Oracle® Application Server TopLink

Application Developer's Guide

10*g* (9.0.4)

Part No. B10313-01

September 2003

Oracle Application Server TopLink Application Developer's Guide, 10*g* (9.0.4)

Part No. B10313-01

Contributing Authors: Jacques-Antoine Dubé, Rick Sapir, Peter Purich, Ellen Siegal (Editor)

Contents

3 Mapping

4 Sessions

5 Data Access

7 Transactions

8 Cache

9 Packaging for Deployment

10 Tuning for Performance

A Application Development Tools

B Configuring OracleAS TopLink for J2EE Containers

C Error Codes and Messages

Send Us Your Comments

Oracle Application Server TopLink Application Developer's Guide, 10*g* (9.0.4)

Part No. B10313-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, please indicate the document title and part number, and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: appserverdocs_us@oracle.com
- FAX: (650) 506-7225 Attn: Java Platform Group, Information Development Manager
- Postal service:
  Oracle Corporation
  Java Platform Group, Information Development Manager
  500 Oracle Parkway, Mailstop 4op9
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

This document provides the information required to build high performance applications. It also introduces the concepts with which you should be familiar to get the most out of Oracle Application Server TopLink.

This preface contains these topics:

- Intended Audience
- Documentation Accessibility
- Organization
- Related Documentation
- Conventions

Intended Audience

The *Oracle Application Server TopLink Application Developer's Guide* is intended for application developers creating Oracle Application Server TopLink applications.

This document assumes that you are familiar with the concepts of object-oriented programming, the Enterprise JavaBeans (EJB) specification, and your own particular Java development environment.

The document also assumes that you are familiar with your particular operating system (such as Windows, UNIX, or other). The general operation of any operating system is described in the user documentation for that system, and is not repeated in this manual.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to evolve over time, and Oracle Corporation is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at

`http://www.oracle.com/accessibility/`

Accessibility of Code Examples in Documentation JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation This documentation may contain links to Web sites of other companies or organizations that Oracle Corporation does not own or control. Oracle Corporation neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Organization

This document contains:

Chapter 1, "Understanding OracleAS TopLink"

This chapter contains general information on OracleAS TopLink. It discusses the OracleAS TopLink application space, development process, components, and the OracleAS TopLink metamodel.

Chapter 2, "OracleAS TopLink Architectures"

This chapter illustrates the five basic OracleAS TopLink architectures in use in projects all over the world.

Chapter 3, "Mapping"

This chapter contains information on creating mappings for your application. It includes discussions on the mapping features and functions you will use to build your project.

Chapter 4, "Sessions"

This chapter contains information on configuring and running sessions. It includes a discussion of the various types of session available in OracleAS TopLink, and the mechanisms and features OracleAS TopLink offers to customize and optimize your application at the session level.

Chapter 5, "Data Access"

This chapter describes how to access the data for your application. It includes discussions on database platforms and drivers, performance issues, and the OracleAS TopLink Software Development Kit (SDK).

Chapter 6, "Queries"

This chapter contains information on building and executing queries in an OracleAS TopLink application.

Chapter 7, "Transactions"

This chapter contains information on OracleAS TopLink transactions. It introduces the concepts of transactions and the OracleAS TopLink *Unit of Work*.

Chapter 8, "Cache"

This chapter contains information on the OracleAS TopLink cache, including discussions on cache isolation, cache synchronization, and other caching issues. It also introduces the concepts associated with running OracleAS TopLink in a clustered environment.

Chapter 9, "Packaging for Deployment"

This chapter contains information on packaging and deploying your OracleAS TopLink application.

Chapter 10, "Tuning for Performance"

This chapter contains information on optimizing your application for maximum efficiency and throughput.

Appendix A, "Application Development Tools"

This appendix contains information on the various tools included with OracleAS TopLink that help you get the most out of your application.

Appendix B, "Configuring OracleAS TopLink for J2EE Containers"

This appendix contains information on configuring OracleAS TopLink for use with J2EE containers.

Appendix C, "Error Codes and Messages"

This appendix contains information on exceptions and error codes you might encounter when building or running an OracleAS TopLink application.

Related Documentation

For more information, see these Oracle resources:

- *Oracle Application Server TopLink Release Notes*
- *Oracle Application Server 10g Release Notes*
- *Oracle Application Server TopLink Getting Started Guide*
- *Oracle Application Server TopLink API Reference*
- *Oracle Application Server TopLink Mapping Workbench User's Guide*

Printed documentation is available for sale in the Oracle Store at

`http://oraclestore.oracle.com`

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

```
http://otn.oracle.com/membership
```

If you already have a username and password for OTN, then you can go directly to the documentation section of the OTN Web site at

```
http://otn.oracle.com/docs
```

Conventions

This section describes the conventions used in the text and code examples of this documentation set. It describes:

- Conventions in Text
- Conventions in Code Examples

Conventions in Text

We use various conventions in text to help you more quickly identify special terms. The following table describes those conventions and provides examples of their use.

Convention	Meaning	Example
Italics	Italic typeface indicates book titles or emphasis.	*Oracle9i Database Concepts* Ensure that the recovery catalog and target database do *not* reside on the same disk.
`UPPERCASE monospace (fixed-width) font`	Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, usernames, and roles.	You can specify this clause only for a `NUMBER` column. You can back up the database by using the `BACKUP` command. Query the `TABLE_NAME` column in the `USER_TABLES` data dictionary view. Use the `DBMS_STATS.GENERATE_STATS` procedure.

Convention	Meaning	Example
`lowercase monospace (fixed-width) font`	Lowercase monospace typeface indicates executables, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, usernames and roles, program units, and parameter values. **Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	Enter `sqlplus` to open SQL*Plus. The password is specified in the `orapwd` file. Back up the datafiles and control files in the `/disk1/oracle/dbs` directory. The `department_id`, `department_name`, and `location_id` columns are in the `hr.departments` table. Set the `QUERY_REWRITE_ENABLED` initialization parameter to `true.` Connect as `oe` user. The `JRepUtil` class implements these methods.
`lowercase italic monospace (fixed-width) font`	Lowercase italic monospace font represents placeholders or variables.	You can specify the *`parallel_clause`*. Run `U`*`old_release`*`.SQL` where *`old_release`* refers to the release you installed prior to upgrading.

Conventions in Code Examples

Code examples illustrate SQL, PL/SQL, SQL*Plus, or other command-line statements. They are displayed in a monospace (fixed-width) font and separated from normal text as shown in this example:

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

The following table describes typographic conventions used in code examples and provides examples of their use.

Convention	Meaning	Example
[ ]	Brackets enclose one or more optional items. Do not enter the brackets.	`DECIMAL (`*`digits`*` [ , `*`precision`*` ])`
{ }	Braces enclose two or more items, one of which is required. Do not enter the braces.	`{ENABLE | DISABLE}`

|	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	`{ENABLE | DISABLE}`
		
		`[COMPRESS | NOCOMPRESS]`

Convention	Meaning	Example

`. . .`	Horizontal ellipsis points indicate either:
	■ That we have omitted parts of the code that are not directly related to the example
	■ That you can repeat a portion of the code
		`CREATE TABLE ... AS` *`subquery`*`;`
		
		`SELECT` *`col1`*`,` *`col2`*`, ... ,` *`coln`* `FROM employees;`

. . .	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	
Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown.	`acctbal NUMBER(11,2);` `acct    CONSTANT NUMBER(4) := 3;`
`Italics`	Italicized text indicates placeholders or variables for which you must supply particular values.	`CONNECT SYSTEM/`*`system_password`* `DB_NAME =` *`database_name`*

`UPPERCASE`	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase.	`SELECT last_name, employee_id FROM employees;`
		`SELECT * FROM USER_TABLES;`
		`DROP TABLE hr.employees;`

`lowercase`	Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files.
	Note: Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.
		`SELECT last_name, employee_id FROM employees;`
		`sqlplus hr/hr`
		`CREATE USER mjones IDENTIFIED BY ty3MU9;`

1 Understanding OracleAS TopLink

Oracle Application Server TopLink is an advanced object-to-relational persistence framework, suitable for a wide range of Java 2 Enterprise Edition (J2EE) and Java application architectures. OracleAS TopLink development tools and runtime capabilities reduce development and maintenance efforts, and increase enterprise application functionality. Use OracleAS TopLink to build high performance applications that store persistent data in a relational database.

This chapter introduces OracleAS TopLink and includes discussions on the following topics:

- Advantages of OracleAS TopLink
- OracleAS TopLink Components
- Application Development With OracleAS TopLink
- OracleAS TopLink Architectures Overview
- General Terms and Concepts

The remainder of this document provides the necessary details required to build J2EE applications with OracleAS TopLink.

Advantages of OracleAS TopLink

Enterprise applications rely on Java-to-database integration to implement objects and logic. OracleAS TopLink enables developers to efficiently develop and refine enterprise applications. To fully understand OracleAS TopLink, you must understand the problems that enterprise application developers face and how OracleAS TopLink resolves them.

OracleAS TopLink Problem Space

Java-to-database integration is a widely underestimated problem in enterprise Java applications. This complex problem involves more than reading from and writing to a database. The database world includes elements such as tables, rows, columns, and primary and foreign keys; the Java and J2EE world contains entity classes (regular Java classes or Enterprise JavaBeans (EJB) entity beans), business rules, complex relationships, and inheritance. Bridging these two fundamentally different technologies is a challenging and resource-intensive problem.

The process of translating object-oriented data into relational data is referred to as *object-relational* (O-R) *mapping*. To enable an O-R solution, developers must resolve the following O-R bridging issues:

- Fundamentally different technologies
- Different skill sets
- Different staff and ownership for each of the technologies
- Different modeling and design principles

Application developers need a product that enables them to integrate Java applications and relational databases, without compromising ideal application design or database integrity. In addition, Java developers need the ability to store (or *persist*) and retrieve business domain objects using a relational database as a repository.

The OracleAS TopLink solution is a persistence framework that manages O-R mapping in a seamless manner and enables developers to rapidly build applications that combine the best aspects of object technology and relational databases.

OracleAS TopLink Solution

OracleAS TopLink provides a mature and powerful solution that addresses the disparity between Java objects and relational databases. OracleAS TopLink enables developers to:

- Persist Java objects in virtually any relational database supported by a JDBC 2.0 compliant driver.
- Map any object model to any relational schema, using the OracleAS TopLink Mapping Workbench graphical mapping tool.
- Use OracleAS TopLink successfully, even if they are unfamiliar with SQL or JDBC, because OracleAS TopLink provides a clean, object-oriented view of relational databases.

Other OracleAS TopLink Advantages

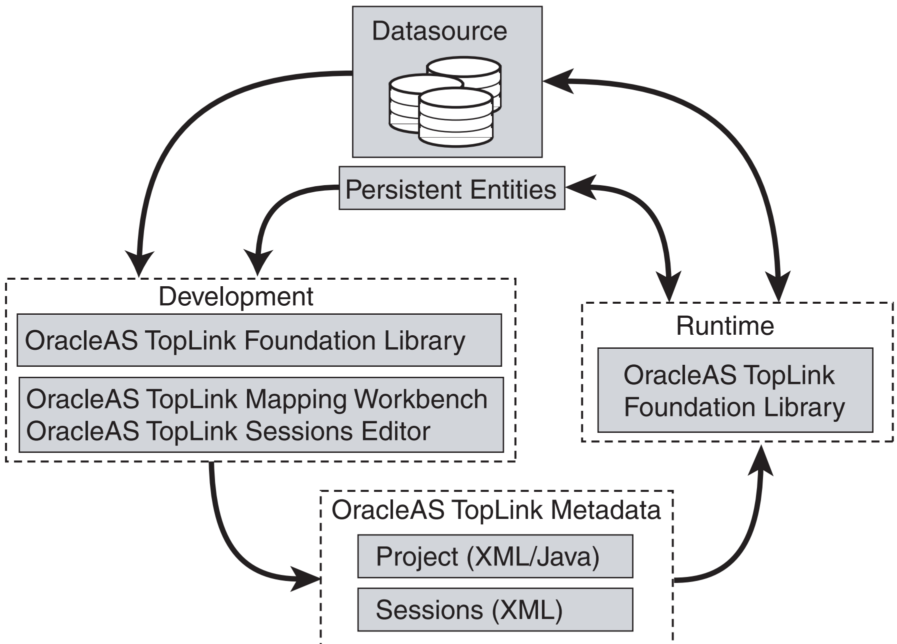
In addition to providing industry leading O-R mapping capabilities, OracleAS TopLink provides flexibility, increases performance and maximizes the productivity of your applications. OracleAS TopLink provides the following features:

- Advanced object caching that improves performance by minimizing database access.
- Rich query support that provides easy access to sophisticated, dynamic query languages and tools such as *query by example*, Java expression-based queries, EJB QL, and SQL.
- A transactional framework that enables developers to easily create and modify mapped objects. This framework integrates the complexities of a shared memory space and caches, and provides scalability that supports multiple server instances (clustering). Although the mechanisms involved are complex, OracleAS TopLink makes it easy to leverage this functionality by simplifying the task of writing transactional code that complies with database referential integrity and optimal access patterns.

OracleAS TopLink Components

At its core, OracleAS TopLink is a runtime engine that provides Java or J2EE applications with access to persistent entities stored in a relational database. In addition to runtime capabilities, the Oracle Application Server TopLink Foundation Library includes the OracleAS TopLink Application Programming Interface (API). This API enables applications to access OracleAS TopLink runtime features, as well as development tools that simplify application development. The tools capture mapping and runtime configuration information in metadata files that OracleAS TopLink passes to the runtime.

Figure 1–1 OracleAS TopLink Components in the Development Cycle



OracleAS TopLink Development Components

OracleAS TopLink application development comprises three elements: the development environment, the OracleAS TopLink runtime, and the metadata that ties them together.

Development

To create an OracleAS TopLink application, map the object and relational models using the OracleAS TopLink Mapping Workbench, and capture the resulting mappings and additional runtime configurations in the OracleAS TopLink project file (the `project.xml` file). Then build a session configuration file (the `sessions.xml` file) in the OracleAS TopLink Sessions Editor. These files together represent your entire OracleAS TopLink project.

During development, developers leverage the OracleAS TopLink API to define query and transaction logic. When developers use EJB entity beans, there is generally little or no direct use of the OracleAS TopLink API.

Runtime

The OracleAS TopLink Foundation Library provides the OracleAS TopLink runtime component. Access the runtime component either directly through the OracleAS TopLink API, or indirectly through a J2EE container when using EJB entity beans. The runtime engine is not a separate or external process; instead, it is embedded within the application. Application calls invoke OracleAS TopLink to provide persistence behavior. This function allows for transactional and thread-safe access to shared database connections and cached objects.

Metadata

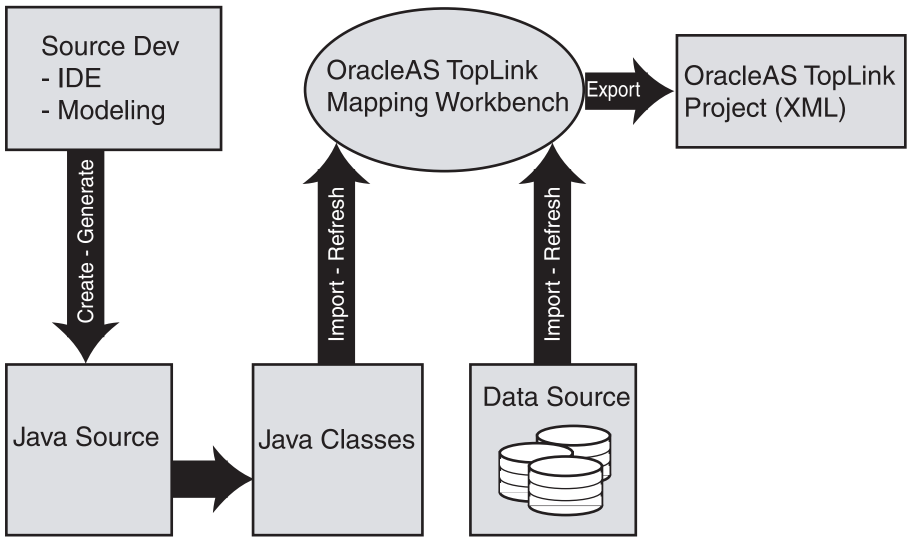
OracleAS TopLink metadata is the bridge between the development of an application and its deployed runtime. Capture the metadata using the OracleAS TopLink Mapping Workbench and OracleAS TopLink Sessions Editor, and pass the metadata to the runtime using deployment `project.xml` and `sessions.xml` files. It is also possible to hand-code these files using Java and the OracleAS TopLink API, but this approach is more labor-intensive.

The metadata, encapsulated in the `project.xml` file and the `sessions.xml` file, allows developers to pass configuration information into the runtime environment. The runtime uses the information in conjunction with the persistent entities (Java objects or EJB entity beans), and the code written with the OracleAS TopLink API, to complete the application.

OracleAS TopLink Mapping Workbench

The OracleAS TopLink Mapping Workbench is a graphical development tool that enables developers to map between the object and relational models, and configure many of the OracleAS TopLink Foundation Library features. The OracleAS TopLink Mapping Workbench creates an OracleAS TopLink project, the primary object in the OracleAS TopLink metamodel. Export the project as a single deployment XML file (the `project.xml` file), which OracleAS TopLink uses in conjunction with the OracleAS TopLink runtime to provide the application-specific persistence capabilities.

Figure 1–2 The OracleAS TopLink Mapping Workbench in an OracleAS TopLink Environment



The OracleAS TopLink Mapping Workbench can import compiled entity classes (Java objects or EJB entity beans), as well as relational schema through a JDBC driver that the developer configures. Because OracleAS TopLink imports the object and relational models for mapping, developers can develop the two models relatively independently from the O-R mapping phase of a project development.

Oracle Application Server TopLink Sessions Editor

Most OracleAS TopLink applications include a session configuration file, the `sessions.xml` file, to simplify the application deployment process. The OracleAS TopLink Sessions Editor provides a graphical environment to configure the `sessions.xml` file.

Use the `sessions.xml` file to configure one or more sessions for the OracleAS TopLink project, and associate the sessions with the project. This approach allows developers to specify individual configurations for each session and to add or modify:

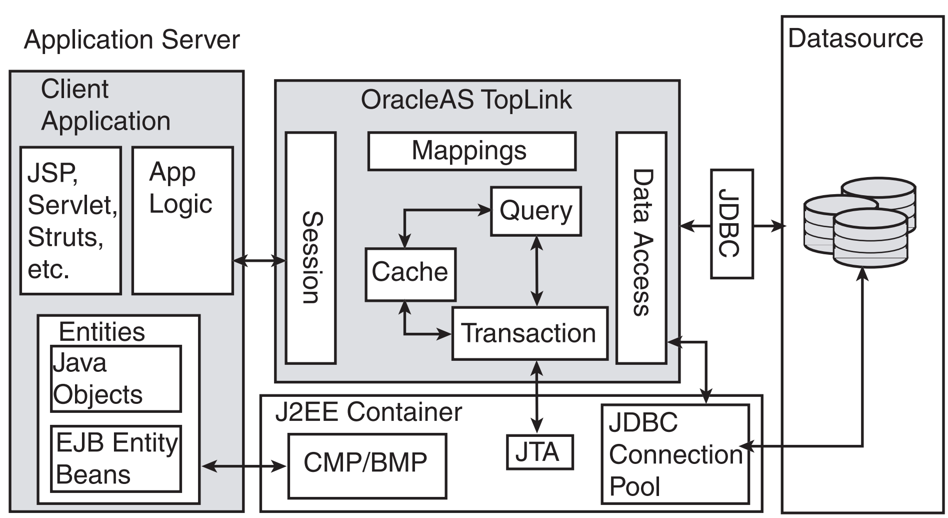
- Database (JDBC) login information different from the login information used during development (for example: external datasources for the host application server's connection pools)

- JTA/JTS transaction usage
- Cache synchronization
- Session broker (enables client applications to view multiple databases and projects as a single OracleAS TopLink session)

Oracle Application Server TopLink Foundation Library

The Oracle Application Server TopLink Foundation Library includes a Java library that forms the runtime component of the product. It provides support and the API for the components that make up an OracleAS TopLink application. The API enables developers to interact with OracleAS TopLink to retrieve and modify their application persistent entities.

Figure 1–3 OracleAS TopLink Application Components



Note: Although this chapter describes how these components fit into J2EE architectures, note that OracleAS TopLink also supports non-J2EE solutions. Chapter 2, "OracleAS TopLink Architectures" describes theses solutions in more detail.

Sessions

A session is the primary interface between the client application and OracleAS TopLink, and represents the connection to the underlying relational database. OracleAS TopLink offers several different session types, each optimized for different design requirements and architectures. The session manager configures and manages the session as a singleton within the application.

The most commonly-used session is the server session, a singleton session that clients access on the server through a client session. The server session provides a shared cache and shared JDBC connection resources. OracleAS TopLink supports sessions for two-tier architectures, distributed applications, and multiple databases.

Data Access

The OracleAS TopLink data access component provides access to JDBC connections through connection pooling, provided either by OracleAS TopLink or a host application server. This component manages the SQL generation required by the various query operations and reconciles any differences between JDBC drivers and SQL dialects. OracleAS TopLink offers many performance tuning options that optimize its data access capabilities.

Caching

OracleAS TopLink supplies an object level cache that guarantees object identity and enhances performance. Developers can configure the OracleAS TopLink cache and maximize application efficiency by reducing the number of times the application accesses the database. In a clustered environment, developers can configure OracleAS TopLink to synchronize changes with other instances of the deployed application.

Queries

The OracleAS TopLink query framework provides developers with the flexibility necessary to manage the complex persistence requirements of enterprise applications. The key features of this query framework include:

- A rich set of query types to allow object retrieval, summary results, and raw data retrieval
- The ability to specify search criteria using OracleAS TopLink Expressions (for object model based queries), EJB QL, SQL, stored procedures, or query by example

- Configuration options that enable developers to specify how the query is executed, and to customize many of its performance optimizing features

Developers can define OracleAS TopLink queries using the OracleAS TopLink Mapping Workbench, in Java code using the OracleAS TopLink API, or, in the case of EJB entity beans, through EJB Finders.

Transactions

OracleAS TopLink provides the ability to write transactional code isolated from the underlying database and schema. OracleAS TopLink achieves this functionality through the Unit of Work.

The Unit of Work isolates changes in a transaction from other threads until it successfully commits the changes to the database. Unlike other transaction mechanisms, the Unit of Work automatically manages changes to the objects in the transaction, the order of the changes, and changes that might invalidate other OracleAS TopLink caches. The Unit of Work manages these issues by calculating a minimal change set, ordering the database calls to comply with referential integrity rules and deadlock avoidance, and merging changed objects into the shared cache. In a clustered environment, the Unit of Work also synchronizes changes with the other servers in the cluster.

If an application uses EJB entity beans, developers do not access the Unit of Work API directly, but they still benefit from its features: the integration between the OracleAS TopLink runtime and the J2EE container leverages the Unit of Work automatically.

JTA/JTS Integration By default, OracleAS TopLink allows the application to create transaction boundaries for all object-level changes. OracleAS TopLink explicitly manages the database transaction, and if it encounters problems, safely rolls back both the database changes and the object-level changes.

In the case of a J2EE application, developers can configure OracleAS TopLink to synchronize with the JTA/JTS subsystem of the host application server. This feature allows an application to use container-managed transactions, rather than the default user-managed transactions.

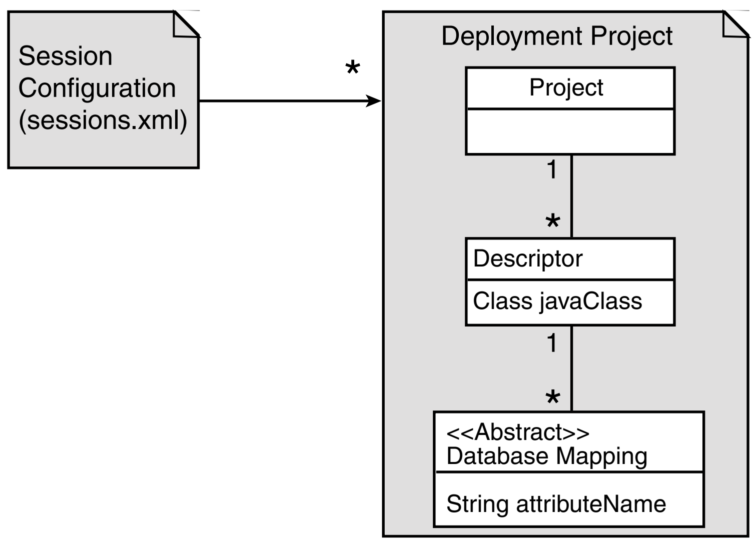
Note that this functionality is not limited to EJB architectures. Developers can configure any OracleAS TopLink architecture to use container-managed transactions.

OracleAS TopLink Metadata

The OracleAS TopLink approach to persistence is based on metadata that defines the class structure (objects) and relational schema, along with other configuration information used by OracleAS TopLink at runtime. Developers can use the OracleAS TopLink Mapping Workbench to define this metadata, and the OracleAS TopLink runtime component uses the metadata to provide the necessary persistence capabilities, using Java's reflective and introspective capabilities.

The OracleAS TopLink application metadata model is based around the OracleAS TopLink project. The project includes descriptors, mappings, and various policies that customize the runtime capabilities.

Figure 1–4 OracleAS TopLink Metadata



Sessions.xml File

Use the `sessions.xml` file to configure sessions for the project. Developers can build and edit these files with the OracleAS TopLink Sessions Editor. The session manager uses the `sessions.xml` configuration file during application initialization.

Project

The OracleAS TopLink deployment project is the primary container for the metadata. A project generally represents an application and contains the mapping information for all persistent classes and their relationships. Each session (excluding the session broker) in the deployed application references a single project. Although developers can build a project by coding it using the OracleAS TopLink API, we recommend that developers create and manage the project in the OracleAS TopLink Mapping Workbench, and use the OracleAS TopLink Mapping Workbench to generate either an XML or Java source version of the project for use at runtime.

Descriptor

A descriptor represents the association between a persistent Java class and a relational table(s). The descriptor contains configuration information for the class level within a project, as well as a set of mappings for each of its persistent attributes. Many of the more advanced configuration options are set at the descriptor level. The OracleAS TopLink Mapping Workbench supports most of these options, but there are a few that developers must set using the OracleAS TopLink API.

Mappings

Mappings describe how the attributes of a mapped class are associated with columns in the database. OracleAS TopLink provides a sophisticated set of flexible and customizable mappings that allow for complex mapping scenarios between the object and relational models.

There are two types of mappings: direct mappings, and relationship mappings.

Direct Mappings Direct mappings relate an attribute or attributes to a column or columns in the relational schema. OracleAS TopLink provides several direct mappings that allow for conversions between the types from the database and the object model's attribute types. Here are the direct mappings and their function:

- *Direct-to-field mappings* map a Java attribute directly to a value database column.
- *Type Conversion mappings* explicitly map a database type to a Java type.
- *Object type mappings* match a fixed number of database values to Java objects.
- *Serialized object mappings* store large data objects, such as multimedia files and BLOBs, in the database.

- *Transformation mappings* offer specialized translations between how a value is represented in Java and in the database, such as when developers map multiple fields into a single attribute.

Relationship Mappings OracleAS TopLink offers sophisticated relationship mapping, which enables developers to represent object relationships based on the database table columns and foreign keys. Here are the relationship mappings and their function:

- *One-to-one mappings* represent simple pointer references between two Java objects. The references use any of foreign keys, target foreign keys, or variable classes to define the pointer.
- *Aggregate object mappings* represent the relationship between a given object and a target object. The objects have a strict one-to-one relationship between the objects, and all the attributes of the second object are retrievable from the same table as the owning object.
- *Aggregate collection mappings* represent the relationship between a single-source object and a collection of target objects. Unlike one-to-many mappings, in which there must be a one-to-one back reference mapping from the target objects to the source object, there is no back reference required for the aggregate collection mappings, because the foreign key relationship is resolved by the aggregation (object & collection).
- *One-to-many mappings* represent the relationship between a single-source object and a collection of target objects.
- *Many-to-many mappings* represent the relationships between a collection of source objects and a collection of target objects. They require an intermediate table for managing the associations between the source and target records.
- *Object-relational mappings* are mappings that leverage databases that support object-relational entity storage within tables.

Application Development With OracleAS TopLink

Using OracleAS TopLink to build an application does not affect the choice of development tools or the creative process. However, OracleAS TopLink does influence how developers approach development. This section highlights some of the key areas in which using OracleAS TopLink affects application development. These areas exist, regardless of whether developers are building an application to support Java objects, EJB entity beans, or both.

Mapping

OracleAS TopLink maps the application's persistent entities to the database, using the descriptors and mappings developers build with the OracleAS TopLink Mapping Workbench. The OracleAS TopLink Mapping Workbench supports several approaches to project development, including:

- Importing classes and tables for mapping
- Importing classes and generating tables and mappings
- Importing tables and generating classes and mappings
- Creating both class and table definitions with mapping creation and model generation

The OracleAS TopLink Mapping Workbench supports all these options; however, the most common solution is to develop the persistent entities using a development tool, such as an integrated development environment (IDE) or modeling tool, and to develop the relational model through appropriate relational design tools. Developers then use the OracleAS TopLink Mapping Workbench to construct mappings that relate these two models.

The OracleAS TopLink Mapping Workbench does offer some facilities for generating persistent entities or the relational model components for an application; however, these utilities are intended only to assist in rapid initial development strategies, rather than complete round-trip application development.

For more information about mapping, see Chapter 3, "Mapping" and the *Oracle Application Server TopLink Mapping Workbench User's Guide*.

Session Management

Sessions are the primary interface between the application and OracleAS TopLink persistence capabilities. When developing an OracleAS TopLink application, developers must ensure that they properly initialize and manage the sessions.

When using EJB entity beans with container-managed persistence (CMP) or bean-managed persistence (BMP), the client code that modifies the entity beans does not access the OracleAS TopLink session directly. Instead, changes occur transparently, through integration with the container or through EJB callbacks.

Well-designed applications that employ Java objects as persistent entities use the session manager provided in the OracleAS TopLink API. This class initializes and manages the singleton session. Developers configure the session manager in the `sessions.xml` file, which allows for easy configuration and customization of the deployed application.

For more information about session management, see Chapter 4, "Sessions".

Querying

OracleAS TopLink provides several object and data query types, and offers flexible options for query selection criteria, including:

- OracleAS TopLink expressions
- EJB QL
- SQL
- Stored procedures
- Query by example

With these options, developers can build any type of query. We recommend that developers use predefined queries to define application queries. Predefined queries are held in the project metadata and referenced by name. This simplifies application development and encapsulates the queries to reduce maintenance costs.

The OracleAS TopLink Mapping Workbench provides the simplest way to define queries. Developers can also build queries in code, using the OracleAS TopLink API.

If the application includes EJB entity beans, developers can code finders completely using EJB QL, which enables the application to comply with the J2EE specification. Alternatively, developers can use any of the other OracleAS TopLink query options. All querying options are available, regardless of the architecture or persistent entity type.

For more information about querying, see Chapter 6, "Queries".

Transactions

In an OracleAS TopLink application, the Unit of Work ensures that OracleAS TopLink transactions comply with the transactional requirements of the application.

The Unit of Work is one of the most sophisticated and powerful components of the OracleAS TopLink Foundation Library. Although developers that use CMP or BMP entity beans do not use the OracleAS TopLink API to apply transactional changes to their persistent entities, the Unit of Work is used behind the scenes. Understanding how the Unit of Work behaves, and developing simple coding patterns to use it, are the keys to building efficient, maintainable applications.

For more information about transaction, see Chapter 7, "Transactions".

Packaging and Deployment

Application packaging (for deployment in the host Java or J2EE environment) influences OracleAS TopLink use and configuration. For example, developers package a J2EE enterprise application in an Enterprise Archive (EAR) file. Within the EAR file, there are several ways to package persistent entities within Web Application (WAR) and Java libraries (JAR). How developers configure OracleAS TopLink depends, in part, on how they package the application and how they use the host application server class loader.

For more information about packaging and deployment, see Chapter 9, "Packaging for Deployment".

Monitoring and Performance Tuning

OracleAS TopLink enables developers to monitor functionality and performance throughout application development, testing, and quality assurance cycles. OracleAS TopLink offers many textual logging, as well as the API required to implement custom logging strategies. Developers can use these features to ensure that the application behaves and performs as they expect.

OracleAS TopLink includes a performance profiler feature, available through the OracleAS TopLink Foundation Library API. This runtime feature tracks query execution time, which developers can use for performance analysis. This tool provides the information necessary to identify bottlenecks that hinder application performance.

OracleAS TopLink also offers a rich set of performance enhancement features. Understanding how to configure these features can have a strong influence on application performance, especially in the later phases of application development.

For more information about monitoring and performance tuning, see Chapter 10, "Tuning for Performance".

OracleAS TopLink Architectures Overview

OracleAS TopLink is designed to work in both Java and J2EE applications. Since it was first introduced, the flexibility OracleAS TopLink provides has led to its use in many architectural styles. This section introduces the five most common architectures associated with OracleAS TopLink. Although this section describes the architectures in relation to J2EE, OracleAS TopLink continues to fully support non-J2EE and Java applications as well.

For more information about OracleAS TopLink's flexible architecture support, see Chapter 2, "OracleAS TopLink Architectures".

Three-Tier

The three-tier (or J2EE Web) application is one of the most common OracleAS TopLink architectures. This architecture is characterized by a server-hosted environment in which the business logic, persistent entities, and the OracleAS TopLink Foundation Library all exist in a single Java virtual machine (JVM).

The most common example of this architecture is a simple three-tier application in which the client browser accesses the application through servlets, Java Server Pages (JSPs) and HTML. The presentation layer communicates with OracleAS TopLink through other Java classes in the same JVM, to provide the necessary persistence logic. This architecture supports multiple servers in a clustered environment, but there is no separation across JVMs from the presentation layer and the code that invokes the persistence logic against the persistent entities using OracleAS TopLink.

EJB Session Bean Facade

A popular variation on the three-tier application involves wrapping the business logic, including the OracleAS TopLink access, in EJB session beans. This architecture provides a scalable deployment and includes integration with transaction services from the host application server. Communication from the presentation layer occurs through calls to the EJB session beans. This architecture separates the application into different tiers for the deployment.

The session bean architecture can persist either Java objects or EJB entity beans.

EJB Entity Beans with CMP

OracleAS TopLink provides CMP support for applications that require the use of EJB entity beans. This support is available on the leading application servers. OracleAS TopLink CMP support provides the developer with an EJB 1.1 and 2.1 CMP solution transparent to the application code, but still offers all the OracleAS TopLink runtime benefits.

Applications can access OracleAS TopLink-enabled EJB entity beans using CMP directly from the client, or from within a session bean layer. OracleAS TopLink also offers the ability to use regular Java objects in relationships with EJB entity beans.

EJB Entity Beans with BMP

Another option for using EJB entity beans is to leverage OracleAS TopLink BMP in the application. This architecture enables developers to access the persistent data through the EJB API, but is platform independent.

The BMP approach is portable—that is, after a developer creates an application, you can move it from one application server platform to another.

Two-Tier

A two-tier (or client-server) application is one in which the OracleAS TopLink application accesses the database directly. Although less common than the others architectures discussed here, OracleAS TopLink supports this architecture for smaller or embedded data processing applications.

General Terms and Concepts

In addition to the OracleAS TopLink specific concepts, familiarity with several industry standard concepts helps you understand and implement OracleAS TopLink applications more effectively.

J2SE

The Java 2 Platform, Standard Edition (J2SE) is the core Java technology platform. It provides software compilers, tools, runtimes, and APIs for writing, deploying, and running applets and applications in Java.

J2EE

The Java 2 Platform, Enterprise Edition (J2EE) is an environment for developing and deploying enterprise applications. J2EE includes a set of services, APIs, and protocols for developing multi-tiered Web-based applications.

J2EE Containers

A J2EE container is a runtime environment for EJBs that includes such basic functions as security, life cycle management, transaction management, and deployment services. J2EE containers are usually provided by a J2EE server, such as Oracle Application Server Containers for J2EE.

Java Transaction API Support

The Java Transaction API (JTA) specifies the interfaces between a transaction manager, a resource manager, an application server, and transactional applications involved in a distributed transaction system.

Java Data Objects

Java Data Objects (JDO) represent a standard Java model for persistence that enables programmers to create code in Java that transparently accesses the underlying data store without using database-specific code. OracleAS TopLink provides support for most of the JDO specification, but, because OracleAS TopLink is a persistence framework, developers may find it easier and more effective to build your applications using OracleAS TopLink functionality rather than JDO.

2

OracleAS TopLink Architectures

This chapter presents an overview of five common enterprise architectures. Each architecture leverages Oracle Application Server TopLink to manage object persistence. The descriptions in this chapter include common usages for each of the architectures, as well as discussions about the technical challenges each architecture presents. Where appropriate, the sections refer to related technical information later in this document.

OracleAS TopLink supports any enterprise architecture that makes use of Java. This chapter focuses on OracleAS TopLink's flexible architecture support, which includes:

- Java application servers and J2EE containers
- Java-supporting databases, such as Oracle9*i* Database Server and IBM DB2 UDB
- Java-compatible browsers, such as Netscape and Internet Explorer
- Server Java platforms, such as AS/400, OS/390, and UNIX

OracleAS TopLink offers you the flexibility you need to choose your database, architecture, mapping strategy, application server and object-relational modeling. This chapter includes sections on:

- How to Use This Chapter
- Architectural Concepts
- Five Key Architectures
- Architecture Details

How to Use This Chapter

This chapter introduces common architectural designs that leverage OracleAS TopLink. This chapter is not intended to give you all the technical information required to build these architectures, but instead introduces the designs and helps you decide which architecture best suits your needs. Other chapters in this document offer details on how to implement the architectures introduced in this chapter, including:

- A typical example illustrating the use of the architecture
- A discussion of some of the technical challenges associated with the architecture
- References to other sections in this document that discuss these challenges in detail, and offer the necessary technical information to resolve them

Architectural Concepts

This section introduces concepts that help you evaluate the architectures presented in this chapter.

Persistent Entity Types

The architectures in this chapter fall into two categories, depending on whether you use Java objects or EJB entity beans to manage the persistent data.

Java Objects

OracleAS TopLink enables developers to use simple Java objects as the persistent mapped entities in your application. To manage them, developers use the OracleAS TopLink API or optionally, the Java Data Objects (JDO) API.

EJB Entity Beans

Enterprise JavaBean (EJB) technology is a component-based architecture that enables developers to develop distributed, object-oriented applications in Java. OracleAS TopLink offers support for EJB entity beans through both bean-managed persistence (BMP) and container-managed persistence (CMP).

Regardless of how you manage persistence, EJB applications require you to integrate the OracleAS TopLink framework with the hosting application server. This integration enables developers to leverage the connection pooling and transaction management offered through the application server's Java Transaction Architecture (JTA) support.

EJB Specification EJBs, developed by Sun Microsystems and its partners, represent a standard in enterprise computing. EJB is not a product, but rather a specification. It provides a framework for developers who create distributed business applications, and vendors who design application servers.

EJB is an important specification because of the widespread support it enjoys from enterprise software vendors.

For more information about EJBs, see the following Web sites:

```
http://java.sun.com/products/ejb/
http://java.sun.com/products/ejb/docs.html
http://java.sun.com/j2ee/white/index.html
```

Multi-Tier Enterprise Applications

An enterprise application integrates multiple heterogeneous systems, such as database servers, legacy applications, and mainframe applications. An enterprise application may also be required to support a diverse range of clients, including:

- Remote Method Invocation (RMI)
- Hypertext Markup Language (HTML)
- Extensible Markup Language (XML)
- Common Object Request Broker Architecture (CORBA)
- Distributed Component Object Model (DCOM)

The multi-tier approach enables developers to build complex enterprise applications that integrate with other systems in the application server tier. Many different types of enterprise architectures use the multi-tier approach.

Java and J2EE applications generally include several tiers, or layers. These layers can include the client tier, the presentation tier, the application tier, and the persistence tier.

Client Tier

An application client tier provides users with access to application functions. Its primary tasks are to present information from the application and to accept user input. For example, Web applications commonly present a browser as the client tier, but may also provide a Java (Swing) interface, a wireless device, or another application.

Presentation Tier

The presentation tier provides information interchange for the application. This tier is often a Java Server Pages (JSP) or servlet front end, an RMI or CORBA interface, or a Web Service.

Application Tier

The application tier holds the application business logic. Users access this tier either directly from the presentation layer using Java calls or through remote interfaces, such as RMI, CORBA, and EJB.

The application interacts with OracleAS TopLink at the application tier to provide application behavior. The user can query for and manipulate persistent entities through this tier.

Persistence Tier

The persistence tier provides access to the underlying datasource, after a relational database. In an OracleAS TopLink-enabled application, OracleAS TopLink provides most of the functionality for this tier. The application developer adds queries, mappings, and persistent entities to complete and enable the tier.

Session Components

The architectures presented in this chapter leverage the different OracleAS TopLink sessions and session components.

For more information about the session components, see Chapter 4, "Sessions".

Session Manager

The session manager is a singleton mechanism that manages the sessions within a given Java virtual machine (JVM). In most systems, the session manager retrieves the sessions from the `sessions.xml` file. This file contains the information required to instantiate sessions and their related mappings.

