deployment descriptor file, `application.xml`.
- `-altwlsappdd <file>`—Use this option to specify the full path and file name of an alternative WebLogic application deployment descriptor, `weblogic-application.xml`.

appc Options

Table D-1 lists `appc` command line options.

Table D-1 appc Command Line Options

Option	Description
`-advanced`	Prints advanced usage options.
`-altappdd`	Designates an alternative Java EE application deployment descriptor.
`-altwlsappdd`	Designates an alternative WebLogic Server application deployment descriptor.
`-basicClientJar`	Does not include deployment descriptors in client JARs generated for EJBs.
`-classpath <path>`	Selects the classpath to use during compilation.
`-clientJarOutputDir <dir>`	Specifies a directory to place generated client jar files. If not set, generated jar files are placed into the same directory location where the JVM is running.
`-compiler <javac>`	Selects the Java compiler to use.

Table D-1 (Cont.) appc Command Line Options

Option	Description
`-deprecation`	Warns about deprecated calls.
`-forceGeneration`	Forces generation of EJB and JSP classes. Without this flag, the classes will not be regenerated unless a checksum indicates that it is necessary.
`-g`	Compiles debugging information into a class file.
`-help`	Prints the standard usage message.
`-idl`	Generates IDL for EJB remote interfaces.
`-idlDirectory <dir>`	Specifies the directory where IDL files will be created (default: target directory or JAR)
`-idlFactories`	Generates factory methods for valuetypes.
`-idlMethodSignatures <>`	Specifies the method signatures used to trigger IDL code generation.
`-idlNoAbstractInterfaces`	Does not generate abstract interfaces and methods/attributes that contain them.
`-idlNoValueTypes`	Does not generate valuetypes and the methods/attributes that contain them.
`-idlOrbix`	Generates IDL somewhat compatible with Orbix C++.
`-idlOverwrite`	Always overwrites existing IDL files.
`-idlVerbose`	Displays verbose information for IDL generation.
`-idlVisibroker`	Generates IDL somewhat compatible with Visibroker C++.
`-iiop`	Generates CORBA stubs for EJBs.
`-iiopDirectory <dir>`	Specifies the directory where IIOP stub files will be written (default: target directory or JAR)
`-J<option>`	Passes flags through to Java runtime.
`-keepgenerated`	Keeps the generated `.java` files.

`-library <file>`	Comma-separated list of libraries. Each library may optionally set its name and versions, if not already set in its manifest, with the syntax:
	`<file>[@name=<string>@libspecver=<version>`
	`@libimplver=<version|string>]`

`-librarydir <dir>`	Registers all files in the specified directory as libraries.
`-lineNumbers`	Adds line numbers to generated class files to aid in debugging.
`-normi`	Passes flags through to Symantec's sj.
`-nowarn`	Compiles without warnings.
`-O`	Compiles with optimization on.
`-output <file>`	Specifies an alternate output archive or directory. If not set, the output is placed in the source archive or directory.
`-plan <file>`	Specifies an optional deployment plan.
`-verbose`	Compiles with verbose output.
`-version`	Prints `appc` version information.

appc and EJBs

`weblogic.appc` performs the following EJB-related functions:

- Generates WebLogic Server container classes for the EJBs.
- Checks all EJB classes and interfaces for compliance with the EJB specification.
- Checks deployment descriptors for potential configuration problems. For example, if there is a cmp field declared in `ejb-jar.xml`, `appc` verifies that the column is mapped in the `weblogic-cmp-rdbms.xml` deployment descriptor.
- Runs each EJB container class through the RMI compiler to create RMI descriptors necessary to dynamically generate stubs and skeletons.

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

For the location of the public version of `weblogic-ejb-jar.xml`, see weblogic-ejb-jar.xml Deployment Descriptor Reference. For the location of the public version of `weblogic-cmp-jar.xml`, see weblogic-cmp-jar.xml Deployment Descriptor Reference.

E

Important Information for EJB 1.1 Users

Examine some of the important information for EJB 1.1. users and the detailed reference to EJB 1.1 deployment descriptors. Oracle strongly recommends that new users implement their distributed business applications using EJB 2.0 beans.
However, if your existing application implements EJB 1.1 beans, read the following sections, which contain important design and implementation information specific to EJB 1.1.

This appendix includes the following topics:

- Writing for RDBMS Persistence for EJB 1.1 CMP
  Learn how to write finders for WebLogic-specific 1.1 EJBs that use RDBMS persistence. You can use EJB QL, a portable query language, to define finder queries for 2.0 EJBs with container-managed persistence.
- Using WebLogic Query Language (WLQL) for EJB 1.1 CMP
  WebLogic Query Language (WLQL) for EJB 1.1 CMP allows you to query 1.1 entity EJBs with container-managed persistence. In the `weblogic-cmp-jar.xml` file, each `finder-query` element must include a WLQL string that defines the query used to return EJBs.
- Using SQL for CMP 1.1 Finder Queries
  WebLogic Server allows you to use a SQL string instead of the standard WLQL query language to write SQL for a CMP 1.1 finder query. The SQL statement retrieves the values from the database for the CMP 1.1 finder query. Use SQL to write a CMP 1.1 finder query when a more complicated finder query is required and you cannot use WLQL.
- Tuned EJB 1.1 CMP Updates in WebLogic Server
  EJB container-managed persistence (CMP) automatically support tuned updates because the container receives `get` and `set` callbacks when container-managed EJBs are read or written. Tuning EJB 1.1 CMP beans helps improve their performance.
- Using is-modified-method-name to Limit Calls to ejbStore()
  The `is-modified-method-name` deployment descriptor element applies to EJB 1.1 container-managed-persistence (CMP) beans only. This element is found in the `weblogic-ejb-jar.xml` file. WebLogic Server CMP implementation automatically detects modifications of CMP fields and writes only those changes to the underlying datastore.
- 5.1 weblogic-ejb-jar.xml Deployment Descriptor File Structure
  `The WebLogic Server 5.1 weblogic-ejb-jar.xml` file defines the EJB document type definitions (DTD) you use with EJB 1.1 beans. These deployment descriptor elements are WebLogic-specific.
- 5.1 weblogic-ejb-jar.xml Deployment Descriptor Elements
  Examine the deployment descriptor elements in `weblogic-ejb-jar.xml`.
- 1.1 weblogic-cmp-jar.xml Deployment Descriptor File Structure
  `weblogic-cmp-jar.xml` defines deployment elements for a single entity EJB that uses WebLogic Server RDBMS-based persistence services.
- 1.1 weblogic-cmp-jar.xml Deployment Descriptor Elements
  Examine the `weblogic-cmp-jar.xml` deployment descriptor elements.