Server Session

The server session manages the persistence for a single OracleAS TopLink project, cached objects, query execution and maintaining shared JDBC resources. The session manager manages the server session.

The server session requires a client session to enable client access.

Client Session

The client session handles client interaction with the server. The server session manages the client session.

Project

The project contains mapping information for the persistence system. OracleAS TopLink stores the project in either a deployment XML format or a generated class. The OracleAS TopLink Mapping Workbench generates the project file in either of these formats.

Database Session

The database session is a singleton session used in a two-tier application instead of the Client-Server model used in the three-tier architectures. The main difference is that the database session manages a single JDBC connection (used for both reading and writing). This approach also assumes that there is only a single client involved and the cache is therefore not shared.

Database Login

The project contains default database login information, including a user name and password. You can also override this information by including alternative login information for a session, either in the `sessions.xml` file or in custom code.

Unit of Work

The Unit of Work, OracleAS TopLink's native transaction mechanism, provides several advantages over a standard database transaction. It is the most efficient mechanism to apply changes to the object model in all OracleAS TopLink usage patterns.

For more information about the Unit of Work, see Chapter 7, "Transactions".

Five Key Architectures

This section summarizes the five basic OracleAS TopLink architectures. These patterns are not mutually exclusive; instead, they are extensions of each other, based on the same core technology. This section introduces:

- Three-Tier
- EJB Session Bean Facade
- EJB Entity Beans Using CMP
- EJB Entity Beans Using BMP
- Two-Tier Architecture

Entity Bean Versus Non-Entity Bean Architectures

Two of the architectures presented in this chapter (EJB Entity Beans Using BMP, and EJB Entity Beans Using CMP) use EJB entity beans. EJB entity bean architectures are slightly different from the other architectures, because the EJB entity bean interfaces hide OracleAS TopLink functionality completely from the client application developer.

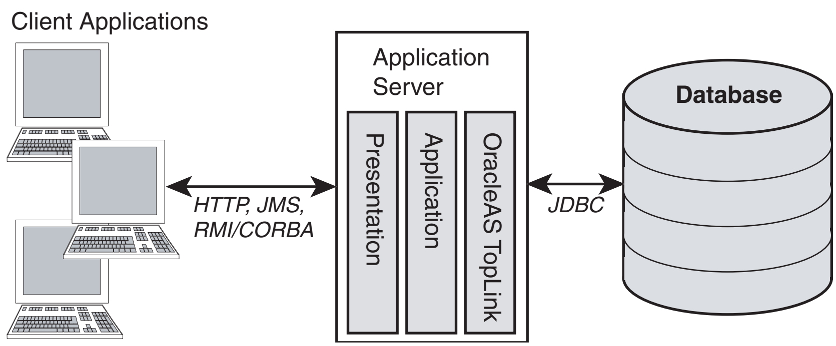
You can use entity beans in almost any J2EE application. From an OracleAS TopLink perspective, how the application uses the entity beans is not important; what is important to OracleAS TopLink is how each entity bean is mapped and implemented.

Three-Tier

The three-tier application is a common architecture in which OracleAS TopLink resides within a Java server (either a J2EE server or a custom server). In this architecture, the server session provides clients with shared access to JDBC connections and a shared object cache. Because it resides on a single JVM, this architecture is simple and easily scalable. The OracleAS TopLink persistent entities in this architecture are generally Java objects.

This architecture often supports Web-based applications in which the client application is a Web client, a Java client, or a server component.

Figure 2–1 Three-Tier Architecture



Not all three-tier applications are Web-based; however, the three-tier application is ideally suited to distributed Web applications. In addition, although it is also common to use EJBs in a Web application, this OracleAS TopLink architecture does not do so.

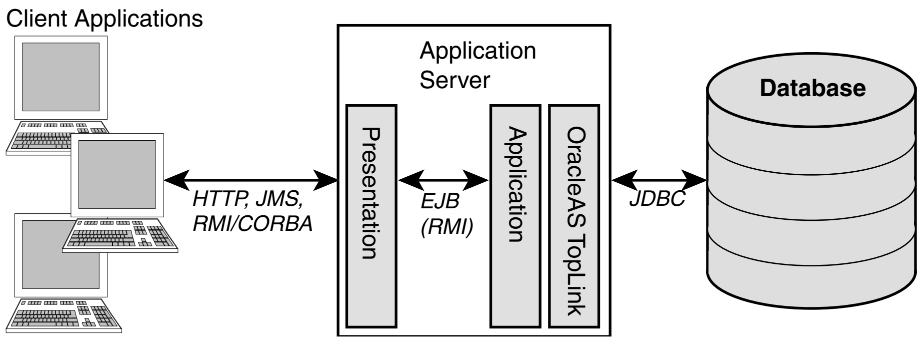
For more information, see "Three-Tier Architecture" on page 2-11.

EJB Session Bean Facade

This architecture is an extension of the three-tier pattern, with the addition of EJB Session Beans wrapping the access to the application tier. The EJB Session Beans provide public API access to application operations, enabling you to separate the presentation tier from the application tier. The architecture also enables you to leverage the EJB session beans within a J2EE container.

This type of architecture generally includes JTA integration, and serialization of data to the client.

Figure 2–2 Three-Tier Architecture Using Session Beans and Java Objects



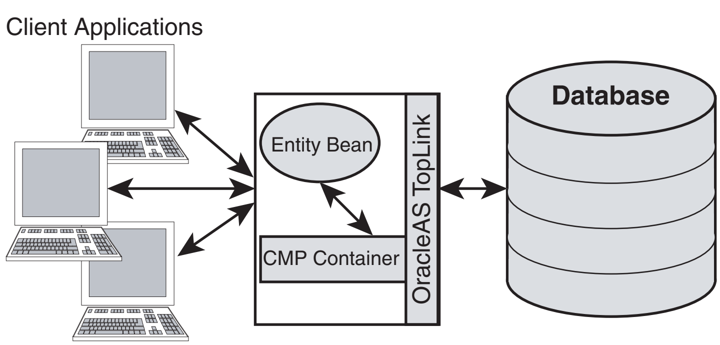
For more information, see "EJB Session Bean Facade Architecture" on page 2-13.

EJB Entity Beans Using CMP

OracleAS TopLink enables developers to leverage EJB entity beans within a J2EE application, using OracleAS TopLink CMP support. This support, which enables OracleAS TopLink to participate in container-managed transactions, requires a tight integration between the J2EE container and the persistence manager.

This architecture is an extension of the three-tier architecture, in which a J2EE container manages OracleAS TopLink mapping, querying, and other calls automatically.

Figure 2–3 Three-Tier Container-Managed Persistence Architecture



For more information, see "EJB Entity Beans with CMP Architecture" on page 2-15.

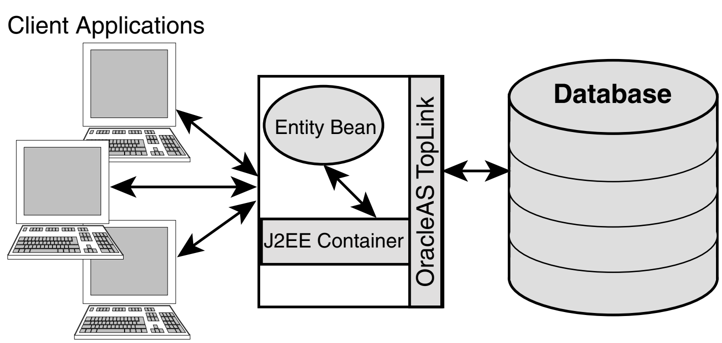
EJB Entity Beans Using BMP

OracleAS TopLink BMP support enables developers to use EJB Entity beans on all application servers that comply with J2EE. This architecture is an extension of the three-tier architecture, in which the persistent data is bean-managed within an entity bean. The client code accesses the data through the entity bean interface.

The BMP architecture enables developers to leverage a J2EE application server. The resulting application is portable—not tied to a particular J2EE application server. However, the BMP architecture is not common because:

- It offers functionality similar to a CMP solution, but BMP is not as transparent or efficient as CMP.
- OracleAS TopLink-only Java Object applications offer the same degree of independence from the application server.
- The developer must create the persistence mechanisms in the bean code.

Figure 2–4 Three-Tier Bean-Managed Persistence Architecture



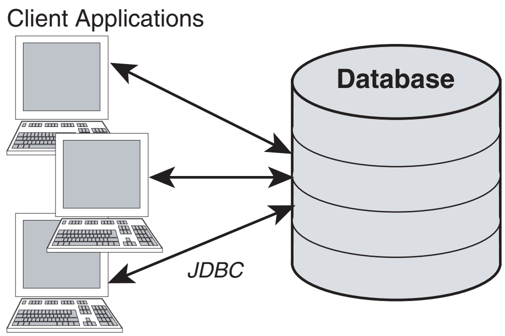
For more information, see "EJB Entity Beans with BMP Architecture" on page 2-17.

Two-Tier

A two-tier application generally includes a Java client that connects directly to the database through OracleAS TopLink. The two-tier architecture is most common in complex user interfaces with limited deployment. The database session provides OracleAS TopLink support for two-tier applications.

For more information, see "Database Session" on page 4-48.

Figure 2–5 Two-Tier Architecture



Although the two-tier architecture is the simplest OracleAS TopLink application pattern, it is also the most restrictive, because each client application requires its own session. As a result, two-tier applications do not scale as easily as other architectures.

For more information, see "Two-Tier Architecture" on page 2-18.

Architecture Details

This section offers a more in-depth look at the five architectures and provides information to help you choose the right design for your application. It includes sections that describe:

- Selecting an Architecture
- Three-Tier Architecture
- EJB Session Bean Facade Architecture
- EJB Entity Beans with CMP Architecture
- EJB Entity Beans with BMP Architecture
- Two-Tier Architecture

Selecting an Architecture

Table 2–1 lists common application feature requirements and indicates which architectures support each feature. Use this information to choose the most appropriate architecture for your application.

Table 2–1 Feature Support in the Five OracleAS TopLink Architectures

Feature	Three-tier Web Application	EJB Session Bean Facade	EJB Entity Bean with CMP	EJB Entity Bean with BMP	Client-Server Two-Tier
Persistent Entity: Java Objects	X	X	X	X	X
Persistent Entity: EJB Entity Beans		X	X	X	
JSP/Servlet Presentation layer	X	X	X	X	
J2EE Compliance	X	X	X	X	
JTA/JTS Transaction Management	X	X	X	X	
Scaling to multiple J2EE Application Server (clustering)	X	X	X	X	
Hosting Web Server and Application Server on Separate JVMs	X	X	X	X	X
Java 2 Standard Edition (J2SE) Application	X				X

Note: Application requires access to multiple datasources and therefore requires the JTA/JTS capabilities of the host application server to support two-phase commit.

About Non-Relational Datasources

The examples and discussions in this guide focus primarily on managing persistent entities on relational databases; however, OracleAS TopLink also offers access to non-relational data through the OracleAS TopLink Software Development Kit (SDK). For example, the OracleAS TopLink installation includes the ability to persist objects to and from XML data stream or file representation.

For more information about OracleAS TopLink with non-relational information, see "OracleAS TopLink XML Support" on page 5-56.

Three-Tier Architecture

The three-tier Web application architecture generally includes the connection of a server-side Java application to the database through a JDBC connection. In this common pattern, OracleAS TopLink resides within a Java server (a J2EE server or a custom server), with several possible server integration points. The application can support Web clients such as servlets, Java clients, and generic clients, using XML or CORBA.

Example Implementations

- A Model View Controller (MVC) Model 2 architectural design pattern that runs in a J2EE container with servlets and JSPs that uses OracleAS TopLink to access data, without EJBs.
- A Swing or AWT client that connects to a server-side Java application through RMI, without an application server or container.

Advantages and Disadvantages

The three-tier Web application architecture offers the following advantages:

- High performance, lightweight persistent objects
- High degree of flexibility in deployment platform and configuration

The disadvantage of this architecture is that it is a less standard approach than EJBs.

A Variation Using Remote Sessions

OracleAS TopLink includes a session type called `RemoteSession`. The remote session offers the full session API and contains a cache of its own, but exists on the client machine rather than the OracleAS TopLink server. Communications can be configured to use RMI or RMI-IIOP.

Remote session operations require a corresponding client session on the server.

Although this is an excellent option for developers who wish to simplify their access from the client tier to the server tier, it is less scalable than using a client session and does not easily allow changes to server-side behavior.

Technical Challenges

If you build the three-tier application with a stateless client, this architecture presents several technical challenges, the following sections discuss.

Managing Transactions in a Stateless Environment A common design practice is to delimit client requests within a single Unit of Work. In a stateless environment, this may affect how you design the presentation layer. For example, if a client requires multiple pages to collect information for a transaction, the presentation layer must retain the information from page to page until the application accumulates the full set of changes or requests. At that point, the presentation layer invokes the Unit of Work to modify the database.

Optimistic Locking in a Stateless Environment In a stateless environment, take extra care to avoid processing out-of-date (*stale*) data. A common strategy for avoiding stale data is to implement optimistic locking and store the optimistic lock values in the object.

This solution requires careful implementation if the stateless application serializes the objects or sends the contents of the object to the client in an alternative format. If this is the case, transport the optimistic lock values to the client in the HTTP contents of an edit page. Developers must then use the returned values in any Write transaction to ensure that the data did not change while the client was performing its work.

For more information about locking, see "Locking Policy" on page 5-20.

EJB Session Bean Facade Architecture

A common extension to the three-tier architecture is to combine session beans and OracleAS TopLink-managed persistent Java objects. The resulting application includes session beans and Java objects on an OracleAS TopLink three-tier architecture.

The three-tier architecture creates a server session and shares it between the session beans in the application. When a session bean needs to access an OracleAS TopLink session, the bean obtains a client session from the shared server session.

Here are the key features in this solution:

- Session beans delimit transactions; developers must configure OracleAS TopLink to work with a JTA system and its associated connection pool.
- Accessing the persistent objects on the client side causes them to be serialized; ensure that when they re-emerge on the server-side, they properly merge into the cache to maintain identity.

Example Implementation

An example of the EJB session bean facade architecture implementation is a Model View Controller (MVC) Model 2 architectural design pattern that runs in a J2EE container with servlets and JSPs that uses the OracleAS TopLink-enabled session bean to access data, without EJBs.

Advantages and Disadvantages

The EJB session bean facade architecture is a popular and an effective compromise between the performance of persistent Java objects and the benefits of EJBs for standardized client development and server scalability. It offers several advantages:

- *Less overhead than an EJB entity bean application*: OracleAS TopLink shares access to the project, descriptor, and login information across the beans in the application.
- *Future compatibility with other servers*: This design isolates login and EJB server-specific information from the beans, which enables you to migrate the application from one application server to another without major recoding or rebuilding.
- *Shared read cache*: This design offers increased efficiency by providing a shared cache for reading objects.

The key disadvantage of this model is the need to transport the persistent model to the client. If the model involves complex object graphs in conjunction with indirection, this can present many of challenges with relationships, inheritance, and indirection.

For more information about managing relationships, inheritance, and indirection, see Chapter 3, "Mapping".

Understanding Session Beans

Session beans model a process, operation, or service and as such, are not persistent entities. However, session beans can use persistence mechanisms to perform the services they model.

Under the session bean model, a client application invokes methods on a session bean that, in turn, performs operations on OracleAS TopLink-enabled Java objects. Session beans execute all OracleAS TopLink-related operations on behalf of the client.

The EJB specification describes session beans as either stateless or stateful.

- Stateful beans maintain a conversational state with a client; that is, they retain information between method calls issued by a particular client. This enables the client to use multiple method calls to manipulate persistent objects.
- Stateless beans do not retain data between method calls. When the client interacts with stateless session beans, it must complete any object manipulations within a single method-call.

Technical Challenges

An application can use both stateful and stateless session beans with an OracleAS TopLink client session or database session. When you use session beans with an OracleAS TopLink session, the type of bean used affects how it interacts with the session.

Stateless Session Beans and the OracleAS TopLink Session Stateless beans store no information between method calls from the client. As a result, re-establish the bean's connection to the session for each client method call. Each method call through OracleAS TopLink obtains a session, makes the appropriate calls, and releases the reference to the session.

Stateful Session Beans and the OracleAS TopLink Session Your EJB Server configuration includes settings that affect the way it manages beans—settings designed to increase performance, limit memory footprint, or set a maximum number of beans. When you use stateful beans, the server may deactivate an OracleAS TopLink-enabled stateful session bean out of the JVM memory space between calls to satisfy one of these settings. The server then reactivates the bean when required and brings it back into memory.

This behavior is important, because an OracleAS TopLink session instance does not survive passivation. To maintain the session between method calls, release it during the passivation process and re-obtain it when you reactivate the bean.

Unit of Work Merge

You can use a Unit of Work to enable your client application to modify objects on the database. The Unit of Work merge functions employ mappings to copy the values from the serialized object into the Unit of Work and to calculate changes.

For more information, see "Merging Changes in Working Copy Clones" on page 7-37.

EJB Entity Beans with CMP Architecture

OracleAS TopLink CMP support enables you to leverage a J2EE container to automate mapping, querying, and other OracleAS TopLink calls. In doing so, developers combine the standard interfaces and power of CMP and a container, with OracleAS TopLink flexibility, performance and productivity. OracleAS TopLink integrates with the EJB container in this architecture, to become the container's persistence manager.

OracleAS TopLink components are transparent to the developer in CMP architectures. The developer interacts with CMP entity beans, and the container uses OracleAS TopLink internally.

Example Implementation

An example of the EJB entity beans with CMP implementation is a Model View Controller (MVC) Model 2 architectural design pattern that runs in a J2EE container, with servlets and JSPs that access either session beans or OracleAS TopLink-enhanced EJB 2.0-compliant CMP entity beans.

Advantages and Disadvantages

This three-tier application offers the following advantages:

- It allows for CMP beans with OracleAS TopLink features such as caching and mapping support. This enables the bean designer to leverage OracleAS TopLink's complex mapping functionality, such as storing bean data across more than one table, composite primary keys, and data conversion.
- The CMP Architecture presents a standard method to access data, which enables developers to create standardized reusable business objects.
- CMP is well-suited to create coarse-grained objects, which OracleAS TopLink relates to dependent lightweight regular Java objects.
- OracleAS TopLink provides for lazy initialization of referenced objects and beans.
- OracleAS TopLink provides functionality for transactional copies of beans, allowing concurrent access by several clients, rather than relying on individual serialization.
- OracleAS TopLink provides advanced query capabilities, as well as dynamic querying.
- OracleAS TopLink maintains bean and object identity.

The disadvantage of this architecture is that pure CMP entity bean architectures can impose a high overhead cost. This is especially true when a data model has a large number of fine-grained classes with complex relationships.

Technical Challenges

The key technical challenge in this architecture lies in integrating components into a cohesive system. For example, this architecture requires a specific OracleAS

TopLink integration with the application server or J2EE container. Other issues include:

- *External JDBC Pools*: By default, OracleAS TopLink manages its own connection pools. Developers can also configure OracleAS TopLink to use connection pooling offered by the host application server. This is useful for shared connection pools and is required for JTA/JTS integration.
- *JTA/JTS Integration*: JTA and JTS are standard Java components that enable sessions to participate in distributed transactions. Developers must configure OracleAS TopLink to use JTA/JTS to leverage session beans in the architecture.
- *Cache Synchronization*: You may choose to use multiple servers to scale your application. In that case, you may require OracleAS TopLink cache synchronization.

EJB Entity Beans with BMP Architecture

OracleAS TopLink BMP support enables you to combine the standard interfaces of BMP entity beans with OracleAS TopLink flexibility, performance, and productivity. OracleAS TopLink provides a base class for BMP entity beans, and the base class implements the required methods for the EJB specification. This greatly simplifies the work of the developer when implementing BMP entity beans.

Example Implementations

An example of the EJB entity beans with BMP implementation is a Model View Controller (MVC) Model 2 architectural design pattern that runs in a J2EE container, with servlets and JSPs that access session beans and OracleAS TopLink-enhanced EJB 2.0-compliant BMP entity beans.

Advantages and Disadvantages

Using BMP with an OracleAS TopLink three-tier architecture offers the following advantages:

- It simplifies the BMP method calls. These can be inherited from an abstract bean class, rather than being generated.
- OracleAS TopLink makes BMP easier to implement.
- It enables developers to implement database-independent code in the bean methods.

- The architecture supports features such as complex relationships, caching, object level and dynamic queries, and the Unit of Work.

Technical Challenges

The key technical challenge in this architecture lies in integrating components into a cohesive system. For example, this architecture requires a specific OracleAS TopLink integration with the application server or J2EE container. Other issues include:

- *External JDBC Pools*: By default, OracleAS TopLink manages its own connection pools. Developers can also configure OracleAS TopLink to use connection pooling offered by the host application server. This is useful for shared connection pools and is required for JTA/JTS integration.
- *JTA/JTS Integration*: JTA and JTS are standard Java components that enable sessions to participate in distributed transactions. Developers must configure OracleAS TopLink to use JTA/JTS to leverage session beans in the architecture.
- *Cache Synchronization*: You may choose to use multiple servers to scale your application. In that case, you may require OracleAS TopLink cache synchronization.

Two-Tier Architecture

Two-tier applications are often implemented as user interfaces that directly access the database. They can also be noninterface processing engines. In either case, the two-tier model is not as common as the three-tier model.

These are key elements of an efficient two-tier (client-server) architecture with OracleAS TopLink:

- Minimal dedicated connections from the client to the database
- An isolated object cache

Example Implementations

An example of a two-tier architecture implementation is a Java User Interface (Swing/AWT) and batch data processing.

Advantages and Disadvantages

The advantage of the two-tier design is its simplicity. The OracleAS TopLink database session that builds the two-tiered architecture provides all the OracleAS

TopLink features in a single session type. This makes the two-tier architecture simple to build and use.

The most important limitation of the two-tired architecture is that it is not scalable, because each client requires its own database session.

Technical Challenges

The current trend towards multi-tiered Web applications makes the two-tier architecture less common in-production systems, but no less viable. However, because there is no shared cache in a two-tier system, you risk encountering stale data if you run multiple instances of the application. This risk increases as the number of individual database sessions increase.

To minimize this problem, OracleAS TopLink offers support for several data locking strategies. These include pessimistic locking and several variations of optimistic locking.

For more information, see "Locking Policy" on page 5-20.

3

Mapping

Mapping enables you to relate objects in your application to data in a database. This chapter describes how you can build mappings for Oracle Application Server TopLink-based applications. It includes descriptions of:

- Introduction to Mapping Concepts
- Basic Mappings
- Inheritance
- Mapping EJB Entity Beans
- Descriptor Validation
- Advanced Mappings
- Customizing the Project
- Writing Mappings in Code

For more information about mappings, see also the *Oracle Application Server TopLink Mapping Workbench User's Guide*.

Introduction to Mapping Concepts

In an OracleAS TopLink application, you persist objects by storing, or *mapping*, information about them in a relational database. A mapping has three components:

- The object being mapped
- The *descriptor*, or object-to-database table translator
- The database table or tables in which you stored the object

Although OracleAS TopLink supports more complex mappings, most OracleAS TopLink classes map to a single database table that defines the type of information available in the class. Each object instantiated from a given class maps to a single row comprising the object's attributes, plus an identifier (the *primary key*) that uniquely identifies the object.

Figure 3–1 How Classes and Objects Map to a Database Table

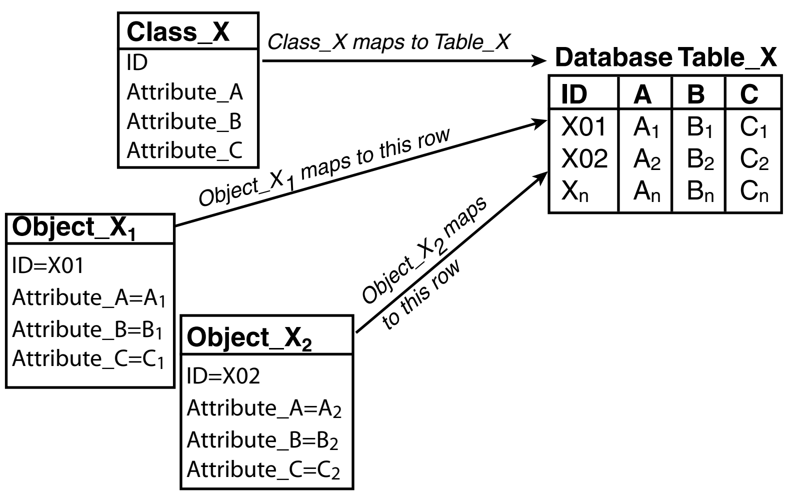


Figure 3–1 illustrates the simplest case in which:

- Table_X in the database represents Class_X.
- Object_X_1 and Object_X_2 are instances of Class_X.
- Individual rows in Table_X represent Object_X_1 and Object_X_2, as well as any other instances of Class_X.

OracleAS TopLink provides you with the tools to build these mappings, from the simple mappings illustrated in Figure 3–1, to complex mappings. OracleAS TopLink addresses the most difficult challenge for mapping—transforming a class or object into database table or row.

The following section describes the basic concepts that you must understand before moving on to the more in-depth information in this chapter, and introduces some of the more complex issues that are part of mapping.

Persistent Entities

Persistent entities are entities that survive, or *persist*, beyond the scope of a given transaction. A key feature of OracleAS TopLink is its ability to persist objects and entities in an application by mapping them to a database.

Metadata Model

OracleAS TopLink implements a *metadata* model, in which OracleAS TopLink uses metadata to define how objects and classes map to tables or rows, as well how tables and rows map to objects and classes. OracleAS TopLink uses the metadata, contained in the *descriptor*, to generate SQL statements that create, read, modify, and delete objects.

The OracleAS TopLink metadata model has three levels of information:

- *Mappings* describe how individual object attributes relate to the fields in a database row. Mappings relate object attributes to the database at the row level, and can involve a complex transformation, or a direct entry.

  For more information, see Primitive Versus Complex Data on page 3-7.

- *Descriptors* describe how a class relates to a database table. Class attributes map to database columns. Descriptors relate object classes to the database at the table level.
- *Projects* are collections of descriptors that make up an OracleAS TopLink application. Projects relate groups of object classes to the database at the schema level.

The metadata model describes the simplest case. There are more complex cases in which objects map to partial or multiple rows, and classes map to multiple tables, which are described later in this chapter. For the purposes of introducing mapping, this simple case forms the basis for understanding how mapping works.

OracleAS TopLink interaction with both object models and databases is unintrusive: OracleAS TopLink adapts to the object model and database schema, rather than requiring developers to design their object model or database schema to suit OracleAS TopLink.

OracleAS TopLink Mapping Workbench

The OracleAS TopLink Mapping Workbench is a graphical tool that gives you access to most OracleAS TopLink features. Although the OracleAS TopLink Mapping Workbench does not support the complete OracleAS TopLink feature set, it does support the basic functions required for mapping your application, as well as most of the advanced features.

The graphical nature of the OracleAS TopLink Mapping Workbench makes it easy to create models and mappings. As such, Oracle recommends that you build as much of your project in the OracleAS TopLink Mapping Workbench as possible.

An important feature of the OracleAS TopLink Mapping Workbench is its ability to generate deployment files from your project, either as deployment XML files or Java source code.

For more information about generating deployment files, see "Exporting Project Information" in the *Oracle Application Server TopLink Mapping Workbench User's Guide*.

Deployment XML Generation

The OracleAS TopLink Mapping Workbench can generate XML files from your project. OracleAS TopLink reads these files at runtime to configure your application. Deployment XML files reduce development time by eliminating the need to regenerate and recompile Java code each time the project changes.

Project Class Generation

The OracleAS TopLink Mapping Workbench can generate Java source files for your project that you compile and run for your application. Often, this generated code deploys faster than XML files, but is less flexible and more difficult to troubleshoot.

OracleAS TopLink Mapping Types

OracleAS TopLink offers several types of mapping, each optimized for different types of information.

Direct Mappings *Direct mappings* define how a persistent object refers to objects and attributes that do not have OracleAS TopLink descriptors, such as the JDK classes, primitive types, and other nonpersistent classes. Direct mappings map primitive data types to database data types on a one-to-one basis.

For more information about direct mappings, see "Direct Mappings" on page 3-8.

Relationship Mappings Relationship mappings describe how you manage relationships on the database. OracleAS TopLink uses several different mechanisms to represents relationships in the database, the most common of which is foreign keys. The OracleAS TopLink descriptors include details on the storage and retrieval mechanisms used for the relationship.

For more information about relationship mappings, see "Relationship Mappings" on page 3-14.

Inheritance

In object modeling, when one class (the superclass) shares its attributes with another class (the subclass), the subclass is said to inherit those attributes from the superclass or table. Similarly, in the database world, when one table shares information with a subordinate table in the database, the subordinate table inherits information from the main table. Although these two types of inheritance are similar, mapping them properly can be difficult.

OracleAS TopLink supports both object and database inheritance, and enables you to easily map object inheritance to database tables. OracleAS TopLink treats both types of inheritance interchangeably, provided that you map the inheritance in the class descriptors for the superclass and subclass.

For more information about inheritance, see "Inheritance" on page 3-47.

Objects and the Database

OracleAS TopLink stores objects in database tables. In most cases, a single row in a database table represents a single object in your OracleAS TopLink application. Several OracleAS TopLink concepts follow from this arrangement, including:

- Primary Keys
- Sequencing
- Foreign Keys and Object Relationships

Primary Keys

A primary key is a column or a combination of columns, in a database table that contains a unique identifier for every record in the table. Persistent objects require a primary key. If a table uses a combination of columns to create a unique identifier, this combination of fields is collectively called a composite primary key. In either case, a primary key uniquely identifies each row.

Sequencing

Sequencing is a mechanism to populate the primary key attribute of new objects and entity beans before inserting them into the database.

For more information, see "Sequencing" on page 3-37.

Foreign Keys and Object Relationships

Objects stored in one database table (the *source* objects) can share a relationship with objects in other tables (the *target* objects). To define these relationships, your tables must include data that identifies which target objects are related to the source object in the relationship.

The target table primary key in the relationship becomes a foreign key in the source table and identifies which objects in the target table are related to the objects in the source table.

For more information, see "Foreign Keys" on page 3-46.

Indirection

The standard object reading behavior in Java is that when you read an object, you also read all its related objects, which can be unnecessarily time consuming. The OracleAS TopLink indirection feature enables you to defer reading related objects until they are required. This is also known as lazy reading, lazy loading, and just-in-time reading.

For more information, see "Indirection" on page 3-27.

Serialization

In OracleAS TopLink, serialization is the act of writing out (*marshalling*) an object from its home OracleAS TopLink Java virtual machine (JVM) to another JVM.

For more information, see "Serialization" on page 3-34.

General Terms and Concepts

This section outlines some of the more common general concepts you will encounter when dealing with mappings.

Primitive Versus Complex Data

OracleAS TopLink treats certain classes as primitive data types for mapping purposes. These include Strings and Integers. Primitive data types correspond directly to representations in the database fields in which they are stored.

Because of this direct correspondence, there is no need to describe how to map the primitive data. As a result, OracleAS TopLink does not require mapping descriptors for primitive data types.

Object attributes represent complex data. OracleAS TopLink requires class descriptors to define how the attributes and relationships of instances of a particular class are stored and retrieved. Descriptors specify where and how attributes are stored in database tables.

Java Objects

Java objects represent the components or business logic of your application. As the basic building blocks in an OracleAS TopLink application, objects can include data, methods, relationships, and inheritance hierarchies.

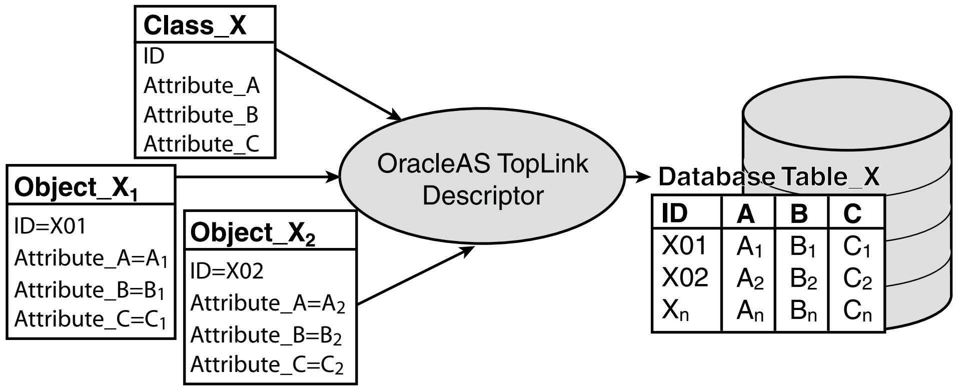
Basic Mappings

The OracleAS TopLink Mapping Workbench enables you to set properties and configure the mappings and OracleAS TopLink descriptors for any given project in a graphical environment. To create mappings, use either the OracleAS TopLink Mapping Workbench or the Java code-based API. However, Oracle recommends the OracleAS TopLink Mapping Workbench whenever possible.

Mappings for each class are stored in the class descriptor. OracleAS TopLink uses the descriptor to instantiate objects from the database and to store new or modified objects on the database. The descriptor describes how to store to or retrieve from the database a given class. Object instantiation uses this information to build and store the instantiated objects.

The relationship among the database, the objects and classes, and the descriptor makes up the OracleAS TopLink metadata model.

Figure 3–2 The OracleAS TopLink Metadata Model



For more information about the OracleAS TopLink Mapping Workbench, see the *Oracle Application Server TopLink Mapping Workbench User's Guide*.

This section presents several topics and techniques to optimize your mapping strategy, including:

- Direct Mappings
- Relationship Mappings
- Indirection
- Primary Keys
- Sequencing
- Foreign Keys
- Multiple Table Mappings
- Mapping and Enterprise JavaBeans

Direct Mappings

Use direct mapping to map primitive object attributes, or nonpersistent regular objects, such as the JDK classes. For example, use a direct-to-field mapping to store a `String` attribute in a VARCHAR field.

You can map entity bean attributes using direct mappings without any special considerations.

Note: When you work with EJBs, do not map the entity context attribute (type `javax.ejb.EntityContext`).

All direct mappings include optional `setGetMethodName()` and `setSetMethodName()` messages. These messages allow OracleAS TopLink to access the attribute through user-defined methods, rather than directly through the attribute.

Direct-to-Field Mappings

The direct-to-field mappings are instances of the `DirectToFieldMapping` class and require the following elements:

- The attribute being mapped, set by sending the `setAttributeName()` message
- The field to store the value of the attribute, set by sending the `setFieldName()` message

The `Descriptor` class provides the `addDirectMapping()` method that creates a new `DirectToFieldMapping`, sets the attribute and field name parameters, and registers the mapping with the descriptor.

You create a direct-to-field mapping in one of two ways:

- map one attribute to one field
- map more than one attribute to one field (to create different views of the same field)

Mapping an Attribute

Example 3–1 and Example 3–2 illustrate common ways of mapping one attribute to one field.

Example 3–1 Creating a Direct-to-Field Mapping in Java and Registering It with the Descriptor

```
// Create a new mapping and register it with the descriptor.
DirectToFieldMapping mapping = new DirectToFieldMapping();
mapping.setAttributeName("city");
mapping.setFieldName("CITY");
descriptor.addMapping(mapping);
```

Example 3–2 Creating a Mapping that Uses Method Access

This mapping example assumes that the persistent class has `getCity()` and `setCity()` methods defined.

```
// Create a new mapping and register it with the descriptor.
DirectToFieldMapping mapping = new DirectToFieldMapping();
mapping.setAttributeName("city");
mapping.setFieldName("CITY");
mapping.setGetMethodName("getCity");
mapping.setSetMethodName("setCity");
descriptor.addMapping(mapping);
```

Example 3–3 Using the Two Overloaded Versions of the Descriptor's addDirectMapping() Method

```
// Alternate method which does the same thing.
descriptor1.addDirectMapping("city", "CITY");
descriptor2.addDirectMapping("city", "getCity", "setCity", "CITY");
```

Mapping Multiple Attributes to the Same Field

You must pay special attention when you map more than one attribute to the same field where some mappings are read-only and some are not. By default, with `DatabaseLogin.setShouldOptimizeDataConversion(true)` — OracleAS TopLink uses the data type of the attribute of the last writable mapping for all subsequent read-only mappings. In this context, "last" is relative to the order in which the attributes are declared in the mapped class.

This behavior can lead to a loss of precision.

Consider the following hypothetical example: you want to map the class that appears in Example 3–4 to create two different views of the same, underlying database field. You want `attribute view1` to represent the database field as an integer and `attribute view2` to represent the same database field as a double. Finally, you want `attribute view1` to be writable and `attribute view2` to be read-only.

Example 3–4 ClassToMap Definition

```
public class ClassToMap
{
private String name;
private long id;
private int view1;
private double view2; // READONLY
```

```
...
}
```

Furthermore, your database administrator decides that both attributes will be mapped to a single database column, `NUM_VIEW` of table `CLASSTOMAP`, declared `NUMBER(20,7)` — that is, with a non-zero sub field size which allows storage of both integer values and floating point values with up to 7 digits of precision.

The corresponding `project.xml` database mapping elements appear in Example 3–5: the first maps `attribute view1` to table `CLASSTOMAP` field `NUM_VIEW` as writable and the second maps `attribute view2` to the same field as read-only.

Example 3–5 project.xml Database Mapping Elements

```
<database-mapping>
<attribute-name>view1</attribute-name>
<read-only>false</read-only>
<field-name>CLASSTOMAP.NUM_VIEW</field-name>
<type>oracle.toplink.mappings.DirectToFieldMapping</type>
</database-mapping>

<database-mapping>
<attribute-name>view2</attribute-name>
<read-only>true</read-only>
<field-name>CLASSTOMAP.NUM_VIEW</field-name>
<type>oracle.toplink.mappings.DirectToFieldMapping</type>
</database-mapping>
```

If the database is loaded with a record with `CLASSTOMAP.NUM_VIEW` value 3.141, you need to use `readAllObjects()` to get this instance of `ClassToMap` as shown in Example 3–6.

Example 3–6 Reading Objects of Type ClassToMap

```
Session sess = SessionManager.getManager().getSession("test");
Vector v = sess.readAllObjects(ClassToMap.class)
```

In the returned instance, the value of `view1` will be 3 and the value of `view2` will be 3.0 instead of 3.141. This loss of precision is the result of OracleAS TopLink's method of applying the data type of the last writable mapping which in this example is integer.

In this case, you can choose from either of the following options:

- Disable data conversion optimization with `DatabaseLogin.setShouldOptimizeDataConversion(false)`.
- Map the attribute with the highest precision as writable.

To change your design so that the `view1` is read-only and `view2` is writable, proceed as follows:

```
<database-mapping>
<attribute-name>view1</attribute-name>
<read-only>true</read-only>
<field-name>CLASSTOMAP.NUM_VIEW</field-name>
<type>oracle.toplink.mappings.DirectToFieldMapping</type>
</database-mapping>

<database-mapping>
<attribute-name>view2</attribute-name>
<read-only>false</read-only>
<field-name>CLASSTOMAP.NUM_VIEW</field-name>
<type>oracle.toplink.mappings.DirectToFieldMapping</type>
</database-mapping>
```

For more information about the available methods for `DirectToFieldMapping`, see the *Oracle Application Server TopLink API Reference*.

Type Conversion Mappings

Type conversion mappings and instances of the `TypeConversionMapping` class and require the following elements:

- The attribute mapped, set by sending the `setAttributeName( )` message
- The field to store the value of the attribute, set by the `setFieldName( )` message
- The Java type stored in the attribute, set by sending the `setAttributeClassification( )` message
- The database type to be written, set by sending the `setFieldClassification( )` message

Example 3–7 Creating a Type Conversion Mapping and Registering It with the Descriptor

```
// Create a new mapping and register it with the descriptor.
TypeConversionMapping typeConversion = new TypeConversionMapping();
typeConversion.setFieldName("J_DAY");
typeConversion.setAttributeName("joiningDate");
typeConversion.setFieldClassification(java.sql.Date.class);
typeConversion.setAttributeClassification(java.util.Date.class);
descriptor.addMapping(typeConversion);
```

For more information about the available methods for `TypeConversionMapping`, see the *Oracle Application Server TopLink API Reference*.

Object Type Mappings

Object type mappings are instances of the `ObjectTypeMapping` class and require the following elements:

- The attribute mapped, set by sending the `setAttributeName( )` message
- The field to store the value of the attribute, set by `setFieldName( )` message
- A set of values and their conversions, added by sending the `addConversionValue( )` message

The following methods are useful in a legacy environment or when you want to change the values of the fields:

- `addToAttributeOnlyConversionValue(Object fieldValue, Object attributeValue)`: This is a one-way mapping from the field to the attribute. Use this mapping if multiple database values map to the same object value. When written to the database, the value entered by `addConversionValue(Object fieldValue, Object attributeValue)` is used, and the original values in the database change.
- `setDefaultAttributeValue Object defaultAttributeValue)`: Substitutes the default value for any unmapped value retrieved from database. When writing to the database, the value entered by `addConversionValue(Object fieldValue, Object attributeValue)` is used, and the original values in the database change.

Example 3–8 Creating an Object Type Mapping and Registering It with the Descriptor

```
// Create a new mapping and register it with the descriptor.
ObjectTypeMapping typeMapping = new ObjectTypeMapping();
typeMapping.setAttributeName("gender");
typeMapping.setFieldName("GENDER");
typeMapping.addConversionValue("M", "Male");
typeMapping.addConversionValue("F", "Female");
typeMapping.setNullValue("F");
descriptor.addMapping(typeMapping);
```

For more information about the available methods for `ObjectTypeMapping`, see the *Oracle Application Server TopLink API Reference*.