Writing for RDBMS Persistence for EJB 1.1 CMP

Learn how to write finders for WebLogic-specific 1.1 EJBs that use RDBMS persistence. You can use EJB QL, a portable query language, to define finder queries for 2.0 EJBs with container-managed persistence.

Clients use finder methods to query and receive references to entity beans that fulfill query conditions.

WebLogic Server provides an easy way to write finders.

1. Write the method signature of a finder in the `EJBHome` interface.
2. Define the finder's query expressions in the `ejb-jar.xml` deployment file.

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

The key components of a finder for RDBMS persistence are:

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

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

Finder Signature

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

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

finder-list Element

The `finder-list` element 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` element using WebLogic Server RDBMS-based persistence:

```
<finder-list>
    <finder>
        <method-name>findBigAccounts</method-name>
        <method-params>
            <method-param>double</method-param>
```

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

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

finder-query Element

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

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

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

Using WebLogic Query Language (WLQL) for EJB 1.1 CMP

WebLogic Query Language (WLQL) for EJB 1.1 CMP allows you to query 1.1 entity EJBs with container-managed persistence. In the `weblogic-cmp-jar.xml` file, each `finder-query` element must include a WLQL string that defines the query used to return EJBs.

Use WLQL for EJBs and their corresponding deployment files that are based on the EJB 1.1 specification.

WLQL Syntax

WLQL strings use the prefix notation for comparison operators, as follows:

```
(operator operand1 operand2)
```

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

WLQL Operators

The following are valid WLQL operators.

Table E-1 WLQL Operators

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

WLQL Operands

Valid WLQL operands include:

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

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

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

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

Examples of WLQL Expressions

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

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

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

  The sample `<finder>` element is:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Using SQL for CMP 1.1 Finder Queries

WebLogic Server allows you to use a SQL string instead of the standard WLQL query language to write SQL for a CMP 1.1 finder query. The SQL statement retrieves the values from the database for the CMP 1.1 finder query. Use SQL to write a CMP 1.1 finder query when a more complicated finder query is required and you cannot use WLQL.

For more information on WLQL, see Using WebLogic Query Language (WLQL) for EJB 1.1 CMP.

To specify this SQL finder query:

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

   `findBigAccounts(double cutoff)` as follows:

   ```
   <finder-sql><![CDATA{balance >$0]]></finder-sql>
   ```

   Use values such as $0 or $1 in the SQL string to reference the parameters to the finder method. The WebLogic Server EJB container replaces the `$` parameters but will not interpret the SQL query.

2. The Container emits the following SQL:

   ```
   SELECT <columns> FROM table WHERE balance > ?
   ```

   The SQL should be the WHERE clause of an SQL statement. The Container prepends the SELECT and FROM clauses. The WHERE clause may contain arbitrary SQL.

If you use characters in your SQL query that may confuse an XML parser, such as the greater than (**>**) symbol and the less than (**<**) symbol, make sure that you declare the SQL query using the CDATA format shown in the preceding sample SQL statement.

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

Tuned EJB 1.1 CMP Updates in WebLogic Server

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

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

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

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

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

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

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

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

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

The `is-modified-method-name` deployment descriptor element applies to EJB 1.1 container-managed-persistence (CMP) beans only. This element is found in the `weblogic-ejb-jar.xml` file. WebLogic Server CMP implementation automatically detects modifications of CMP fields and writes only those changes to the underlying datastore.

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

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

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

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

5.1 weblogic-ejb-jar.xml Deployment Descriptor File Structure

The WebLogic Server 5.1 `weblogic-ejb-jar.xml` file defines the EJB document type definitions (DTD) you use with EJB 1.1 beans. These deployment descriptor elements are WebLogic-specific.

The top level elements in the WebLogic Server 5.1 `weblogic-ejb-jar.xml` are:

- `description`
- `weblogic-version`
  - `weblogic-enterprise-bean`
  - `ejb-name`
  - `caching-descriptor`
  - `persistence-descriptor`
  - `clustering-descriptor`
  - `transaction-descriptor`
  - `reference-descriptor`
  - `jndi-name`
  - `transaction-isolation`
- `security-role-assignment`

5.1 weblogic-ejb-jar.xml Deployment Descriptor Elements

Examine the deployment descriptor elements in `weblogic-ejb-jar.xml`.

The following sections describe WebLogic-Server 5.1 `weblogic-ejb-jar.xml` deployment descriptor elements.

caching-descriptor

The `caching-descriptor` element 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 element, as well as each of its child elements, is optional. WebLogic Server uses default values where no elements are defined.

The following is a sample `caching-descriptor` element that shows the caching elements described in this section:

```
<caching-descriptor>
    <max-beans-in-free-pool>500</max-beans-in-free-pool>
    <initial-beans-in-free-pool>50</initial-beans-in-free-pool>
    <max-beans-in-cache>1000</max-beans-in-cache>
    <idle-timeout-seconds>20</idle-timeout-seconds>
    <cache-strategy>Read-Write</cache-strategy>
    <read-timeout-seconds>0</read-timeout-seconds>
</caching-descriptor>
```

max-beans-in-free-pool

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

WebLogic Server maintains a free pool of EJBs for every bean class. This optional element defines the size of the pool. By default, `max-beans-in-free-pool` has no limit; the maximum number of beans in the free pool is limited only by the available memory.

initial-beans-in-free-pool

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

If you specify a value for `initial-bean-in-free-pool`, WebLogic Server populates the free pool with the specified number of bean instances at startup. Populating the free pool in this way improves initial response time for the EJB, since initial requests for the bean can be satisfied without generating a new instance.

`initial-bean-in-free-pool` defaults to 0 if the element is not defined.

max-beans-in-cache

> **Note:**
>
> This element is valid only for stateful session EJBs and entity EJBs.