Relationship Mappings

Relationship mappings define how persistent objects reference other persistent objects. OracleAS TopLink supports several relationship mapping types, as described in this section.

Relationships and Entity Beans

Persistent objects use *relationship mappings* to store references to instances of other persistent classes. The appropriate mapping type is selected based on the cardinality of the relationship (for example: a one-to-one or one-to-many). Entity beans can have relationships to regular Java objects, other entity beans, or both.

Mappings Between Entity Beans A bean that has a relationship to another bean acts as a client of that bean—it does not access the actual bean directly but acts through the remote (EJB 1.1) or local (EJB 2.0) interface of the bean. For example, if an `OrderBean` is related to a `CustomerBean`, it has an instance variable of type `Customer` (the `Local` or `Remote` interface of the `CustomerBean`) and accesses only those methods defined on the `Customer` interface.

> **Note:** Although beans must refer to each other through their remote (EJB 1.1) or local (EJB 2.0) interface, all OracleAS TopLink descriptors and projects refer to the bean class. For example, if you map beans and define relationships between them, you need load only the bean classes into the OracleAS TopLink Mapping Workbench—not the `Remote`, `Local`, or `Home` interfaces. When you define a relationship mapping in both the OracleAS TopLink Mapping Workbench and code API, the reference class is always the bean class.

Most OracleAS TopLink relationship mapping functionality is available regardless of the EJB specification supported by your J2EE container or application server. However, there are some differences between OracleAS TopLink support for EJB 1.1 and EJB 2.0.

Relationship Mappings Under EJB 1.1 The EJB 1.1 specification does not specify how entity beans store an object reference to another entity bean; as a result, if you are using an EJB 1.1-compliant container, this normally prevents you from mapping relationships between entity beans. However, OracleAS TopLink includes support for relationships that exceeds what is available in the EJB 1.1 specification, and allows the creation of inter-bean relationships.

Relationship Mappings Under EJB 2.0 The EJB 2.0 specification defines methods for relating beans to one another. OracleAS TopLink support for the EJB 2.0 specification includes the following concepts:

- The persistence layer manages bean relationships, and the relationships do not require any internal use of finder methods.
- You can define one-to-one, one-to-many, and many-to-many relationships between beans.
- You can use dependent objects (regular Java objects) to model fine-grained objects that are associated with a particular entity.

The EJB 2.0 specification also imposes many restrictions on CMP relationships, some of which are not enforced by OracleAS TopLink. Therefore, although OracleAS TopLink offers more flexibility in developing applications, if the application must be fully EJB 2.0-compliant, be careful about which features you include in your application.

Some of the EJB 2.0 restrictions that OracleAS TopLink does not enforce include:

- CMP beans must be abstract and have only virtual fields.
- Collections of entities used in relationship mappings must not be implemented by the bean developer, and must never be exposed directly to the client.
- Beans referenced by other beans must be related through `Local` interfaces.
- The EJB 2.0 specification does not support method access (such as get and set methods) for mappings.

The EJB 2.0 specification describes additional restrictions to the mapping and runtime behavior of EJB 2.0 CMP beans.

For more information about the Enterprise JavaBeans and the EJB 2.0 specification, see

```
http://java.sun.com/products/ejb/
http://java.sun.com/products/ejb/docs.html
http://java.sun.com/j2ee/white/index.html
```

In addition, although EJB 2.0 support for indirection is limited, OracleAS TopLink does enable you to implement OracleAS TopLink valueholder indirection for one-to-one relationships, and transparent indirection for one-to-many and many-to-many relationships.

For more information, see "Indirection" on page 3-27.

Importing EJB 2.0 Relationship Metadata in the OracleAS TopLink Mapping Workbench The OracleAS TopLink Mapping Workbench can obtain relationship metadata from the `ejb-jar.xml` file.

For more information on how to update OracleAS TopLink relationships in the OracleAS TopLink Mapping Workbench from the `ejb-jar.xml` deployment descriptor, see “Working with project properties” in the *Oracle Application Server TopLink Mapping Workbench User's Guide*.

Mappings Between Entity Beans and Java Objects Entity beans represent independent business objects. Objects that depend on the entity bean are often implemented as Java classes, and included as part of the entity bean on which they depend. The following relationship mappings may exist between an entity bean and regular Java objects:

- One-to-one, privately owned mappings (bean is source, Java object is target)
- One-to-many, privately owned mappings (bean is source, Java objects are target)
- Aggregate mappings (bean is source, Java object is target)
- Direct collection mappings (bean is source, Java object is target and is a base data type, such as String or Date)

Notes:

- Relationships from entity beans to regular Java objects must be dependent.
- If you expose dependent objects to the client, these objects must be serializable.

One-to-One Mappings

One-to-one mappings represent simple pointer references between two objects. One-to-one mappings for relationships between entity beans, or between an entity bean and a regular Java object, where the entity bean is the source and the regular Java object is the target of the relationship.

One-to-one mappings are instances of the `OneToOneMapping()` class and require the following elements:

- The attribute mapped, set by sending the `setAttributeName( )` message
- The reference class, set by sending the `setReferenceClass( )` message
- The foreign key information, normally specified by sending the `setForeignKeyFieldName( )` message and passing the foreign key field from the source table that references the primary key of the target table

Note: If the target primary key is composite, send the `addForeignKeyFieldName()` message for each of the foreign fields and target primary key fields that make up the relationship.

Bidirectional Relationships If the mapping has a bidirectional relationship in which the two classes in the relationship reference each other with one-to-one mappings, then set up the foreign key information as follows:

- One mapping must send the `setForeignKeyFieldName()` message.
- The other must send the `setTargetForeignKeyFieldName()` message.

It is also possible to set up composite foreign key information by sending the `addForeignKeyFieldName()` and `addTargetForeignKeyFieldName()` messages. Because OracleAS TopLink enables indirection by default, the attribute must be a `ValueHolderInterface`.

Caution: When your application does not use a cache, enable indirection for at least one object in a bidirectional relationship. In rare cases, disabling indirection on both objects in the bidirectional relationship can lead to infinite loops.

Example 3–9 Creating a Simple One-to-One Mapping and Registering It with the Descriptor

```
// Create a new mapping and register it with the descriptor.
OneToOneMapping oneToOneMapping = new OneToOneMapping();
oneToOneMapping.setAttributeName("address");
oneToOneMapping.setReferenceClass(Address.class);
oneToOneMapping.setForeignKeyFieldName("ADDRESS_ID");
descriptor.addMapping(oneToOneMapping);
```

Example 3–10 Implementing a Bidirectional Mapping Between Two Classes that Reference Each Other

The foreign key is stored in the Policy's table referencing the composite primary key of the Carrier.

```
// In the Policy class, which will hold the foreign key, create the mapping that
references the Carrier class.
OneToOneMapping carrierMapping = new OneToOneMapping();
carrierMapping.setAttributeName("carrier");
carrierMapping.setReferenceClass(Carrier.class);
carrierMapping.addForeignKeyFieldName("INSURED_ID", "CARRIER_ID");
carrierMapping.addForeignKeyFieldName("INSURED_TYPE", "TYPE");
descriptor.addMapping(carrierMapping);. . .
// In the Carrier class, create the mapping that references the Policy class.
OneToOneMapping policyMapping = new OneToOneMapping();
policyMapping.setAttributeName("masterPolicy");
policyMapping.setReferenceClass(Policy.class);
policyMapping.addTargetForeignKeyFieldName("INSURED_ID", "CARRIER_ID");
policyMapping.addTargetForeignKeyFieldName("INSURED_TYPE", "TYPE");
descriptor.addMapping(policyMapping);
```

For more information about the available methods for `OneToOneMapping`, see the *Oracle Application Server TopLink API Reference*.

For more information about one-to-one mappings, see the *Oracle Application Server TopLink Mapping Workbench User's Guide*.

One-to-One Mappings and EJBs To maintain EJB compliance, the object attribute that points to the target of the relationship must be the remote (EJB 1.1) or local (EJB 2.0) interface type—not the bean class.

OracleAS TopLink provides variations on one-to-one mappings that allow you to define complex relationships when the target of the relationship is a dependent Java object. For example, *variable one-to-one mappings* enable you to specify variable target objects in the relationship. These variations are not available for entity beans, but are valid for dependent Java objects.

For more information, see the "Variable One-to-One Mappings" on page 3-72.

Aggregate Object Mappings

Two objects are related by aggregation if there is a strict one-to-one relationship between the objects, and if all the attributes of the second object can be retrieved from the same table(s) as the owning object. So if the target (or child) object exists, then the source (or parent) object must also exist. The child object cannot exist without its parent.

Aggregate object mappings are instances of the `AggregateObjectMapping` class. This mapping relates to an attribute in each of the parent classes. Aggregate object mappings require the following information:

- The attribute mapped, set by sending the `setAttributeName( )` message
- The target (child) class, set by sending the `setReferenceClass( )` message

Aggregate object mappings also require the following modifications to the target class descriptor:

- Send the `descriptorIsAggregate( )` message to the descriptor to indicate that all information must come from the row(s) of its parent object's r
- Include no table or primary key information for the target class

By default, the mapping allows null references to its target class, so it does not create an instance of the target object. To prevent a parent from having a null reference, send the `dontAllowNull( )` message, which results in an instance of the child with its attributes set to null.

Example 3–11 Creating an Aggregate Object Mapping for the Employee Source Class and Registering It with the Descriptor

```
// Create a new mapping and register it with the source descriptor.
AggregateObjectMapping aggregateMapping = new AggregateObjectMapping();
aggregateMapping.setAttributeName("employPeriod");
```

```
aggregateMapping.setReferenceClass(Period.class);
descriptor.addMapping(aggregateMapping);
```

Example 3–12 Creating the Descriptor of the Period Aggregate Target Class

The aggregate target descriptor does not need a mapping to its parent, nor does it need any table or primary key information.

```
// Create a descriptor for the aggregate class. The table name and primary key
are not specified in the aggregate descriptor.
Descriptor descriptor = new Descriptor();
descriptor.setJavaClass(Period.class);
descriptor.descriptorIsAggregate();

// Define the attribute mappings or relationship mappings.
descriptor.addDirectMapping("startDate", "START_DATE");
descriptor.addDirectMapping("endDate", "END_DATE");
return descriptor;
```

Example 3–13 Creating an Aggregate Object Mapping for the Project, Which is Another Source Class that Contains a Period

The field names must be translated in the Project descriptor. No changes need to be made to the `Period` class descriptor to implement this second parent.

```
// Create a new mapping and register it with the parent descriptor.
AggregateObjectMapping aggregateMapping = new AggregateObjectMapping();
aggregateMapping.setAttributeName("projectPeriod");
aggregateMapping.setReferenceClass(Period.class);
aggregateMapping.addFieldNameTranslation("S_DATE", "START_DATE");
aggregateMapping.addFieldNameTranslation("E_DATE", "END_DATE");
descriptor.addMapping(aggregateMapping);
```

For more information about the available methods for `AggregateObjectMapping`, see the *Oracle Application Server TopLink API Reference*.

Aggregate Object Mappings and EJBs You can use aggregate mappings with entity beans when the source of the mapping is an entity bean and the target is a regular Java object. An entity bean cannot be the target of an aggregate object mapping.

Note: Aggregate objects are privately owned and must not be shared or referenced by other objects.

For more information about aggregate object mappings, see the *Oracle Application Server TopLink Mapping Workbench User's Guide*.

One-to-Many Mappings

One-to-many mappings represent the relationship between a single source object and a collection of target objects.

For more information about one-to-many mappings, see the *Oracle Application Server TopLink Mapping Workbench User's Guide*.

One-to-many mappings are instances of the `OneToManyMapping` class and require the following elements:

- The attribute being mapped, set by sending the `setAttributeName()` message
- The reference class, set by sending the `setReferenceClass()` message
- The foreign key information, which you specify by sending the `setTargetForeignKeyFieldName()` message and passing a field in the target object's associated table that refers to the primary key in the owning object's table

> **Note:** If the target primary key is composite, send the `addTargetForeignKeyFieldName()` message for each of the fields that make up the key.

- A one-to-one mapping in the target class back to the source class.

  For more information, see "One-to-One Mappings" on page 3-17.

> **Note:** Because indirection is enabled by default for a one-to-many mapping, the attribute must implement `ValueHolderInterface`.

Example 3–14 Creating a Simple One-to-Many Mapping and Registering It with the Descriptor

```
// In the Employee class, create the mapping that references the Phone class.
oneToManyMapping = new OneToManyMapping();
oneToManyMapping.setAttributeName("phoneNumbers");
oneToManyMapping.setReferenceClass(PhoneNumber.class);
oneToManyMapping.setTargetForeignKeyFieldName("EMPID");
descriptor.addMapping(oneToManyMapping);
```

```
. . .
// In the Phone class, which will hold the foreign key, create the mapping that
references the Employee class.
OneToOneMapping oneToOneMapping = new OneToOneMapping();
oneToOneMapping.setAttributeName("owner");
oneToOneMapping.setReferenceClass(Employee.class);
oneToOneMapping.setForeignKeyFieldName("EMPID");
descriptor.addMapping(oneToOneMapping);
```

In addition to the API Example 3–14 illustrates, other common API for use to implement indirection in aggregate collection include:

- `useBasicIndirection()`: implements OracleAS TopLink valueholder indirection
- `useTransparentCollection()`: if you use transparent indirection, this element places a special collection in the source object's attribute
- `dontUseIndirection()`: implements no indirection

For more information about the available methods for `OneToManyMapping`, see the *Oracle Application Server TopLink API Reference*.

One-to-Many Mappings and EJBs Use one-to-many mappings for relationships between entity beans or between an entity bean and a collection of privately owned regular Java objects. When you create one-to-many mappings, also create a one-to-one mapping from the target objects back to the source. The object attribute that contains a pointer to the bean must be the remote (EJB 1.1) or local (EJB 2.0) interface type—not the bean class.

OracleAS TopLink automatically maintains back-pointers when you create or update bidirectional relationships between beans.

For more information, see "Maintaining Bidirectional Relationships" on page 3-61.

Aggregate Collections

Aggregate collection mappings represent the aggregate relationship between a single-source object and a collection of target objects. Unlike the OracleAS TopLink one-to-many mappings, there is no back reference required for the aggregate collection mappings because the foreign key relationship is resolved by the aggregation.

Aggregate collection mappings require a target table for the target objects.

To implement an aggregate collection mapping:

- The descriptor of the target class must declare itself as an aggregate collection object. Unlike the aggregate object mapping, in which the target descriptor does not have a specific table to associate with, there must be a target table for the target object.
- The descriptor of the source class must add an aggregate collection mapping that specifies the target class.

When to Use Aggregate Collections Although similar in behavior to one-to-many mappings, an aggregate collection is not a replacement for one-to-many mappings. Use aggregate collections when the target collections are reasonable in size and a one-to-one mapping from the target to the source proves difficult.

Because one-to-many relationships offer better performance and are more robust and scalable, consider using a one-to-many relationship rather than an aggregate collection. In addition, aggregate collections are privately owned by the source of the relationship and must not be shared or referenced by other objects.

Aggregate Collections and Inheritance Aggregate collection descriptors can make use of inheritance, but you must declare the subclasses as aggregate collections as well. The subclasses can have their own mapped tables, or share the table with their parent class.

In a Java vector, the owner references its parts; in a relational database, the parts reference their owners. Relational databases use this implementation to make querying more efficient.

Java Implementation Aggregate collection mappings are instances of the `AggregateCollectionMapping` class and require the following elements:

- The attribute mapped, set by sending the `setAttributeName( )` message
- The reference class, set by sending the `setReferenceClass( )` message
- The foreign key information, specified by sending the `addTargetForeignKeyFieldName( )` message and passing the field name of the target foreign key and the source of the primary key in the source table

---

Notes:

- If the source primary key is composite, send the `addTargetForeignKeyFieldName( )` message to each of the fields that make up the key.
- Because indirection is enabled by default for an aggregate collection mapping, the attribute must implement `ValueHolderInterface`.

Example 3–15 Creating a Simple Aggregate Collection Mapping and Registering It with the Descriptor

```
// In the PolicyHolder class, create the mapping that references the Phone class
AggregateCollectionMapping phonesMapping = new AggregateCollectionMapping();
phonesMapping.setAttributeName("phones");
phonesMapping.setGetMethodName("getPhones");
phonesMapping.setSetMethodName("setPhones");
phonesMapping.setReferenceClass("Phone.class");
phonesMapping.dontUseIndirection();
phonesMapping.privateOwnedRelationship;
phonesMapping.addTargetForeignKeyFieldName("INS_PHONE.HOLDER_SSN","HOLDER.SSN");
descriptor.addMapping(phonesMapping);
```

In addition to the API Example 3–15 illustrates, other common API for use to implement indirection in aggregate collection mappings include:

- `useBasicIndirection()`: implements OracleAS TopLink valueholder indirection
- `useTransparentCollection()`: if you use transparent indirection, this element places a special collection in the source object's attribute
- `dontUseIndirection()`: implements no indirection

For more information about the available methods for `AggregateCollectionMapping`, see the *Oracle Application Server TopLink API Reference*.

Aggregate Collection Mappings and EJBs You can use aggregate collection mappings with entity beans if the source of the relationship is an entity bean or Java object, and the mapping targets are regular Java objects. Entity beans cannot be the target of an aggregate object mapping.

Direct Collection Mappings

Direct collection mappings store collections of Java objects that are not OracleAS TopLink-enabled. Direct collections usually store Java types, such as `strings`.

Direct collection mappings are instances of the `DirectCollectionMapping` class and require the following elements:

- The attribute mapped, set by sending the `setAttributeName( )` message
- The database table that holds the values to be stored in the collection, set by sending the `setReferenceTableName( )` message
- The field in the reference table from which the values are read and placed into the collection; this is called the direct field and is set by sending the `setDirectFieldName( )` message
- The foreign key information, which you specify by sending the `setReferenceKeyFieldName( )` message and passing the name of the field that is a foreign reference to the primary key of the source object

> **Note:** If the target primary key is composite, send the `addReferenceKeyFieldName()` message for each of the fields that make up the key.

Example 3–16 Creating a Simple Direct Collection Mapping

```
DirectCollectionMapping directCollectionMapping = new DirectCollectionMapping();
directCollectionMapping.setAttributeName ("responsibilitiesList");
directCollectionMapping.setReferenceTableName ("RESPONS");
directCollectionMapping.setDirectFieldName("DESCRIP");
directCollectionMapping.setReferenceKeyFieldName ("EMP_ID");
directCollectionMapping.useCollectionClass (Vector.class); // the default
descriptor.addMapping(directCollectionMapping);
```

In addition to the API Example 3–16 illustrates, other common API for use with direct collection mappings include:

- `useBasicIndirection()`: implements OracleAS TopLink valueholder indirection
- `useTransparentCollection()`: if you use transparent indirection, this element places a special collection in the source object's attribute
- `dontUseIndirection()`: implements no indirection

For more information about the available methods for `DirectCollectionMapping`, see the *Oracle Application Server TopLink API Reference*.

Many-to-Many Mappings

Many-to-many mappings represent the relationships between a collection of source objects and a collection of target objects. This requires an intermediate table that manages the associations between the source and target records.

Many-to-many mappings are instances of the `ManyToManyMapping` class and requires the following elements:

- The attribute mapped, set by sending the `setAttributeName( )` message
- The reference class, set by sending the `setReferenceClass( )` message
- The relation table, set by sending the `setRelationTableName( )` message
- The foreign key information (for non composite target primary keys), which you specify by sending the `setSourceRelationKeyFieldName( )` and `setTargetRelationKeyFieldName( )` messages
- The foreign key information if the source or target primary keys are composite, which you specify by sending the `addSourceRelationKeyFieldName()` or `addTargetRelationKeyFieldName()` messages

Example 3–17 Code that Creates a Simple Many-to-Many Mapping

```
// In the Employee class, create the mapping that references the Project class.
ManyToManyMapping manyToManyMapping = new ManyToManyMapping();
manyToManyMapping.setAttributeName("projects");
manyToManyMapping.setReferenceClass(Project.class);
manyToManyMapping.setRelationTableName("PROJ_EMP");
manyToManyMapping.setSourceRelationKeyFieldName ("EMPID");
manyToManyMapping.setTargetRelationKeyFieldName ("PROJID");
descriptor.addMapping(manyToManyMapping);
```

In addition to the API Example 3–17 illustrates, other common API for use with many-to-many mappings include:

- `useBasicIndirection()`: implements OracleAS TopLink valueholder indirection
- `useTransparentCollection()`: if you use transparent indirection, this element places a special collection in the source object's attribute

- `dontUseIndirection()`: implements no indirection

For more information about the available methods for `ManyToManyMapping`, see the *Oracle Application Server TopLink API Reference*.

Many-to-Many Mappings and EJBs When you use CMP, many-to-many mappings are valid only between entity beans, and cannot be privately owned. The only exception is when a many-to-many mapping is used to implement a logical one-to-many mapping with a relation table.

OracleAS TopLink automatically maintains back-pointers when you create or update bidirectional relationships.

For more information, see "Maintaining Bidirectional Relationships" on page 3-61.

For more information about `ManyToManyMapping`, see the *Oracle Application Server TopLink Mapping Workbench User's Guide*.

Indirection

By default, when an OracleAS TopLink application reads an object, it also reads all its related objects. For example, given an object, CAR, with related objects, TIRES and RADIO, reading the CAR object forces reading of the TIRES and RADIO objects at the same time. This method is inefficient if the reason for reading in the CAR object has nothing to do with the related objects (for example: when you read CAR to check one of its attributes, such as COLOR).

OracleAS TopLink indirection gives you the ability to replace the related objects (TIRES and RADIO, in this example) with an indirection object. An indirection object is a placeholder that represents related objects, but prevents them from being read until they are actually required. If you never need the related objects, they are never read from the database.

OracleAS TopLink supports three main types of indirection:

- *Valueholder indirection*: places a special OracleAS TopLink object with an interface between the pair of related objects.
- *Proxy indirection*: uses a dynamically constructed object with the same interface as the class of the object referenced in the relationship.
- *Transparent indirection*: a special OracleAS TopLink collection that prevents instantiation of the objects it contains until they are called. The collections conform to `Vector`, `Hashtable`, or `Collection` interfaces.

Indirection represents an effective way to improve the efficiency of your application and we recommend it be implemented wherever it is supported by your application and its usage patterns.

For more information about implementing indirection in code, see "Implementing Indirection in Java" on page 3-100.

Valueholder Indirection

Valueholder indirection is a native OracleAS TopLink feature that implements the OracleAS TopLink `ValueHolderInterface` on your objects to achieve indirection. A valueholder represents an instance of a related class and stores the information necessary to retrieve the object it represents from the database. If the application does not access the valueholder, the replaced object is never read from the database.

If you use method access, the `get` and `set` methods specified for the mapping must access an instance of `ValueHolderInterface`, rather than the object that the valueholder is referencing. To obtain the object represented by the valueholder, use the `getValue()` and `setValue()` methods of the `ValueHolderInterface` class. You can hide the `getValue` and `setValue` methods of the `ValueHolderInterface` inside `get` and `set` methods.

You can change the attribute types in the class editor, but if you do, also change the attribute types in your Java code, as well as their accessor methods.

If the instance variable returns a vector instead of an object, define the valueholder in the constructor as follows:

```
addresses = new ValueHolder(new Vector());
```

The application uses the `getAddress()` and `setAddress()` methods to access the `Address` object. When you use indirection, OracleAS TopLink uses the `getAddressHolder()` and `setAddressHolder()` methods to save instances to and retrieve instances from the database.

Example 3–18   Implementing the Employee Class Using Indirection with Method Access for a One-to-One Mapping to Address

This example modifies the class definition so that the `address` attribute of `Employee` is a `ValueHolderInterface`, rather than an `Address`, and supplies the appropriate `get` and `set` methods.

```
// Initialize ValueHolders in Employee Constructor
public Employee() {
address = new ValueHolder();
}
```

```
protected ValueHolderInterface address;

// 'Get' and 'Set' accessor methods registered with the mapping and used by
OracleAS TopLink.
public ValueHolderInterface getAddressHolder() {
return address;
}
public void setAddressHolder(ValueHolderInterface holder) {
address = holder;
}

// 'Get' and 'Set' accessor methods used by the application to access the
attribute.
public Address getAddress() {
return (Address) address.getValue();
}
public void setAddress(Address theAddress) {
address.setValue(theAddress);
}
```

Proxy Indirection

Proxy indirection enables you to use dynamic proxy objects as stand-ins for a defined interface. You can configure all the following mapping types to use proxy indirection, which gives you the benefits of indirection without the need to include OracleAS TopLink classes in your domain model:

- One-to-one mapping
- Variable one-to-one mapping
- Reference mapping
- Transformation mapping

Note that all these mapping types map one-to-one relationships.

The `useProxyIndirection()`method indicates that OracleAS TopLink must use proxy indirection for the current mapping. When you read the source object from the database, OracleAS TopLink creates a proxy for the target object and uses it in place of the target object. When you call any method other than `toString()` on the proxy, OracleAS TopLink reads the target object from the database.

Proxy indirection is not directly supported in the OracleAS TopLink Mapping Workbench. To implement proxy indirection, use the `useProxyIndirection` method in an amendment method.

Proxy indirection does not use the OracleAS TopLink `ValueholderInterface`, and nor are target objects typed as `ValueHolderInterface`. Instead, to implement proxy indirection, make changes to both the object model and the descriptor mapping for the source object.

To use proxy indirection, your domain model must satisfy the following criteria:

- The target class of the one-to-one relationship must implement a defined public interface.
- The one-to-one attribute on the source class must be of the interface type defined in the target class.
- If you employ method accessing, the `get()` and `set()` methods must use the interface.

In the descriptor, invoke the `useProxyIndirection` method in the source object descriptor that defines mapping between the source and target objects.

Example 3–19 Implementing Proxy Indirection on the Source Descriptor

The `Employee` class has an attribute, `ADDRESS`, of type Address. The `Address` attribute is mapped using a one-to-one mapping from `Employee` (source) to `Address` (target) and uses proxy indirection. The code includes the steps for building this relationship.

```
//Step 1. Define an interface "IAddress" for "Address"
public interface IAddress {
    public String getCity();
    public void setCity(String aCity);
}

// Step 2. Implement this interface on the "Address" class
public class Address implements IAddress {
    String city;
    public String getCity() { return city;}
    public void setCity(String aCity){city = aCity;}

    public Address() {
    ...
}

}
```

```
//Step 3. Declare the attribute "address" as interface "IAddress" on the
Employee.

public class Employee {
    public BigInteger id;
    public String firstName;
    public String lastName;
    public IAddress address;

//4. Configure the Set and get methods "getAddress()", "setAddress()" to use
interface IAddress"

    // get and set methods for instance variables
    public IAddress getAddress() {return this.address;}
    public void setAddress (IAddress newAddress)
    {this.address=newAddress;}


    public Employee() {
    ...
    }
}

//5. The mapping between Employee and Address must invoke the
useProxyIndirection() api. Also, the target class, which implements the
interface must be passed to the setReferenceClass() method as the argument.

//Define the 1:1 mapping, and specify that ProxyIndirection should be used
OnetoOnemapping addressMapping = new OneToOneMapping();
addressMapping.setAttributeName("address");
addressMapping.setReferenceClass(Address.class);
addressMapping.setForeignKeyFieldName("ADDRESS_ID");
addressMapping.setSetMethodName("setAddress");
addressMapping.setGetMethodName("getAddress");
addressMapping.useProxyIndirection();
descriptor.addMapping(addressMapping);
```

Proxy Indirection Restrictions You cannot register the target of a proxy indirection implementation with a Unit of Work. Instead, first register the source object with the Unit of Work. This enables you to retrieve a target object clone with a `get...()` call against the source objects clone.

For example:

```
UnitOfWork uow = session.acquireUnitOfWork();
Employee emp = (Employee)session.readObject(Employee.class);

// Register the source object
Employee empClone = (Employee)uow.registerObject(emp);

// All of source object's relationships are cloned when source object is cloned
Address addressClone = empClone.getAddress();
addressClone.setCity("Toronto");
```

For more information about clones and the Unit of Work, see "Understanding the Unit of Work" on page 7-5.

Transparent Indirection

Transparent indirection enables you to declare any relationship attribute of a persistent class that holds a collection of related objects as a `java.util.Collection`, `java.util.Map`, `java.util.Vector`, or `java.util.Hashtable`. OracleAS TopLink uses an indirection object that implements the appropriate interface, and performs just-in-time reading of the related objects.

When using transparent indirection, you do not have to declare the attributes as `ValueHolderInterface`.

You can specify transparent indirection from the OracleAS TopLink Mapping Workbench. Newly created collection mappings use transparent indirection by default if their attribute is not a `ValueHolderInterface`.

Do not include OracleAS TopLink classes in the domain class for transparent indirection.

Choosing Your Indirection Type

Although there are no universal rules for the use of indirection, the following guidelines illustrate when indirection is beneficial, and helps you choose the appropriate type of indirection.

Choosing No Indirection Because it delays database reads until they are required, indirection produces an increase in performance. However, if you have a relationship between objects that are always called together, the benefit does not apply. For example, if you have a pair of objects that are always called together to populate a Web page, there is no benefit to delay the reading of the target of the relationship, because it will be called at the same time as the source object every time. If you have objects that you always call together, do not implement indirection.

Choosing Valueholder Indirection Use valueholder indirection if at least one of the following conditions exists:

- Your application can tolerate the addition of OracleAS TopLink classes to your model.
- The relationship to which you are applying indirection involves EJB 2.0 entity beans.

Choosing Proxy Indirection Use proxy indirection if you are not applying indirection with EJB entity beans as targets.

Choosing Transparent Indirection When you create one-to-many or many-to-many relationships in the OracleAS TopLink Mapping Workbench, OracleAS TopLink automatically implements transparent indirection. This provides the best possible performance for large relationship graphs and must not be disabled.

Indirection and EJBs

OracleAS TopLink offers mechanisms to implement indirection for relationships between EJBs. As with regular Java objects, these mechanisms include:

- The use of indirection objects
- Transparent indirection
- Proxy indirection

The *Oracle Application Server TopLink Mapping Workbench User's Guide* describes these indirection mechanisms.

Consider the following guidelines when you use indirection with EJBs, particularly when you migrate objects between client and server:

- Uninstantiated valueholders (indirection objects) do not survive serialization. If you send a valueholder from the server to the client, it will no longer function unless it has been previously triggered.
- You can use valueholders in bean-to-bean relationships and bean-to-object relationships, but avoid them in relationships in which the source is likely to be serialized to the client.

- Do not serialize collections that use untriggered transparent indirection to the client application. These collections do not function if they are serialized.
- Proxy indirection is unavailable for relationships whose target is an entity bean. The proxies used for this kind of indirection interfere with the RMI stubs and skeletons generated for the entity. If proxies exist, instantiate them before serializing to the client.
- Use valueholders for bean-to-bean relationships and for bean-to-object relationships. You can also use transparent indirection for collections that are not exposed to the client application.

EJB 2.0 and Indirection When both the source and target are entity beans, the indirection policies for container-managed relationship fields under the EJB 2.0 specification must be one of the following:

- Transparent indirection for one-to-many or many-to-many relationships
- Valueholder indirection for one-to-one relationships

Because subclasses are code-generated, all indirection is hidden from the user.

Serialization

OracleAS TopLink supports Java serialization, which enables you to write objects out to one JVM and read objects back from the other JVM. Preparing the objects for transport is known as *marshalling*; receiving objects back is known as *unmarshalling*. In an OracleAS TopLink application, serialization occurs between a JVM with OracleAS TopLink and a non-OracleAS TopLink JVM. If you serialize to another JVM with OracleAS TopLink, consider using a remote session instead.

For more information, see "Remote Session" on page 4-58.

Serialization and Indirection

A common cause of problems with serialization is the use of indirection in serialized objects. Indirection valueholders rely on the OracleAS TopLink session for context (mapping information, JDBC connectivity, and so on); however, because the OracleAS TopLink session is stored in the object in a transient variable, it does not survive serialization, leaving the serialized valueholders with no context and no way to resolve the links to the data they represent. As a result, when you marshall the object for serialization, the values held by valueholders are replaced with a null value. If the application on the receiving JVM invokes the valueholder, the result is a null pointer exception.

Note that no null pointer exception is thrown during the serialization process, nor does OracleAS TopLink prevent you from serializing an untriggered valueholder. This enables you to serialize objects and retain the efficiency advantages of indirection if you know that the receiving JVM does not use the valueholders.

Triggering Valueholders During Marshalling A common way to avoid null pointer exceptions in the receiving JVM is to selectively trigger valueholders before serializing them.

Java serialization supports a callback mechanism that enables you to execute a special type of method on an object before serializing it. Specifically, if a `writeObject` method exists on the object Java serialization executes the method.

For example:

```
protected void writeObject(ObjectOutputStream out) throws IOException
```

This mechanism presents an opportunity to selectively trigger valueholders. You can:

- Add triggering methods directly on the object.
- Build helper classes to trigger valueholders, and use a method on the serializable object to call the helper classes.

Helper classes are the most flexible way to trigger valueholders, because you can use a single helper class for several objects. When deciding how to trigger valueholders, we recommend the following methods:

- *Trigger no Valueholders*: This method requires no extra work; however, it assumes that the application that receives the serializable object does not call any valueholders, including those in the serialized object.
- *Trigger a set of Valueholders specific to the purpose of the receiving application*: This method requires the OracleAS TopLink developer to know exactly what the receiving application does with the serialized object and to manually trigger all required valueholders.
- *Trigger a set of Valueholders to make the serialized object generically useful*: For example, you can choose to trigger all valueholders in the object itself, but none in the related objects. This method does not require that the OracleAS TopLink developer know what the receiving application does with the serialized objects, but imposes a predictable limit on the receiving application.
- *Trigger all Valueholders, traversing all relationships to the leaf class*: This method makes the object completely fail-safe to the receiving application, but imposes

the potentially resource-intensive overhead associated with triggering all objects in the relationship hierarchy on the OracleAS TopLink application.

Merging Clones on Deserialization

Unmarshalling a serialized object always occurs in the context of a Unit of Work when you integrate changes made outside of the JVM with the affected objects in the OracleAS TopLink application. Several options are available for the merge:

- *Merge only the direct attributes of the object being read*: Use the `shallowMergeClone(java.lang.Object rmiClone)` method to capture changes in the deserialized object only. Use this option when you know that changes to the object do not extend to related objects.
- *Merge the deserialized object and its privately owned parts*: Use the `mergeClone(java.lang.Object rmiClone)` method to capture changes in the deserialized object and any of its privately owned objects.
- *Merge the deserialized object and all referenced objects*: Use the `mergeCloneWithReferences(java.lang.Object rmiClone)` method to capture changes to the deserialized object, its privately owned objects, and all its referenced objects. Note that the referenced objects include only those objects with a direct relationship to the deserialized object.
- *Merge the entire relationship graph of the deserialized object*: Use the `deepMergeClone(java.lang.Object rmiClone)` method to capture all changes to the deserialized object's relationship graph. This method causes OracleAS TopLink to traverse all relationships from the deserialized object to its leaf objects and merge any changes it finds.

Limitations on Merge

To maintain data integrity, OracleAS TopLink imposes a restriction on merging back serialized objects. If the outside JVM adds objects to the structure passed to it, and then passes back the new objects, OracleAS TopLink merges those objects into the model if one of the following conditions is met:

- The new objects do not exist in the OracleAS TopLink model.
- The new objects exist in the OracleAS TopLink model and are registered with the Unit of Work.

Primary Keys

A primary key is a column (or combination of columns) that contains a unique identifier for every record in the table. OracleAS TopLink requires that every table that stores persistent objects has a primary key. The following concepts and techniques apply to primary keys:

- If a table uses a combination of columns to create a primary key (a *composite* primary key), declare all the necessary fields as primary keys.
- Sequencing is the most common method to implement a primary key.
- Descriptors must always provide mappings for a primary key. These mappings can be direct, transformation, or one-to-one.
- You do not have to define a primary key constraint in the database, but you must ensure that the fields you specify for the primary key are unique.

Under most circumstances, you set primary key information in the OracleAS TopLink Mapping Workbench for persistent Java objects and EJB entity beans. Alternatively, set the primary key manually in Java code.

For more information, see "Implementing Primary Keys in Java" on page 3-94.

Primary Keys and EJB Entity Beans

A primary key is a mechanism by which OracleAS TopLink and other applications identify persistent objects and entity beans. EJB entity beans use primary keys in much the same way as regular Java objects, and as with Java objects, you usually set primary keys for entity beans in the OracleAS TopLink Mapping Workbench.

EJB entity beans support both simple primary keys, which are composed of information from a single field in the bean, and composite primary keys, which are composed of information from one or more fields and are stored in a custom class.

Sequencing

When you create tables that do not include a unique key suitable for use as a primary key, use sequencing to assign an identifier to each record. In most cases, you configure sequencing through the OracleAS TopLink Mapping Workbench.

For more information, see "Working with Sequencing" in the *Oracle Application Server TopLink Mapping Workbench User's Guide*. For more information about implementing sequencing in Java code, see "Implementing Sequence Numbers in Java" on page 3-107.

This section describes how to assign primary keys to objects that use sequencing, and includes discussions on:

- Sequencing and Database Tables
- Sequencing and Preallocation Size
- Table Sequencing
- Oracle Native Sequencing
- Native Sequencing with Other Databases
- Sequencing with CMP Entity Beans

Sequencing and Database Tables

OracleAS TopLink offers three ways to implement sequencing. Although each method is unique, the three techniques have some commonality.

You store persistent objects for your application in database tables that represent the class of instantiated object. Each row of the table represents an instantiated object from that class, and one column in that table holds the primary key for each object. Sequencing populates the primary key row in the table.

When you configure sequencing, specify two settings for these tables, regardless of the type of sequencing you plan to use:

- The name of the table that stores the primary key for the class
- The name of the column in the table that stores the primary key for each object (the sequencing column)

Figure 3–3 Sequencing Elements in a Class Table

Table Name → **VEHICLE_POOL**

Sequencing Column →

VEH_ID	COLOR	MAKE	MODEL	YEAR
1	red	Chev	Malibu	2000
2	white	Ford	Focus	2001
3	white	Hyundai	Accent	2000
4	yellow	Dodge	3500 Tow Truck	1998
5	blue	Pontiac	Bonneville	2003
6	green	BMW	325i	2002

In addition to these two elements, table sequencing requires you to specify a `SEQ_NAME` (a name to identify the class in a special sequencing table name) for each sequenced object. You configure these elements in the OracleAS TopLink Mapping Workbench.

Sequencing and Preallocation Size

To improve sequencing efficiency, OracleAS TopLink allows you to preallocate sequence numbers. Preallocation enables OracleAS TopLink to build a pool of available sequence numbers that are assigned to new objects as they are created and inserted into the database. OracleAS TopLink assigns numbers from the pool until the pool is exhausted.

The preallocation size specifies the size of the pool of available numbers. Preallocation improves sequencing efficiency by substantially reducing the number of database accesses required by sequencing. By default, OracleAS TopLink sets preallocation size to 50. You can specify preallocation size either in the OracleAS TopLink Mapping Workbench or as part of the session login.

For more information about setting sequencing parameters at session login, see "Setting Sequencing at Login" on page 5-11.

Preallocation is available in table sequencing and is required for Oracle native sequencing.

Table Sequencing

Table sequencing involves creating and maintaining an extra database table that includes sequencing information for sequenced objects in the project. OracleAS TopLink maintains this table to track sequence numbers.

Sequencing information appears in this table for any class that uses sequencing. The default table is called `SEQUENCE` and contains two columns:

- `SEQ_NAME`, which specifies the class type to which the selected row refers
- `SEQ_COUNT`, which specifies the highest sequence number currently allocated for the object represented in the selected row

Figure 3–4 OracleAS TopLink SEQUENCE Table

SEQUENCE

SEQ_NAME	SEQ_COUNT
SEQ_V_POOL	350
SEQ_MACHINERY	800
SEQ_PURCH_ORDER	1550
SEQ_WORK_ORDER	2400

The rows of the `SEQUENCE` table represent every class that participates in sequencing. When you configure sequencing in the OracleAS TopLink Mapping Workbench, you specify the `SEQ_NAME` for the class. OracleAS TopLink adds a row with that name to the `SEQUENCE` table and initializes the `SEQ_COUNT` column to the value `1`.

You can create the `SEQUENCE` table on the database in one of two ways:

- Use the OracleAS TopLink Mapping Workbench to create the table, in the same way as you do any other table.

  For more information about specifying tables in the OracleAS TopLink Mapping Workbench, see "Working with Database Tables in the Navigator Pane" in the *Oracle Application Server TopLink Mapping Workbench User's Guide*.

- Use the OracleAS TopLink table creator to create and update the table manually.

  For more information, see "Creating the Sequence Table" on page A-21.

Using the SEQ_COUNT Column OracleAS TopLink includes an internal mechanism that manages table sequencing. This mechanism maintains a pool (a vector or array) of preallocated values for each sequenced class. When OracleAS TopLink exhausts this pool of values, it acquires a new pool of values, as follows:

1. OracleAS TopLink accesses the database, requesting that the `SEQ_COUNT` for the given class (identified by the `SEQ_NAME`) be incremented by the preallocation size and the result returned.

   For example, consider the `SEQUENCE` table in Figure 3–4. If you create a new purchase order and OracleAS TopLink has exhausted its pool of sequence numbers, then OracleAS TopLink executes SQL to increment `SEQ_COUNT` for `SEQ_PURCH_ORDER` by the preallocation size (in this case, the OracleAS TopLink default of 50). The database increments `SEQ_COUNT` for `SEQ_PURCH_ORDER` to 1600 and returns this number to OracleAS TopLink.

2. OracleAS TopLink calculates a maximum and minimum value for the new sequence number pool and creates the vector of values.
3. OracleAS TopLink populates the object sequence attribute with the first number in the array and writes the object to the class table.

As you add new objects to the class table, OracleAS TopLink continues to assign values from the pool until it exhausts the pool. When the pool is exhausted, OracleAS TopLink again requests new values from the table.

Default Versus Custom Tables In most cases, you implement table sequencing using the default table parameters. However, you may want to leverage the Custom Table option if:

- You want to use an existing sequence table for sequencing.
- You do not want to use the default naming convention for the table and its columns.

Oracle Native Sequencing

OracleAS TopLink support for native sequencing with Oracle databases is similar to table sequencing, except that OracleAS TopLink does not maintain a table in the database. Instead, the Oracle database contains a `SEQUENCE` object that stores the current maximum number and preallocation size for sequenced objects.

Understanding the Oracle SEQUENCE Object The Oracle `SEQUENCE` object implements a strategy that closely resembles OracleAS TopLink sequencing: it implements an `INCREMENT` construct that parallels the OracleAS TopLink preallocation size, and a `sequence.nextval` construct that parallels the `SEQ_COUNT` field in the OracleAS TopLink `SEQUENCE` table in table sequencing. This implementation enables OracleAS TopLink to use the Oracle `SEQUENCE` object as if it were an OracleAS TopLink `SEQUENCE` table, but eliminates the need for OracleAS TopLink to create and maintain the table.

As with table sequencing, OracleAS TopLink creates a pool of available numbers by requesting that the Oracle `SEQUENCE` object increment the `sequence.nextval` and return the result. Oracle adds the value, `INCREMENT`, to the `sequence.nextval`, and OracleAS TopLink uses the result to build the sequencing pool.

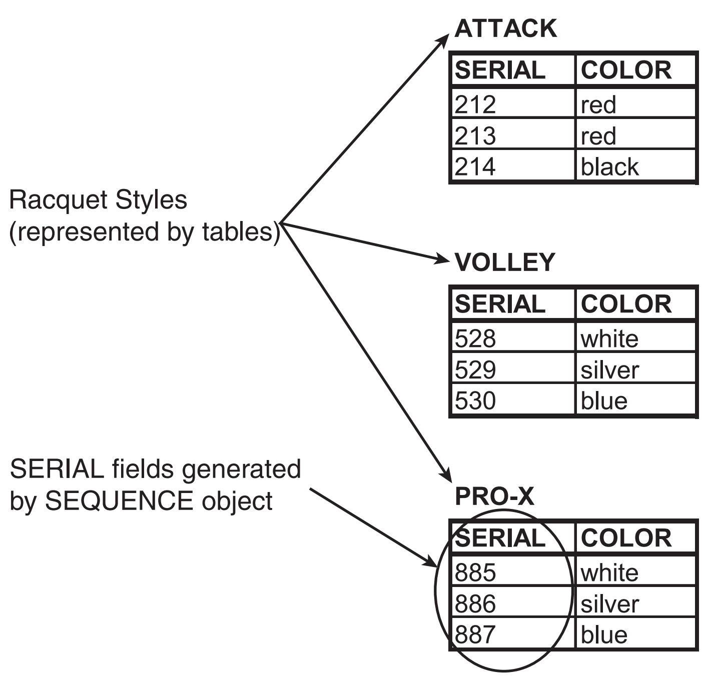
The key difference between this process and the process involved in table sequencing is that OracleAS TopLink is unaware of the `INCREMENT` construct on the `SEQUENCE` object. OracleAS TopLink sequencing and the Oracle SEQUENCE object operate in isolation. To avoid sequencing errors in the application, set the OracleAS TopLink preallocation size and the Oracle `SEQUENCE` object `INCREMENT` to the same value.

Using SEQUENCE Objects Your Database Administrator (DBA) must create a `SEQUENCE` object on the database for every sequencing series your application requires. If every class in your application requires its own sequence, the DBA creates a `SEQUENCE` object for every class; if you design several classes to share a sequence, the DBA need only create one `SEQUENCE` object for those classes.

For example, consider the case of a sporting goods manufacturer that manufactures three styles of tennis racquet. The data for these styles of racquet are stored in the database as follows:

- Each style of racquet has its own class table.
- Each manufactured racquet is an object, represented by a line in the class table.
- The system assigns serial numbers to the racquets, that use sequencing.

Figure 3–5 Example of Database Tables—Racquet Information



The manufacturer can:

- *Use separate sequencing for each racquet style*. The DBA builds three separate SEQUENCE objects, perhaps called ATTACK_SEQ, VOLLEY_SEQ, and PROX_SEQ. Each different racquet line has its own serial number series, and there may be duplication of serial numbers between the lines (for example: all three styles may include a racquet with serial number 1234).
- *Use a single sequencing series for all rackets*. The DBA builds a single SEQUENCE object (perhaps called RACQUET_SEQ). The manufacturer assigns serial numbers to racquets as they are produced, without regard for the style of racquet.

> **Note:** If the manufacturer chooses this second option, he might also choose to combine the three tables into a single table to improve database efficiency.

Native Sequencing with Other Databases

Several databases support a type of native sequencing in which the database management system (DBMS) generates the sequence numbers. When you create a class table for a class that uses sequencing, include a specified primary key column, and set the column type as follows:

- For Sybase and Microsoft SQL Server databases, set the primary key field to the type IDENTITY.
- For IBM Informix databases, set the primary key field to the type SERIAL.

> **Note:** OracleAS TopLink does not support native sequencing in IBM DB2 databases.

When you insert a new object into the table, OracleAS TopLink populates the object before insertion into the table, but does not include the sequence number. As the database inserts the object into its table, the database automatically populates the primary key field, with a value equal to the primary key of the previous object, plus 1.

At this point, and before the transaction closes, OracleAS TopLink reads back the primary key for the new object so that the object has an identity in the OracleAS TopLink cache.

Sequencing with CMP Entity Beans

To implement sequencing for CMP entity beans, use a sequencing strategy that implements preallocation, such as table sequencing or Oracle native sequencing. Preallocation ensures that the bean primary key is available at `ejbPostCreate()` time. If use native sequencing as offered in Sybase, Microsoft SQL Server, or Informix databases, be aware that:

- Native sequencing does not strictly conform to any EJB specification, because it does not initialize the primary key for a created object until you commit the transaction that creates the object. EJB specifications expect that the primary key is available at `ejbPostCreate()` time.
- OracleAS TopLink CMP integration for IBM WebSphere does not support native sequencing other than Oracle native sequencing.
- BEA WebLogic supports native sequencing; however, this type of native sequencing does not assign or return a primary key for a created object until you commit the transaction in which the object is created. Because of this, if you use native sequencing, commit a transaction immediately after calling the `ejbCreate` method to avoid problems with object identity in the OracleAS TopLink cache and the container.

OracleAS TopLink CMP Integration with IBM WebSphere The OracleAS TopLink CMP integration with IBM WebSphere does not automatically provide the primary key after calling the `ejbCreate` method. If you deploy to a WebSphere server, explicitly set the primary key in the `ejbCreate` method. Example 3–20 illustrates this call in a WebSphere integration.

Example 3–20 Setting Primary Key in IBM WebSphere

```
public Integer ejbCreate() throws CreateException {
    oracle.toplink.ejb.cmp.was.SessionLookupHelper.getHelper().getSession(this)
      .getActiveUnitofWork().assignSequenceNumber(this);
    return null
}
```

OracleAS TopLink CMP Integration with BEA WebLogic In the OracleAS TopLink CMP integration with BEA WebLogic, OracleAS TopLink automatically sets the primary key field on the bean. You do not pass the key value as a parameter to the `create()` method, nor set them in the `create()` method.

Example 3–21 Setting Primary Key in BEA WebLogic

```
public Integer ejbCreate() throws CreateException {
    return null;
}
```

The additional line of code looks up the correct session and uses it to assign a sequence number to the bean.

For more information about how to configure sequencing, see the *Oracle Application Server TopLink Mapping Workbench User's Guide*.

Sequencing with Stored Procedures

If you have stored procedures that perform sequencing for your application, use an amendment method to direct sequencing queries to use the stored procedures.

Example 3–22 Calling a Stored Procedure for Sequencing

```
DataModifyQuery seqUpdateQuery = new DataModifyQuery();
StoredProcedureCall call = new StoredProcedureCall();
call.setProcedureName("UPDATE_SEQ");
seqUpdateQuery.addArgument("SEQ_NAME"};
seqUpdateQuery.setCall(call)
project.getLogin().setUpdateSequenceQuery(seqUpdateQuery));
```

Example 3–22 illustrates specifying a stored procedure for sequence updates. The name of the stored procedure must match the name specified in the `setProcedureName` call (in this case, `UPDATE_SEQ`). The `seqUpdateQuery.addArgument` contains one argument, the sequence name.

Example 3–23 illustrates the use of a stored procedure for sequence selects.

Example 3–23 Using a Stored Procedure for Sequence Selects

```
ValueReadQuery seqReadQuery = new ValueReadQuery();
StoredProcedureCall call = new StoredProcedureCall();
call.setProcedureName("SELECT_SEQ");
seqReadQuery.addArgument("SEQ_NAME"};
seqReadQuery.setCall(call)
project.getLogin().setSelectSequenceNumberQuery (seqReadQuery));
```

The name of the stored procedure must match the name specified in the `setProcedureName` call (in this case, `SELECT_SEQ`). The `seqUpdateQuery.addArgument` contains one argument, the sequence name.

Foreign Keys

A foreign key is a combination of columns that reference a unique key, usually the primary key, in another table. As with a primary key, a foreign key can be any number of fields, all of which are treated as a unit. A foreign key and the parent key it references must have the same number and types of fields.

OracleAS TopLink enables you to specify two types of foreign keys:

- *Foreign key*: key added to the table associated with the mapping's own descriptor.
- *Target foreign key:* key that references the target object's table back to the key from the mapping descriptor's table. The key in the mapping descriptor table is a foreign key in the target table that the target table uses to reference the mapping descriptor's table.

Relationship mappings use foreign keys to search the database for the information it requires to instantiate the target object or objects. For example, if every Employee has an attribute, `address`, that contains an instance of `Address` (which has its own descriptor and table), then the one-to-one mapping for the `address` attribute specifies foreign key information to find an address for a particular Employee.

Multiple Table Mappings

OracleAS TopLink enables you to store the information for a single class in multiple tables. This feature offers you the flexibility to create the objects for your application without imposing any new design requirements on your database schema.

For example, you can create a class called `EMPLOYEE` that contains not just personal information about the employees, but also business information, such as salary. If your database schema stores salaries in a separate table from basic employee information, OracleAS TopLink multiple table mappings support enables you to create the class you require. Use multiple tables when either of the following is true:

- You have a subclass with a superclass mapped to one table and the subclass has additional attributes that are also mapped to a second table.
- A class is not involved in inheritance and its data is spread out across multiple tables.

You can associate information for the class using primary keys or foreign keys.

For more information about mapping a class to multiple tables, see "Working with Multiple Tables" in the *Oracle Application Server TopLink Mapping Workbench User's Guide*.

For more information about implementing multiple table mappings in code, see "Implementing Multiple Tables in Java" on page 3-102.

Mapping and Enterprise JavaBeans

To enable container-managed persistent (CMP) storage of entity beans in an Enterprise JavaBean (EJB) application, map the attributes on the bean implementation class. The implementation class is the class specified in the `ejb-class` element for the specified bean in the `ejb-jar.xml` deployment descriptor file. Do not map the `Home` or `Remote` interface classes, or the primary key classes.

EJBs and the OracleAS TopLink Mapping Workbench

If you use the OracleAS TopLink Mapping Workbench to build projects with entity beans, you can load the bean classes themselves into the OracleAS TopLink Mapping Workbench. You do not need to load the `Remote`, `Local`, `Home`, and `localHome` interfaces, or the primary key class, nor must you use these classes to define mappings.

To avoid errors when you load the beans, ensure that classes referenced by the entity beans are on the project class path used by the OracleAS TopLink Mapping Workbench project. The `Remote`, `Local`, `Home`, and `localHome` interfaces must also be on the class path, because they may be used during EJB validation.

Inheritance

Inheritance enables you to share attributes between objects such that a subclass inherits attributes from its parent class. OracleAS TopLink provides several methods to preserve inheritance relationships, and enables you to override mappings that are specified in a superclass, or to map attributes that are not mapped in the superclass. Subclasses must include the same database field (or fields) as the parent class for their primary key (although the primary key can have different names in these two tables). As a result, when you are mapping relationships to a subclass stored in a separate table, the subclass table must include the parent table primary key, even if the subclass primary key differs from the parent primary key.

This section describes OracleAS TopLink inheritance, and introduces several topics and techniques to leverage inheritance in your own applications, including:

- Understanding Object Inheritance

- Representing Inheritance in the Database
- Class Types
- Class Indicators
- Class Extraction Methods
- Entity Bean Inheritance Restrictions

For more information about implementing inheritance in code, see "Implementing Inheritance in Java" on page 3-95.

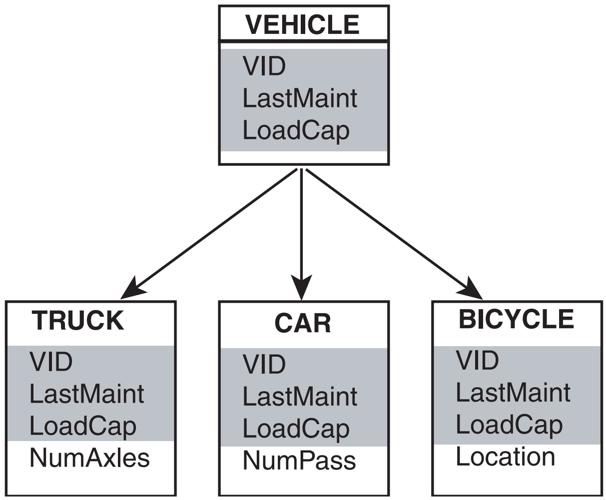
Understanding Object Inheritance

Consider a simple database used by a courier company. It contains registration information for three types of vehicles: trucks, cars, and bicycles. For each vehicle type, your application requires the following information:

- VID (Vehicle Identification)
- LastMaint (mileage since last maintenance)
- LoadCap (load capacity)

If these are all the attributes shared by all vehicles in the application, then these attributes must all appear in the super class, Vehicle. You can then build subclasses for each of the vehicle types that reflects their differences. For example, the Truck class might have an attribute indicating whether the local department of transportation considers it to be a commercial vehicle (NumAxles), the Car class might require a NumPass (number of passengers) attribute, and the Bicycle class, by virtue of its more limited range, may require a Location attribute. Through inheritance, each vehicle automatically inherits the basic vehicle information, but by being separate subclasses, also have unique characteristics.

Figure 3–6 Inheritance in a Courier Application



Representing Inheritance in the Database

You can represent inheritance in the database in one of two ways:

- Multiple tables that represent the parent class and each child class.
- A single table that comprises the parent and all child classes.

Figure 3–7 Inheritance in the Database in Individual Tables

VEHICLE Table

VID	LastMaint	LoadCap
1	2002	850
2	2000	30
3	2001	920
4	1998	1700
5	2003	35
6	2001	2250

TRUCK Table

VID	LastMaint	LoadCap	NumPass
4	1998	1700	5
6	2001	2250	3

CAR Table

VID	LastMaint	LoadCap	NumPass
1	2002	850	5
3	2001	920	7

BICYCLE Table

VID	LastMaint	LoadCap	NumPass
2	2000	30	1
5	2003	35	1

If your database already represents the objects in the inheritance hierarchy this way, you can map the objects and relationships without modifying the tables. However, it is most efficient to represent all classes from a given inheritance hierarchy in a single table, because it substantially reduces the number of table reads and eliminates joins when querying on objects in the hierarchy.

Figure 3–8 Inheritance in the Database in a Single Table

VEHICLE Table

VID	LastMaint	LoadCap	Class	NumPass
1	2002	850	C	5
2	2000	30	B	1
3	2001	920	C	7
4	1998	1700	T	5
5	2003	35	B	1
6	2001	2250	T	3

To consolidate tables in the database this way, determine the class type of the objects represented by the rows in the table. There are two ways to determine class type:

- If you can add columns to the database table, add a class indicator column that represents the vehicle class type (Truck, Car, or Bicycle).

For more information about class indicators, see "Class Indicators" on page 3-52.

- If you cannot modify the table, build a class extraction method that executes an appropriate login to determine the class type.

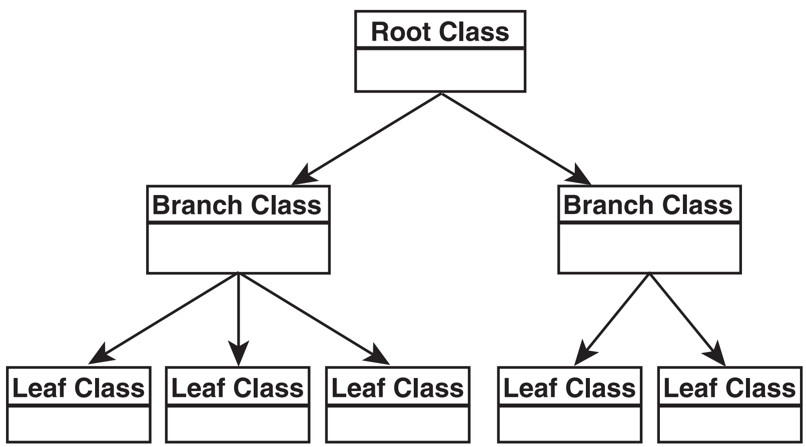
  For more information about class extraction methods, see "Class Extraction Methods" on page 3-53.

Class Types

The OracleAS TopLink inheritance hierarchy includes three types of classes:

- Root Class
- Branch Class
- Leaf Class

Figure 3–9 Inheritance Hierarchy Class Types



Root Class

The root class stores information for all instantiable classes in its subclass hierarchy. By default, queries performed on the root class return instances of the root class and its instantiable subclasses. However, you can also configure the root class to return only instances of itself, without instances of its subclasses when queried. All class types beneath the root class inherit from the root class.

Branch Class

Branch classes have a persistent superclass and subclasses. By default, queries performed on the branch class return instances of the branch class and any of its subclasses. As with the root class, you can configure the branch class to return only instances of itself without instances of its subclasses when queried. All classes below the branch class inherit attributes from the branch class, including any attributes the branch class inherits from classes above it in the hierarchy.

Leaf Class

Leaf classes have a persistent superclass in the hierarchy, but do not have subclasses. Queries performed on the leaf class return only instances of the leaf class.

Class Indicators

A class indicator is a mechanism for determining the class or type of an object. For example, a Person table might include an indication of whether the person represented by the table row is an Employee or a Manager. Use the class indicator to select the appropriate subclass to be instantiated from a set of available subclasses.

Class Indicator Field

A class indicator field is a number or string stored in a database table that indicates the class or type of an object. OracleAS TopLink uses this information to determine the correct type of object to instantiate when building an object from that data in the row. For example, an `EMPLOYEE` table might contain a field, the value of which indicates whether the employee is permanent or contract, and determines whether OracleAS TopLink instantiates a `PermanentEmployee` object or a `ContractEmployee` object.

You can use strings or numbers as values in the class indicator field in the database. The root class descriptor must specify how the value in the class indicator field translates into the class to be instantiated.

Class Indicators and Mappings

Class indicator fields do not have an associated direct mapping unless it is set to read-only. Mappings defined for the write-lock or class indicator field *must* be read-only, unless the write-lock is configured not to be stored in the cache and the class indicator is part of the primary key.

For more information about transformation mappings, see "Transformation Mappings" on page 3-68.

Class Extraction Methods

Class extraction enables you to determine the correct class type to instantiate from a table that includes several classes. Unlike a class indicator, however, a class extraction method does not rely on a single column in the table to determine class type. Instead, you can apply logic to the information in several fields to determine class type.

This method is useful when you use a legacy database with a new application. Table 3–1 illustrates a sample use of the a class extraction method.

Table 3–1 Sample use of the a Class Extraction Method

ID	NAME	JOB_TYPE	JOB_TITLE
732	Bob Jones	1	Manager
733	Sarah Smith	3	Technical Writer
734	Ben Ng	2	Director
735	Sally Johnson	3	Programmer

The inheritance hierarchy is designed such that Employee is the root class, and Director is a branch class that inherits from Employee. All employees, other than directors, are represented as instances of Employee, but directors must be represented by an instance of the Director class. Because values other than 2 can appear in the JOB_TYPE field, you cannot use OracleAS TopLink's class indicator mechanism for mapping this data.

To resolve this, add a class extraction method to the root class, Employee. The method executes custom logic to determine the correct class to instantiate. The method is static, returns a Class object, and takes `DatabaseRow` as a single parameter.

Example 3–24 Simple Class Extraction Method

```
// Return the Director class for TYPE values of 2,
// Employee class for any other value

public static Class getClassFromRow(DatabaseRow row) {
    if (row.get("JOB_TYPE").equals(new Integer(2)) {
        return Director.class;
```

```
    }
    else { return Employee.class;
    }
}
```

This simple case enables you to determine whether the selected person is of the Director class or the Employee class. You can also implement complex logic that combines information from several columns in the table to infer class type. For example, consider a table that represents vehicles in a municipal vehicle pool. In addition to other information, Table 3–2 illustrates, the database includes data that indicates gross vehicle weight and number of axles.

Table 3–2 Gross Vehicle Weight and Number of Axles Example

Gross Vehicle Weight	Number Of Axles
2650	3
800	2
2730	2
2400	2
3580	4

Although there is no direct indication of vehicle type in the data, you can build logic into a class extraction method to infer the vehicle type. This is made easier if you are familiar with the available types in the database. In this example, you can use a class extraction method to implement the following logic:

- If `NumberOfAxles` is greater than 2, then return the class `HeavyTruck`.
- If `NumberOfAxles` is 2 or less and `GrossVehicleWeight` is greater than 1000, then return the class type `PassengerVehicle`.
- In all other cases, return the class `Motorcycle`.

Example 3–25 Complex Class Extraction Method

```
public static Class getClassFromRow(DatabaseRow row) {
    if (row.get("NumberOfAxles").intValue()>2){
        return HeavyTruck.class;
    }
    else {
        if (row.get("GrossVehicleWeight").intValue()>1000) {
        return PassengerVehicle.class;
```

```
            }
            else { return Motorcycle.class;
            }
        }
}
```

In addition to implementing logic to determine object class, you can also use class extraction methods to execute other methods unrelated to class determination. This is an unusual use for class extraction methods but, provided that the method ultimately returns a class type, it is possible.

To implement the class extraction method in the OracleAS TopLink Mapping Workbench, open the inheritance settings for the root descriptor in the subclass hierarchy (`EMPLOYEE` in this case), and select the class extraction method in the **Use Class Extraction** Method box.

Entity Bean Inheritance Restrictions

The following restrictions apply to entity beans when using inheritance:

- The `Home` interfaces cannot inherit. The `findByPrimaryKey` method must be overloaded to have the correct return type, but this is not allowed. As a result, inheritance is not applicable to the `Home` interfaces.
- The primary key of the subclass must be the same as that of the parent class.

The Application Server EJB 1.1 and 2.0 CMP Advanced Examples illustrate inheritance. For more information, see the OracleAS TopLink Examples at `<`*`ORACLE_HOME`*`>\toplink\doc\examples.htm`.

> **Note:** Because the existing EJB specifications offer no implementation guidelines for inheritance, exercise caution when implementing inheritance, especially if EJB compliance is an issue for your application.

Mapping EJB Entity Beans

EJB Entity beans represent a *business entity*. Entity beans can be shared by many users and are long-lived, able to survive a server failure. Essentially, entity beans *are* persistent data objects (objects with durable state that exist from one moment in time to the next).

This section describes entity bean development, as well as the following mapping topics and techniques:

- Terminology and Definitions
- Overview of Bean-Managed Persistence
- Overview of Container-Managed Persistence
- Maintaining Bidirectional Relationships
- Managing Dependent Objects Under EJB 1.1
- Managing Dependent Objects Under EJB 2.0
- Managing Collections of EJBObjects Under EJB 1.1

Terminology and Definitions

Enterprise JavaBeans

An EJB implements a business task or a business entity. EJBs are server-side domain objects that fit into a component-based architecture for building enterprise applications using the Java language. EJBs are Java objects that the developer can install in an EJB server to make them distributed, transactional, and secure. OracleAS TopLink supports three kinds of EJBs under the EJB 2.0 specification: session beans, entity beans, and message-driven beans. Note that EJB 1.1 does not support message-driven beans.

EJB Server and Container

An EJB bean resides in an EJB container that in turn resides in an EJB server. Although the EJB 2.0 specification does not define the container-server relationship, the accepted paradigm is that the server provides the bean with access to various services (transactions, security, and so on), and the container provides the execution context for the bean by managing its life cycle.

Deployment Descriptors

Deployment descriptors supply additional information that is required to install an EJB within its server. The deployment descriptors are of a set of XML files that provide the security, transaction, relationship, and persistence information for the bean.

Session Beans

Session beans represent a business operation, task, or process. Although the use of a session bean may involve database access, the beans are not in themselves persistent because they do not directly represent a database entry. Session beans do not always retain conversational state; they can be stateful and retain client information between calls. They can be stateless and retain information only within a single method call.

You can use OracleAS TopLink to make the regular Java objects that are accessed by a session bean persistent, or to access OracleAS TopLink persistent entity beans. Session beans may also act as wrappers to other legacy applications.

Entity Beans

Entity beans represent a persistent data object that exists from one access to the next. Accomplish persistence by storing the object in a relational database, object database, or some other storage facility.

Two schemes exist for making entity beans persistent: bean-managed persistence (BMP) and container-managed persistence (CMP). BMP requires the bean developer to hand-code the methods that perform the persistence work. CMP uses information supplied by the developer to handle all aspects of persistence.

Message-Driven Beans

Message-driven beans process asynchronous Java Message Service (JMS) messages. A bean method is transactionally-invoked by a JMS message sent to the objects registered against the given topic. From a client perspective, a message-driven bean is simply a JMS consumer with no conversational state and no `Home` or `Remote` interfaces.

Overview of Bean-Managed Persistence

OracleAS TopLink provides a class `oracle.toplink.ejb.bmp.BMPEntityBase`. This class provides developers with a starting point when developing beans. The `BMPEntityBase` class provides implementation for all EJB specification required methods except `ejbPassivate()` which is excluded because of special requirements. By subclassing the `BMPEntityBase`, developers have an OracleAS TopLink-enabled entity bean.

To use the `BMPEntityBase`, create the `sessions.xml` file. For information about the `sessions.xml` file, see "Session Manager" on page 4-29. In addition, add an `oracle.toplink.ejb.bmp.BMPWrapperPolicy` to each descriptor that represents an Entity Bean. This `BMPWrapperPolicy` provides OracleAS TopLink with the information to create Remote objects for entity beans and to extract the data out of a Remote object. After this is performed, the user must create the `Home` and `Remote` interfaces, create deployment descriptors, and deploy the beans.

If a more customized approach is required, OracleAS TopLink provides a hook into its functionality through the `oracle.toplink.ejb.bmp.BMPDataStore` class. Use this class to translate EJB-required functionality into simple calls.

The `BMPDataStore` provides implementations of `LOAD` and `STORE`, multiple finders, and `REMOVE` functionality. The `BMPDataStore`, requires a `sessions.xml` file and the session manager. A single instance of `BMPDataStore` must exist for each bean type deployed within a session. When creating a `BMPDataStore`, pass in the session name of the session that the `BMPDataStore` must use to persist the beans and the class of the Bean type being persisted. Store the `BMPDataStore` in a global location so that each instance of a Bean type uses the correct Store.

If you use a customized implementation, the full functionality of the server session and the `UnitOfWork` is available.

BMP Support with EJB 2.0

To use BMP support with EJB 2.0, the `Home` interface must inherit from the `oracle.toplink.ejb.EJB20Home`. To make calls to the `oracle.toplink.ejb.bmp.BMPEntityBase`, the `FindAll()` method must call the EJB 2.0 version of the methods. These methods are prefixed with `ejb20`.

For example, in the EJB 2.0 version, the `findAll()` method appears as `ejb20FindAll`.

Using Local beans

To use local beans, use the `oracle.toplink.ejb.EJB20LocalHome` setting instead of the default `oracle.toplink.ejb.EJB20Home`.

Instead of the `oracle.toplink.ejb.BMPWrapperPolicy` setting, use the `oracle.toplink.ejb.bmp.BMPLocalWrapperPolicy` setting.

To accommodate both local and remote configurations, ensure the following:

- For a bean that has a single interface, use the corresponding wrapper policy (local or remote) for the descriptor.
- Beans can only participate in relationships as either `Local` or `Remote` interfaces, not both.

Overview of Container-Managed Persistence

OracleAS TopLink CMP is an extension of the OracleAS TopLink persistence framework. OracleAS TopLink CMP support provides container-managed persistence for EJBs deployed in a J2EE container.

OracleAS TopLink CMP support enables complex mappings from entity beans to relational database tables and enables you to model bean-to-bean and bean-to-regular Java object relationships. OracleAS TopLink provides a rich set of querying options and allows query definition at the bean-level rather than the database level. OracleAS TopLink CMP supports the specification as defined by Sun Microsystems.

Understanding CMP

This section introduces the concepts required to use CMP facilities. It highlights the features that are specific to OracleAS TopLink CMP, and explains any differences in the use of other core features.

OracleAS TopLink and CMP Entity Beans

The common mechanism for developers to make beans persistent is to map beans to a relational database. The EJB specification describes the CMP entity bean as a type of bean for which the designer does not have to include calls to any particular persistence mechanism in the bean itself. The EJB Server and its tools use meta-information in the deployment descriptor to describe how the bean is to be persisted to a database. This is commonly referred to as *automatic* persistence.

EJB 2.0 Support OracleAS TopLink provides support for EJB 2.0 entity beans. Here are some specific features of EJB 2.0 that OracleAS TopLink supports:

- Local interfaces and local relationships
- Generation of concrete bean subclasses
- EJB QL
- Automatic management of bidirectional relationships
- Initializing a project from the `ejb-jar.xml` file
- Finders
- Home methods
- `ejbSelect`

Java Objects and Entity Beans

Table 3–3 illustrates the components that Java objects contain:

Table 3–3 Java Object Components

Component	Function
Attributes	Stores primitive data such as integers, as well as simple Java types such as String and Date.
Relationships	Stores references to other OracleAS TopLink-enabled classes. An OracleAS TopLink-enabled class (also known as a persistent class) has a descriptor and can be stored in the database.
Methods	Stores paths of execution that can be invoked in a Java environment. Methods are not stored in the database.

Table 3–4 illustrates the components that entity beans contain:

Table 3–4 Entity Bean Components

Component	Function
Bean instance	An instance of an entity bean class supplied by the bean developer. It is a regular Java object whose class implements the `javax.ejb.EntityBean` interface. The bean instance has persistent state. The client application must never access the bean instance directly.
EJBObject	An instance of a generated class that implements the `Remote` interface defined by the bean developer. This instance wraps the bean and provides client interaction with the bean. The `EJBObject` does not have persistent state.
EJBHome	An instance of a class that implements the `Home` interface supplied by the bean developer. This instance, accessible from JNDI, provides all create and finder methods for the EJB. The `EJBHome` does not have persistent state.
EJBLocalObject (EJB 2.0 only)	An instance of a generated class that implements the `Local` interface defined by the bean developer. The key difference between an `EJBLocalObject` and an `EJBObject` is that the `EJBLocalObject` is accessed only from within the same server on which the beans are deployed. The `EJBLocalObject` does not have persistent state.

Table 3–4 Entity Bean Components (Cont.)

Component	Function
EJBLocalHome (EJB 2.0 only)	An instance of a class that implements the `localHome` interface supplied by the bean developer. This instance, accessible from JNDI, provides all `create` and `finder` methods for the EJB. The key difference between an `EJBLocalHome` and an `EJBHome` is that access to the `EJBLocalHome` is only available from within the same server on which the beans are deployed, even when using JNDI. The `EJBLocalHome` does not have persistent state.
EJB Primary Key	An instance of the primary key class provided by the bean developer. The primary key is a serializable object whose fields match the primary key fields in the bean instance. Although the EJB primary key shares some data with the bean instance, it does not have persistent state. If the key consists of a single field, the bean does not have to have a separate primary key class under the EJB 1.1 or later specifications.

For more information about the Enterprise JavaBeans and the EJB specification, see

```
http://java.sun.com/products/ejb/
http://java.sun.com/products/ejb/docs.html
http://java.sun.com/j2ee/white/index.html
```

Maintaining Bidirectional Relationships

When one-to-one or many-to-many mappings are bidirectional, maintain the back-pointers as the relationships change. When the relationship is between two entity beans (in EJB 2.0), OracleAS TopLink automatically maintains the relationship. However, when the relationship is between an entity bean and a Java object, or when the application is built to the EJB 1.1 specification, the relationship must be maintained manually. To set the back-pointer under the EJB 1.1 specification, do one of the following:

- The entity bean can maintain the back-pointer when the relationship is established or modified.
- The client can explicitly set the back-pointer.

If you set back-pointers within the entity bean, the client is freed of this responsibility. This has the advantage of encapsulating the mapping maintenance implementation in the bean.

Note: Under the EJB 1.1 specification, you must manually update all back-pointers.

One-to-Many Relationship

In a one-to-many mapping, a source bean might have several dependent target objects. For example, an `EmployeeBean` might have several dependent `phoneNumbers`. When a new dependent object (a `phoneNumber`, in this example) is added to an employee record, the `phoneNumber`'s back-pointer to its owner (the employee) must also be set.

Example 3–26 Setting the Back-Pointer in the Entity Bean

Maintaining a one-to-many relationship in the entity bean involves getting the local object reference from the context of the `EmployeeBean`, and then updating the back-pointer. The following code illustrates this technique:

```
// obtain owner and phoneNumber
owner = empHome.findByPrimaryKey(ownerId);
phoneNumber = new PhoneNumber("cell", "613", "5551212");
// add phoneNumber to the phoneNumbers of the owner
owner.addPhoneNumber(phoneNumber);
```

The Employee's `addPhoneNumber( )` method maintains the relationship, as follows:

```
public void addPhoneNumber(PhoneNumber newPhoneNumber) {
    //get, then set the back pointer to the owner
    Employee owner = (Employee)this.getEntityContext().getEJBLocalObject();
    newPhoneNumber.setOwner(owner);
    //add new phone
    getPhoneNumbers().add(newPhoneNumber);
}
```

Managing Dependent Objects Under EJB 1.1

The EJB 1.1 specification recommends that you model entity beans so that all dependent objects are regular Java objects and not other entity beans. If you expose a dependent or privately owned object to the client application, it must be serializable (that is, it implements the `java.io.Serializable interface`) so that it can be sent over to the client and back to the server.

Serializing Java Objects Between Client and Server

Because entity beans are remote objects, they are referenced remotely in a *pass-by-reference* fashion. When an entity bean is returned to the client, a remote reference to the bean is returned.

Unlike entity beans, regular Java objects are not remote objects. As a result, when regular Java objects are referenced remotely, they are passed by value (rather than by reference) and serialized (copied) from the remote machine on which they originally resided.

Merging Changes to Regular Java Objects One of the effects of serializing regular Java objects between servers and clients is a loss of object identity, due to the copying semantics inherent in serialization. When you serialize a dependent object from the server to the client and then back, two objects with the same primary key but different object identities exist in the server cache. These objects must be merged to avoid exceptions.

If relationships exist between entity beans and Java objects, and these objects are serialized back and forth between the client and server, either:

- Use the OracleAS TopLink `SessionAccessor` utility class to perform the merge for you.
- Merge the objects yourself by adding merge methods on your regular Java objects and within your *set* methods.

Using Session Accessor to Merge Dependent Objects Use the class `oracle.toplink.ejb.WebLogic.SessionAccessor` to perform merges for you within the *set* methods (on your bean class) that take regular Java objects as their arguments.

Two static methods are defined on the `SessionAccessor` that allow you to do the register and merge operation:

registerOrMergeObject() This method requires two arguments: the object to merge and the `EntityContext` for the bean.

Example 3–27 Using the registerOrMergeObject() Method

```
public void setAddress(Address address) {
    this.address = (Address)SessionAccessor
      .registerOrMergeObject(address,this.ctx);
}
```

The `registerOrMergeAttribute()` method requires three arguments: the Java object to be merged, the name of the attribute, and the `EntityContext` for the bean:

```
public void setAddress(Address address) {
    this.address = (Address) SessionAccessor.registerOrMergeAttribute
      (address, “address”, this.ctx);
}
```

To use the `registerOrMergeAttribute()` call for collection mappings, pass the entire collection as the attribute object.

For example:

```
public void setPhones(Vector phones) {
    this.phones = (Vector)SessionAccessor.registerOrMergeAttribute(phones,
      "phones", this.ctx);
    //... additional logic to set back-pointers on the phones
}
```

The `registerOrMergeObject()` method is not as simple to use for setters of collection mappings. It requires that you iterate through the collection and invoke the `registerOrMergeObject()` for each element in the collection. You must also create a new collection, set in the entity bean, to hold the return values of the call.

Merging code may be required in methods that add elements to a collection.

For example:

```
/* The old version of this phone number is removed from the collection. It is
assumed that equals() returns true for phones with the same primary key value.
If this is not true, you must iterated through the phones to see if a phone with
the same primary key already exists in the collection. */
public void addPhoneNumber(PhoneNumber phone) {
phone.setOwner((Employee)this.ctx.getEJBObject());
//add to collection
//merge new phone
PhoneNumber serverSidePhone =
(PhoneNumber)SessionAccessor.registerOrMergeObject(phone,this.ctx);
//set back pointer
getPhoneNumbers().addElement(serverSidePhone);
}}
```

Note: This example requires merging code only if there is a risk that a **Phone** with the same primary key can be added twice. If the elements in a collection cannot be added more than once, then merging code is not required.

Merging Dependent Objects without Session Accessor There are several ways to merge objects manually. For example, you can use a `set()` method as follows:

```
public void setAddress(Address address) {
    if(this.address == null){
        this.address = address;
    } else{
        this.address.merge(address);
    }
}
```

You must merge objects when they are added to a collection on the entity bean, unless the objects cannot be added more than once to a collection in which case, merging is not necessary.

Merging a collection requires more work. Determine if a copy of each object already exists in the collection, and if so, merge the two copies. If not, you need only add the new object to the collection.

Managing Dependent Objects Under EJB 2.0

Unlike EJBs, OracleAS TopLink dependent persistent objects can be sent back and forth between a client and the server. When objects are serialized, the risk exists the objects can cause the cache to lose the identity of the objects or attempt to cache duplicate identical objects. To avoid potential problems, use the bean set methods when adding dependent objects to relationship collections. This enables OracleAS TopLink to handle merging of objects in the cache.

Example 3–28 Adding a Dependent Object

```
addPhoneNumber(PhoneNumber phone) {
    Collection phones = this.getPhoneNumbers();
    Vector newCollection = new Vector();
    newCollection.addAll(phones);
    newCollection.add(phone);
    this.setPhones(newCollection);
}
```

Managing Collections of EJBObjects Under EJB 1.1

Collections generally use the `equals()` method to compare objects. However, in the case of a Java object that contains a collection of entities, the `EJBObjects` do not respond as expected to the `equals()` method. If you manage a collection of entities under EJB 1.1, we recommend the use of the `isIdentical()` method to avoid problems.

In addition, the standard collection methods, such as `remove()` or `contains()`, frequently return unexpected results and so must be avoided.

> **Note:** This issue does not arise in the case of an entity contains a collection of entities, because the EJB 2.0 container collection used handles equality appropriately.

Several options are available when dealing with collections of `EJBObjects`. One option is to create a helper class to assist with collection-type operations. Example 3–29 illustrates the use of a helper in the `EJBCollectionHelper` distribution.

Example 3–29 Using a Helper Class to Manage a Collection of EJBObjects

```
public void removeOwner(Employee previousOwner){
EJBCollectionHelper.remove(previousOwner, getOwners());
}
```

Example 3–30 illustrates the implementation of `remove()` and `indexOf()` in `EJBCollectionHelper`.

Example 3–30 Using remove() and indexOf() in the EJBCollectionHelper

```
public static boolean remove(javax.ejb.EJBObject ejbObject, Vector vector) {
    int index = -1;
    index = indexOf(ejbObject, vector);
    // indexOf returns -1 if the element is not found.
    if(index == -1){
        return false;
    }
    try{
        vector.removeElementAt(index);
    } catch(ArrayIndexOutOfBoundsException badIndex){
        return false;
    }
```

```
    return true;
}
public static int indexOf(javax.ejb.EJBObject ejbObject, Vector vector) {
    Enumeration elements = vector.elements();
    boolean found = false;
    int index = 0;
    javax.ejb.EJBObject current = null;
    while(elements.hasMoreElements()){
        try{
            current = (javax.ejb.EJBObject)
            elements.nextElement();
            if(ejbObject.isIdentical(current)){
            found = true;
            break;
            }
        }catch(ClassCastException wrongTypeOfElement){
            . . .
        }catch (java.rmi.RemoteException otherError){
            . . .
        }
        index++; //increment index counter
    }
    if(found){
        return index;
    } else{
        return -1;
    }
}
```

You can create a special Collection class that uses `isIdentical()` instead of `equals()` for its comparison operations. To use `isIdentical()`, properly define the `equals()` method for the primary key class.