This element specifies the maximum number of objects of this class that are allowed in memory. When `max-beans-in-cache` is reached, WebLogic Server passivates some EJBs that have not been recently used by a client. `max-beans-in-cache` also affects when EJBs are removed from the WebLogic Server cache.

The default value of `max-beans-in-cache` is 100.

idle-timeout-seconds

`idle-timeout-seconds` defines the maximum length of time a stateful EJB should remain in the cache. After this time has elapsed, WebLogic Server may remove the bean instance if the number of beans in cache approaches the limit of `max-beans-in-cache`.

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

cache-strategy

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

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

The default value is `Read-Write`.

read-timeout-seconds

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

persistence-descriptor

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

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

is-modified-method-name

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

Providing a method and setting it as appropriate can improve performance. However, any errors in the method's return value can cause data inconsistency problems.

delay-updates-until-end-of-tx

Set this property to `true` (the default), to update the persistent store of all beans in a transaction at the completion of the transaction. This generally improves performance by avoiding unnecessary updates. However, it does not preserve the ordering of database updates within a database transaction.

If your datastore uses an isolation level of `TransactionReadCommittedUncommitted`, you may want to allow other database users to view the intermediate results of in-progress transactions.

In this case, set `delay-updates-until-end-of-tx` to `false` to update the bean's persistent store at the conclusion of each method invoke.

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

persistence-type

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

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

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

  > **Note:**
  >
  > The specified version must *exactly* match the RDBMS persistence version for the WebLogic Server release. Specifying an incorrect version results in the error:
  >
  > ```
  > weblogic.ejb.persistence.PersistenceSetupException: Error
  > initializing the CMP Persistence Type for your bean: No installed
  > Persistence Type matches the signature of (identifier
  > 'Weblogic_CMP_RDBMS', version 'version_number').
  > ```

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

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

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

```
<persistence-type>
    <type-identifier>WebLogic_CMP_RDBMS</type-identifier>
    <type-version>5.1.0</type-version>
    <type-storage>META-INF\weblogic-cmp-jar.xml</type-storage>
</persistence-type>
```

db-is-shared

The `db-is-shared` element applies only to entity beans. When set to `true` (the default value), WebLogic Server assumes that EJB data could be modified between transactions and reloads data at the beginning of each transaction. When set to `false`, WebLogic Server assumes that it has exclusive access to the EJB data in the persistent store.

stateful-session-persistent-store-dir

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

persistence-use

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

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

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

clustering-descriptor

The `clustering-descriptor` element defines the replication properties and behavior for EJBs deployed in a WebLogic Server cluster. The `clustering-descriptor` element and each of its child elements are optional, and are not applicable to single-server systems.

The following shows all elements contained in the `clustering-descriptor` element:

```
<clustering-descriptor>
    <home-is-clusterable>. . .</home-is-clusterable>
    <home-load-algorithm>. . .</home-load-algorithm>
    <home-call-router-class-name>. . .</home-call-router-class-name>
    <stateless-bean-is-clusterable>. . .</stateless-bean-is-clusterable>
    <stateless-bean-load-algorithm>. . .</stateless-bean-load-algorithm>
    <stateless-bean-call-router-class-name>. . .
    </stateless-bean-call-router-class-name>
    <stateless-bean-methods-are-idempotent>. . .
    </stateless-bean-methods-are-idempotent>
</clustering-descriptor>
```

home-is-clusterable

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

home-load-algorithm

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

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

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

home-call-router-class-name

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

stateless-bean-is-clusterable

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

stateless-bean-load-algorithm

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

stateless-bean-call-router-class-name

This property is similar to `home-call-router-class-name`, but it is applicable only to stateless session EJBs.

stateless-bean-methods-are-idempotent

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

> **Note:**
>
> This property is applicable only to stateless session EJBs.

transaction-descriptor

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

```
<transaction-descriptor>
    <trans-timeout-seconds>20</trans-timeout-seconds>
</transaction-descriptor>
```

trans-timeout-seconds

The `trans-timeout-seconds` element specifies the maximum duration for the EJB's container-initiated transactions. If a transaction lasts longer than `trans-timeout-seconds`, WebLogic Server rolls back the transaction.

If you specify no value for `trans-timeout-seconds`, container-initiated transactions timeout after five minutes, by default.

reference-descriptor

The `reference-descriptor` element 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` element contains one or more additional elements to define resource factory references and EJB references. The following shows the organization of these elements:

```
<reference-descriptor>
    <resource-description>
        <res-ref-name>. . .</res-ref-name>
        <jndi-name>. . .</jndi-name>
    </resource-description>
    <ejb-reference-description>
        <ejb-ref-name>. . .</ejb-ref-name>
        <jndi-name>. . .</jndi-name>
    </ejb-reference-description>
</reference-descriptor>
```

resource-description

The following elements define an individual `resource-description`:

- `res-ref-name` specifies a resource reference name. This is the reference that the EJB provider places within the `ejb-jar.xml` deployment file.
- `jndi-name` specifies the JNDI name of an actual resource factory available in WebLogic Server.

ejb-reference-description

The following elements define an individual `ejb-reference-description`:

- `ejb-ref-name` specifies an EJB reference name. This is the reference that the EJB provider places within the `ejb-jar.xml` deployment file.
- `jndi-name` specifies the JNDI name of an actual EJB available in WebLogic Server.

enable-call-by-reference

By default, EJB methods called from within the same EAR pass arguments by reference. This increases the performance of method invocation since parameters are not copied.

If you set `enable-call-by-reference` to `false`, parameters to EJB methods are copied (pass by value) in accordance with the EJB 1.1 specification. Pass by value is always necessary when the EJB is called remotely (not from within the same application).

jndi-name

The `jndi-name` element specifies a jndi-name for a bean, resource, or reference.

transaction-isolation

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

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

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

isolation-level

The `transaction-isolation` element defines method-level transaction isolation settings for an EJB. Allowable values include:

- `TransactionSerializable`—Simultaneously executing this transaction multiple times has the same effect as executing the transaction multiple times in a serial fashion.