Descriptor Validation

You can validate descriptors in two ways:

- Run the project in a test environment and watch for and interpret any exceptions that occur.

  For more information about descriptor exceptions, see "Descriptor Exception" on page C-4.

- Run the OracleAS TopLink Integrity Checker.

For more information about the Integrity Checker, see "Using the Integrity Checker" on page 4-66.

Advanced Mappings

Several complex mappings are available in OracleAS TopLink. This section discusses the following mapping types:

- Transformation Mappings
- Serialized Object Mappings
- Variable One-to-One Mappings
- Object Relational Mappings
- Direct Map Mappings

Transformation Mappings

Transformation mappings enable you to create specialized translations between how a value is represented in Java and in the database. Use transformation mappings only when mapping multiple fields into a single attribute. Transformation mapping is often appropriate when you use values from multiple fields to create an object.

Note: Because of the complexity of transformation mappings, it is often easier to perform the transformation with `get` and `set` methods of a direct-to-field mapping.

After you create the required transformation method, use the OracleAS TopLink Mapping Workbench to implement transformation mappings.

For more information, see "Working with Transformation Mappings" in *Oracle Application Server TopLink Mapping Workbench User's Guide*.

Implementing Transformation Mappings in Java

Transformation mappings are instances of the `TransformationMapping` class and require the following elements:

- The attribute mapped, set by sending the `setAttributeName( )` message; not required for write-only mappings

- The method to be invoked that sets the value of the attribute from information in the database row; set by sending the `setAttributeTransformation( )` message that expects one or two parameters: a `DatabaseRow` and optionally a `Session`
- A set of methods associated to fields in the database, where the value for each field is the result of invoking the associated method; associations are made by sending the `addFieldTransformation( )` message, passing along the database field name and the method name

Use the optional `setGetMethodName( )` and `setSetMethodName( )` messages to access the attribute through user-defined methods, rather than directly.

Example 3–31 Creating a Transformation Mapping and Registering It with the Descriptor

This example provides custom support for two fields. You can use this approach to map any number of fields.

```
// Create a new mapping and register it with the descriptor.
TransformationMapping transformation1 = new TransformationMapping();
transformation1.setAttributeName ("dateAndTimeOfBirth");
transformation1.setAttributeTransformation ("buildDateAndTime");
transformation1.addFieldTransformation("B_DAY", "getDateOfBirth");
transformation1.addFieldTransformation("B_TIME", "getTimeOfBirth");
descriptor.addMapping(transformation1);
// Define attribute transformation method to read from the database row
public java.util.Date buildDateAndTime(DatabaseRow row) {
    java.sql.Date sqlDateOfBirth = (java.sql.Date)row.get("B_DAY");
    java.sql.Time timeOfBirth = (java.sql.Time)row.get("B_TIME");
    java.util.Date utilDateOfBirth = new java.util.Date(
        sqlDateOfBirth.getYear(),
        sqlDateOfBirth.getMonth(),
        sqlDateOfBirth.getDate(),
        timeOfBirth.getHours(),
        timeOfBirth.getMinutes(),
        timeOfBirth.getSeconds());
    return utilDateOfBirth;
}

// Define a field transformation method to write to the database
public java.sql.Time getTimeOfBirth()
{
    return new java.sql.Time this.dateAndTimeOfBirth.getHours(),
       this. dateAndTimeOfBirth.getMinutes(),
```

```
      this.dateAndTimeOfBirth.getSeconds());
}

// Define a field transformation method to write to the database
public java.sql.Date getDateOfBirth()
{
    return new java.sql.DateOfBirth this.dateAndTimeOfBirth.getYear(),
      this.dateAndTimeOfBirth.getMonth(), this.dateAndTimeOfBirth.getDate());
}
```

Example 3–32 Creating a Transformation Mapping Using Indirection

```
// Create a new mapping and register it with the descriptor.
TransformationMapping transformation2 = new
transformation2.setAttributeName("designation");
transformation2.setGetMethodName ("getDesignationHolder");
transformation2.setSetMethodName ("setDesignationHolder");
transformation2.setAttributeTransformation ("getRankFromRow");
transformation2.addFieldTransformation("RANK", "getRankFromObject");
transformation2.useIndirection();
descriptor.addMapping(transformation2);

//Define an attribute transformation method to read from database row.
public String getRankFromRow()
{
    Integer value = new Integer(((Number)row.get("RANK)).intValue());
    String rank = null;
    if (value.intValue() == 1) {
        rank = "Executive";
    }
    if (value.intValue() == 2) {
        rank = "Non-Executive";
    }
    return rank;
}
//Define a field transformation method to write to the database.
public Integer getRankFromObject()
{
    Integer rank = null;

    if (getDesignation().equals("Executive")) rank = new Integer(1);
    if (getDesignation().equals("Non-Executive")) rank = new Integer(2);
    return rank;
}
```

```
//Provide accessor methods for the indirection.
private ValueHolderInterface designation;
public ValueHolderInterface getDesignationHolder()
{
    return designation;
}
public void setDesignationHolder(ValueHolderInterface value)
{
    designation = value;
}
```

For more information about the available methods for `TransformationMapping`, see the *Oracle Application Server TopLink API Reference*.

Serialized Object Mappings

Serialized object mappings are used to store large data objects, such as multimedia files and BLOBs, in the database. Serialization transforms these large objects as a stream of bits.

Serialized object mappings are instances of the `SerializedObjectMapping` class and require the following elements:

- The attribute mapped, set by sending the `setAttributeName( )` message
- The field that stores the value of the attribute, set by the `setFieldName( )` message

Use the optional `setGetMethodName( )` and `setSetMethodName( )` messages to access the attribute through user-defined methods, rather than directly. You do not have to define accessors when you use Java 2.

Example 3–33 Creating a Serialized Object Mapping and Registering It with the Descriptor

```
// Create a new mapping and register it with the descriptor.
SerializedObjectMapping serializedMapping = new SerializedObjectMapping();
serializedMapping.setAttributeName("jobDescription");
serializedMapping.setFieldName("JOB_DESC");
descriptor.addMapping(serializedMapping);
```

For more information about the available methods for `SerializedObjectMapping`, see the *Oracle Application Server TopLink API Reference*.

Variable One-to-One Mappings

Variable one-to-one mappings are instances of the `VariableOneToOneMapping()` class and require the following elements:

- The attribute mapped, set by sending the `setAttributeName( )` message
- The reference class, set by sending the `setReferenceClass( )` message
- The foreign key and target query key information, normally specified by sending the `setForeignQueryKeyName( )` message and passing the source foreign key field name and the target abstract query key name on the interface descriptor

---

Note: If the target implementor descriptors' primary keys are composite, send the `addForeignQueryKeyName( )` message for each of the foreign key fields, and target query keys that make up the relationship.

If the mapping uses a class indicator field:

- Specify a type indicator field.
- Specify the class indicator values on the mapping so that mapping can determine the class of object to create.

Note: Because Indirection is enabled by default, this requires the attribute be a `ValueHolderInterface`.

Example 3–34 Defining a Variable One-to-One Mapping Using a Class Indicator Field

```
VariableOneToOneMapping variableOneToOneMapping = new VariableOneToOneMapping();
variableOneToOneMapping.setAttributeName("contact");
variableOneToOneMapping.setReferenceClass (Contact.class);
variableOneToOneMapping.setForeignQueryKeyName ("C_ID", "id");
variableOneToOneMapping.setTypeFieldName("TYPE");
variableOneToOneMapping.addClassIndicator(Email.class, "Email");
variableOneToOneMapping.addClassIndicator(Phone.class, "Phone");
variableOneToOneMapping.dontUseIndirection();
variableOneToOneMapping.privateOwnedRelationship();
```

Example 3–35 Defining a Variable One-to-One Mapping Using a Primary Key

```
VariableOneToOneMapping variableOneToOneMapping = new VariableOneToOneMapping();
variableOneToOneMapping.setAttributeName("contact");
variableOneToOneMapping.setReferenceClass (Contact.class);
variableOneToOneMapping.setForeignQueryKeyName ("C_ID", "id");
variableOneToOneMapping.dontUseIndirection();
variableOneToOneMapping.privateOwnedRelationship();
```

For more information about the available methods for `VariableOneToOneMapping`, see the *Oracle Application Server TopLink API Reference*.

Object Relational Mappings

Relational mappings defines the reference between persistent objects. Object relational mappings enable you to persist an object model into an object-relational data model. The OracleAS TopLink Mapping Workbench does not directly support these mappings—you must define them in code through amendment methods.

OracleAS TopLink supports the following object-relational mappings:

- Array
- Object array
- Structure
- Reference
- Nested table

Array Mappings

In an object-relational data-model, structures can contain *arrays* (collections of other data types). These arrays can contain primitive data types or collections of other structures. OracleAS TopLink stores the arrays with their parent structure in the same table.

All elements in the array must be the same data type. The number of elements in an array controls the size of the array. An Oracle database allows arrays of variable sizes (called Varrays).

Oracle8*i* or higher offers two collection types:

- Varray – Used to represent a collection of primitive data or aggregate structures.

- Nested table – Similar to varrays except they store information in a separate table from the parent structure's table

OracleAS TopLink supports arrays of primitive data through the `ArrayMapping` class. This is similar to `DirectCollectionMapping`—it represents a collection of primitives in Java. However, the `ArrayMapping` class does not require an additional table to store the values in the collection.

OracleAS TopLink supports arrays of aggregate structures through the `ObjectArrayMapping` class.

OracleAS TopLink supports nested tables through the `NestedTableMapping` class.

Implementing Array Mappings in Java Array mappings are instances of the `ArrayMapping` class and require the following elements:

- The attribute mapped, set by sending the `setAttributeName( )` message
- The field mapped, set by sending the `setFieldName( )` message
- The name of the array, set by sending the `setStructureName( )` message

Example 3–36 Creating an Array Mapping for the Employee Source Class and Registering It with the Descriptor

```
// Create a new mapping and register it with the source descriptor.
ArrayMapping arrayMapping = new ArrayMapping();
arrayMapping.setAttributeName("responsibilities");
arrayMapping.setStructureName("Responsibilities_t");
arrayMapping.setFieldName("RESPONSIBILITIES");
descriptor.addMapping(arrayMapping);
```

In addition to the API Example 3–36 illustrates, other common API for use with implement array mapping include:

- `setReferenceClass(Class referenceClass)`: to set the parent class
- `setGetMethodName(String name)` and `setSetMethodName(String name)`: to provide method access

For more information about the available methods for `ArrayMapping`, see the *Oracle Application Server TopLink API Reference*.

Object Array Mappings

In an object-relational data-model, object arrays allow for an array of object types or structures to be embedded into a single column in a database table or an object table.

OracleAS TopLink supports object array mappings to define a collection-aggregated relationship in which the target objects share the same row as the source object.

Implementing Object Array Mappings in Java Object array mappings are instances of the `ObjectArrayMapping` class. You must associate this mapping to an attribute in the parent class. Object array mappings require the following elements:

- The attribute mapped, set by sending the `setAttributeName( )` message
- The field mapped, set by sending the `setFieldName( )` message
- The name of the array, set by sending the `setStructureName( )` message

Use the optional `setGetMethodName( )` and `setSetMethodName( )` messages to access the attribute through user defined methods, rather than directly.

Example 3–37 Creating an Object Array Mapping for the Insurance Source Class and Registering It with the Descriptor

```
// Create a new mapping and register it with the source descriptor.
ObjectArrayMapping phonesMapping = new ObjectArrayMapping();
phonesMapping.setAttributeName("phones");
phonesMapping.setGetMethodName("getPhones");
phonesMapping.setSetMethodName("setPhones");
phonesMapping.setStructureName("PHONELIST_TYPE");
phonesMapping.setReferenceClass(Phone.class);
phonesMapping.setFieldName("PHONES");
descriptor.addMapping(phonesMapping);
```

For more information about the available methods for `ObjectArrayMapping`, see the *Oracle Application Server TopLink API Reference*.

Structure Mappings

In an object-relational data-model, structures are user defined data types or object-types. This is similar to a Java class—it defines attributes or fields in which each attribute is either:

- A primitive data type
- Another structure

- Reference to another structure

OracleAS TopLink maps each structure to a Java class defined in your object model and defines a descriptor for each class. A `StructureMapping` maps nested structures, similar to an `AggregateObjectMapping`. However, the structure mapping supports null values and shared aggregates without requiring additional settings (because of the object-relational support of the database).

Implementing Structure Mappings in Java Structure mappings are instances of the `StructureMapping` class. You must associate this mapping to an attribute in each of the parent classes. Structure mappings require the following elements:

- The attribute mapped, set by sending the `setAttributeName( )` message
- The field mapped, set by sending the `setFieldName( )` message
- The target (child) class, set by sending the `setReferenceClass( )` message

Use the optional `setGetMethodName( )` and `setSetMethodName( )` messageto access the attribute through user-defined methods, rather than directly.

Make the following changes to the target (child) class descriptor:

- Send the `descriptorIsAggregate( )` message to indicate it is not a root level
- Remove table or primary key information

Example 3–38 Creating a Structure Mapping for the Employee Source Class and Registering It with the Descriptor

```
// Create a new mapping and register it with the source descriptor.
StructureMapping structureMapping = new StructureMapping();
structureMapping.setAttributeName("address");
structureMapping.setReferenceClass(Address.class);
structureMapping.setFieldName("address");
descriptor.addMapping(structureMapping);
```

Example 3–39 Creating the Descriptor of the Address Aggregate Target Class

The aggregate target descriptor does not need a mapping to its parent, or any table or primary key information.

```
// Create a descriptor for the aggregate class. The table name and primary key
are not specified in the aggregate descriptor.
ObjectRelationalDescriptor descriptor = new ObjectRelationalDescriptor ();
descriptor.setJavaClass(Address.class);
descriptor.setStructureName("ADDRESS_T");
descriptor.descriptorIsAggregate();
```

```
// Define the field ordering
descriptor.addFieldOrdering("STREET");
descriptor.addFieldOrdering("CITY");
...

// Define the attribute mappings or relationship mappings.
...
```

In addition to the API Example 3–39 illustrates, other common API for use with structure mapping include:

- `readWrite()`
- `readOnly()`
- `setIsReadOnly(boolean readOnly)`

For more information about the available methods for `StructureMapping`, see the *Oracle Application Server TopLink API Reference*.

Reference Mappings

In an object-relational data-model, structures reference each other through *refs*—not through foreign keys (as in a traditional data model). *Refs* are based on the target structure's `ObjectID`.

OracleAS TopLink supports refs through the `ReferenceMapping`. They represent an object reference in Java, similar to a `OneToOneMapping`. However, the reference mapping does not require foreign key information.

Implementing Reference Mappings in Java Reference mappings are instances of the `ReferenceMapping` class. You must associate this mapping to an attribute in the source class. Reference mappings require the following elements:

- The attribute mapped, set by sending the `setAttributeName( )` message
- The field mapped, set by sending the `setFieldName( )` message
- The target class, set by sending the `setReferenceClass ( )` message

Use the optional `setGetMethodName( )` and `setSetMethodName( )` messages to access the attribute through user-defined methods, rather than directly.

Example 3–40 Creating a Reference Mapping for the Employee Source Class and Registering It with the Descriptor

```
// Create a new mapping and register it with the source descriptor.
ReferenceMapping referenceMapping = new ReferenceMapping();
referenceMapping.setAttributeName("manager");
referenceMapping.setReferenceClass(Employee.class);
referenceMapping.setFieldName("MANAGER");
descriptor.addMapping(refrenceMapping);
```

In addition to the API Example 3–40 illustrates, other common API for use with reference mappings include:

- `useBasicIndirection()`: implements OracleAS TopLink valueholder indirection
- `dontUseIndirection()`
- `readWrite()`
- `readOnly()`
- `setIsReadOnly(boolean readOnly)`

For more information about the available methods for `ReferenceMapping`, see the *Oracle Application Server TopLink API Reference*.

Nested Table Mappings

Nested table types model an unordered set of elements. These elements may be built-in or user-defined types. You can view a nested table as a single-column table or, if the nested table is an object type, as a multi-column table (with a column for each attribute of the object type).

Nested tables represent a one-to-many or many-to-many relationship of references to another independent structure. They support querying and joining better than `Varrays` that are inlined to the parent table.

OracleAS TopLink supports nested table through the `NestedTableMapping`. They represent a collection of object references in Java, similar to a `OneToManyMapping` or `ManyToManyMapping`. However, the nested table mapping does not require foreign key information (such as a one-to-many mapping) or the relational table (such as a many-to-many mapping).

Implementing Nested Table Mappings in Java Nested table mappings are instances of the `NestedTableMapping` class. This mapping is associated to an attribute in the parent class. Nested table mapping require the following elements:

- The attribute mapped, set by sending the `setAttributeName( )` message
- The field mapped, set by sending the `setFieldName( )` message
- The name of the array structure, set by sending the `setStructureName( )` message

Use the optional `setGetMethodName( )` and `setSetMethodName( )` messages to allow OracleAS TopLink to access the attribute through user-defined methods, rather than directly.

Example 3–41 Creating a Nested Table Mapping for the Insurance Source Class and Registering It with the Descriptor

```
// Create a new mapping and register it with the source descriptor.
NestedTableMapping policiesMapping = new NestedTableMapping();
policiesMapping.setAttributeName("policies");
policiesMapping.setGetMethodName("getPolicies");
policiesMapping.setSetMethodName("setPolicies");
policiesMapping.setReferenceClass(Policy.class);
policiesMapping.dontUseIndirection();
policiesMapping.setStructureName("POLICIES_TYPE");
policiesMapping.setFieldName("POLICIES");
policiesMapping.privateOwnedRelationship();
policiesMapping.setSelectionSQLString("select p.* from policyHolders ph,
  table(ph.policies) t, policies p where ph.ssn=#SSN and ref(p) = value(t)");
descriptor.addMapping(policiesMapping);
```

In addition to the API Example 3–41 illustrates, other common API for use with nested table mappings include:

- `useBasicIndirection()`: implements OracleAS TopLink valueholder indirection
- `dontUseIndirection()`
- `setUsesIndirection(boolean usesIndirection)`
- `independentRelationship()`
- `privateOwnedRelationship()`
- `setIsPrivateOwned(Boolean isPrivateOwned)`

For more information about the available methods for `NestedTableMapping`, see the *Oracle Application Server TopLink API Reference*.

Direct Map Mappings

Direct map mappings store instances that implement `java.util.Map`. Unlike one-to-many or many-to-many mappings, the keys and values of the map in this type of mapping are Java objects that do not have descriptors. The object type stored in the key and the value of direct map are Java primitive wrapper types such as String objects.

Support for primitive data types such as `int` is not provided because Java maps only hold objects.

Direct map mappings are instances of the `DirectMapMapping` class and require the following elements:

- The attribute mapped, set by sending the `setAttributeName( )` message
- The database table that holds the keys and values to be stored in the map, set by sending the `setReferenceTableName( )` message
- The field in the reference table from which the keys are read and placed into the map; this is called the direct key field and is set by sending the `setDirectKeyFieldName( )` message
- The foreign key information, which you specify by sending the `setReferenceKeyFieldName( )` message and passing the name of the field that is a foreign reference to the primary key of the source object

> **Note:** If the target primary key is composite, send the `addReferenceKeyFieldName( )` message for each of the fields that make up the key.

- The field in the reference table from which the values are read and placed into the map; this is called the direct field and is set by sending the `setDirectFieldName( )` message
- The Java type of key in map from which the keys are converted from keys that are read from the database placed into the map; this is set by sending the `setKeyClass( )` message
- The Java type of value in map from which the values are converted from values that are read from the database placed into the map; this is set by sending the `setValueClass( )` message

Example 3–42 Creating a Simple Direct Map Mapping

```
DirectMapMapping directMapMapping = new DirectMapMapping();
directMapMapping.setAttributeName("cities");
directMapMapping.setReferenceTableName("CITY_TEMP");
directMapMapping.setReferenceKeyFieldName("RECORD_ID");
directMapMapping.setDirectKeyFieldName("CITY");
directMapMapping.setDirectFieldName("TEMPERATURE");
directMapMapping.setKeyClass(String.class);
directMapMapping.setValueClass(Integer.class);

descriptor.addMapping(directMapMapping);
```

In addition to the API Example 3–42 illustrates, other common API for use with direct map mappings include:

- `useBasicIndirection()`: implements OracleAS TopLink valueholder indirection
- `useTransparentCollection()`: if you use transparent indirection, this element places a special collection in the source object's attribute
- `dontUseIndirection()`: implements no indirection

For more information about the available methods for `DirectMapMapping`, see the *Oracle Application Server TopLink API Reference*.

Customizing the Project

OracleAS TopLink projects, descriptors, and mapping are normally created using the OracleAS TopLink Mapping Workbench. The output of the OracleAS TopLink Mapping Workbench is an XML file that contains the mapping information required to store persistent objects in the database.

The OracleAS TopLink Mapping Workbench does not offer access to all the customization available to the OracleAS TopLink descriptors that make up the project. In these situations, to customize the mapping information, you can specify an *amendment method* to be run at deployment time. Each OracleAS TopLink descriptor can have an amendment method.

This section describes some of the available customization topics and techniques, including:

- Customizing OracleAS TopLink Descriptors with Amendment Methods
- Using After Load Methods

- Descriptor Events
- Descriptor Copy Policy
- Descriptor Query Manager
- Instantiation Policy
- Setting the Wrapper Policy Using Java Code
- Creating EJB Projects and OracleAS TopLink Descriptors in Java

Customizing OracleAS TopLink Descriptors with Amendment Methods

Amendment methods are static methods that run at deployment time and enable you to implement descriptor customization code. You can modify the OracleAS TopLink descriptor of any persistent class with an amendment method when you first instantiate the descriptor. For container-managed persistence, this happens when the entity beans are deployed into the EJB server.

For more information about amendment methods, see the *Oracle Application Server TopLink Mapping Workbench User's Guide*.

Using After Load Methods

Some OracleAS TopLink features cannot be configured from the OracleAS TopLink Mapping Workbench. To use these features, amend the descriptor after it is loaded as part of the project. *After load* methods are a type of amendment method that enables you to modify descriptors in code after you create the project object (either from an XML project or a project class).

To access descriptors from the project object or the session object (after the session object is created from the project), write a Java method that takes the name of the descriptor as a single parameter. You can then send messages to the descriptor or any of its specific mappings to configure advanced features. Make all descriptor changes before the session logs in. Any descriptor change made after login is ignored.

For more information, see "Amending Descriptors After Loading in *Oracle Application Server TopLink Mapping Workbench User's Guide*.

Use any of the following APIs to implement after load methods:

- `project.getDescriptors();`
- `session.getDescriptors();`

- `session.getDescriptor(Class domainClass);`

For more information about these APIs, see the *Oracle Application Server TopLink API Reference*.

Descriptor Events

The descriptor event manager enables you to create events that trigger other events in your application. You use the Event Manager to invoke specific events when OracleAS TopLink reads, updates, deletes, or inserts objects on the database.

Descriptor events enable you to:

- Synchronize persistent objects with other systems, services, and frameworks.
- Maintain non persistent attributes of which OracleAS TopLink is not aware.
- Notify other objects in the application when the persistent state of an object changes.
- Implement complex mappings or optimizations not directly supported by OracleAS TopLink mappings.

You specify descriptor events in the OracleAS TopLink Mapping Workbench.

For more information, see "Specifying Events" in the *Oracle Application Server TopLink Mapping Workbench User's Guide*.

Receiving Descriptor Events

Applications receive descriptor events in several ways:

Implement the Descriptor Event Listener Interface Register objects that implement the `DescriptorEventListener` interface with the descriptor event manager. The descriptor event manager then notifies the object when any event occurs for that descriptor.

Subclass the Descriptor Event Adapter Class Use the `DescriptorEventAdapter` class if your application does not require all the methods defined in the `DescriptorEventListener` interface. The `DescriptorEventAdapter` implements the `DescriptorEventListener` interface and defines an empty method for each method in the interface. To use the adapter, subclass it and then register your new object with the descriptor event manager.

Register an Event Method with a Descriptor Register a public method as an event method. The descriptor then calls the event method when a database operation occurs. The event method must:

- Be public so that OracleAS TopLink can call it
- Return void
- Take a `DescriptorEvent` as a parameter

Registering Descriptor Event Listeners If you want an object other than the domain object to handle these events, register it as a *listener* with the descriptor event manager. If you want a `LockManager` to receive events for all `Employees`, then modify your descriptor amendment to register the `LockManager` as the listener.

Any object you register as a listener must implement the `DescriptorEventListener` interface. The amendment method appears in Example 3–43.

Example 3–43 Registering a Descriptor Event Listeners

```
public static void addToDescriptor(Descriptor descriptor)
{
    descriptor.getEventManager().addListener(LockManager.activeManager());
}
```

Reference Table 3–5 summarizes the most common public methods for `DescriptorEventManager`. For more information about the available methods for `DescriptorEventManager`, see the *Oracle Application Server TopLink API Reference*.

Table 3–5 Elements for the Descriptor Event Manager

Element	Default	Method Name

Events selectors (Defaults specified in listener interface implementation)	All events take `DescriptorEvent`:
	`postBuild`
	`postRefresh`
	`preWrite`
	`postWrite`
	`preDelete`
	`postDelete`
	`preInsert`
	`postInsert`
	`preUpdate`
	`postUpdate`
	`aboutToInsert`
	`aboutToUpdate`
	`postClone`
	`postMerge`
		All events take **`(String methodName)`**:
		`setPostBuildSelector`
		`setPostRefreshSelector`
		`setPreWriteSelector`
		`setPostWriteSelector`
		`setPreDeleteSelector`
		`setPostDeleteSelector`
		`setPreInsertSelector`
		`setPostInsertSelector`
		`setPreUpdateSelector`
		`setPostUpdateSelector`
		`setAboutToInsertSelector`
		`setAboutToUpdateSelector`
		`setPostCloneSelector`
		`setPostMergeSelector`

Listener registration Descriptor-Event reference (available methods on Descriptor-Event)	Source object if it implements the listener interface only; `aboutToInsert/` Update, / Build only; postMerge / Clone / write events within a Unit of Work	`addListener` **`(DescriptorEventListener listener)`** `getSource`**`()`** `getSession`**`()`** `getQuery`**`()`** `getDescriptor`**`()`** `getRow`**`()`** `getOriginalObject`**`()`**

Supported Events

The `DescriptorEventManager` supports several methods, including those in Table 3–6.

Table 3–6 Supported Events

Triggering Method Type	Supported Events	Description
Post-X Method	Post-Build	Occurs after an object is built from the database.
Post-X Method	Post-Clone	Occurs after an object has been cloned into a Unit of Work.
Post-X Method	Post-Merge	Occurs after an object has been merged from a Unit of Work.
Post-X Method	Post-Refresh	Occurs after an object is refreshed from the database.
Updating Method	Pre-Update	Occurs before an object is updated in the database. This may be called in a Unit of Work even if the object has no changes and does not require an update.
Updating Method	About-to-Update	Occurs when an object's row is updated in the database. This method is called only if the object has changes in the Unit of Work.
Updating Method	Post-Update	Occurs after an object is updated in the database. This may be called in a Unit of Work even if the object has no changes and does not require an update.
Inserting Method	Pre-Insert	Occurs before an object is inserted in the database.
Inserting Method	About-to-Insert	Occurs when an object's row is inserted in the database.
Inserting Method	Post-Insert	Occurs after an object is inserted into the database.
Writing Method	Pre-Write	Occurs before an object is inserted or updated into the database. This occurs before Pre-Insert/Update.
Writing Method	Post-Write	Occurs after an object is inserted or updated into the database. This occurs after Pre-Insert/Update.

Table 3–6 Supported Events (Cont.)

Triggering Method Type	Supported Events	Description
Deleting Method	Pre-Delete	Occurs before an object is deleted from the database.
Deleting Method	Post-Delete	Occurs after an object is deleted from the database.

Descriptor Copy Policy

The OracleAS TopLink Unit of Work feature uses copies of object (*clones*) rather than the original objects to perform its tasks. You can construct clones as follows:

- The Unit of Work calls the object default constructor to create a copy. This is the default method to create a clone.
- You specify a method on the object, and the Unit of Work calls this method to generate the clone. For example, add the following method to the descriptor:

  ```
  descriptor.createCopyPolicy("clone");
  ```

  When the Unit of Work requires a clone of this object, it calls the `clone()` method to create the copy.
- You specify the method by adding the following code to the descriptor:

  ```
  useCloneCopyPolicy(String)
  ```

  The `String` in this method is the name of another method that clones the object.

The most common way to use any policy other than the default (using the object default constructor) is to create an amendment method and specify it in the OracleAS TopLink Mapping Workbench when you configure the class.

For more information about amendment methods, see "Customizing OracleAS TopLink Descriptors with Amendment Methods" on page 3-82.

For more information about implementing descriptor copy policy in code, see "Setting the Copy Policy in Java" on page 3-101.

Descriptor Query Manager

You can add queries to a descriptor (*named queries*) for execution later in the application. For example, you can add the following code to a descriptor:

```
ReadObjectQuery aQuery = new ReadObjectQuery(Employee.class);
descriptor.getQueryManager().addQuery("readAnEmployee", aQuery);
```

You can accomplish this with an amendment method.

For more information about amendment methods, see "Customizing OracleAS TopLink Descriptors with Amendment Methods" on page 3-82.

Replacing Descriptor Queries

You can replace all queries in an OracleAS TopLink descriptor with user defined queries. Doing this enables you to change query behavior or to substitute stored procedures for the queries.

Example 3–44 Substituting a Stored Procedure for a Query

This example illustrates how to force the read object descriptor call to use a stored procedure.

```
ReadObjectQuery query = new ReadObjectQuery();
StoredProcedureCall call = new StoredProcedureCall();
call.setProcedureName("READ_RDM_EMP");
query.setCall(call);
descriptor.getQueryManager().setReadObjectQuery
(query);
```

Instantiation Policy

Overriding the Instantiation Policy Using Java Code

The `Descriptor` class provides the following methods to specify how objects get instantiated:

- `useDefaultConstructorInstantiationPolicy()`: Instructs OracleAS TopLink to use the default constructor to create new instances of objects built from the database. This method can be private, protected, or default/package.
- `useFactoryInstantiationPolicy(Object, String)`: Instructs OracleAS TopLink to send the message specified by the `String` parameter to an object factory specified by the `Object` parameter to create objects from the

database. The object factory method can be public, private, protected, or default/package and requires no arguments.

- `useMethodInstantiationPolicy(String)`: Instructs OracleAS TopLink to send the message contained in the string parameter to create objects that are populated with data from the database. This method can be a public, static method on the descriptor class, or it can be private, protected, or default/package. It must return a new instance of the class.
- `useFactoryInstantiationPolicy(Class factoryClass, String methodName)`: Instructs OracleAS TopLink to send the message contained in the `String` parameter to an instance of the specified `factoryClass`. This method must be return a new instance of the descriptor class. To instantiate the factory, OracleAS TopLink invokes the default constructor of the specified `factoryClass`. Both the `factoryClass` default constructor and the method invoked on the factory can be private, protected, or default/package.
- `useFactoryInstantiationPolicy(Class factoryClass, String methodName, String factoryMethodName)`: Instructs OracleAS TopLink to send the message contained in the first `String` parameter, `methodName`, to an instance of the specified `factoryClass`. This method must return a new instance of the descriptor class. To instantiate the factory, OracleAS TopLink invokes the second `String`, `methodName` on the specified `factoryClass`. This method must be a static method on the `factoryClass` and must return an instance of the `factoryClass`. The factory class static factory method and the method invoked on the factory can be private, protected, or default/package.

Setting the Wrapper Policy Using Java Code

The `Descriptor` class provides methods used in conjunction with the wrapper policy:

- `setWrapperPolicy(oracle.toplink.descriptors.WrapperPolicy)`: can be invoked to provide a wrapper policy for the descriptor
- `getWrapperPolicy()`: returns the wrapper policy for a descriptor

Creating EJB Projects and OracleAS TopLink Descriptors in Java

Create mappings and OracleAS TopLink descriptors to access features that are not available in the OracleAS TopLink Mapping Workbench.

To define a project using Java code:

1. Implement a project class that extends the `oracle.toplink.sessions.Project` class.
2. Compile the project class.
3. Edit the `toplink-ejb-jar.xml` deployment descriptor so that the value for the `project-class` element is the fully-qualified Project class name.

   For more information about creating project classes, see the *Oracle Application Server TopLink Mapping Workbench User's Guide*.

> **Note:** Use the OracleAS TopLink Mapping Workbench to create a Java Project class from an existing project. This provides a starting point for a custom project class. For more information, see the *Oracle Application Server TopLink Mapping Workbench User's Guide*.
>
> You can also use the OracleAS TopLink Mapping Workbench **Export Project to Java Source...** menu command to create a starting point for coding the project class manually.

Example 3–45 illustrates how you can specify OracleAS TopLink projects in code.

Example 3–45 Specifying an OracleAS TopLink Project in Code

```
/**
* The class EmployeeProject is an example of an OracleAS TopLink project defined
in Java code. The individual parts of the project - the Login and the
descriptors, are built inside of methods that are called by the constructor.
Note that EmployeeProject extends the class oracle.toplink.sessions.Project.
*/
public class EmployeeProject extends oracle.toplink.sessions.Project{

/**
* Supply a zero argument constructor that initializes all aspects of the
project. Make sure that the login and all the descriptors are initialized and
added to the project.
*/
public EmployeeProject(){
applyPROJECT();
applyLOGIN();
buildAddressDescriptor();
buildEmployeeDescriptor();
// other methods to build all descriptors for the project
/**
* Project-level properties, such as the name of the project, should be specified
here.
*/
protected void applyPROJECT(){
setName("Employee");
}
protected void applyLOGIN()
{
    oracle.toplink.sessions.DatabaseLogin login =
      new oracle.toplink.sessions.DatabaseLogin();

    // use platform appropriate for underlying database
    login.setPlatformClassName( "oracle.toplink.internal.databaseaccess.
      OraclePlatform");

    // if no sequencing is used, setLogin() will suffice
    setLoginAndApplySequenceProperties(login);
}

/**
* Descriptors are built by defining table info, setting properties (caching,
etc.) and by adding mappings to the descriptor.
*/
```

```
protected void buildEmployeeDescriptor() {
    oracle.toplink.publicinterface.Descriptor descriptor =
      new oracle.toplink.publicinterface.Descriptor();
}

// SECTION: DESCRIPTOR
// specify the class to be made persistent
descriptor.setJavaClass(examples.ejb.cmp11.advanced.EmployeeBean.class);

// specify the tables to be used and primary key
Vector tables = new Vector();
tables.addElement("EJB_EMPLOYEE");
descriptor.setTableNames(tables);
descriptor.addPrimaryKeyFieldName("EJB_EMPLOYEE.EMP_ID");

// SECTION: PROPERTIES
descriptor.setIdentityMapClass(
oracle.toplink.internal.identitymaps. FullIdentityMap.class);
descriptor.setExistenceChecking("Check cache");
descriptor.setIdentityMapSize(100);

// SECTION: COPY POLICY
descriptor.createCopyPolicy("constructor");

// SECTION: INSTANTIATION POLICY
descriptor.createInstantiationPolicy("constructor");

// SECTION: DIRECTTOFIELDMAPPING
oracle.toplink.mappings.DirectToFieldMapping firstNameMapping =
  new oracle.toplink.mappings .DirectToFieldMapping();
firstNameMapping.setAttributeName("firstName");
firstNameMapping.setIsReadOnly(false);
firstNameMapping.setFieldName("EJB_EMPLOYEE.F_NAME");
descriptor.addMapping(firstNameMapping);

// … Additional mappings are added to the descriptor using the addMapping()
method.
},}
```

To deploy the OracleAS TopLink project, specify the project class name in the `project-class` element in the `toplink-ejb-jar.xml` file for your entity beans.

For example:

```
<session>
    <name>EmployeeDemo</name>
```

```
    <project-class>oracle.toplink.demos.ejb.cmp.wls.employee.EmployeeProject
    </project-class>
    <login>
        <connection-pool>ejbPool</connection-pool>
    </login>
</session>
```

Writing Mappings in Code

In most cases, the OracleAS TopLink Mapping Workbench is the preferred tool to create OracleAS TopLink elements however, OracleAS TopLink also supports building components of your application in Java code. You can code components ranging in size from small elements to complete projects. This section illustrates the techniques required for building several of these components, and includes discussions on:

- Implementing Object-Relational Descriptors in Java
- Implementing Primary Keys in Java
- Implementing Inheritance in Java
- Implementing Indirection in Java
- Implementing Interfaces in Java
- Setting the Copy Policy in Java
- Implementing Multiple Tables in Java
- Implementing Sequence Numbers in Java
- Implementing Locking in Java

Implementing Object-Relational Descriptors in Java

Use the `ObjectRelationalDescriptor` class to define object-relational descriptors. This descriptor subclass contains the following additional properties:

- *Structure name*: Name of the object-type structure representing the class
- *Field ordering*: Field index of the object-type (required because object-type can be returned through JDBC as indexed arrays)

The OracleAS TopLink Remote (RMI) Example illustrates an object-relational data model and descriptors. For more information, see the OracleAS TopLink Examples at <*ORACLE_HOME*>`\toplink\doc\examples.htm`.

Example 3–46 Creating an Object-Relational Descriptor

```
import oracle.toplink.objectrelational.*;
ObjectRelationalDescriptor descriptor = new ObjectRelationalDescriptor()
descriptor.setJavaClass(Employee.class);
descriptor.setTableName("EMPLOYEES");
descriptor.setStructureName("EMPLOYEE_T");
descriptor.setPrimaryKeyFieldName("OBJECT_ID");

descriptor.addFieldOrdering("OBJECT_ID");
descriptor.addFieldOrdering("F_NAME");
descriptor.addFieldOrdering("L_NAME");
descriptor.addFieldOrdering("ADDRESS");
descriptor.addFieldOrdering("MANAGER");
descriptor.addDirectMapping("id", "OBJECT_ID");
descriptor.addDirectMapping("firstName", "F_NAME");
descriptor.addDirectMapping("lastName", "L_NAME");
//Refer to the mappings section for examples of object relational mappings.
...
```

Implementing Primary Keys in Java

If a single field constitutes the primary key, send the `setPrimaryKeyFieldName()` message to the descriptor. For a composite primary key, send the `addPrimaryKeyFieldName()` message for each field that makes up the primary key.

Alternatively, use the `setPrimaryKeyFieldNames( )` message that sends a `Vector` of the fields used as the primary key.

Example 3–47 Setting a Single-Field Primary Key in Java

```
// Define a new descriptor and set the primary key.
descriptor.setPrimaryKeyFieldName("ADDRESS_ID");
```

Example 3–48 Setting a Composite Primary Key in Java

```
// Define a new descriptor and set the primary key.
descriptor1.addPrimaryKeyFieldName("PHONE_NUMBER");
descriptor1.addPrimaryKeyFieldName("AREA_CODE");
```

Implementing Inheritance in Java

Although you can implement inheritance hierarchy in Java, under most circumstances, we recommend you use the OracleAS TopLink Mapping Workbench.

To implement an inheritance hierarchy completely in Java, modify the descriptors for the superclass and its subclasses. The inheritance implementation for a descriptor is encapsulated in an `InheritancePolicy` object, which is accessed by sending `getInheritancePolicy()` to the descriptor:

- Unless you use a class extraction method, send the `setClassIndicatorFieldName()` message to the `InheritancePolicy` of the root class. The parameter is a string that indicates the table column that holds the subclass type information.
- In the root class, define the values written to the database indicate the class type. Do this by:
  - Sending the `addClassIndicator()` message for each of the instantiable subclasses in the hierarchy. This message requires two parameters—the indicator value and the subclass it represents.
  - Sending the `useClassNameAsIndicator()` message. This stores the full name of the class in the class indicator field.
- Send the `setParentClass()` message to the descriptor for each subclass.
- Configure a root or branch class to return only instances of itself, by calling the `dontReadSubclassesOnQueries()` method.

Note: Descriptors that inherit table names from a parent are not sent the `setTableName()` and `addTableName()` messages for the tables they inherit. Only the root class defines the primary key.

Queries for Inherited Superclasses and Multiple Tables

If a superclass is configured to read subclasses and its subclasses define additional tables, build multiple queries to obtain all the rows for all the subclasses. For best performance in this situation, create a view against which to execute the query using the `setReadAllSubclassesViewName()` method. The view must internally perform an outer join or union on all the subclass tables and return a single result set with all the data.