> **Note:**
>
> For Oracle databases, Oracle recommends that you use the `TransactionReadCommittedForUpdate` isolation level instead of the `TransactionSerializable` isolation level. This is because Oracle databases do not lock read data at the `TransactionSerializable` isolation level. Additionally, at the `TransactionSerializable` isolation level, it is possible for concurrent transactions on Oracle databases to proceed without throwing the Oracle exception ORA-08177 "`can't serialize access for this transaction`"). For more information on the `TransactionReadCommittedForUpdate` isolation level, see Oracle-Only Isolation Levels.

- `TransactionReadCommitted`—The transaction can view only committed updates from other transactions
- `TransactionReadUncommitted`—The transaction can view uncommitted updates from other transactions.
- `TransactionRepeatableRead`—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.

Oracle-Only Isolation Levels

These additional values are supported only for Oracle databases, and only for container-managed persistence (CMP) EJBs:

- `TransactionReadCommittedForUpdate`—Supported only for Oracle databases, and only for container-managed persistence (CMP) EJBs. This value sets the isolation level to `TransactionReadCommitted`, and for the duration of the transaction, all SQL `SELECT` statements executed in any method are executed with `FOR UPDATE` appended to them. This causes the selected rows to be locked for update. If Oracle cannot lock the rows affected by the query immediately, then it waits until the rows are free. This condition remains in effect until the transaction does a `COMMIT` or `ROLLBACK`.

  This isolation level can be used to avoid the error:

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

  which can (but does not always) occur when using the `TransactionSerializable` isolation level with Oracle databases.

  > **Note:**
  >
  > For Oracle databases, Oracle recommends that you use this isolation level (`TransactionReadCommittedForUpdate`) instead of the `TransactionSerializable` isolation level. This is because Oracle databases do not lock read data at the `TransactionSerializable` isolation level.

- `TransactionReadCommittedNoWait`—Supported only for Oracle databases, and only for container-managed persistence (CMP) EJBs.

This value sets the isolation level to `TransactionReadCommitted`, and for the duration of the transaction, all SQL `SELECT` statements executed in any method are executed with `FOR UPDATE NO WAIT` appended to them. This causes the selected rows to be locked for update.

In contrast to the `TransactionReadCommittedForUpdate` setting, `TransactionReadCommittedForUpdateNoWait` causes the Oracle DBMS to `NOT WAIT` if the required locks cannot be acquired immediately—the affected `SELECT` query will fail and an exception will be thrown by the Container.

Refer to your database documentation for more information support for different isolation levels.

method

The `method` element 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 element 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 parent `method-params` element.

For example, the following method element designates all methods in the "AccountBean" EJB:

```
<method>
    <ejb-name>AccountBean</ejb-name>
    <method-name>*</method-name>
</method>
```

The following element designates all methods in the remote interface of "AccountBean:"

```
<method>
    <ejb-name>AccountBean</ejb-name>
    <method-intf>Remote</method-intf>
    <method-name>*</method-name>
</method>
```

security-role-assignment

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

`security-role-assignment` can contain one or more pairs of the following elements:

- `role-name` is the application role name that the EJB provider placed in the `ejb-jar.xml` deployment file.
- `principal-name` specifies the name of an actual WebLogic Server principal.

1.1 weblogic-cmp-jar.xml Deployment Descriptor File Structure

`weblogic-cmp-jar.xml` defines deployment elements for a single entity EJB that uses WebLogic Server RDBMS-based persistence services.

The top-level element of the WebLogic Server 1.1 `weblogic-cmp-jar.xml` consists of a `weblogic-enterprise-bean` element:

```
description
weblogic-version
<weblogic-enterprise-bean>
    <pool-name>finance_pool</pool-name>
    <schema-name>FINANCE_APP</schema-name>
    <table-name>ACCOUNT</table-name>
    <attribute-map>
        <object-link>
            <bean-field>accountID</bean-field>
            <dbms-column>ACCOUNT_NUMBER</dbms-column>
        </object-link>
        <object-link>
            <bean-field>balance</bean-field>
            <dbms-column>BALANCE</dbms-column>
        </object-link>
    </attribute-map>
    <finder-list>
        <finder>
            <method-name>findBigAccounts</method-name>
            <method-params>
                <method-param>double</method-param>
            </method-params>
            <finder-query><![CDATA[(> balance $0)]]></finder-query>
            <finder-expression>. . .</finder-expression>
        </finder>
    </finder-list>
</weblogic-enterprise-bean>
```

1.1 weblogic-cmp-jar.xml Deployment Descriptor Elements

Examine the `weblogic-cmp-jar.xml` deployment descriptor elements.

This section describes the deployment descriptor elements.

- RDBMS Definition Elements
- enable-tuned-updates
- pool-name
- schema-name
- table-name
- EJB Field-Mapping Elements
- attribute-map
- object-link
- bean-field
- dbms-column

- Finder Elements
- finder-list
- finder
- method-name
- method-params
- method-param
- finder-query
- finder-expression

RDBMS Definition Elements

This section describes the RDBMS definition elements.

enable-tuned-updates

`enable-tuned-updates` specifies that when `ejbStore` is called the EJB container automatically determines which container-managed fields have been modified and then writes only those fields back to the database.

pool-name

`pool-name` specifies name of the WebLogic Server connection pool to use for this EJB's database connectivity.

schema-name

`schema-name` specifies the schema where the source table is located in the database. This element is required only if you want to use a schema that is not the default schema for the user defined in the EJB's connection pool.

> **Note:**
>
> This field is case sensitive, although many SQL implementations ignore case.

table-name

`table-name` specifies the source table in the database. This element is required in all cases.

> **Note:**
>
> The user defined in the EJB's connection pool must have read and write privileges to the specified table, though not necessarily schema modification privileges. This field is case sensitive, although many SQL implementations ignore case.

EJB Field-Mapping Elements

This section describes the EJB field-mapping elements.

attribute-map

The `attribute-map` element 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` element, which represents a link between a column in the database and a field in the EJB instance.

bean-field

`bean-field` specifies the field in the EJB instance that should be populated from the database. This element is case sensitive and must precisely match the name of the field in the bean instance.

The field referenced in this tag must also have a `cmp-field` element defined in the `ejb-jar.xml` file for the bean.

dbms-column

`dbms-column` specifies the database column to which the EJB field is mapped. This tag is case sensitive, although many databases ignore the case.

> **Note:**
>
> WebLogic Server does not support quoted RDBMS keywords as entries to `dbms-column`. For example, you cannot create an attribute map for column names such as "create" or "select" if those names are reserved in the underlying datastore.

Finder Elements

This section describes the finder elements.

finder-list

The `finder-list` element defines the set of all finders that are generated to locate sets of beans.

`finder-list` must contain exactly one entry for each finder method defined in the home interface, except for `findByPrimaryKey`. If an entry is not provided for `findByPrimaryKey`, one is generated at compilation time.

> **Note:**
>
> If you 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` element describes a finder method defined in the home interface. The elements contained in the `finder` element 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` element defines the list of parameters to the finder method being specified in `method-name`.

> **Note:**
>
> WebLogic Server compares this list against the parameter types for the finder method in the EJB's home interface; the order and type for the parameter list must exactly match the order and type defined in the home interface.

method-param

`method-param` defines the fully-qualified name for the parameter's type. The type name is evaluated into a `java.lang.Class` object, and the resultant object must precisely match the respective parameter in the EJB's finder method.

You can specify primitive parameters using their primitive names (such as "double" or "int"). If you use a non-primitive data type in a `method-param` element, you must specify a fully qualified name. For example, use `java.sql.Timestamp` rather than `Timestamp`. If you do not use a qualified name, `appc` generates an error message when you compile the deployment unit.

finder-query

`finder-query` specifies the WebLogic Query Language (WLQL) string that is used to retrieve values from the database for this finder.

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

finder-expression

`finder-expression` specifies a Java language expression to use as a variable in the database query for this finder.

Future versions of the WebLogic Server EJB container will use the EJB QL query language (as required by the EJB 2.0 specification at `http://java.sun.com/products/ejb/docs.html`). 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.

F

EJB Query Language (EJB-QL) and WebLogic Server

EJB QL is a portable query language that defines finder methods for 2.0 entity EJBs with container-managed persistence. Use this SQL-like language to select one or more entity EJB objects or fields in your query. You can create queries in the deployment descriptor for any finder method other than `findByPrimaryKey()`. `findByPrimaryKey` is automatically handled by the EJB container.
This appendix includes the following topics:

- EJB QL Requirement for EJB 2.x Beans
  The deployment descriptors must define each finder query for EJB 2.x entity beans by using an EJB QL query string. You cannot use WebLogic Query Language (WLQL) with EJB 2.x entity beans. WLQL is intended for use with EJB 1.1 container-managed persistence.
- Using the EJB 2.x WebLogic QL Extension for EJB QL
  WebLogic Server has an SQL-like language, called WebLogic QL, that extends the standard EJB QL. You define the `query` in the `weblogic-cmp-jar.xml` deployment descriptor using the `weblogic-ql` element.
- Properties-Based Methods of the Query Interface
  The `Query` interface contains both find and execute methods. The find methods work like standard EJB methods, in that they return EJBObjects. The execute methods work more like `Select` statements in that you can select individual fields.
- Migrating from WLQL to EJB QL
  If you have an existing application that uses EJB 1.1, your container-managed entity EJBs can use WLQL for finder methods. You can map the WLQL syntax to EJB QL syntax.
- Known Issue with Implied Cross Products
  The known issue when an EJB QL query contains an implied cross product—as opposed to an explicit one—the EJB-QL query can return an empty result.
- EJB QL Error-Reporting
  You can identify which part of the query is in error using the compiler error messages in EJB QL.

EJB QL Requirement for EJB 2.x Beans

The deployment descriptors must define each finder query for EJB 2.x entity beans by using an EJB QL query string. You cannot use WebLogic Query Language (WLQL) with EJB 2.x entity beans. WLQL is intended for use with EJB 1.1 container-managed persistence.

For more information on WLQL and EJB 1.1 container-managed persistence, see Using WebLogic Query Language (WLQL) for EJB 1.1 CMP.

Using the EJB 2.x WebLogic QL Extension for EJB QL

WebLogic Server has an SQL-like language, called WebLogic QL, that extends the standard EJB QL. You define the `query` in the `weblogic-cmp-jar.xml` deployment descriptor using the `weblogic-ql` element.

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

These topics provide guidelines for using the WebLogic QL extension to EJB 2.x QL:

upper and lower Functions

The EJB WebLogic QL `upper` and `lower` extensions convert the case of arguments to allow finder methods to return results that match the characters in a search expression but not the case. The case change is transient, for the purpose of string matching, and is not persisted in database. The underlying database must also support `upper` and `lower` functions.

upper

The `upper` function converts characters in its arguments from any case to upper case before string matching is performed. Use the `upper` function with an upper-case expression in a query to return all items that match the expression, regardless of case. For example:

```
select name from products where upper(name)='DETERGENT';
```

lower

The `lower` function converts characters in its arguments from any case to lower case before string matching is performed. Use the `lower` function with a lower-case expression in a query to return all items that match the expression, regardless of case.

```
select type from products where lower(name)='domestic';
```

Using ORDER BY

The EJB QL `ORDER BY` clause is a keyword that works with the `Finder` method to specify the CMP field selection sequence for your selections.

Example F-1 ORDER BY Showing Order by id

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

You can specify an `ORDER BY` with ascending [`ASC`] or descending [`DESC`] order for multiple fields as follows. If you do not specify an order, `ORDER BY` defaults to ascending order.

Example F-2 ORDER BY Showing Order by id with ASC and DESC

```
ORDER BY <field> [ASC|DESC], <field> [ASC|DESC]
SELECT OBJECT(A) from A for Account.Bean, OBJECT(B) from B for Account.Bean
ORDER BY A.id ASC; B.salary DESC
```

Using Subqueries

WebLogic Server supports the use of the following features with subqueries in EJB QL:

- Subquery return type
  - Single `cmp-field`s
  - Aggregate functions
  - Beans with simple primary keys
- Subqueries as comparison operands
- Correlated subqueries
- Uncorrelated subqueries
- `DISTINCT` clauses with subqueries

The relationship between WebLogic QL and subqueries is similar to the relationship between SQL queries and subqueries. Use WebLogic QL subqueries in the `WHERE` clause of an outer WebLogic QL query. With a few exceptions, the syntax for a subquery is the same as a WebLogic QL query.