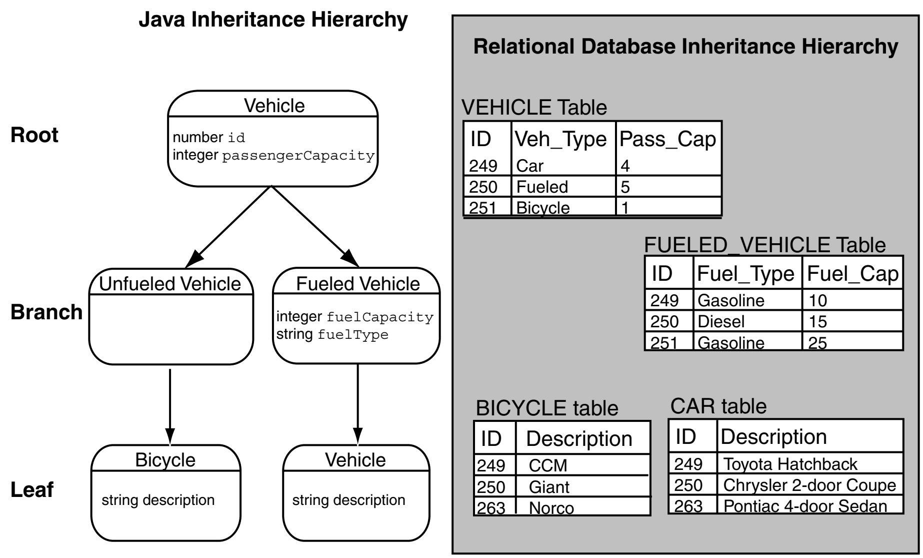
Customizing Inheritance

Occasionally, using the default OracleAS TopLink inheritance mechanism is not possible. For these cases, you can customize the inheritance mechanism. Instead of using a class indicator field and mapping, use a class extraction method. This method takes the object's row and returns the class to be used for that row. The `setClassExtractionMethodName()` method is used to accomplish this.

Queries for inherited classes usually also require filtering of the table rows. By default, OracleAS TopLink generates this from the class indicator information. If you provide the class extraction method, specify the filtering expressions. These can be set for concrete classes through `setOnlyInstancesExpression()` and for branch classes through `setWithAllSubclassesExpression()`.

Figure 3–10 illustrates an example of an inheritance hierarchy. The `Vehicle-Bicycle` branch demonstrates how you can store all subclass information in one table. The `FueledVehicle-Car` branch demonstrates how you can store subclass information in two tables.

Figure 3–10 Inheritance Hierarchy



The `Car` and `Bicycle` classes are leaf classes. Queries performed on them return instances of `Car` and `Bicycle` respectively.

`FueledVehicle` is a branch class. By default, branch classes are configured to read instances and subclass instances. Queries for `FueledVehicle` return instances of `FueledVehicle` and instances of `Car`.

`NonFueledVehicle` is a branch class and is configured to read subclasses. Because it does not have a class indicator defined in the root, it cannot be written to the database. Queries performed on `NonFueledVehicle` return instances of its subclasses.

`Vehicle` is a root class, which is configured to read instances of itself and instances of its subclass by default. Queries performed on the `Vehicle` class return instances of any of the concrete classes in the hierarchy.

Example 3–49 Implementing Descriptors for the Classes in the Inheritance Hierarchy

```
// Vehicle is a root class. Because it is the root class, it must add the class
indicators for its subclasses.
public static Descriptor descriptor()
{
Descriptor descriptor = new Descriptor();
descriptor.setJavaClass(Vehicle.class);
descriptor.setTableName("VEHICLE");
descriptor.setPrimaryKeyFieldName("ID");

// Class indicators must be supplied for each of the subclasses in the hierarchy
that can have instances.
InheritancePolicy policy = descriptor.getInheritancePolicy();
policy.setClassIndicatorFieldName("TYPE");
policy.addClassIndicator(FueledVehicle.class, "Fueled");
policy.addClassIndicator(Car.class, "Car");
policy.addClassIndicator(Bicycle.class, "Bicycle");

descriptor.addDirectMapping("id", "ID");
descriptor.addDirectMapping("passengerCapacity", "CAP");

return descriptor;
}

// FueledVehicle descriptor; it is a branch class and a subclass of Vehicle.
Queries made on this class will return instances of itself and instances of its
subclasses.
public static Descriptor descriptor()
{
Descriptor descriptor = new Descriptor();
descriptor.setJavaClass(FueledVehicle.class);
descriptor.addTableName("FUEL_VEH");
descriptor.getInheritancePolicy().setParentClass(Vehicle.class);
descriptor.addDirectMapping("fuelCapacity", "FUEL_CAP");
descriptor.addDirectMapping("fuelType", "FUEL_TYPE");
return descriptor;
}

// Car descriptor; it is a leaf class and subclass of FueledVehicle.
public static Descriptor descriptor()
{
Descriptor descriptor = new Descriptor();
descriptor.setJavaClass(Car.class);
descriptor.addTableName("CAR");
descriptor.getInheritancePolicy().setParentClass(FueledVehicle.class);
```

```
// Next define the attribute mappings.
descriptor.addDirectMapping("description", "DESCRIP");
descriptor.addDirectMapping("fuelType", "FUEL_VEH.FUEL_TYPE");
return descriptor;
}

// NonFueledVehicle descriptor; it is a branch class and a subclass of Vehicle.
Queries made on this class will return instances of its subclasses.
public static Descriptor descriptor()
{
Descriptor descriptor = new Descriptor();
descriptor.setJavaClass(NonFueledVehicle.class);
descriptor.getInheritancePolicy().setParentClass(Vehicle.class);
return descriptor;
}

// Bicycle descriptor; it is a leaf class and subclass of NonFueledVehicle.
public static Descriptor descriptor()
{
Descriptor descriptor = new Descriptor();
descriptor.setJavaClass(Bicycle.class);
descriptor.getInheritancePolicy().setParentClass(NonFueledVehicle.class);
descriptor.addDirectMapping("description", "BICY_DES");
return descriptor;
}

// FueledVehicle class; If a class extraction method is used, the following
needs to be added to specify that only the branch class itself needs to be
returned. This example is just specifying the class indicator field, which can
also be specified in the OracleAS TopLink Mapping Workbench in the Descriptor
Advanced Properties dialog.
public void addToDescriptor(Descriptor descriptor)
{
ExpressionBuilder builder = new ExpressionBuilder();
descriptor.getInheritancePolicy().setOnlyInstancesExpression(builder.getField
  ("VEHICLE.TYPE").equal("F"));
}
```

Reference Table 3–7 summarizes the most common public methods for `InheritancePolicy`. For more information about the available methods for `InheritancePolicy`, see the *Oracle Application Server TopLink API Reference*.

Table 3–7 Elements for the Inheritance Policy

Element	Default	Method Name
Class indicators	use indicator mapping	`setClassIndicatorFieldName`**`(String fieldName)`**
Parent classes	not applicable	`setParentClass`**`(Class parentClass)`**

Implementing Indirection in Java

To create indirection objects in code, the application must replace the relationship reference with a `ValueHolderInterface`. It must also call the `useIndirection()` method of the mapping if the mapping does not use indirection by default. Likewise, call the `dontUseIndirection()` method to disable indirection. `ValueHolderInterface` is defined in the `oracle.toplink.indirection`.

Example 3–50 A Mapping that Does Not Use Indirection

```
// Define the One-to-One mapping. Note that One-to-One mappings have indirection
enabled by default, so the "dontUseIndirection()" method must be called if
indirection is not used.
OneToOneMapping oneToOneMapping = new OneToOneMapping();
oneToOneMapping.setAttributeName("address");
oneToOneMapping.setReferenceClass(Address.class);
oneToOneMapping.setForeignKeyFieldName("ADDRESS_ID");
oneToOneMapping.dontUseIndirection();
oneToOneMapping.setSetMethodName("setAddress");
oneToOneMapping.setGetMethodName("getAddress");
descriptor.addMapping(oneToOneMapping);
```

The following code illustrates a mapping using indirection.

```
// Define the One-to-One mapping. One-to-One mappings have indirection enabled
by default, so the "useIndirection()" method is unnecessary if indirection is
used.
OneToOneMapping oneToOneMapping = new OneToOneMapping();
oneToOneMapping.setAttributeName("address");
oneToOneMapping.setReferenceClass(Address.class);
oneToOneMapping.setForeignKeyFieldName("ADDRESS_ID");
oneToOneMapping.setSetMethodName("setAddressHolder");
oneToOneMapping.setGetMethodName("getAddressHolder");
```

```
descriptor.addMapping(oneToOneMapping);
```

Implementing Interfaces in Java

Descriptors can own their parent interfaces. They can set multiple interfaces if they have implemented multiple interfaces. The query keys are defined in a normal way except that they must define the abstract query key from the interface descriptor in their descriptors. An abstract query key on the interface descriptor enables it to write expression queries on the interface.

Example 3–51 Using an Abstract Query Key on the Interface Descriptor

```
ExpressionBuilder contact = new ExpressionBuilder();
session.readObject(Contact.class, contact.get("id").equal(2));
```

Setting the Copy Policy in Java

The `Descriptor` class provides three methods that determine how an object is cloned:

- `useInstantiationCopyPolicy()`: the default method; OracleAS TopLink creates a new instance of the object using the technique indicated by the descriptor's instantiation policy. The default behavior is to use the default constructor. The new instance is then populated by using the descriptor's mappings to copy attributes from the original object.

> **Note:** `Descriptor.useInstantiationCopyPolicy()` replaces `Descriptor.useConstructorCopyPolicy()` available in previous versions of OracleAS TopLink. The old method is still supported, but it has been deprecated.

- `useCloneCopyPolicy()`: OracleAS TopLink calls the `clone()` method of the object; ensure that the clone method is written correctly and returns a logical shallow clone of the object
- `useCloneCopyPolicy(String)`: this method is called by passing in a string that contains the name of a method that clones the object; ensure that the method specified returns a logical shallow clone of the object

Implementing Multiple Tables in Java

To define a multiple table descriptor, call the `addTableName()` method for each table the descriptor maps to. If the descriptor inherits its primary table and is defining only a single additional one, then the descriptor is mapped normally to this table.

Primary Keys Match

Normally, the primary key is defined only for the primary table of the descriptor. The primary table is the first table specified through `addTableName()`. The primary key is not defined for the additional tables and is required to be the same as in the primary table. If the additional table's key is different, refer to the next example.

By default, all the fields in a mapping are presumed to be part of the primary table. If a mapping's field is for one of the additional tables, it must be fully qualified with the field's table name.

Example 3–52 Implementing a Multiple Table Descriptor In Which the Primary Keys Match

```
//Define a new descriptor that uses three tables.
Descriptor descriptor = new Descriptor();
descriptor.setJavaClass(Employee.class);
descriptor.addTableName("PERSONNEL"); // Primary table
descriptor.addTableName("EMPLOYMENT");
descriptor.addTableName("USERS");

descriptor.addPrimaryKeyFieldName("PER_NUMBER");
descriptor.addPrimaryKeyFieldName("DEP_NUMBER");

descriptor.addDirectMapping("id", "PER_NUMBER");
descriptor.addDirectMapping("firstName", "F_NAME");
descriptor.addDirectMapping("lastName", "L_NAME");

OneToOneMapping department = new OneToOneMapping();
department.setAttributeName("department");
department.setReferenceClass(Department.class);
department.setForeignKeyFieldName("DEP_NUMBER");
descriptor.addMapping(department);
// Mapping the primary key fields in the additional tables is not required
descriptor.addDirectMapping("salary", "EMPLOYMENT.SALARY");

AggregateObjectMapping period = new AggregateObjectMapping();
period.setAttributeName(period);
```

```
period.setReferenceClass(EmployementPeriod.class);
period.addFieldNameTranslation("EMPLOYMENT.S_DATE", "S_DATE");
period.addFieldNameTranslation("EMPLOYMENT.E_DATE", "E_DATE");
descriptor.addMapping(period);

descriptor.addDirectMapping("userName", "USERS.NAME");
descriptor.addDirectMapping("password", "USERS.PASSWORD");
```

Primary Keys are Named Differently

If the additional table's primary key is named differently, then call the descriptor method `addMultipleTablePrimaryKeyName()`, which provides:

- The field of the primary key from the primary table
- The additional table name
- The field in the additional table that the primary key maps to

Example 3–53 Implementing a Multiple Table Descriptor In Which the Additional Table Primary Keys are Named Differently

```
//Define a new descriptor that uses three tables.
Descriptor descriptor = new Descriptor();
descriptor.setJavaClass(Employee.class);
descriptor.addTableName("PERSONNEL");
// Primary table
descriptor.addTableName("EMPLOYMENT");
descriptor.addTableName("USERS");

descriptor.addPrimaryKeyFieldName("PER_NUMBER");
descriptor.addPrimaryKeyFieldName("DEP_NUMBER");

descriptor.addMultipleTablePrimaryKeyName("PERSONEL.PER_NUMBER",
  "USERS.PERSONEL_NO");
descriptor.addMultipleTablePrimaryKeyName("PERSONEL.DEP_NUMBER",
  "USERS.DEPARTMENT_NO");

// Assumed EMPLOYMENT uses same primary key
descriptor.addDirectMapping(id, PER_NUMBER);

OneToOneMapping department = new OneToOneMapping();
department.setAttributeName("department");
department.setReferenceClass(Department.class);
department.setForeignKeyFieldName("DEP_NUMBER");
descriptor.addMapping(department);
```

```
// Primary key does not have to be mapped for additional tables.
...
```

Tables Related by Foreign Key Relationships

For OracleAS TopLink to support read, insert, update, and delete operations on an object mapped to multiple tables:

- Specify the foreign key information on the descriptor.
- Specify the foreign keys and primary keys in the object.

The API is `addMultipleTableForeignKeyFieldName()`. This method builds the join expression and adjusts the table insertion order to respect the foreign key constraints.

Example 3–54 illustrates the setup of a descriptor for an object mapped to multiple tables in which the tables are related by a foreign key relationship from the primary table to the secondary table. The `addMultipleTableForeignKeyFieldName()` method is used to specify the direction of the foreign key relationship.

If the foreign key is in the secondary table and refers to the primary table, then the order of the arguments to `addMultipleTableForeignKeyFieldName()` is reversed.

Note: To allow read, insert, update, and delete operation to be performed on the Employee object, map the foreign key field in the primary table and the primary key in the secondary table.

Example 3–54 Implementing Multiple Tables In Which a Foreign Key from the Primary Table to the Secondary Table is Used to Join the Tables

```
Descriptor descriptor = new Descriptor();
descriptor.setJavaClass(Employee.class);
Vector vector = new Vector();
vector.addElement("EMPLOYEE");
vector.addElement(ADDRESS");
descriptor.setTableNames(vector);
descriptor.addPrimaryKeyFieldName("EMPLOYEE.EMP_ID");
// Map the foreign key field of the employee table and the primary key of the
address table.
descriptor.addDirectMapping("addressID", "EMPLOYEE.ADDR_ID");

// Setup the join from the address table to the country employee table to the
```

```
address table by specifying the FK info to the descriptor. Set the foreign key
info from the address table to the country table.
descriptor.addMultipleTableForeignKeyFieldName("EMPLOYEE.ADDR_ID",
  "ADDRESS.ADDR_ID");
```

Non Standard Table Relationships

Occasionally the join condition can be nonstandard. In this case, the descriptor's query manager can be used to provide a custom multiple table join expression. The `getQueryManager()` method is called on the descriptor to obtain its query manager, and the `setMultipleTableJoinExpression()` method is used to customize the join expression.

Simply specifying the join expression allows OracleAS TopLink to perform read operations for the object. Insert operations can also be supported if the table insertion order is specified and the primary key of the additional tables is mapped manually.

The insertion order is required to conform to foreign key constraints when inserting to the multiple tables. Specify the insert order using the descriptor method `setMultipleTableInsertOrder`().

Example 3–55 illustrates the use of the `setMultipleTableJoinExpression()` and `setMultipleTableInsertOrder()` methods. In addition, it illustrates the use of a custom join expression without specifying the table insert order.

Note: Using these methods does not support update or delete operations because of the lack of primary key information for the secondary table(s). If update and delete operations are required, perform them with custom SQL, or explicitly specify the foreign key information as explained in the previous section.

Example 3–55 Implementing Multiple Tables In Which You Specify a Join Expression and the Table Insert Order

Using this method allows only read and insert operations to be performed on Employee objects. Note that the primary key of the secondary table, and the foreign key of the primary table must be mapped and maintained by the application for insert operations to work.

```
Descriptor descriptor = new Descriptor();
descriptor.setJavaClass(Employee.class);
Vector vector = new Vector();
vector.addElement("EMPLOYEE");
```

```
vector.addElement(ADDRESS");
descriptor.setTableNames(vector);

// Specify the primary key information for each table.
descriptor.addPrimaryKeyFieldName("EMPLOYEE.EMP_ID");

// Map the foreign key field of the employee table and the primary key of the
address table.
descriptor.addDirectMapping("employee_addressID", "EMPLOYEE.ADDR_ID");
descriptor.addDirectMapping("address_addressID", "ADDRESS.ADDR_ID");
// Setup the join from the employee table to the address table using a custom
join expression and specifying the table insert order.
ExpressionBuilder builder = new ExpressionBuilder();
descriptor.getQueryManager().setMultipleTableJoinExpression(builder.getField
  ("EMPLOYEE.ADDR_ID").equal(builder.getField("ADDRESS.ADDR_ID")));
Vector tables = new Vector(2);
tables.addElement(new DatabaseTable("ADDRESS"));
tables.addElement(new DatabaseTable("EMPLOYEE"));
descriptor.setMultipleTableInsertOrder(tables);
...
```

Example 3–56 Mapping a Multiple Table Descriptor In Which a Custom Join Expression is Required

In this example, only read operations are supported.

```
//Define a new descriptor that uses three tables.
Descriptor descriptor = new Descriptor();
descriptor.setJavaClass(Employee.class);
descriptor.addTableName("PERSONNEL");
// Primary table
descriptor.addTableName("EMPLOYMENT");
descriptor.addPrimaryKeyFieldName("PER_NO");
descriptor.addPrimaryKeyFieldName("DEP_NO");

ExpressionBuilder builder = new ExpressionBuilder();
descriptor.getQueryManager().setMultipleTableJoinExpression((builder.getField
  ("PERSONEL.EMP_NO").equal(builder.getField("EMPLOYMENT.EMP_NO")));
descriptor.addDirectMapping("personelNumber", "PER_NO");

OneToOneMapping department = new OneToOneMapping();
department.setAttributeName("department");
department.setReferenceClass(Department.class);
department.setForeignKeyFieldName("DEP_NO");
descriptor.addMapping(department);
// The primary key field on the EMPLOYMENT does not have to be mapped.
```

```
...
```

Implementing Sequence Numbers in Java

To implement sequence numbers using Java code, send the `setSequenceNumberFieldName()` message to the descriptor to register the name of the database field that holds the sequence number. The `setSequenceNumberName()` method also holds the name of the sequence. This name can be one of the entries in the `SEQ_NAME` column or the name of the sequence object (if you are using Oracle native sequencing).

Notes:

- The sequence field must be in the first (primary) table if multiple tables are used.
- If you use Sybase, Microsoft SQL Server, or IBM Informix native sequencing, this implementation has no direct meaning but must still be set for compatibility reasons.

Implementing Locking in Java

Use the API to set optimistic locking completely in code. All the API is on the descriptor:

- `useVersionLocking(String)`: sets this descriptor to use version locking and increments the value in the specified field name for update or delete
- `useChangedFieldsLocking()`: tells this descriptor to compare only modified fields for an update or delete
- `useTimestampLocking(String)`: sets this descriptor to use timestamp locking and writes the current server time in the field every update or delete
- `useAllFieldsLocking()`: tells this descriptor to compare every field for an update or delete
- `useSelectedFieldsLocking(Vector)`: tells this descriptor to compare the field names specified in this vector of Strings for an update or delete

Example 3–57 Implementing Optimistic Locking Using the Version Field of Employee Table as the Version Number of the Optimistic Lock

```
// Set the field that control optimistic locking. No mappings are set for fields
which are version fields for optimistic locking.
```

```
descriptor.useVersionLocking("VERSION");
```

The code in Example 3–57 stores the optimistic locking value in the identity map. If the value must be stored in a nonread only mapping, then the code can be:

```
descriptor.useVersionLocking("VERSION", false);
```

The `false` indicates that the lock value is not stored in the cache, but is stored in the object.

Java Implementation of Optimistic Locking

Use the API to set optimistic locking in code. All the API is on the descriptor:

- `useVersionLocking(String)`: sets this descriptor to use version locking and increments the value in the specified field name for every update or delete
- `useTimestampLocking(String)`: sets this descriptor to use timestamp locking and writes the current server time in the specified field name for every update or delete
- `useChangedFieldsLocking()`: tells this descriptor to compare only modified fields for an update or delete
- `useAllFieldsLocking()`: tells this descriptor to compare every field for an update or delete
- `useSelectedFieldsLocking(Vector)`: tells this descriptor to compare the field names specified in this vector of Strings for an update or delete

Example 3–58 illustrates how to implement optimistic locking using the `VERSION` field of `EMPLOYEE` table as the version number of the optimistic lock.

Example 3–58 Implementing Optimistic Locking Example

```
descriptor.useVersionLocking("VERSION");
```

The code in Example 3–58 stores the optimistic locking value in the identity map. If the value must be stored in a nonread only mapping, then the code appears as follows:

```
descriptor.useVersionLocking("VERSION", false);
```

The `false` indicates that the lock value is not stored in the cache, but is stored in the object.

4 Sessions

Sessions are a key component of the Oracle Application Server TopLink application—they provide OracleAS TopLink with access to the database. Sessions enable you to execute queries, and they return persistent objects and other results for client applications. This chapter introduces OracleAS TopLink sessions, and describes:

Introduction to Session Concepts

A session represents the connection between an application and the relational database that stores its persistent objects. OracleAS TopLink provides different session classes, each optimized for different design requirements and data access strategies. OracleAS TopLink session types range from a simple database session that gives one user one connection to the database, to the session broker that provides access to several databases for multiple clients.

To understand the OracleAS TopLink session, you must be familiar with several session concepts.

sessions.xml File

In most cases, the developer pre configures sessions for the application in a session configuration file. This file, known as the `sessions.xml` file, is an Extensible Markup Language (XML) file that contains all sessions that are associated with the application. The `sessions.xml` file can contain any number of sessions and session types.

Session Types

Several session types each provide a particular set of functionality to the application.

Server Session

A server session is the most common OracleAS TopLink session type, because it supports the three-tier architectures that are common to enterprise applications. Server sessions manage the server side of client-server communications. They work together with the client session to provide complete client-server communication.

The server session provides shared resources to a multithreaded environment, including a shared cache and connection pools. The server session also provides transaction isolation.

For more information about the server session, see "Server Session and Client Session" on page 4-37.

Client Session

A client session is a client-side communications mechanism that works together with the server session to provide the client-server connection. Each client session serves one client.

For more information about the client session, see "Server Session and Client Session" on page 4-37.

Remote Session

A remote session offers database access to clients that do not reside on the OracleAS TopLink Java virtual machine (JVM). The remote session connects to a client session, which, in turn, connects to the server session.

For more information, see "Remote Session" on page 4-58.

Database Session

A database session is a unique session type because it provides both client and server communications. It is a relatively simple session type that supports only a single client and a single database connection. The database session is not scalable; however, if you have an application with a single client that requires only one database connection, the database session is usually your best choice.

For more information, see "Database Session" on page 4-48.

Session Broker

The OracleAS TopLink session broker is a mechanism that enables client applications to communicate with multiple databases. A session broker makes multiple database access transparent to the client.

For more information, see "Session Broker" on page 4-53.

Session Manager

When a client application requires a session, it requests the session from the OracleAS TopLink session manager. The two main functions of the session manager are to instantiate OracleAS TopLink sessions for the server, and to hold the sessions for the life of the application. The session manager instantiates database sessions, server sessions, or session brokers based on the configuration information in the `sessions.xml` file.

The session manager instantiates sessions as follows:

1. The client application request a session by name.
2. The session manager looks up the session name in the `sessions.xml` file. If the session name exists, the session manager instantiates the specified session; otherwise, it raises an exception.

3. After instantiation, the session remains viable until you shut down the application.

Connection Pool

A connection pool is a collection of reusable database connections. OracleAS TopLink manages these connections for the application, provides connections to processes as needed, and returns connections to the pool when the process is complete. When it is returned to the pool, the connection is available for other processes.

A properly configured connection pool significantly improves performance.

For more information about configuring connection pools, see "Working with Connection Pools" in the *Oracle Application Server TopLink Mapping Workbench User's Guide.*

Caching

OracleAS TopLink sessions provide an object cache. This cache, known as the *session cache*, retains information about objects that are read from or written to the database, and is a key element for improving the performance of an OracleAS TopLink application.

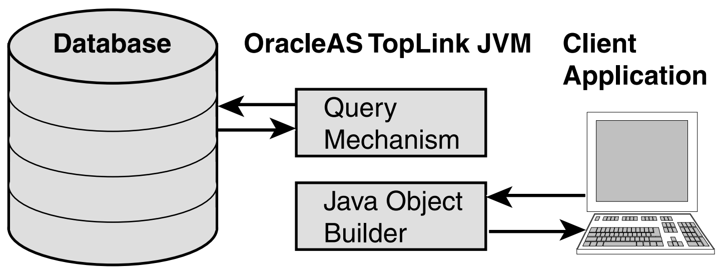
Profiling

OracleAS TopLink profiling enables you to identify performance bottlenecks in your application. When enabled, the profiler logs a summary of the performance statistics for every query that the application executes.

Session Architectures

A session in an OracleAS TopLink application includes a query mechanism that interacts with the database, and an object construction mechanism that builds objects from the data that is stored in the database. The data interaction and object construction components both reside on a JVM. A client application uses these mechanisms to query the database and retrieve objects.

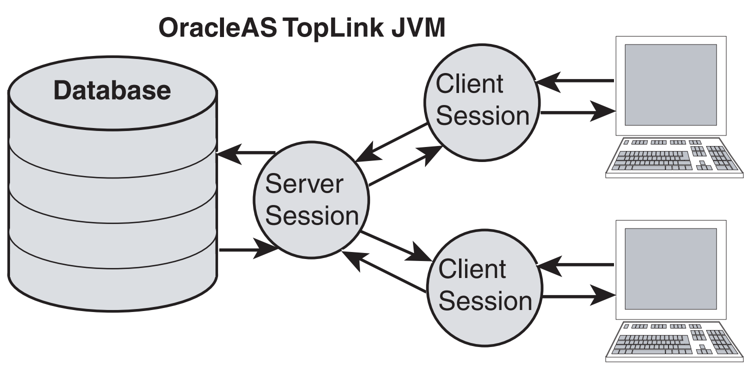
Figure 4–1 Simple OracleAS TopLink Session Architecture



Server Session

A server session provides a connection with the database, and makes the extracted data available to one or more client session (either client session or remote sessions). A server session usually appears as part of an OracleAS TopLink three-tier architecture. It uses a JDBC connection pool configured to provide a query mechanism to clients. Client applications communicate with the server session through a client session.

Figure 4–2 Typical OracleAS TopLink Server Session with Client Session Architecture



For more information about the server session, see "Server Session and Client Session" on page 4-37.

Client Session

A client session communicates with the server session on behalf of the client application (see Figure 4–2). A server session creates client sessions on request, and the client sessions share an object cache.

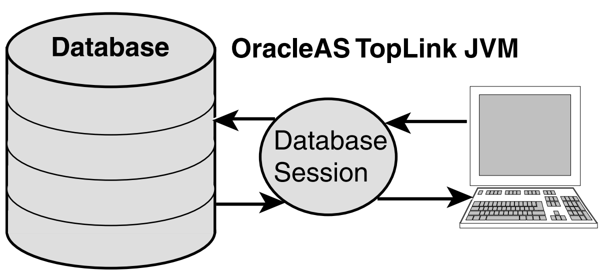
Together, the client session and server session provide a three-tier architecture that you can scale easily, by adding more client sessions. Because of this scaleability, we recommend you use the three-tier architecture to build your OracleAS TopLink applications.

For more information about the client session, see "Server Session and Client Session" on page 4-37.

Database Session

A database session provides a client application with a single JDBC database connection, for simple, standalone applications in which a single connection services all database requests for one user.

Figure 4–3 OracleAS TopLink Database Session Architecture

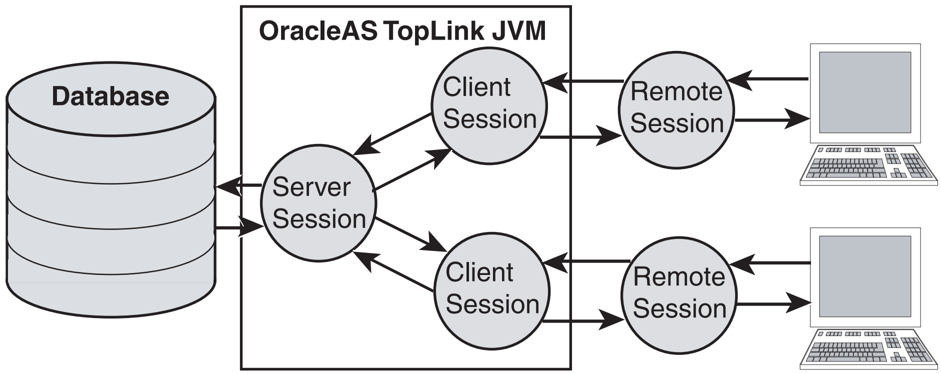


For more information about the database session, see "Database Session" on page 4-48.

Remote Session

A remote session is a client-side session that resides on the client rather than the OracleAS TopLink JVM. The remote session does not replace the client session; rather, a remote session requires a client session to communicate with the server session. A remote session can also communicate directly with a database session.

Figure 4–4 Typical OracleAS TopLink Server Session with Remote Session Architecture



The remote session provides a full OracleAS TopLink session, complete with a session cache, on the client system. OracleAS TopLink manages the remote session cache and enables client applications to execute operations on the OracleAS TopLink JVM.

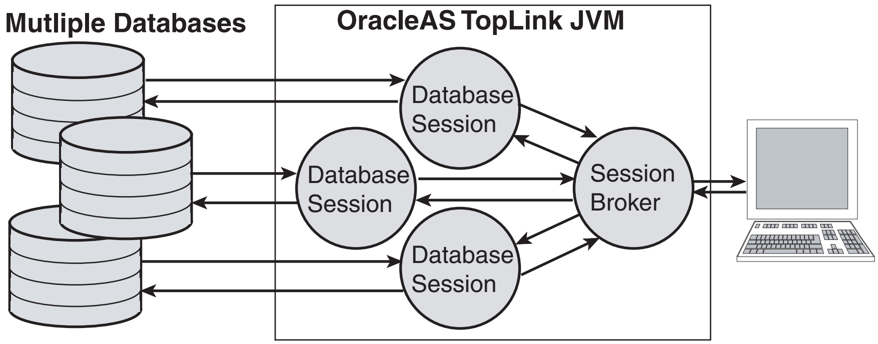
For more information about the remote session, see "Remote Session" on page 4-58.

Session Broker

The OracleAS TopLink session broker enables client applications to view several databases through a single session. If you store objects in your application on multiple databases, the session broker, which provides seamless communication for client applications, enables the client to view multiple databases as if they are a single database.

The session broker connects to the databases through either a database session or a server session.

Figure 4–5 OracleAS TopLink Session Broker with Server Session Architecture



For more information about the session broker, see "Session Broker" on page 4-53.

Configuring Sessions with the sessions.xml File

OracleAS TopLink provides two ways to preconfigure your sessions: you can export and compile Java source code from the OracleAS TopLink Mapping Workbench, or use the OracleAS TopLink Sessions Editor to build a session configuration file, the `sessions.xml` file. For the following reasons, we recommend you use the `sessions.xml` file to deploy an OracleAS TopLink application:

- It is easy to create and maintain in the OracleAS TopLink Sessions Editor.
- It is easy to troubleshoot.
- It provides access to most session configuration options.
- It offers excellent flexibility, including the ability to modify deployed applications.

This section describes the `sessions.xml` file and illustrates the options that are available when you build the file. This section discusses editing the file manually, but the simplest way to build the `sessions.xml` file is to use the OracleAS TopLink Sessions Editor in the OracleAS TopLink Mapping Workbench.

This section explains how to configure the `sessions.xml` file, and includes discussions on:

- Navigating the sessions.xml File
- XML Header
- toplink-configuration Element

- session Element
- session-broker Element
- JTA Configuration

For more information about creating configuration files in the OracleAS TopLink Mapping Workbench, see "Understanding the OracleAS TopLink Sessions Editor" in the *Oracle Application Server TopLink Mapping Workbench User's Guide.*

Navigating the sessions.xml File

The `sessions.xml` file's Document Type Definition (DTD) defines the file structure. If you use the OracleAS TopLink Sessions Editor, you need not concern yourself with that structure. However, if you do create or edit the file, you must understand its structure.

The main structure of the `sessions.xml` file is the `toplink-configuration` element. This element includes all session configuration options. Within the `toplink-configuration` element, you configure sessions and session brokers. The session broker contains only sessions defined in the `sessions.xml` file; the bulk of session configuration occurs within the `session` element.

Example 4–1 offers a navigational view of the `sessions.xml` file, illustrating the file's structure:

Example 4–1 Navigating the sessions.xml File

```
<toplink-configuration>
   <session>
     <name>
     <project-class> or <project-xml>
     <session-type>
     <login>
        [Login Options including Sequencing and Cache Sysnchronization]
        <uses-external-connection-pool>
        <uses-external-transaction-controller>
     </login>
     <event-listener-class>
     <profiler-class>
     <data-source>
     <external-transaction-controller-class>
     <exception-handler-class>
     <connection-pool>
        [Connection Pool Options]
```

```
    </connection-pool>
    <enable-logging>
       [Logging Options]
    </enable-logging>>
  </session>
</toplink-configuration>
```

XML Header

The `sessions.xml` file begins with a header section that describes the file, and specifies the location of the DTD for file validation.

If you use third-party parsers with the `sessions.xml` file, be aware that some parsers require a fully qualified path to the DTD in the XML header. If you are using one of these parsers, include the full path to the DTD in the system identifier, as follows:

```
<!DOCTYPE toplink-configuration PUBLIC "-//Oracle Corp.//DTD TopLink Sessions
9.0.4//EN" "file://<ORACLE_HOME>/toplink/config/dtds/sessions_9_0_4.dtd">
```

toplink-configuration Element

The `toplink-configuration` element is the root XML element for the `sessions.xml` file. It encapsulates the rest of the session configuration information.

Example 4–2 The toplink-configuration Element

```
<toplink-configuration>
    ...
    //Session configuration information
    ...
</toplink-configuration>
```

session Element

The `session` element contains configuration information for an OracleAS TopLink session. It includes several tags that specify the options for the session. The `sessions.xml` file normally contains at least one `session` element, and can include several elements if the application requires it.

The `session` element supports the configuration tags listed in Table 4–1.

Table 4–1 Tags Within the Session Element

Tag	Description
`name`	Specifies the name of the session. Assign a unique name to each session in the `sessions.xml` file to enable the session manager to retrieve it correctly. The `name` tag is mandatory.
`project-class`	Specifies the name of the class that contains the OracleAS TopLink project metadata. Use this tag (and not the `project-xml` tag) to deploy a project that uses exported and compiled Java code. Specify the fully qualified Java class name, but do not include the `.class` or `.java` extension.
`project-xml`	Specifies the name of the XML file that contains the OracleAS TopLink project metadata. Use this tag (and not the `project-class` tag) to deploy your project that uses an exported XML file. Specify the fully qualified file name, including the .xml extension.

Example 4–3 Using a Project Class Element

```
<toplink-configuration>
    <session>
        <name>mysession</name>
        <project-class>com.mycompany.MyProject</project-class>
        ...
    </session>
</toplink-configuration>
```

Example 4–4 Using the project.xml File

```
<toplink-configuration>
    <session>
        <name>mysession</name>
        <project-xml>C:/myproject/myproject.xml</project-xml>
        ...
    </session>
</toplink-configuration>
```

In addition to the preceding tags, the `session` element includes several tags that contain session configuration information:

- session-type Element
- login Element
- event-listener-class Element
- cache-synchronization-manager Element
- profiler-class Element
- external-transaction-controller-class Element
- exception-handler-class Element
- connection-pool Element
- enable-logging Element

session-type Element

The `session-type` element appears inside of a `session` element and specifies the session type with the tags listed in Table 4–2.

Table 4–2 Tags Within the Session-Type Element

Tag	Description
`session-type`	Specifies the type of OracleAS TopLink session the `SessionManager` will instantiate. Valid options include `server-session` and `database-session`. The `session-type` tag is mandatory.
`server-session`	In the `session-type` element, indicates that the `SessionManager` instantiates and returns the named session as a `ServerSession` (Server).
`database-session`	In the `session-type` element, indicates that the `SessionManager` instantiates and returns the named session as a `DatabaseSession`.

Example 4–5 Defining a Server Session

```
<session>
    <name>myServerSession</name>
    <project-class>com.mycompany.MyProject</project-class>
    <session-type>
        <server-session/>
    </session-type>
    ...
</session>
```

Example 4–6 Defining a Database Session

```
<session>
    <name>myDatabaseSession</name>
    <project-class>com.mycompany.MyProject</project-class>
    <session-type>
        <database-session/>
    </session-type>
    ...
</session>
```

login Element

The `login` element tags listed in Table 4–3 are optional for the session. If you do not include the `login` element in the `sessions.xml` file, set a default login in the OracleAS TopLink Mapping Workbench.

Table 4–3 Basic Configuration Tags Within the Login Element

Tag	Description
`license-path`	Specifies the license path for pre-TopLink 4.6 licensing. Because OracleAS TopLink no longer requires this tag, OracleAS TopLink does not process this element. If you are using the `sessions.xml` file from an OracleAS TopLink version that required a licence file, this tag will not prevent the `sessions.xml` file from running under the current version of OracleAS TopLink, but you should consider rebuilding your `sessions.xml` file. **Note:** If you are using a `sessions.xml` file from an older version of OracleAS TopLink, you can delete this tag.

Table 4–3 Basic Configuration Tags Within the Login Element (Cont.)

Tag	Description
`driver-class`	Specifies the JDBC driver class to use to log in to the database. The `driver-class` tag is optional and is not required when you implement the `data-source` tag.
`connection-url`	Specifies the JDBC connection URL for the database. This tag is optional. Do not use the `connection-url` tag if you implement the `data-source` tag.
`data-source`	Specifies the datasource name if you are using a JNDI datasource. This tag is optional. Do not use the `data-source` tag if you implement the `connection-url` and `driver-class` tag.
`platform-class`	Specifies the OracleAS TopLink platform class for the session. This tag is optional. For more information about platform classes, see "SDK Platform and Sequencing" on page 5-54.
`user-name`	The user name to log in to the database. The `user-name` tag is optional and is not required if you use a datasource.
`password`	The password to log in to the database. The `password` tag is optional and is not required if you use a datasource.
`encrypted-password`	The password of the user name used to log into the database. The `<encrypted-password>` tag.
`encryption-class-name`	When you use an encrypted password, select the specific encryption class. The `<encryption-class-name>` tag.

Example 4–7 Basic Configuration Using JDBC

```
<session>
  <name>myServerSession</name>
  <project-class>com.mycompany.MyProject</project-class>
  <session-type>
    <server-session/>
  </session-type>
  <login>
    <license-path>C:/myproject/license/</license-path>
    <driver-class>oracle.jdbc.driver.OracleDriver</driver-class>
```

```
    <connection-url>jdbc:oracle:thin@dbserver:1521:dbname</connection-url>
    <platform-class>oracle.toplink.internal.databaseaccess.OraclePlatform</platform-class>
    <user-name>scott</user-name>
    <password>tiger</password>
  </login>
...
</session>
```

Example 4–8 Basic Configuration Using a Datasource

```
<session>
  <name>myServerSession</name>
  <project-class>com.mycompany.MyProject</project-class>
  <session-type>
    <server-session/>
  </session-type>
  <login>
    <data-source>jdbc/MyApplicationDS</data-source>
    <platform-class>oracle.toplink.internal.databaseaccess.OraclePlatform</platform-class>
  </login>
...
</session>
```

Optional Login Tags The `login` element offers several optional tags that enable you to customize your session login.

Optional tags the `login` element offers include:

- `encryption-class-name`: Specifies the name of the custom class used to encrypt and decrypt the password. The `encryption-class-name` must be fully qualified and the class must be on the class path.
- `encrypted-password`: Specifies the encrypted password.

Other optional `login` tags accept TRUE or FALSE as valid values. Table 4–4 describes these tags.

Table 4–4 Optional Tags Within the Login Element

Tag	Description
`should-bind-all-parameters`	Enables parameter binding for all parameters. Use parameter binding with statement caching. The default value is FALSE. For more information about Parameter Binding, see "Binding and Parameterized SQL" on page 5-17.
`should-cache-all-statements`	Enables statement caching. The default value is FALSE. Statement caching requires you to set the `should-bind-all-parameters` tag to TRUE.
`uses-byte-array-binding`	Specifies whether OracleAS TopLink uses binding for byte arrays. The default value is FALSE.
`uses-string-binding`	Specifies whether OracleAS TopLink uses binding for String objects. The default value is FALSE.
`uses-streams-for-binding`	Specifies whether OracleAS TopLink uses streams for binding byte array parameters. The default value is FALSE.
`should-force-field-names-to-uppercase`	Specifies whether OracleAS TopLink converts field names to uppercase when generating SQL. The default value is FALSE.
`should-optimize-data-conversion`	Specifies whether the session should optimize driver-level data conversion. The default value is TRUE.
`should-trim-strings`	Specifies whether OracleAS TopLink removes any trailing white spaces from the end of strings. The default value is TRUE.
`uses-batch-writing`	Specifies whether the session uses batch writing to write to the database. The default value is FALSE.
`uses-jdbc20-batch-writing`	Specifies whether the session's database connection(s) uses JDBC 2.0 batch writing or OracleAS TopLink batch writing. The default value is TRUE. If you enable this option, enable the `uses-batch-writing` option as well.
`uses-external-connection-pool`	Specifies whether the session uses external connection pooling. The default value is FALSE.
`uses-native-sql`	Specifies whether the session uses database-specific SQL grammar. The default value is FALSE.