To specify WebLogic QL, see Using the EJB 2.x WebLogic QL Extension for EJB QL. Use those instructions with a `SELECT` statement that specifies a subquery as shown the following sample.

The following query selects all above average students as determined by the provided grade number:

```
SELECT OBJECT(s) FROM studentBean AS s WHERE s.grade > (SELECT AVG(s2.grade) FROM
StudentBean AS s2)
```

> **Note:**
>
> In the above query the subquery, (SELECT AVG(s2.grade) FROM StudentBean AS s2), has the same syntax as an EJB QL query.

You can create nested subqueries.The depth is limited by the underlying database's nesting capabilities.

In a WebLogic QL query, the identifiers declared in the `FROM` clauses of the main query and all of its subqueries must be unique. This means that a subquery may not re-declare a previously declared identifier for local use within that subquery.

For example, the following example is not legal because `EmployeeBean` is being declared as emp in both the query and the subquery:

```
SELECT OBJECT(emp)
FROM EmployeeBean As emp
WHERE emp.salary=(SELECT MAX(emp.salary) FROM
EmployeeBean AS emp WHERE employee.state=MA)
```

Instead, this query should be written as follows:

```
SELECT OBJECT(emp)
FROM EmployeeBean As emp
WHERE emp.salary=(SELECT MAX(emp2.salary) FROM
EmployeeBean AS emp2 WHERE emp2.state=MA)
```

The above examples correctly declare the subquery's employee bean to have a different identifier from the main query's employee bean.

Subquery Return Types

The return type of a WebLogic QL subquery can be one of a number of different types, such as:

Single cmp-field Type Subqueries

WebLogic Server supports a return type consisting of a `cmp-field`. The results returned by the subquery can consist of a single value or collection of values. An example of a subquery that returns value(s) of the type `cmp-field` is as follows:

```
SELECT emp.salary FROM EmployeeBean AS emp WHERE emp.dept = 'finance'
```

This subquery selects all of the salaries of employees in the finance department.

Aggregate Functions

WebLogic Server supports a return type consisting of an aggregate of a `cmp-field`. As an aggregate always consist of a single value, the value returned by the aggregate is always a single value. An example of a subquery that return a value of the type aggregate (`MAX`) of a `cmp-field` is as follows:

```
SELECT MAX(emp.salary) FROM EmployeeBean AS emp WHERE emp.state=MA
```

This subquery selects the single highest employee salary in Massachusetts.

For more information on aggregate functions, see Using Aggregate Functions.

Beans with Simple Primary Key

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

The following example illustrates a subquery that returns the value(s) of the type bean with a simple primary key:

```
SELECT OBJECT(emp) FROM EMployeeBean As emp WHERE emp.department.budget>1,000,000
```

This subquery provides a list of all employee in departments with budgets greater than $1,000,000.

> **Note:**
>
> Beans with compound primary keys are NOT supported. Attempts to designate the return type of a subquery to a bean with a compound primary key will fail when you compile the query.

Subqueries as Comparison Operands

Use subqueries as the operands of comparison operators and arithmetic operators. WebLogic QL supports subqueries as the operands of:

- these comparison operators: `[NOT]IN, [NOT]EXISTS`

and

- these arithmetic operators: `<, >, <=, >=, =, <> with ANY and ALL`

[NOT]IN

The `[NOT]IN` comparison operator tests whether the left-had operand is or is not a member of the subquery operand on the right-hand side.

An example of a subquery which is the right-hand operand of the `NOT IN` operator is as follows:

```
SELECT OBJECT(item)
FROM ItemBean AS item
WHERE item.itemId NOT IN
(SELECT oItem2.item.itemID
FROM OrderBean AS orders2, IN(orders2.orderItems)oIttem2
```

The subquery selects all items from all orders.

The main query's `NOT IN` operator selects all the items that are not in the set returned by the subquery. So the end result is that the main query selects all unordered items.

[NOT]EXISTS

The `[NOT]EXISTS` comparison operator tests whether the set returned by the subquery operand is or is not empty.

An example of a subquery which is the operand of the `NOT EXISTS` operand is as follows:

```
SELECT (cust) FROM CustomerBean AS cust
WHERE NOT EXISTS
(SELECT order.cust_num FROM OrderBean AS order
WHERE cust.num=order_num)
```

This is an example of a query with a correlated subquery. See Correlated and Uncorrelated Subqueries for more information. the following query returns all customers that have not placed an order.

```
SELECT (cust) FROM CustomerBean AS cust
WHERE cust.num NOT IN
(SELECT order.cust_num FROM OrderBean AS order
WHERE cust.num=order_num)
```

Arithmetic Operators

Use arithmetic operators for comparison when the right-hand subquery operand returns a single value. If the right hand subquery instead returns multiple values, then the qualifiers `ANY` or `ALL` must precede the subquery.

An example of a subquery which uses the '=' operator is as follows:

```
SELECT OBJECT (order)
FROM OrderBean AS order, IN(order.orderItems)oItem
WHERE oItem.quantityOrdered =
(SELECT MAX (subOItem.quantityOrdered)
FROM Order ItemBean AS subOItem
WHERE subOItem,item itemID = ?1)
AND oItem.item.itemId = ?1
```

For a given `itemId`, the subquery returns the maximum quantity ordered of that item. Note that this aggregate returned by the subquery is a single value as required by the '=' operator.

For the same given `itemId`, the main query's `'='` comparison operator checks which order's `OrderItem.quantityOrdered` equals the maximum quantity returned by the subquery. The end result is that the query returns the OrderBean that contains the maximum quantity of a given item that has been ordered.

Use arithmetic operators in conjunction with `ANY` or `ALL`, when the right-hand subquery operand may return multiple values.

An example of a subquery which uses `ANY` and `ALL` is as follows:

```
SELECT OBJECT (order)
FROM OrderBean AS order, IN(order.orderItems)oItem
WHERE oItem.quantityOrdered > ALL
(SELECT subOItem.quantityOrdered
FROM OrderBean AS suborder IN (subOrder.orderItems)subOItem
WHERE subOrder,orderId = ?1)
```

For a given `orderId`, the subquery returns the set of `orderItem.quantityOrdered` of each item ordered for that `orderId`. The main query's '>' `ALL` operator looks for all orders whose `orderItem.quantityOrdered` exceeds all values in the set returned by the subquery. The end result is that the main query returns all orders in which all `orderItem.quantityOrdered` exceeds every `orderItem.quantityOrdered` of the input order.

> **Note:**
>
> Since the subquery can return multi-valued results that they '>'`ALL` operator is used rather then the `'>'` operator.

Correlated and Uncorrelated Subqueries

WebLogic Server supports both correlated and Uncorrelated subqueries.

Uncorrelated Subqueries

Uncorrelated subqueries may be evaluated independently of the outer query. An example of an uncorrelated subquery is as follows:

```
SELECT OBJECT(emp) FROM EmployeeBean AS emp
WHERE emp.salary>
(SELECT AVG(emp2.salary) FROM EmployeeBean AS emp2)
```

This example of a uncorrelated subquery selects the employees whose salaries are above average. This example uses the '>' arithmetic operator.

Correlated Subqueries

Correlated subqueries are subqueries in which values from the outer query are involved in the evaluation of the subquery. An example of a correlated subquery is as follows:

```
SELECT OBJECT (mainOrder) FROM OrderBean AS mainOrder
WHERE 10>
(SELECT COUNT (DISTINCT subOrder.ship_date)
FROM OrderBean AS subOrder
WHERE subOrder.ship_date>mainOrder.ship_date
AND mainOrder.ship_date IS NOT NULL
```

This example of a correlated subquery selects the last 10 shipped orders. This example uses the `NOT IN` operator.

> **Note:**
>
> Keep in mind that correlated subqueries can involve more processing overhead the uncorrelated subqueries.

- DISTINCT Clause with Subqueries

DISTINCT Clause with Subqueries

Use the `DISTINCT` clause in a subquery to enable an SQL `SELECT DISTINCT` in the subquery's generated SQL. Using a `DISTINCT` clause in a subquery is different from using one in a main query because the EJB container enforces the `DISTINCT` clause in a main query; whereas the `DISTINCT` clause in the subquery is enforced by the generated SQL `SELECT DISTINCT`. The following is an example of a `DISTINCT` clause in a subquery:

```
SELECT OBJECT (mainOrder) FROM OrderBean AS mainOrder
WHERE 10>
(SELECT COUNT (DISTINCT subOrder.ship_date)
FROM OrderBean AS subOrder
WHERE subOrder.ship_date>mainOrder.ship_date
AND mainOrder.ship_date IS NOT NULL
```

Using Arithmetic Functions

WebLogic Server supports arithmetic functions with WebLogic QL. To specify WebLogic QL, see Using the EJB 2.x WebLogic QL Extension for EJB QL. Use those examples with a SELECT statement that specifies an arithmetic function.

Table F-1 Arithmetic Functions

Arithmetic Function	Description
`ABS(number)`	Returns the absolute value of a (int, double, or float) number.

Table F-1 (Cont.) Arithmetic Functions

Arithmetic Function	Description
`MOD(int, int)`	Returns the value of `x` modulo `y`.
`SQRT(double)`	Returns the square root.

> **Note:**
>
> EJB QL arithmetic functions may not work with query parameters on DB2.

Using Aggregate Functions

WebLogic Server supports aggregate functions with WebLogic QL. You only use these functions as `SELECT` clause targets, not as other parts of a query, such as a `WHERE` clause. The aggregate functions behave like SQL functions. They are evaluated over the range of the beans returned by the `WHERE` conditions of the query

To specify WebLogic QL, see Using the EJB 2.x WebLogic QL Extension for EJB QL. Use those instructions with a SELECT statement that specifies an aggregate function as shown in the samples shown in the following table.

A list of the supported functions and sample statements follows:

Table F-2 Aggregate Functions

Aggregate Function	Valid Argument Data Types	Description	Sample Statement
`MIN(x)`	• character • date • numeric • string	Returns the minimum value of this field.	`SELECT MIN(t.price) FROM TireBean AS t WHERE t.size=?1` This statement selects the lowest price for a tire of a given input size.
`MAX(x)`	• character • date • numeric • string	Returns the maximum value of this field.	**`SELECT MAX(s.customer_count) FROM SalesRepBean AS s WHERE s.city='Los Angeles'`** This statement selects the maximum number of customers served by any single sales representative in Los Angeles.
`AVG( [DISTINCT] x)`	numeric	Returns the average value of this field	`SELECT AVG(b.price) FROM BookBean AS b WHERE b.category='computer_science'` This statement selects the Average Price of a book in the Computer Science category.

Table F-2 (Cont.) Aggregate Functions

Aggregate Function	Valid Argument Data Types	Description	Sample Statement
`SUM( [DISTINCT] x)`	numeric	Returns the sum of this field.	`SELECT SUM(s.customer_count) FROM SalesRepBean AS s WHERE s.city='Los Angeles'` This statement retrieves the total number of customers served by sales representatives in Los Angeles.
`COUNT( [DISTINCT] x)`	numeric	Returns the number of occurrences of a field.	`SELECT COUNT(s.deal.amount) FROM SalesRepBean AS s, IN(deal)s WHERE s.deal.status='closed' AND s.deal.amount>=1000000` This statement retrieves the number of closed deals for at least 1 million dollars.

Note:

In this release of WebLogic Server, you receive an `ObjectNotFoundException` if all of the following are true:

- your aggregate query uses the `SUM`, `AVG`, `MAX`, or `MIN` operator
- the result type of the select method is a primitive
- there are no values to which the aggregate function can be applied

  In pre-9.0 releases of WebLogic Server, you received a return value of `0` when the above conditions were all true.

The SQL Specification requires queries that select individual fields along with aggregates to include a `GROUP BY` clause.

You can return aggregate functions in ResultSets as described below.