Table 4–4 Optional Tags Within the Login Element (Cont.)

Tag	Description
`uses-external-transaction-controller`	Specifies whether the session uses an external transaction controller. The default value is FALSE.
`non-jts-connection-url`	Specifies the URL for sequencing connection pooling. Used in conjunction with the `non-jts-datasource` tag when you set the `uses-sequence-connection-pool` tag to TRUE.
`non-jts-datasource`	Specifies the non-JTS datasource for the sequencing connection pool. Used in conjunction with the `non-jts-connection-url` tag when you set the `uses-sequence-connection-pool` tag to TRUE.
`uses-sequence-connection-pool`	Specifies whether the session creates and uses a separate connection pool for sequencing. The default value is FALSE. If you set this element to TRUE, you must also configure the `non-jts-connection-url` and `non-jts-datasource` tags.

Sequencing Elements You can configure sequencing as part of the session login, although it is not a requirement. If you do not configure sequencing in the `sessions.xml` file, then the application uses the configuration that is specified in the OracleAS TopLink Mapping Workbench project.

Configure sequencing in the `sessions.xml` file when you want to use custom sequencing for a given session.

Table 4–5 lists the elements you use to configure sequencing in the `sessions.xml` file. All these elements are optional.

Table 4–5 Optional Sequencing Configuration Tags Within Login

Tag	Description
`uses-native-sequencing`	Specifies whether the session uses native sequencing. This tag accepts TRUE or FALSE as values. The default is FALSE. Note that not all database platforms support native sequencing.
`sequence-preallocation-size`	Specifies the sequence preallocation size. If you use native sequencing, this value must match the sequence preallocation size set on your database. The default value is 50.
`sequence-table`	For table sequencing, specifies the name of the sequencing table. The default name is `SEQUENCE`.
`sequence-name-field`	For table sequencing, specifies the column in the sequencing table that contains the names of the sequenced objects. The default name is `SEQ_NAME`.
`sequence-counter-field`	For table sequencing, specifies the column in the sequence table that stores the current sequence count for each sequenced object. The default name is `SEQ_COUNT`.

For more information, see "Sequencing" on page 3-37.

Example 4–9 Configuring Native Sequencing

```
<session>
    <login>
...
        <uses-native-sequencing>true</uses-native-sequencing>
        <sequence-preallocation-size>50</sequence-preallocation-size>
    </login>
...
</session>
```

Example 4–10 Configuring Table-Based Sequencing

```
<session>
...
    <login>
        <uses-native-sequencing>false</uses-native-sequencing>
        <sequence-table>SEQUENCE</sequence-table>
```

```
        <sequence-name-field>SEQ_NAME</sequence-name-field>
        <sequence-counter-field>SEQ_COUNT</sequence-counter-field>
    </login>
...
</session>
```

cache-synchronization-manager Element You configure cache synchronization as part of the `login`. Use the `cache-synchronization-manager` element and the tags listed in Table 4–6 to configure cache-synchronization for your application.

Table 4–6 Cache Synchronization Manager Configuration Tags

Tag	Description
`clustering-service`	Specifies the class name of the clustering service. This tag is required for cache synchronization.
`multicast-port`	Specifies the port for listening for connection messages over IP multicast. Ensure that all servers in your OracleAS TopLink cache synchronization group use the same multicast port. This tag is required only if you also use the `multicast-group-address` element. The default value is 6018.
`multicast-group-address`	Specifies the IP address for sending connection messages over IP multicast. Ensure that all servers in your OracleAS TopLink cache synchronization group use the same multicast address. This tag is required only if you also use the `multicast-port` element. The default value is 226.18.6.18.
`packet-time-to-live`	Specifies the number of network hops that cache synchronization discovery packets traverse. This optional tag defaults to 2.
`is-asynchronous`	Specifies whether cache synchronization is performed asynchronously (TRUE) or synchronously (FALSE). This optional tag defaults to TRUE.
`should-remove-connection-on-error`	Specifies whether OracleAS TopLink removes a remote connection if a communications exception occurs with a remote server. This optional tag defaults to FALSE.

Table 4–6 Cache Synchronization Manager Configuration Tags (Cont.)

Tag	Description
`jndi-user-name`	Specifies the user name to use for binding the Cache Synchronization Manager into JNDI. Use this tag to support JNDI in non application server applications. This optional tag requires the `jndi-password` tag.
`jndi-password`	Specifies the password to use for binding the cache synchronization manager into JNDI. Use this tag to support JNDI in non application server applications. This optional tag requires the `jndi-user-name` tag.
`jms-topic-connection-factory-name`	Specifies the topic connection factory name for JMS cache synchronization. This tag is required only when you use JMS cache synchronization.
`jms-topic-name`	Specifies the topic name for JMS cache synchronization. This tag is required only when you use JMS cache synchronization.
`naming-service-initial-context-factory-name`	Specifies the initial context factory for accessing JNDI. Use this tag only if OracleAS TopLink encounters difficulties connecting to JNDI or JMS.
`naming-service-url`	Specifies the URL of the naming service that supports cache synchronization. The value for this element depends on how you implement cache synchronization: ■ For JNDI clustering services, this is the scheme, host IP address, and port of the JNDI service. ■ For the RMI clustering service, this is the host IP address and port of the RMI registry. This optional tag may resolve problems that occur when you implement cache synchronization inside an application server with a JNDI clustering service. If you do not encounter any problems, do not use this tag.

Example 4–11 Using the Cache Synchronization Manager

```
<session>
...
  <login>
    <cache-synchronization-manager>
      <clustering-service>oracle.toplink.remote.rmi.RMIClusteringService</clustering-servi
ce>
      <multicast-port>6020</multicast-port>
```

```
      <multicast-group-address>226.18.6.18</multicast-group-address>
      <is-asynchronous>true</is-asynchronous>
      <should-remove-connection-on-error>true</should-remove-connection-on-error>
      <naming-service-url>localhost:1099</naming-service-url>
    </cache-synchronization-manager>
  </login>
...
</session>
```

event-listener-class Element

If your applications need to know when session events take place, use event listeners to register for event notification. Event listeners can be configured in the `sessions.xml` file.

The `event-listener-class` tag enables you to configure listener classes that either implement the `oracle.toplink.sessions.SessionEventListener` interface, or extend the `oracle.toplink.sessions.SessionEventAdapter` class. Configure multiple event listener classes by including multiple `event-listener-class` tags and specifying the implementing class name for each tag.

OracleAS TopLink automatically registers event listeners in the `sessions.xml` file with the session event manager.

For more information, see "Customizing Session Events" on page 4-67.

Example 4–12   Setting the Event Listener Class in Code

```
package examples;
import oracle.toplink.sessions.*;
public class MyEventListener extends SessionEventAdapter {
    public void preLogin(SessionEvent event) {
        Session session = event.getSession();
        /* custom code goes here */
    }
}
```

Example 4–13   Setting the Event Listener Class in the sessions.xml File

```
<session>
    ...
    <event-listener-class>examples.MyEventListener</event-listener-class>
    ...
```

```
</session>
```

OracleAS TopLink registers the `examples.MyEventListener` class with the session event manager for the session. OracleAS TopLink invokes the `MyEventListener` class `preLogin` method when the `preLogin` event occurs on the session.

profiler-class Element

OracleAS TopLink provides a profiler to optimize your application and identify performance bottlenecks. To implement the performance profiler, use the `profiler-class` tag to include the performance profiler in your session.

Example 4–14 Implementing the Performance Profiler in the sessions.xml File

```
<session>
    ...
    <profiler-class>oracle.toplink.tools.profiler.PerformanceProfiler</profiler-class>
    ...
</session>
```

The `profiler-class` tag supports any class that implements the `oracle.toplink.sessions.SessionProfiler` interface. Because of this, you can build your own profiler and add it to your session—provided that your profiler implements the `oracle.toplink.sessions.SessionProfiler` interface.

> **Note:** You can implement only one profiler a session.

external-transaction-controller-class Element

If your system includes external transactions (under JTA, for example), specify an OracleAS TopLink external transaction controller using the `external-transaction-controller-class` tag.

To use an external transaction controller, specify the following in the session login:

- The external transaction controller
- A datasource on the session
- An external connection pool

Example 4–15 Configuring the External Transaction Controller

```
<session>
    ...
    <login>
        ...
        <uses-external-transaction-controller>true</uses-external-transaction-controller>
        <data-source>jdbc/MyApplicationDS</data-source>
        <uses-external-connection-pool>true</uses-external-connection-pool>
        ...
    </login>
    <external-transaction-controller-class>oracle.toplink.jts.oracle9i.Oracle9iJTSExternal
TransactionController</external-transaction-controller-class>
    ...
</session>
```

exception-handler-class Element

The `exception-handler-class` tag specifies a class that handles exceptions for the session. This tag accepts any class that implements the `oracle.toplink.exceptions.ExceptionHandler`.

Example 4–16 Configuring the Exception Handler in Code

```
package examples;
import oracle.toplink.exceptions.*;
public class MyExceptionHandler implements ExceptionHandler {
   public Object handleException(RuntimeException exception) {
      /*custom code goes here */
   }
}
```

Example 4–17 Configuring the Exception Handler in the sessions.xml File

```
<session>
  ...
  <exception-handler-class>examples.MyExceptionHandler</exception-handler-class>
  ...
</session>
```

connection-pool Element

You can explicitly configure a single connection pool or multiple connection pools for your OracleAS TopLink application with the `connection-pool` element in the `sessions.xml` file. If you do not configure a connection pool for a session, then the session uses the default connection pool that you defined for the project.

Define a login for each `connection-pool` that you define manually. Table 4–7 lists the elements you use to configure the `connection-pool` element in the `sessions.xml` file.

For more information about configuring the connection pool for the project, see "Working with Connection Pools" in the *Oracle Application Server TopLink Mapping Workbench User's Guide.*

For more information about configuring a login, see "login Element" on page 4-13.

Table 4–7 Connection Pool Element Tags

Tag	Description
`is-read-connection-pool`	Specifies whether the connection pool contains read connections (true) - (non-transactional) or for write connections (false) - (transactional). The `is-read-connection-pool` tag is mandatory, and accepts TRUE or FALSE as values.
`name`	Specifies the name of the connection pool. If the name is the same as another existing OracleAS TopLink connection pool (such as the default OracleAS TopLink read pool), the existing connection pool is replaced with the new one. The `name` tag is mandatory.
`max-connections`	Specifies the maximum number of database connections that the connection pool can use. This tag is optional and accepts integer values. The default is 10.
`min-connections`	Specifies the minimum number of database connections that the connection pool should use at startup. This tag optional and accepts integer values. The default is 5.

Example 4–18 Configuring the Connection Pool Element

```
<session>
...
    <connection-pool>
        <is-read-connection-pool>true</is-read-connection-pool>
        <name>additionalReadPool</name>
        <max-connections>20</max-connections>
        <min-connections>10</min-connections>
        <login>
        ...
```

```
        </login>
    </connection-pool>
...
</session>
```

enable-logging Element

OracleAS TopLink does not automatically enable logging for a session unless you explicitly request it. To enable logging in a session, include the `enable-logging` element as part of your session definition in the `sessions.xml` file and set it to TRUE.

After you enable logging, you can customize the logging behavior on the session by including one or more logging options in the `sessions.xml` file. The available logging options appear in Table 4–8, and accept TRUE or FALSE as arguments.

Table 4–8 Logging Option Tags

Tag	Description
`log-debug`	Specifies whether the session logs debug information in addition to standard log entries.
`log-exceptions`	Specifies whether the session logs uncaught exception messages.
`log-exception-stacktrace`	Specifies whether the session logs exception stack traces.
`print-session`	Specifies whether the session logs session identifiers.
`print-thread`	Specifies whether the session logs thread identifiers.
`print-connection`	Specifies whether the session logs connection identifiers.
`print-date`	Specifies whether the session logs the date and time of each log entry.

Example 4–19 Configuring Logging and Logging Options

```
<session>
    ...
    <enable-logging>true</enable-logging>
    <logging-options>
        <log-debug>false</log-debug>
        <log-exceptions>true</log-exceptions>
        <log-exception-stacktrace>true</log-exception-stacktrace>
        <print-session>true</print-session>
        <print-thread>false</print-thread>
        <print-connection>true</print-connection>
        <print-date>true</print-date>
```

```
    </logging-options>
    ...
</session>
```

session-broker Element

The session broker enables client applications to view several databases through a single session. The `session-broker` element enables you to configure a session broker in the `sessions.xml` file, as follows:

1. Configure the session broker sessions in the `sessions.xml` file. These sessions are the database sessions or server sessions that the session broker uses to communicate with the databases.
2. Add the session broker to the `sessions.xml` file using the `session-broker` element.
3. Populate the `session-broker` element with a name and the sessions that you configured in Step 1.

Example 4–20 Configuring a Session Broker in the sessions.xml File

```
/* Configure the sessions for the SessionBroker */
<session>
    <name>EmployeeSession</name>
    ...
</session>
<session>
    <name>ProjectSession</name>
    ...
</session>
/* Configure the SessionBroker */
<session-broker>
/* Name the SessionBroker */
    <name>EmployeeAndProjectBroker</name>
/* Specify the sessions contained in the SessionBroker */
        <session-name>EmployeeSession</session-name>
        <session-name>ProjectSession</session-name>
</session-broker>
    ...
```

JTA Configuration

OracleAS TopLink J2EE integration includes support for JTA external connection pools and external transaction controllers. To enable a JTA external transaction controller, set the login to use an external transaction controller, and configure the following in your `sessions.xml` file:

- A JTA `DataSource` (in the `login` element)
- An external connection pool (in the `login` element)
- An external transaction controller (in the `session` element)

For more information about the OracleAS TopLink JTA integration, see "J2EE Integration" on page 7-44.

Example 4–21 Configuring for JTA in the sessions.xml File

```
<session>
    ...
    <login>
        ...
        <uses-external-transaction-controller>true</uses-external-transaction-controller>
        <data-source>jdbc/MyApplicationDS</data-source>
        <uses-external-connection-pool>true</uses-external-connection-pool>
        ...
    </login>
    <external-transaction-controller-class>oracle.toplink.jts.oracle9i.Oracle9iJTSExternal
TransactionController</external-transaction-controller-class>
    ...
</session>
```

Example 4–22 Configuring for JTA in Code

```
DatabaseLogin login = null;
project = null;

/*note that useExternalConnectionPooling and useExternalTransactionController
must be set before Session is created */
project = new SomeProject();
login = project.getLogin();
login.useExternalConnectionPooling();
login.useExternalTransactionController();

/* usually, other login configuration such as user, password, JDBC URL comes
from the project but these can also be set here
session = new Session(project);
```

```
/* other session configuration, as necessary: logging, ETC
session.SetExternalTransactionController(new
SomeJTSExternalTransactionController());
session.login();
```

Registering Descriptors

How you add descriptors depends on how you created them. You can create project descriptors in the OracleAS TopLink Mapping Workbench and export them to a single descriptor file, set the `sessions.xml` file to reference the descriptor file. As a result, OracleAS TopLink can load the descriptors into the session automatically. A project class can also be specified in the `sessions.xml` file. For all other options, use the add descriptors method to register the descriptors, as Table 4–9, "addDescriptors Options" illustrates.

Table 4–9 addDescriptors Options

Format	Description
`addDescriptors(Project)`	Enables you to manually add additional descriptor to the session in the form of a project.
`addDescriptors(Vector)`	Enables you to add a vector of individual descriptor files to the session in the form of a project.
`addDescriptor(Descriptor)`	Enables you to add individual descriptors to the session.

Registering Descriptors after Login You can register descriptors after the session logs in. Doing this enables you to load self-contained sub-systems after the session connects. Descriptors that are registered this way are independent of descriptors that are already registered.

- To change a descriptor and redeploy it with a minimum of down time, you can also re-register descriptors that are loaded in the session. You must also re-register all related descriptors at the same time, because changes to one descriptor may affect the initialization of other descriptors.

Caching Objects

Database sessions include an identity map that maintains object identity, and acts as a cache. When the session reads objects from the database, it instantiates them and stores them in the identity map. When the application subsequently queries for the same object, OracleAS TopLink returns the object in the cache rather than read the object from the database again.

You can force OracleAS TopLink to flush all objects from the cache. To do so, first ensure that none of the objects are in use within the database session. Then call the `initializeIdentityMaps()` method.

To improve performance, you can customize the identity map. For more information about using the identity map and caching, see the *Oracle Application Server TopLink Mapping Workbench User's Guide.*

Session Manager

The OracleAS TopLink session manager enables developers to build a series of sessions that are maintained under a single entity. The session manager is a static utility class that loads OracleAS TopLink sessions from the `sessions.xml` file, caches the sessions by name in memory, and provides a single access point for OracleAS TopLink sessions.

The session manager supports the following session types:

- `ServerSession` (see "Server Session and Client Session" on page 4-37)
- `DatabaseSession` (see "Database Session" on page 4-48)
- `SessionBroker` (see "Session Broker" on page 4-53)

The session manager has two main functions: it creates instances of these sessions and it ensures that only a single instance of each named session exists for any instance of a session manager.

Instantiate the session manager as follows:

```
SessionManager.getManager()
```

This section describes techniques for working with the session manager and includes discussions of the following topics:

- Retrieving a Session from a Session Manager
- Storing Sessions in the Session Manager Instance

Retrieving a Session from a Session Manager

OracleAS TopLink maintains only one instance of the session manager class. The singleton session manager maintains all the named OracleAS TopLink sessions at runtime. When an application requests a session by name, the session manager retrieves the specified session from the configuration file.

To access the session manager instance, invoke the static `getManager()` method on the `oracle.toplink.tools.sessionmanagement.SessionManager` class. You can then use the session manager instance to load OracleAS TopLink sessions.

Example 4–23 Loading a Session Manager Instance

```
import oracle.toplink.tools.sessionmanagement.SessionManager;
SessionManager sessionManager = SessionManager.getManager();
```

OracleAS TopLink uses a class loader to load the session manager. The session manager, in turn, uses that same class loader to load named sessions that are not already initialized in the session manager cache.

> **Note:** To fully leverage the methods associated with the session type that is being instantiated, cast the session that is returned from the `getSession()` method. This type must match the session type that is defined in the `sessions.xml` file for the named session.

Example 4–24 Loading a Named Session from Session Manager Using Defaults

```
/* This example loads a named session (mysession) defined in the sessions.xml
file. */
SessionManager manager = SessionManager.getManager();
Server server = (Server) manager.getSession("myserversession");
```

Loading a Session with an Alternative Class Loader

You can use an alternative class loader to load sessions. This is common when your OracleAS TopLink application integrates with a J2EE container. If the session is not already in the session manager's in-memory cache of sessions, the session manager creates the session and logs in.

Example 4–25 Loading a Session Using an Alternative Class Loader

```
/* This example uses the specified ClassLoader to load a session (mysession)
defined in the sessions.xml file. */
ClassLoader classLoader = YourApplicationClass.getClassLoader();
SessionManager manager = SessionManager.getManager();
Session session = manager.getSession("mysession", // session nameclassLoader);
```

```
// classloader
```

Loading an Alternative Session Configuration File

You can use the XML Loader to load any XML configuration file on the application class path. This enables you to use files other than the standard `sessions.xml` file to load sessions.

You can use the XML loader to load different sessions, and even different class loaders, from configuration files. The `XMLLoader` class defines two constructors:

- The *zero-argument* constructor loads the default `sessions.xml` file.
- The single argument constructor includes a parameter (a `String`) that specifies an alternative configuration file.

Example 4–26 Loading an Alternative Configuration File

```
/* XMLLoader loads the toplink-sessions.xml file */
XMLLoader xmlLoader = new XMLLoader("toplink-sessions.xml");
ClassLoader classLoader = YourApplicationClass.getClassLoader();
SessionManager manager = SessionManager.getManager();
Session session = manager.getSession(
    xmlLoader, // XML Loader
    "mysession", // session name
    classLoader); // classloader
```

Reusing the Configuration File If your application maintains the XML loader instance, then OracleAS TopLink reads sessions from the configuration file with the first `getsession ()`, but does not reparse the file with each subsequent `getsession ()` calls. If OracleAS TopLink uses a different XML loader to call a session, or if you invoke the API to refresh the configuration file, then OracleAS TopLink reparses the configuration file, but sessions already in the session manager do not change.

Opening Sessions without Logging In The XML loader enables you to call a session using `getSession()`, without invoking the `login()` method. This enables you to prepare a session for use and leave login to the application.

Example 4–27 Open Session with No Login

```
SessionManager manager = SessionManager.getManager();
Session session = manager.getSession(
    new XMLLoader(), // XML Loader (sessions.xml file)
```

```
    "mysession", // session name
    YourApplicationClass.getClassLoader(), // classloader
    false, // log in session
    false); // refresh session
```

Reparsing the Session Configuration File The XML loader can force OracleAS TopLink to reparse the session configuration file for sessions that do not exist in its in-memory cache. This function is useful when you want to add a session to an in-production `sessions.xml` file that already exists in the session manager cache. When the session manager attempts to load a session that is not in its in-memory cache, it reparses the XML file.

Example 4–28 Forcing a Reparse of the sessions.xml File

```
//In this example, the XML loader loads the sessions.xml file from the class
path.
SessionManager manager = SessionManager.getManager();
Session session = manager.getSession(
    new XMLLoader(), // XML Loader (sessions.xml file)
    "mysession", // session name
    YourApplicationClass.getClassLoader(), // classloader
    true, // log in session
    true); // refresh session
```

Storing Sessions in the Session Manager Instance

You can manually create a session in your application, rather than loading a preconfigured session from the session configuration file. Use the `SessionManager` class as a singleton to store the manually created session. Use the `getSession()` API with the single `String [session name]` argument on session manager to load the session.

Note: The `getSession()` API is not necessary if you are loading sessions from a session configuration file.

Example 4–29 Storing Sessions Manually in the Session Manager

```
// create and log in session programmatically
Session theSession = project.createDatabaseSession();
theSession.login();
// store the session in the SessionManager instance
```

```
SessionManager manager = SessionManager.getManager();
manager.addSession("mysession", theSession);
// retrieve the session
Session session = SessionManager.getManager().getSession("mysession");
```

Destroying Sessions in the Session Manager Instance

The Session Manager provides two utility methods for destroying stored sessions.

Example 4–30 Destroying Sessions in the Session Manager

```
// create and log in session programmatically
Session theSession = project.createDatabaseSession();
theSession.login();
// store the session in the SessionManager instance
SessionManager manager = SessionManager.getManager();
manager.addSession(“mysession”, theSession);
…
// destroying the session
// this will throw a validation exception if the session name
// is not found
manager.destroySession(“mySession”);

OR

// if multiple sessions have been stored and all need to be
// destroyed, then use the destroyAllSessions API
manager.destroyAllSessions();
```

Session Querying

The Session class and its subclasses provide *query methods* that enable you to run queries against the object model rather than the relational model. You can invoke query methods using any of the following:

- Simple Query API
- Query Objects
- Predefined Queries

This section introduces query methods.

For more in-depth information, see "Session Queries" on page 6-37.

Simple Query API

The Session class offers the following methods to access the database:

- The `readObject()` method uses a primary key to search for a single object in the database or the session cache. Specify the class of the queried object.

  For example:

  ```
  session.readObject(MyDomainObject.class);
  ```

  This example returns the first instance of `MyDomainObject` found in the table that contains the `MyDomainObject` class. If the query does not find an object that matches the criteria, it returns null. For more complex `readObject()` queries, augment the query with an OracleAS TopLink Expression.

  For more information, see "Using Expressions in Session Queries" on page 4-34.

- The `readAllObjects()` method retrieves a `Vector` of objects from the database. Specify the class of the queried object.

  For example:

  ```
  session.readAllObjects(MyDomainObject.class)
  ```

  If the query does not find any objects that match the criteria, it returns an empty vector. For more complex `readAllObjects()` queries, augment the query with an OracleAS TopLink Expression.

  For more information, see "Using Expressions in Session Queries" on page 4-34.

  The `readAllObjects()` method does not order the objects, but instead returns objects in the order in which they are found.

Using Expressions in Session Queries

To form more complex queries, include expressions in session query methods. Expression support makes up two public classes:

- The *Expression* class enables you to build either simple or complex logic into the expression, You can also combine multiple expressions in a query method.
- The *ExpressionBuilder* class is the factory that constructs new expressions.

To combine expressions with query methods, use the Expression Builder to create an expression and set the expressions as the selection criterion for the query.

Example 4–31 The readObject() Method Using an Expression

```
Employee employee = (Employee) session.readObject(Employee.class, new
ExpressionBuilder().get("lastName").equal("Smith"));
```

Example 4–32 The readAllObjects() Method Using an Expression

```
Vector employees = session.readAllObjects(Employee.class,new
ExpressionBuilder.get("salary").greaterThan(10000));
```

For more information about the OracleAS TopLink Expression Builder, see "Expressions" on page 6-12.

Custom SQL Queries

You can execute custom SQL queries and stored procedure calls from within an OracleAS TopLink application. This is useful when you call stored procedures on the database and to access raw data. Use custom SQL strings and stored procedure calls in either of the following ways:

- Use the `executeSelectingCall()` and `executeNonSelectingCall()` session methods to execute SQL queries directly on the database.

  For example:

  ```
  Vector rows = session.executeSelectingCall(new SQLCall("SELECT USER, SYSDATE
  FROM DUAL"));
  ```

- Call the `executeQuery()` method on the `DatabaseSession`. The following code example uses SQL to read all employee IDs:

  ```
  DirectReadQuery query = new DirectReadQuery();
  query.setSQLString("SELECT EMP_ID FROM EMPLOYEE");
  Vector ids = (Vector) session.executeQuery(query);
  ```

Session Methods and the Unit of Work If you call a session method to execute native SQL or invoke a stored procedure within a Unit of Work, then the Unit of Work is aware that you called a session method. However, it does not know about any changes the SQL or stored procedure makes to the database outside of the Unit of Work context, and so cannot roll back those changes if the `commit` call fails. Avoid using session methods inside a Unit of Work.

Query Objects

A query object is an OracleAS TopLink querying mechanism that offers full database querying access. Query objects support search criteria specified in several ways, including OracleAS TopLink expressions.

Use query objects to perform complex querying. An application creates query objects by instantiating the object and defining its querying criteria with either `Expression` objects or SQL strings.

You can:

- Execute the query objects directly, by calling the `executeQuery()` method on the `DatabaseSession`.
- Define new querying routines and add the routines to the session. Because you name these queries when you add them to the session, you can call them by name.
- Change the default querying behavior for read or write operations. An application can customize how the session's queries operate by supplying query objects to the descriptor's query manager.
- Change the default querying behavior for complex relationship mappings such as selection queries.

For more information about creating and using query objects, see "Query Objects" on page 6-41.

Predefined Queries

Predefined queries are queries you store in either the `sessions.xml` file or the OracleAS TopLink descriptor. Because they are part of the session or descriptor, OracleAS TopLink stores predefined queries in memory after you initially invoke them. Use predefined queries to maintain frequently-called queries.

For more information about predefined queries, see "Predefined Queries" on page 6-48.

Session Types

OracleAS TopLink provides several session types that enable you to tailor the session to your application needs. This section describes the following OracleAS TopLink session types:

- Server Session and Client Session

- Database Session
- Session Broker
- Remote Session

Server Session and Client Session

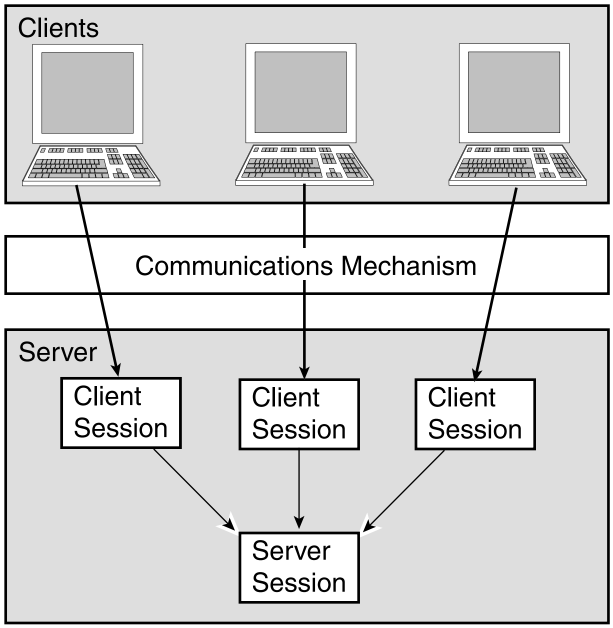
The server session and client session architecture is known collectively as a *three-tier* architecture. In this type of architecture, the server session provides session management for the clients, and the client session acts as an dedicated database session for each client or request.

Although they are two separate session types, use the client sessions and server sessions together. You define the server session in the `sessions.xml` file. After you instantiate the server session, you acquire client sessions from it. Each client session can have only one associated server session, but a server session can support any number of client sessions.

Three-Tier Architecture Overview

In an OracleAS TopLink three-tier architecture, client sessions and server sessions both reside on the server. Client applications access the OracleAS TopLink application through a client session, and the client session communicates with the database using the server session.

Figure 4–6 Server Session and Client Session Usage



EJBs and Server Session

The Enterprise JavaBean (EJB) container manages interaction with the database and OracleAS TopLink. The server session manages all aspects of persistence, such as caching, reading and writing, but does so behind the scenes.

General Concepts for the OracleAS TopLink Three-Tier Design

Although the server session and the client session are two different session types, you can treat them as a single unit in most cases, because they are both required to provide three-tier functionality to the application. The server session provides the client session to client applications, and also supplies the bulk of the session functionality. This section discusses some of the advantages and general concepts associated with the OracleAS TopLink three-tier design.

Shared Resources The three-tier design enables multiple clients to share persistent resources. The server session provides its client sessions with a shared live object cache, read and write connection pooling, and parameterized named queries. Client sessions also share descriptor metadata.

You can use client sessions and server sessions in any application server architecture that allows for shared memory and supports multiple clients. These architectures can include HTML, Servlet, JSP, RMI, CORBA, DCOM, and EJB.

To support a shared object cache, client sessions must:

- Implement any changes to the database with the OracleAS TopLink Unit of Work.
- Share a common database login for reading (you can implement separate logins for writing).

For more information, see "Sessions and the Cache" on page 4-64.

Providing Read Access To read objects from the database the client must first acquire a client session from the server session. Acquiring a client session gives the client access to the session cache and the database through the server session.

Example 4–33 Acquiring a Client Session

```
ClientSession myClientSession = myServerSession.acquireClientSession();
```

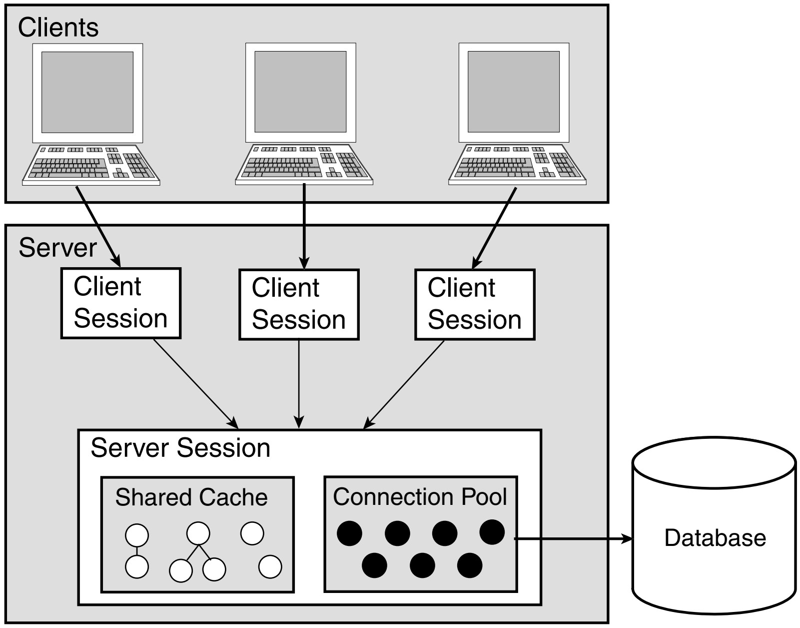
After the client acquires a client session, it can send read requests to the server. The server session responds to these requests as follows:

- If the object or data is in the session cache, then the server session returns the information back to the client.
- If the object or data is not in the cache, then the server session reads the information from the database and stores the object in the session cache. The objects are then available for retrieval from the cache.

Because a server session processes each client request in a separate thread, this enables multiple clients to access the database connection pool concurrently.

Figure 4–7 illustrates how multiple clients read from the database using the server session.

Figure 4–7 Multiple Client Sessions Reading the Database Using the Server Session



To read objects from the database using a Client Session:

1. Start the application server.
2. Create a `ServerSession` object and call `login()`.
3. Call `acquireClientSession()` to acquire a `ClientSession` from the `ServerSession`.
4. Execute read operations on the `ClientSession` object.

Note: Do not use the `ServerSession` object directly to read objects from the database.

Providing Write Access Because the client session disables all database modification methods, a client session cannot create, change, or delete objects directly. Instead, the client must obtain a Unit of Work from the client session to perform database modification methods.

To write to the database, the client acquires a client session from the server session and then acquires a `UnitOfWork` within that client session. The Unit of Work acts as an exclusive transactional object space, and also ensures that any changes that are committed to the database also occur in the session cache.

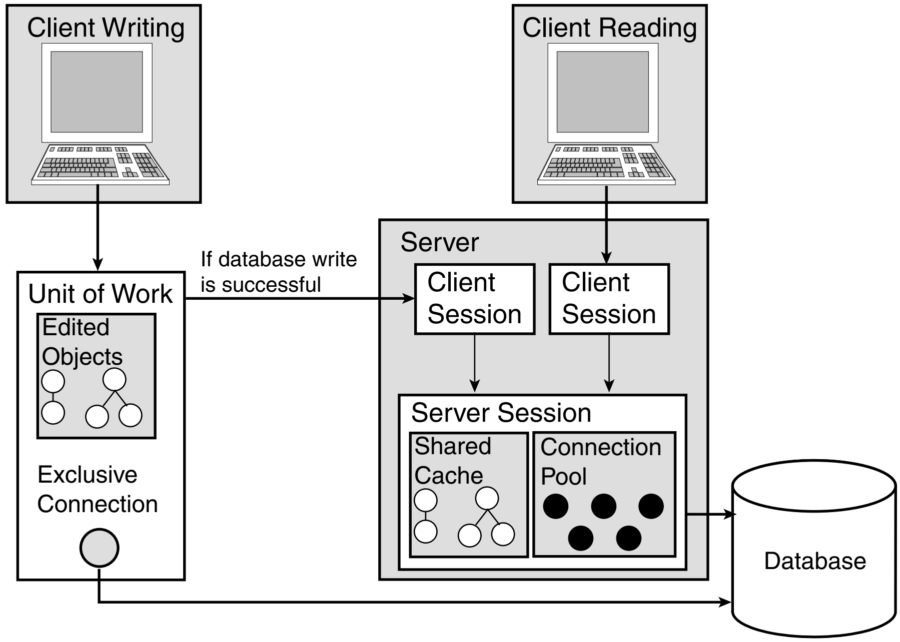
Caution: Although client sessions are thread-safe, do not use them to write across multiple threads. Multi-thread writes from the same client session can result in errors and a loss of data.

To write to the database using a Unit of Work:

1. Start the application server.
2. Create a `ServerSession` object and call `login()`.
3. Call `acquireClientSession()` to acquire a `ClientSession` from the `ServerSession`.
4. Acquire a `UnitOfWork` object from the `ClientSession` object.

   For more information about the Unit of Work, see Chapter 7, "Transactions" on page 7-1.
5. Perform the required updates, and then commit the `UnitOfWork`.

Figure 4–8 Writing with Client Sessions and Server Sessions



Parallel Units of Work The Unit of Work ensures that the client edits objects in a separate object transaction space. This feature enables clients to perform object transactions in parallel. When transactions commit, the Unit of Work makes any required changes in the database and then merges the changes into the shared OracleAS TopLink session cache. The modified objects are then available to all other users.

For more information about the Unit of Work, see to Chapter 7, "Transactions".

Security and User Privileges You can define several different server sessions in your application to support users with different data access rights. For example, your application may serve a group called "Managers," who has access rights to salary information, and a group called "Employees," who do not. Because each session you define in the `sessions.xml` file has its own login information, you can create multiple sessions, each with its own login credentials, to meet the needs of both of these groups.

Concurrency The server session supports concurrent clients by providing each client with a dedicated thread of execution. Dedicated threads enable clients to operate asynchronously—that is, client processes execute as they are called and do not wait for other client processes to complete.

OracleAS TopLink safeguards thread safety with a concurrency manager. The concurrency manager ensures that no two threads interfere with each other when performing operations such as creating new objects, executing a transaction on the database, or accessing valueholders.

Not all JDBC drivers support concurrency. Those that do not may require a thread to have exclusive access to a JDBC connection when reading. Configure the server session to use exclusive read connection pooling in these cases.

Connection Pooling When you instantiate the server session, it creates a pool of database connections. It then manages the connection pool based on your session configuration, and shares the connections among its client sessions. The server session provides connections to client sessions on an as-needed basis. When the client session releases the connection, the server session recovers the connection and makes it available to other client processes. Reusing connections reduces the number of connections required by the application and allows a server session to support a larger number of clients.

By default, the OracleAS TopLink write connection pool maintains a minimum of five connections and a maximum of ten. You can change these settings as follows:

- *To change the settings for the entire project*, adjust these settings in the OracleAS TopLink Mapping Workbench.

  For more information, see "Working with Connection Pools" in the *Oracle Application Server TopLink Mapping Workbench User's Guide.*

- *To change the settings for a particular server session*, adjust these settings in the `sessions.xml` file. You can make these changes using the OracleAS TopLink Sessions Editor, or add the following lines to the `session` element in the file manually:

```
 <session>
...
    <connection-pool>
        ...
        <max-connections>20</max-connections>
        <min-connections>10</min-connections>
        ...
    </connection-pool>
...
</session>
```

Tip: To maintain compatibility with JDBC drivers that do not support many connections, the default number of connections is small. If your JDBC driver supports it, use a larger number of connections for reading and writing.

The server session also supports multiple write connection pools and non-pooled connections. If your application server or JDBC driver also supports write connection pooling, you can configure the server session to use this feature. Set these options at the session level, modify the `session` element in the `sessions.xml` file.

For more information, see "Configuring Sessions with the sessions.xml File" on page 4-8.

Read Connections Although a single connection supports multiple threads reading asynchronously, some JDBC drivers perform better with multiple read connections. OracleAS TopLink enables you to allocate multiple read connections, and balances the load across the connections using a *least-busy* algorithm.

Server Session Connection Options The server session maintains a pool of read connections and a pool of write connections for its client sessions. You can customize the following options either in the `sessions.xml` file or in Java code:

- Create a new connection pool and add it to the pools the server session

  ```
  addConnectionPool(String poolName, JDBCLogin login, int
  minNumberOfConnections, int maxNumberOfConnections)
  ```

- In Java code, configure the read connection pool:

  ```
  useReadConnectionPool(int minNumberOfConnections, int
  maxNumberOfConnections)
  ```

- In Java code, configure the read connection pool to allow only a single thread to access each connection:

  ```
  useExclusiveReadConnectionPool(int minNumberOfConnections, int
  maxNumberOfConnections)
  ```

- In Java code, set the maximum number of nonpooled connections:

  ```
  setMaxNumberOfNonPooledConnections(int maxNumber)
  ```

Client Session Connection Options The three ways to get connections from within a client session object correspond to three arguments you can pass with the `acquireClientSession()` method on the server session are:

- Pass no argument (the *zero argument*). The acquired `ClientSession` uses the default connection pool.
- Pass a `poolName` as an argument. The acquired `ClientSession` uses a connection from the specified pool.
- Pass a `DatabaseLogin` object as an argument. The acquired `ClientSession` uses a specified `DatabaseLogin` object to obtain a connection.

By default, the server session does not allocate database connections for these client session until a Unit of Work commits to the database (a *lazy* database connection).

If you need to establish database connection immediately, configure the `ConnectionPolicy` object to specify a connection option more suited to your needs, and pass the `ConnectionPolicy` object as an argument.

Connection Policy The `ConnectionPolicy` class provides the following methods to configure a client connection:

- `setPoolName(String poolName)`: Creates a connection from the named connection pool. You can also use the `ConnectionPolicy(String poolName)` method.
- `setLogin(DatabaseLogin login)`: Sets up a connection by logging directly into the database. You can also use the `ConnectionPolicy(DatabaseLogin login)` method from the connection policy constructor.
- `useLazyConnection()`: Specifies whether the application uses a lazy connection (a connection that OracleAS TopLink instantiates only when required).
- `setLazyConnection(boolean isLazy)`: Specifies a lazy connection.
- `dontUseLazyConnection()`: Creates an active connection.

If you request a database connection when none is available, the method waits for the next available connection, rather than time out or return an error.