Using Queries that Return ResultSets

WebLogic Server supports `ejbSelect()` queries that return the results of multi-column queries in the form of a `java.sql.ResultSet`. To support this feature, WebLogic Server allows you to use the `SELECT` clause to specify a comma delimited list of target fields as shown in the following query:

```
SELECT emmp.name, emp.zip FROM EmployeeBean AS emp
```

This query returns a `java.sqlResultSet` with rows whose columns are the values Employee's Name and Employee's Zip.

To specify WebLogic QL, see Using the EJB 2.x WebLogic QL Extension for EJB QL. Use those instructions with a query specifying a ResultSet as shown in the above query. Use those instructions with a SELECT statement that specifies an aggregate query like the samples shown in the following table.

ResultSets created in EJB QL can return `cmp-field` values or aggregates of `cmp-field` values, they cannot return beans.

In addition, you can create powerful queries, as described in the following example, when you combine `cmp-fields` and aggregate functions.

The following rows (beans) show the salaries of employees in different locations:

Table F-3 CMP Fields Showing Salaries of Employees in California

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

Table F-4 CMP Fields Showing Salaries of Employees in Arizona

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

Table F-5 CMP Fields Showing Salaries of Employees in Texas

Name	Location	Salary
Curly	TX	70,000
Larry	TX	180,000
Moe	TX	80,00

> **Note:**
>
> Each row represents a bean.

The following `SELECT` statement shows a query that uses ResultSets and the aggregate function (`AVG`) along with a `GROUP BY` statement and an `ORDER BY` statement using a descending sort to retrieve results from a multi-column query.

```
SELECT e.location, AVG(e.salary)
    FROM Finder EmployeeBean AS e
        GROUP BY e.location
            ORDER BY 2 DESC
```

The query shows the average salary of employees at each location in descending order. The number, 2, means that the `ORDER BY` sort is on the second item in the `SELECT` statement. The `GROUP BY` clause specifies the `AVERAGE` salary of employees with a matching `e.location` attribute.

The ResultSet, in descending order is as follows:

Location	Average
AZ	100,000
CA	105,000
TX	110,000

> **Note:**
>
> You can only use integers as `ORDER BY` arguments in queries that return ResultSets. WebLogic Server does not support the use of integers as `ORDER BY` arguments in any Finder or `ejbselect()` that returns beans.

Using Oracle SELECT HINTS

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

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

The WebLogic QL keyword and usage is as follows:

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

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

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

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

"get" and "set" Method Restrictions

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

Properties-Based Methods of the Query Interface

The `Query` interface contains both find and execute methods. The find methods work like standard EJB methods, in that they return EJBObjects. The execute methods work more like `Select` statements in that you can select individual fields.

The `Query` interface return type is a disconnected `ResultSet`, meaning you access the information from the returned object the same way you would access it from a `ResultSet`, except that the `ResultSet` does not hold open a database connection.

The `Query` interface's properties-based methods offer an alternate way of specifying settings particular to a query. The `QueryProperties` interface holds standard EJB query settings while the `WLQueryProperties` interface holds WebLogic-specific query settings.

Although the `Query` interface extends `QueryProperties`, the actual `Query` implementation extends `WLQueryProperties` so it can be safely cast, as in the example in Example F-3, which sets field group settings:

Example F-3 Setting Field Group Settings with WLQueryProperties

```
Query query=qh.createQuery(); ((WLQueryProperties) query).setFieldGroupName("myGroup");
Collection results=query.find(ejbql);
```

or

```
Query query=qh.createQuery(); Properties props = new Properties();
props.setProperty(WLQueryProperties.GROUP_NAME, "myGroup"); Collection
results=query.find(ejbql, props);
```

Migrating from WLQL to EJB QL

If you have an existing application that uses EJB 1.1, your container-managed entity EJBs can use WLQL for finder methods. You can map the WLQL syntax to EJB QL syntax.

This section provides a quick reference to common WLQL operations. Use this table to map the WLQL syntax to EJB QL syntax.

Table F-6 Migrating from WLQL to EJB QL

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

Known Issue with Implied Cross Products

The known issue when an EJB QL query contains an implied cross product—as opposed to an explicit one—the EJB-QL query can return an empty result.

Consider this example query:

```
SELECT OBJECT(e) FROM EmployeeBean AS e WHERE e.name LIKE 'Joe' OR e.acct.balance < 100
```

This query references `AccountEJB`, but `AccountEJB` is not listed in the `FROM` clause. The result of this query is identical to that of a query with `AccountEJB` explicitly listed in the `FROM` clause.

EJB QL Error-Reporting

You can identify which part of the query is in error using the compiler error messages in EJB QL.

Compiler error messages in EJB QL provide a visual aid to identify which part of the query is in error and allow the reporting of more than one error per compilation.

- Visual Indicator of Error in Query
- Multiple Errors Reported after a Single Compilation

Visual Indicator of Error in Query

When an error is reported, EJB QL indicates the location of the problem within these symbols: =>> <<=. These symbols are highlighted in red in the following sample compiler error report.

```
ERROR: Error from appc: Error while reading 'META-INF/FinderEmployeeBeanRDBMS.xml'. The error was:
Query:
EJB Name: FinderEmployeeEJB
Method Name: findThreeLowestSalaryEmployees
Parameter Types: (java.lang.String)
Input EJB Query: SELECT OBJECT(e) FROM FinderEmployeeBean e WHERE f.badField = '2' O
R (e.testId = ?1) ORDER BY e.salary
SELECT OBJECT(e ) FROM FinderEmployeeBean e
WHERE =>> f.badField <<= = '2' OR ( e.testId = ?1 ) ORDER BY e.salary
Invalid Identifier in EJB QL expression:
Problem, the path expression/Identifier 'f.badField' starts with an identifier: 'f'.
The identifier 'f', which can be either a range variable identifier or a collection member identifier,
is required to be declared in the FROM clause of its query or in the FROM clause of a parent query.
'f' is not defined in the FROM clause of either its query or in any parent query.
Action, rewrite the query paying attention to the usage of 'f.badField'.
```

Multiple Errors Reported after a Single Compilation

If a query contains multiple errors, EJB QL is now capable of reporting more than one of these after a single compilation. Previously, the compiler could only report one error per compilation. Reporting of subsequent errors required recompilation.

The compiler is not guaranteed to report all errors after a single compilation.