Reference

Table 4–10 and Table 4–11 summarize the most common public methods for `ClientSession` and `ServerSession`. For more information about the available methods for `ClientSession` and `ServerSession`, see the *Oracle Application Server TopLink API Reference*.

Table 4–10 Elements for Client Session

Element	Method Name
Executing a query object	`executeQuery(DatabaseQuery query, Vector parameters)`
Reading from the database	`readAllObjects(Class domainClass, Expression expression)` `readObject(Class domainClass, Expression expression)`
Release	`release()`
Unit of Work	`acquireUnitOfWork()`

Table 4–11 Elements for Server Session

Element	Method Name
Acquire `ClientSessions`	`acquireClientSession()`
Logging (Logging is not turned on, by default)	`logMessages()`
Login / logout	`login()` `logout()`

Customizing Server Session and Database Login

You can use a session amendment class to configure the server session and database login in ways not available through the deployment descriptor file. For example, you can:

- Specify special settings for the JDBC driver. For example, if you are working with an incompatible database driver, you can implement *parameter binding*, to enable a different data conversion routine.
- Access regular OracleAS TopLink features, such as database connections or caching, directly.
- Define custom finder queries on one or more OracleAS TopLink descriptors (under EJB 1.1).

- Enable native SQL support if your JDBC bridge does not support the JDBC standard SQL syntax.
- Enable binding and parameterized SQL, to specify whether values are inlined directly into the generated SQL or are parameterized.
- Enable batch writing, forcing the application to send groups of insert, update, and delete statements to the database in a single batch.
- Optimize data conversion.

Working with Login

Databases generally require a valid user name and password to login successfully. OracleAS TopLink applications maintain this information in the `DatabaseLogin` class. All sessions must have a valid `DatabaseLogin` instance before logging in to the database.

For more information about the `DatabaseLogin`, see "Database Session" on page 4-48.

Registering Event Listeners for EJB 1.1

To customize an EJB 1.1 application, register a session listener class that extends `oracle.toplink.sessions.SessionEventAdaptor`. Configure the listener to listen for various session events, such as `pre_login` and `post_commit_unit_of_work`. To register the OracleAS TopLink session, define the `event_listener_class` tag in the `toplink-ejb-jar.xml` file, as follows:

```
<session>
    <event_listener_class>
    oracle.toplink.ejb.cmp.demos.sessionlistener
    </event_listener_class>
</session>
```

Specify the fully-qualified name of the class that you want to use for this purpose in the `customization-class` element of the `toplink-ejb-jar.xml` deployment descriptor.

Example 4–34 illustrates the project portion of `the toplink-ejb-jar.xml` deployment descriptor that specifies a `customization class`.

Example 4–34 Customization Class in the toplink-ejb-jar.xml File

```
<session>
    <name>EmployeeDemo</name>
    <project-class>
    oracle.toplink.demos.ejb.cmp.wls.employee.EmployeeProject.class
    </project-class>
    <login>
        <connection-pool>ejbPool</connection-pool>
    </login>
    <customization-class>
    oracle.toplink.demos.ejb.cmp.wls.employee.EmployeeCustomizer
    </customization-class>
</session>
```

Database Session

A database session is the simplest session OracleAS TopLink offers. The database session offers functionality for a single user and a single database connection.

> **Note:** Use server sessions and client sessions for three-tier applications; applications that are built using database sessions may be difficult to migrate to a scalable architecture in the future.

A database session contains and manages the following information:

- An instance of `Project` and `DatabaseLogin`, which stores database login and configuration information
- The JDBC connection and the database access
- The descriptors for each of the application persistent classes
- Identity maps that maintain object identity and act as a cache

Creating a Database Session

An application opens a database session by creating an instance of the `DatabaseSession` class, and initializing the project with the appropriate database login parameters. After initialization, the session:

- Registers the OracleAS TopLink descriptors (see "Registering Descriptors" on page 4-28)

- Connects to the database
- Establishes the session cache

Connecting to the Database

After you register the descriptors, use the `DatabaseSession` class to connect to the database, using the `login()` method. If the login parameters in the `DatabaseLogin` class are incorrect, or if the connection cannot be established, OracleAS TopLink throws a `DatabaseException`.

After a connection is established, the application can use the session to access the database. To test the connection, invoke the `isConnected()` method. If the connection is functions, that method returns TRUE.

To interact with the database, the application use the session querying methods or executes query objects. The interactions between the application and the database are collectively known as the query framework. For more information about querying, see Chapter 6, "Queries" on page 6-1.

Although session query methods work well with database sessions, concurrency issues make the database session unsuited for three-tier applications.

Logging Out of the Database

To log out the session, use the `logout()` method. To disconnect the session from the relational database and flush the session's identity maps, call the `logout()` method.

Because logging in to the database can be time consuming, log out only when all database interactions are complete.

Applications that log out from the database do not have to reregister their descriptors if they log back in to the database.

Using Manual Transaction Control

Certain versions of Sybase JConnect prevent the execution of stored procedures with JDBC auto-commit. If you use OracleAS TopLink with a version of JConnect that causes this problem, use the `handleTransactionsManuallyForSybaseJConnect()` method to handle the transactions manually.

To add transaction processing to a set of database operations:

1. At the start of the transaction set, call `beginTransaction()`.

2. Specify a try-catch block that calls `rollbackTransaction()` if a database exception is thrown.
3. At the end of the transaction set, call `commitTransaction()`.

> **Note:** The Unit of Work is already transaction bound and does not require these calls.

Example 4–35 A Typical Manual Transaction

```
/** Update a group of employee records*/
void writeEmployees(Vector employees, Session session)
{
    Employee employee;
    Enumeration employeeEnumeration = employees.elements();
    try {
        session.beginTransaction();
        while (employeeEnumeration.hasMoreElements())
        {
            employee=(Employee) employeeEnumeration.nextElement();
            session.writeObject(employee);
        }
        session.commitTransaction();
    } catch (DatabaseException exception) {
    // If a database exception has been thrown, roll back the transaction.
        session.rollbackTransaction();
    }
}
```

Creating Database Sessions: Examples

Example 4–36 Creating a Session from a OracleAS TopLink Mapping Workbench Project

```
import oracle.toplink.tools.workbench.*;
import oracle.toplink.sessions.*

// Create the project object
Project project = XMLProjectReader.read("C:\TopLink\example.xml");
DatabaseLogin loginInfo = project.getLogin();
loginInfo.setUserName("scott");
loginInfo.setPassword("tiger");
```

```
//Create a new instance of the session and login
DatabaseSession session = project.createDatabaseSession();
try {
    session.login();
    } catch (DatabaseException exception) {
    throw new RuntimeException("Database error occurred at login: " +
    exception.getMessage());
    System.out.println("Login failed");
    }

/* Do any database interaction using the query framework, transactions or units
of work */
...

// Log out when database interaction is over
session.logout();
Creating and using a session from coded descriptors
import oracle.toplink.sessions.*;

//Create the project object.
DatabaseLogin loginInfo = new DatabaseLogin();
loginInfo.useJDBCODBCBridge();
loginInfo.useSQLServer();
loginInfo.setDataSourceName("MS SQL Server");
loginInfo.setUserName("scott");
loginInfo.setPassword("tiger");
Project project = new Project(loginInfo);

//Create a new instance of the session, register the descriptors, and login
DatabaseSession session = project.createDatabaseSession();
session.addDescriptors(this.buildAllDescriptors());
try {
    session.login();
} catch (DatabaseException exception) {
    throw new RuntimeException("Database error occurred at login: " +
    exeption.getMessage());
    System.out.println("Login failed");
    }

//Do any database interaction using the query framework, transactions or units
of work
...
//Log out when database interaction is over
session.logout();
}
```

Reference

Table 4–12 summarizes the most common public methods for the `DatabaseSession` class. For more information about the available methods for the `DatabaseSession` class, see the *Oracle Application Server TopLink API Reference*.

Table 4–12 Elements for Database Session

Description	Method Name
Construction methods	`Project.createDatabaseSession()`
log in to the database (Defaults to the `user name` and `password` from project login)	`login()`
Log out of the database	`logout()`
Executing predefined queries	`executeQuery(String queryName)`
Executing a query object	`executeQuery(DatabaseQuery query)`
Reading from the database	`readAllObjects(Class domainClass, Expression expression)` `readObject(Class domainClass, Expression expression)`
SQL logging (logging is off by default)	`logMessages()`
Debugging	`printIdentityMaps()`

Transactions	`beginTransaction()`
	`commitTransaction()`
	`rollbackTransaction()`

Exception handlers (throws exceptions by default)	`setExceptionHandler(ExceptionHandler handler)`
JTA/JTS (Defaults to use JDBC transactions)	`setExternalTransactionController(ExternalTransactionController controller)`
Unit of Work	`acquireUnitOfWork()`
Writing to the database	`deleteObject(Object domainObject)` `writeObject(Object domainObject)`

Session Broker

OracleAS TopLink provides the session broker to enable multiple database access. Use the session broker to access objects that are stored on multiple databases. The session broker:

- Provides transparent multiple database access through a single OracleAS TopLink session
- Enables objects to reference objects on other databases
- Transparently manages new object storage in a multiple databases environment
- Manages single Unit of Work and transaction across multiple databases
- Supports two-phase commit when integrated with a compliant JTA driver; otherwise, uses a two stage algorithm

Multiple Sessions

The session broker is a powerful tool that enables you to use data that is split across multiple databases for a given application. An alternative to the session broker is to use multiple sessions to work with multiple databases:

- If the data on each database is unrelated to data on the other databases, and relationships do not cross database boundaries, then you can create a separate session for each database. For example, you might have individual databases and associated sessions dedicated to each cost center.

  This arrangement requires that you to manage each session manually and ensure that the class descriptors for your project reside in the correct session.
- You can use additional sessions to house a regular batch job. In this case, you can create two or more sessions on the same database. In addition to the main session that supports client queries, you create other sessions that support batch inserts at low-traffic times in your system. This enables you to maintain the client cache.

Configuring the Session Broker in Code

After the session broker is set up and logged in, it functions like a session, making multiple database access transparent. Because a session broker is more complex than a regular database session, it is requires more work to create and configure.

Configuring the Session Broker in the Sessions.xml file To configure the session broker in the `sessions.xml` file, configure sessions for use in the session broker, and then reference the sessions from within the `session-broker` element. When the session manager instantiates the session broker, it also instantiates the referenced sessions.

For more information, see "session-broker Element" on page 4-26.

Configuring Session Broker in Java Code Because the session broker references other sessions, configure these sessions before instantiating the session broker. Add all required descriptors to the session, but do not initialize the descriptors, or log the sessions in. The session broker manages these issues when you instantiate it.

After you configure a session, use the `registerSession(String name, Session session)` method to register it with a `SessionBroker`.

Example 4–37 Adding Sessions to a Session Broker

This code prepares and adds two sessions to a session broker.

```
Project p1 = ProjectReader.read(("C:\Test\Test1.project"));
Project p2 = ProjectReader.read(("C:\Test\Test2.project"));

/* modify the user name and password if they are not correct in the
.project file */
p1.getLogin().setUserName("User1");
p1.getLogin().setPassword("password1");
p2.getLogin().setUserName("User2");
p2.getLogin().setPassword("password2");
DatabaseSession session1 = p1.createDatabaseSession();
DatabaseSession session2 = p2.createDatabaseSession();

SessionBroker broker = new SessionBroker();
broker.registerSession("broker1", session1);
broker.registerSession("broker2", session2);

broker.login();
```

When you call the `login()` method on the session broker, session broker logs in all contained sessions and initializes the descriptors in the sessions. After login, the session broker appears and functions as a regular session. OracleAS TopLink handles the multiple database access transparently.

Example 4–38 Writing to the Database

```
UnitOfWork uow = broker.acquireUnitOfWork();
Test test = (Test) broker.readObject(Test.class);
```

```
Test testClone = uow.registerObject(test);
. . .
//change and manipulate the clone and any of its references
. . .
uow.commit();

//log out when finished
broker.logout();
```

Committing a Transaction with a Session Broker

If you use a session broker, incorporate a JTA external transaction controller wherever possible. The external transaction controller provides a *two-phase commit*, which passes the SQL statements that are required to commit the transaction to the JTA driver. The JTA driver handles the entire commit process.

JTA guarantees that the transaction commits or rolls back completely, even if the transaction involves more than one database. If the commit to any one database fails, then all database transactions roll back. The two-phase commit is the safest method available to commit a transaction to the database.

Two-phase commit support requires integration with a compliant JTA driver.

For more information about the JTA drivers, see "JTA" on page 5-8.

Committing a Session without a JTA Driver: Two-stage Commits If there is no JTA driver available, then the session broker provides a *two-stage commit* algorithm. A two-stage commit differs from a two-phase commit in that it only guarantees data integrity up to the point of the final commit of the transaction. If the SQL executes successfully on all databases, but the commit then fails on one database, only the database that experiences the commit failure rolls back.

Although unlikely, this scenario is possible. As a result, if your system does not include a JTA driver and you use a two-stage commit, build a mechanism into your application to deal with this type of potential problem.

Using the Session Broker in a Three-tier Architecture

The session broker operates in a seamless manner in a tree-tier environment. To use session broker in a three-tier application, configure server sessions for the session broker.

Although you can configure your session broker in code as illustrated in Example 4–39, we recommend you use the OracleAS TopLink Sessions Editor to specify a session broker in the `sessions.xml` file.

For more information, see the *Oracle Application Server TopLink Mapping Workbench User's Guide.*

Example 4–39   Configuring a Session Broker in a Three-Tier Architecture in Java Code

```
Project p1 = ProjectReader.read(("C:\Test\Test1.project"))

Project p2 =
ProjectReader.read(("C:\Test\Test2.project"));

/* Create Sessions for the SessionBroker */
Server sSession1 = p1.createServerSession();
Server sSession2 = p2.createServerSession();


/* Create the SessionBroker and assign the sessions to it */
SessionBroker broker = new SessionBroker();
broker.registerSession("broker1", sSession1);
broker.registerSession("broker2", sSession2);
broker.login();
```

Clients with a Three-Tier Session Broker When a three-tier session broker application uses server sessions to communicate with the database, clients require a client session to write to the database. Similarly, when you implement a session broker, the client requires a *client session broker* to write to the database.

A client session broker is a collection of client sessions, one from each server session associated with the session broker. When a client acquires a client session broker, the session broker collects one client session from each associated server session, and wraps the client sessions so that they appear as a single client session to the client application.

To request a client session broker, the client calls the `acquireClientSessionBroker()` method.

Example 4–40   Sample Client Request Code

```
Session clientBroker = broker.acquireClientSessionBroker();
```

Limitations

Using the session broker is not the same as linking databases at the database level. If your database allows linking, use that functionality to provide multiple database access.

The session broker has the following limitations:

- You cannot split multiple table descriptors across databases.
- Each class must reside on only one database.
- You cannot use joins through expressions across databases.
- Many-to-many join tables and direct collection tables must reside on the same database as the source object.

> **Note:** The "Advanced Use" section describes a work-around for this limitation. It uses an amendment to the descriptor.

Advanced Use

Many-to-many join tables and direct collection tables must be on the same database as the source object, because reading these tables requires a join that spans both databases. To get around this problem, use the `setSessionName(String sessionName)` method on `ManyToManyMapping` and `DirectCollectionMapping`. This method indicates that the join table or direct collection table is on the same database as the target table.

```
Descriptor desc = session1.getDescriptor(Employee.class);
((ManyToManyMapping)desc.getObjectBuilder().getMappingForAttributeName("projects
")).setSessionName("broker2");
```

`DatabaseQuery` offers a similar method that supports non object queries.

Reference

Table 4–13 summarizes the most common public methods for `SessionBroker`. For more information about the available methods for `SessionBroker`, see the *Oracle Application Server TopLink API Reference*.

Table 4–13 Elements for the Session Broker

Element	Method Name
Writing objects	`acquireUnitOfWork()`

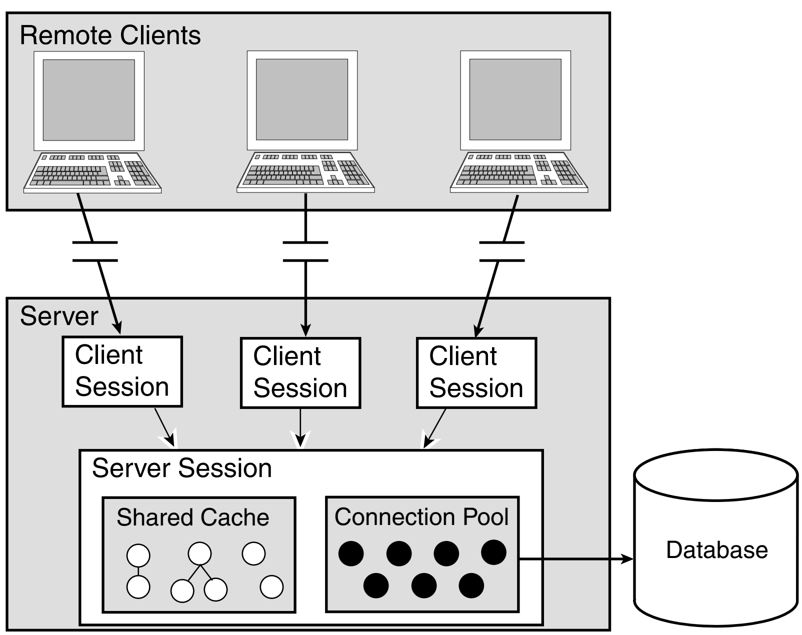
Table 4–13 Elements for the Session Broker (Cont.)

Element	Method Name
Acquiring `ClientSessions`	`acquireClientSessionBroker()`
Database connection	`login()` `logout()`

Remote Session

A remote session is a session that resides on the client. It communicates with a client session on the server, and the client session communicates with the server session on its behalf. Remote sessions handle object identity, proxies, and the communication between the client and server layer.

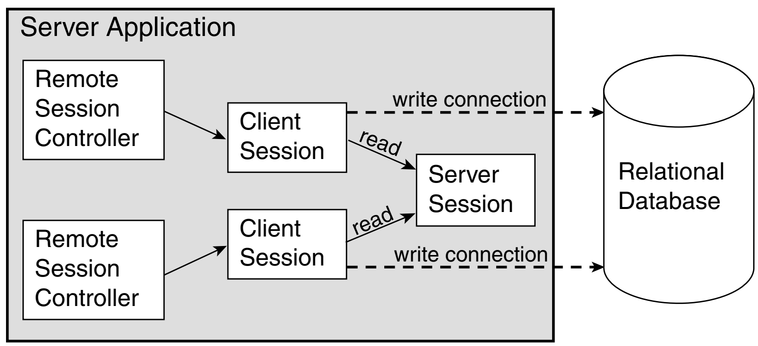
Figure 4–9 Remote Session Model for a Three-tier Application



The remote session can also interact with a database session rather than a client session. The user sets this up on the server side.

When choosing between a client session and a database session, you should be aware that the database session is not suited to a distributed environment, because the database session enables only one user to interact with the database. However, if the remote session interacts with a client session, then multiple remote sessions can share a single database connection. The remote session also benefits from connection pooling.

Figure 4–10 Remote Session and a Database or a Client Session



Architectural Overview

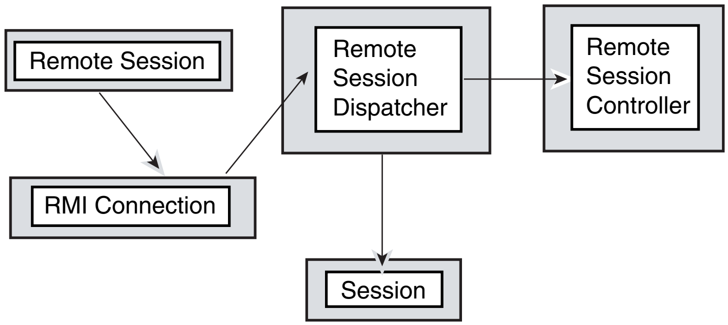
The remote session model consists of the following layers (also see Figure 4–11):

- The application layer—a client side application talking to remote session
- The transport layer—a communication layer, RMI or RMI-IIOP
- The server layer—OracleAS TopLink session communicating with a database

The request from the client application to the server travels down through the layers of a distributed system. A client that makes a request to the server session uses the remote session as a conduit to the server session. The client references the remote session, and the remote session forwards a request to the server session through the transport layer.

At runtime, the remote session builds its knowledge base by reading descriptors and mappings from the server side as they are needed. These descriptors and mappings are lightweight because not all information is passed on to the remote session. The information needed to traverse an object tree and to extract primary keys from the given object is passed with the mappings and descriptors.

Figure 4–11 An Architectural Overview of the Remote Session



Application layer The application layer includes the application and the remote session. The remote session is a subclass of the session and maintains all the public protocols of the session, giving the appearance of working with the local database session.

The remote session maintains its own identity map and a hash table of all the descriptors read from the server. If the remote session can handle a request by itself, the request is not passed to the server. For example, a request for an object that is in the Remote session cache is processed by the remote session. However, if the object is not in the remote session cache, the request passes to the server session.

Transport Layer The transport layer is responsible for carrying the semantics of the invocation. It is a layer that hides all the protocol dependencies from the application and server layer.

The transport layer includes a remote connection that is an abstract entity through which all requests to the server are forwarded. Each remote session maintains a single remote connection that marshals and unmarshals all requests and responses on the client side.

The remote session supports communications over RMI and CORBA. It includes deployment classes and stubs for RMI, BEA WebLogic RMI, VisiBroker, OrbixWeb, BEA WebLogic EJB, and Oracle 10i EJB.

Server Layer The server layer includes a remote session controller dispatcher and a session. The remote session controller dispatcher marshals and unmarshals all responses and requests from the server side. This is a client side component.

The remote session controller dispatcher is an interface between the session and transport layers. It hides the specifics of the transport layer from the session.

Securing Remote Session Access

The remote session represents a potential security risk because it requires you to register a remote session controller dispatcher as a service that anyone can access. This can expose the entire database.

To reduce this threat, run a server manager as a service to hold the remote controller session dispatcher. All the clients must then communicate through the server manager, which implements the security model for accessing the remote session controller dispatcher.

On the client side, the user requests the remote session controller dispatcher. The manager returns a remote session controller dispatcher only if the user has access rights according to the security model built into the server manager.

To access the system, the remote session controller dispatcher on the client side creates a remote connection, and acquires remote session from the remote connection. The API for the remote session is the same as for the session, and there is no user-visible difference between working on a session or a remote session.

Queries

Read queries are publicly available on the client side, but queries that modify objects must be performed using the Unit of Work.

Refreshing

Calling refresh methods on the remote session causes database reads, and may also cause cache updates if the data being refreshed is modified in the database. This can lead to poor performance.

To improve performance, configure refresh methods to run against the server session cache, by configuring the descriptor to always remotely refresh the objects in the cache on all queries. This technique ensures that all queries against the remote session refresh the objects from the server session cache, without the database access.

Cache hits on remote sessions still occur on read object queries based on the primary keys. If you want to avoid this, disable the remote session cache hits on read object queries based on the primary key.

Example 4–41 Refreshing on the Server Session Cache

```
// Get the PolicyHolder descriptor
Descriptor holderDescriptor = remoteSession.getDescriptor(PolicyHolder.class);

// Set refresh on the ServerSession cache
holderDescriptor.alwaysRefreshCachedOnRemote();

// Disable remote cache hits, ensure all queries go to the ServerSession cache
holderDescriptor.disableCacheHitsOnRemote();
```

Indirection

The remote session supports indirection objects. An indirection object is a valueholder that can be invoked remotely on the client side. When invoked, the valueholder first checks to see if the requested object exists on the remote session. If not, then the associated valueholder on the server is instantiated to get the value that is then passed back to the client. Remote valueholders are used automatically; the application's code does not change.

Cursored Streams

Remote session supports cursored streams, but not scrollable cursors.

For more information about enabling cursored streams, see "Java Streams" on page 6-60.

Unit of Work

Use a Unit of Work acquired from the remote session to modify objects on the database. A Unit of Work acquired from the remote session offers the user the same functionality as a Unit of Work acquired from the client session or the database session.

Creating a Remote Connection Using RMIConnection

Example 4–42 and Example 4–43 demonstrate how to create a remote OracleAS TopLink session on a client that communicates with a remote session controller on a server that uses RMI. After creating the connection, the client application uses the remote session as it does with any other OracleAS TopLink session.

These examples assume that a class called `RMIServerManager` exists on the server. It is not an OracleAS TopLink-enabled class. This class has a method that instantiates and returns an `RMIRemoteSessionController` (an OracleAS TopLink server side interface).

Example 4–42 Client Acquiring RMIRemoteSessionController from Server

The client-side code gets a reference to the `RMIServerManager` and uses this code to get the `RMIRemoteSessionController` running on the server. The reference to the session controller is then used to create the `RMIConnection` from which it acquires a remote session.

```
RMIServerManager serverManager = null;
// Set the client security manager
try {
    System.setSecurityManager(new RMISecurityManager());
    } catch(Exception exception) {
    System.out.println("Security violation " + exception.toString());
}
// Get the remote factory object from the Registry
try {
    serverManager = (RMIServerManager) Naming.lookup("SERVER-MANAGER");
    } catch (Exception exception) {
    System.out.println("Lookup failed " + exception.toString());
}
// Start RMIRemoteSession on the server and create an RMIConnection
RMIConnection rmiConnection = null;
try {
    rmiConnection = new
    RMIConnection(serverManager.createRemoteSessionController());
    } catch (RemoteException exception) {
    System.out.println("Error in invocation " + exception.toString());
}
// Create a remote session which we can then use as a normal OracleAS TopLink
Session
Session session = rmiConnection.createRemoteSession();
```

Example 4–43 Server Creating RMIRemoteSessionController for Client

The `RMIServerManager` uses this code to create and return an instance of an `RMIRemoteSessionController` to the client. The controller sits between the remote client and the local OracleAS TopLink session.

```
RMIRemoteSessionController controller = null;
```

```
try {
    /* Create instance of RMIRemoteSessionControllerDispatcher which implements
    RMIRemoteSessionController. The constructor takes an OracleAS TopLink
    session as a parameter */
    controller = new RMIRemoteSessionControllerDispatcher (localTopLinkSession);
}
catch (RemoteException exception) {
    System.out.println("Error in invocation " + exception.toString());
}
return controller;
```

Sessions and the Cache

OracleAS TopLink automatically caches any data that is returned when a client reads an object. The cache resides with the session, which enables any associated client sessions to share the cache. This cache plays an important role in the performance of your application.

To define how the cache manages objects, specify a strategy for cache management in the OracleAS TopLink Mapping Workbench.

For more information, see "Working with Identity Maps" in the *Oracle Application Server TopLink Mapping Workbench User's Guide.*

Session Utilities

The OracleAS TopLink session provides several utilities to test and troubleshoot your application. This section introduces these tools and describes techniques for using them:

- Logging SQL and Messages
- Using the Profiler
- Using the Integrity Checker
- Using Exception Handlers

Logging SQL and Messages

OracleAS TopLink accesses the database using SQL strings that it generates internally. This feature enables applications to use the session methods or query objects without having to perform their own SQL translation.

If, for debugging purposes, you want to review a record of the SQL that is sent to the database, sessions provide these methods to log generated SQL to a writer. OracleAS TopLink disables SQL and message logging by default. To enable it, use the `logMessages()` method on the session. The default writer is a stream writer to `System.out`, but you can configure the log destination using the `setLog()` method on the session.

The session logs:

- Debug print statements
- Exceptions/error messages sent to system out
- Any other output sent to the system log

Logging Chained Exceptions

The logging chained exception facility enables you to log causality when one exception causes another as part of the standard stack back-trace. If you build your applications with JDK 1.4, causal chains appear automatically in your logs.

Logging and the Oracle Enterprise Manager

You can view OracleAS TopLink logs with all the other Oracle Application Server 10*g* log files using the Oracle Enterprise Manager.

For more information, see "Managing Diagnostic Log Files" in the *Oracle Application Server 10g Administrator's Guide*.

If you install OracleAS TopLink in the same Oracle Home directory as Oracle Application Server, OracleAS TopLink logs appear automatically with the other Oracle Application Server component log files in the Oracle Enterprise Manager. If you install OracleAS TopLink in a different Oracle Home directory, use the following procedure:

1. Locate the `toplink.xml` file in the `<*ORACLE_HOME*>\toplink\config\` directory.
2. Ensure that the `log path` tag reflects the location of your OracleAS TopLink log file, and is properly configured.

   For example:

   ```
   - <log path="toplink/config/toplink.log"
      componentId="TOPLINK" encoding="utf-8">
   ```

3. Copy the `toplink.xml` file to the following directory:

```
<ORACLE_HOME>\diagnostics\config\registration\
```

Using the Profiler

The OracleAS TopLink Profiler is a high-level logging service. Instead of logging SQL statements, the Profiler logs a summary of each query you execute. The summary includes a performance breakdown of the query that enables you to identify performance bottlenecks. The Profiler also provides a report summarizing the query performance for an entire session.

Access Profiler reports and profiles through the **Profile** tab in the OracleAS TopLink Web Client, or create your own application or applet to view the Profiler logs.

For more information about the Web Client, see "OracleAS TopLink — Web Client" on page A-2.

Using the Integrity Checker

When you connect a session or add descriptors to a session after connection, OracleAS TopLink initializes and validates the descriptor information. The integrity checker allows you to customize the validation process. The integrity checker offers the following configuration options:

Catch All Exceptions This option specifies whether or not the integrity checker catches all exceptions in the session. The settings for this option are `catchExceptions` (the default setting), and `dontcatchExceptions`.

Catch Instantiation Policy Exceptions This option catches only errors that are associated with instantiation policy, and:

- Throws the first error that it encounters, including the error's stack trace.
- Validates the state of the database schema to ensure that it matches the information in the descriptors.
- Disables the instance creation check.

Example 4–44 Using the Integrity Checker

```
session.getIntegrityChecker().checkDatabase();
session.getIntegrityChecker().catchExceptions();
session.getIntegrityChecker().dontCheckInstantiationPolicy();
```

```
session.login();
```

Using Exception Handlers

Exception handlers process database exceptions, generally to process connection timeouts or database failures. To use exception handlers, register an implementor of the `ExceptionHandler` interface with the session. If a database exception occurs during the execution of a query, the exception passes to the exception handler. The exception handler then either handles the exception, retries the query, or throws an unchecked exception.

For more information about exceptions, see Appendix C, "Error Codes and Messages".

Example 4–45 Implementing an Exception Handler

```
session.setExceptionHandler(newExceptionHandler(){
    public Object handleException(RuntimeException exception) {
        if ((exception instanceof DatabaseException) &&
            (exception.getMessage().equals("connection reset by peer."))) {
            DatabaseException dbex = (DatabaseException) exception;
            dbex.getAccessor().reestablishConnection
            (dbex.getSession());
            return dbex.getSession().executeQuery(dbex.getQuery());
        }
        throw exception;
    }
});
```

Note: Unhandled exceptions must be re-thrown by the handler code.

Customizing Session Events

Sessions, such as database sessions, Unit of Work, client sessions, server sessions, and remote sessions raise *session events* for most session operations. Session events help you debug or coordinate the actions of multiple sessions.

This section illustrates how you customize session events, and discusses:

- Session Event Listeners
- Session Event Manager

- Implementing Events Using Java

Session Event Listeners

One approach to customizing session events is to create session event listeners that detect and respond to session events. To register objects as listeners for session events, implement the `SessionEventListener` interface and register it with the `SessionEventManager` using `addListener()`.

Table 4–14 Session Event Manager Events

Event	Description
`PreExecuteQuery`	Raised before the execution of every query on the session
`PostExecuteQuery`	Raised after the execution of every query on the session
`PreBeginTransaction`	Raised before a database transaction starts
`PostBeginTransaction`	Raised after a database transaction starts
`PreCommitTransaction`	Raised before a database transaction commits
`PostCommitTransaction`	Raised after a database transaction commits
`PreRollbackTransaction`	Raised before a database transaction rolls back
`PostRollbackTransaction`	Raised after a database transaction rolls back
`PreLogin`	Raised before the Session initializes and acquires connections
`PostLogin`	Raised after the Session initializes and acquires connections

Table 4–15 Unit of Work Events

Event	Description
`PostAcquireUnitOfWork`	Raised after a `UnitOfWork` is acquired
`PreCommitUnitOfWork`	Raised before a `UnitOfWork` commits
`PrepareUnitOfWork`	Raised after the a `UnitOfWork` flushes its SQL, but before it commits its transaction
`PostCommitUnitOfWork`	Raised after a `UnitOfWork` commits
`PreReleaseUnitOfWork`	Raised on a `UnitOfWork` before it releases
`PostReleaseUnitOfWork`	Raised on a `UnitOfWork` after it releases
`PostResumeUnitOfWork`	Raised on a `UnitOfWork` after it resumes

Table 4–16 Server Session and Client Session Events (Three-Tier Applications)

Event	Description
`PostAcquireClientSession`	Raised after a `ClientSession` is acquired
`PreReleaseClientSession`	Raised before releasing a `ClientSession`
`PostReleaseClientSession`	Raised after releasing a `ClientSession`
`PostConnect`	Raised after connecting to the database
`PostAcquireConnection`	Raised after acquiring a connection
`PreReleaseConnection`	Raised before releasing a connection

Table 4–17 Database Access Events

Event	Description
`OutputParametersDetected`	Raised after a stored procedure call with output parameters executes. This event enables you to retrieve a result set and output parameters from a single stored procedure.
`MoreRowsDetected`	Raised when a ReadObjectQuery detects more than one row returned from the database. This event can indicate a possible error condition in your application.

Session Event Manager

The session event manager handles information about session events. Applications register listeners with the session event manager to receive session event data.

Example 4–46 Registering a Listener

```
public void addSessionEventListener(SessionEventListener listener)
{
    // Register specified listener to receive events from mySession
    mySession.getEventManager().addListener(listener);
}
```

Example 4–47 Using the Session Event Adapter to Listen for Specific Session Events

```
...
    SessionEventAdapter myAdapter = new SessionEventAdapter() {
        // Listen for PostCommitUnitOfWork events
        public void postCommitUnitOfWork(SessionEvent event) {
        // Call my handler routine
```

```
        unitOfWorkCommitted();
    }
};
mySession.getEventManager().addListener(myAdapter);
...
```

Implementing Events Using Java

You can implement custom events and event handlers in Java code. The code in Example 4–48 checks for lock conflicts when the application builds an instance of `Employee` from information in the database.

Example 4–48 Implementing an Event in Code

```
/*In the employee class, declare the event method which will be invoked when the
event occurs */
public void postBuild(DescriptorEvent event) {
    // Uses object row to integrate with some application level locking service.
    if ((event.getRow().get("LOCKED")).equals("T")) {
        LockManager.checkLockConflict(this);
   }
}
```

OracleAS TopLink Support for Java Data Objects (JDO)

Java Data Objects (JDO) is an API for transparent database access. The JDO architecture is a standard API for data, both in local storage systems and enterprise information systems. It unifies access to heterogeneous systems, such as mainframe transaction processing, and database systems. JDO enables programmers to create Java code that accesses the underlying data store transparently and does not require database-specific code.

OracleAS TopLink provides basic JDO support based on the JDO specification. OracleAS TopLink support includes much of the JDO API, but does not require you to enhance or modify the class to leverage JDO.

This section includes information on:

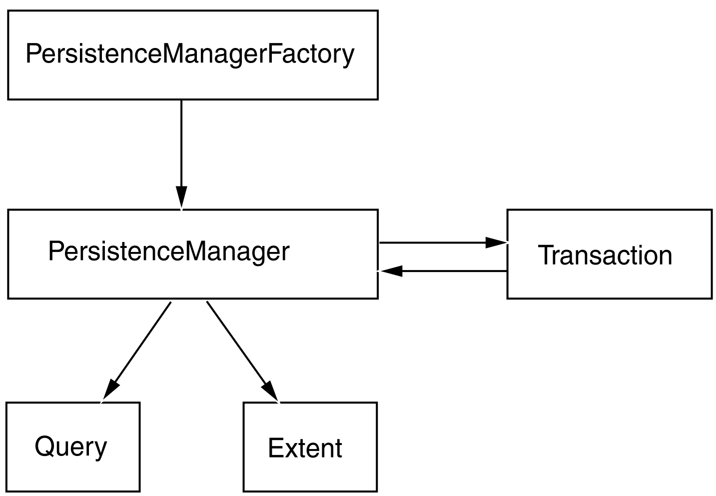
- Understanding the JDO API
- JDO Implementation
- Running the OracleAS TopLink JDO Example

Understanding the JDO API

The JDO API includes four main interfaces:

- The *PersistenceManagerFactory* is a factory that generates `PersistenceManagers`. It has a configuration and login API.
- The *PersistenceManager* is the main point of contact from the application. It provides an API for accessing the transaction, queries, and object life cycle API (`makePersistent`, `makeTransactional`, `deletePersistent`).
- The *Transaction* defines a basic begin, commit, rollback API.
- The *Query* defines the API to configure the query (filter, ordering, parameters, and variables) and to execute the query.

Figure 4–12 Understanding the JDO API



JDO Implementation

OracleAS TopLink implements the `PersistenceManagerFactory`, `PersistenceManager`, and `Transaction` interfaces, and extends the query functionality to include the complete OracleAS TopLink query framework.

JDOPersistenceManagerFactory

To create a `JDOPersistenceManagerFactory`, call the constructor and include a session name string, or an OracleAS TopLink session or project. If you construct the factory from a project, then OracleAS TopLink creates a new database session and attaches it to the `PersistenceManager` every time you obtain the `PersistenceManager` with the `getPersistenceManager` method.

The `PersistenceManager` is not multi-threaded. In a multi-threaded application, assign each thread its own `PersistenceManager`. In addition, construct the `JDOPersistenceManagerFactory` from a server session, rather than a database session or project. Doing this enables you to use the lightweight client session and more scalable connection pooling.

Creating a JDOPersistenceManagerFactory Example 4–49 illustrates how to create a factory from an OracleAS TopLink session named `jdoSession`. A session manager manages a singleton instance of the OracleAS TopLink server session or database session.

For more information, see "Session Manager" on page 4-29.

Example 4–49 Creating a JDOPersistenceManagerFactory

```
JDOPersistenceManagerFactory factory= new
JDOPersistenceManagerFactory("jdoSession");
/*Create a persistence manager factory from an instance of OracleAS TopLink
ServerSession or DatabaseSession that is managed by the user */
ServerSession session = (ServerSession) project.createServerSession();
JDOPersistenceManagerFactory factory= new JDOPersistenceManagerFactory(session);
/* Create a persistence manager factory with ties to a DatabaseSession that is
created from OracleAS TopLink project */
JDOPersistenceManagerFactory factory= new JDOPersistenceManagerFactory(new
EmployeeProject());
```

Obtaining PersistenceManager To create new PersistenceManagers, call the `getPersistentManager` method. If you construct the factory from a Project instance, use the `getPersistentManager(String userid, String password)` method to configure the `userid` and `password`.

Reference Table 4–18 summarizes the most common public methods for `PersistenceManagerFactory`. For more information about the available methods for `PersistenceManagerFactory`, see the *Oracle Application Server TopLink API Reference*.

Table 4–18 Elements for Persistence Manager Factory

Method Name	Description
`JDOPersistenceManagerFactory() persistence`	Constructs a factory from a session manager session
`JDOPersistenceManagerFactory(String sessionName)`	Constructs a factory from the named session
`JDOPersistenceManagerFactory(Session session)`	Constructs a factory from a user session
`JDOPersistenceManagerFactory(Project project)`	Constructs a factory from a project
`getIgnoreCache()setIgnoreCache` `(boolean ignoreCache)`	Query mode that specifies whether cached instances are considered when evaluating the filter expression. The default is set to FALSE.
`getNontransactionalRead()setNontransactionalRead` `(boolean nontransactionalRead)`	Transaction mode that allows you to read instances outside a transaction. The default is set to FALSE.

`getConnectionUserName()`
`setConnectionUserName(String userName)`
`getConnectionPassword()`
`setConnectionPassword(String password)`
`getConnectionURL()`
`setConnectionURL(String URL)`
`getConnectionDriverName()`
`setConnectionDriverName(String driverName)`
	Available settings if the factory is constructed from an OracleAS TopLink project.
	Derives the default user name, password, URL, driver from project login.

`getPersistenceManager()` `getPersistenceManager(String userid, String password)`	Accesses `PersistenceManager`, and sets the user ID and password if the factory is constructed from an OracleAS TopLink project (uses default values in the absence of a project). Derives the default user ID, password from session login, or project login.
`getProperties()`	Nonconfigurable properties
`supportedOptions()`	Collection of supported option String

JDOPersistenceManager

The `JDOPersistenceManager` class is the factory for the `Query` interface and contains methods to access transactions, and manage the persistent life cycle instances.

Inserting JDO objects To make new JDO objects persistent, use the `makePersistent()` or `makePersistentAll()` method. If you do not manually begin the transaction, then OracleAS TopLink begins and commits the transaction when you invoke either `makePersistent()` or `makePersistentAll()`. If the object is already persisted, then calling these methods has no effect.

Example 4–50 Persist a New Employee Named Bob Smith

```
Server serverSession = new EmployeeProject().createServerSession();
PersistenceManagerFactory factory = new
JDOPersistenceManagerFactory(serverSession);
PersistenceManager manager = factory.getPersistenceManager();
Employee employee = new Employee();
employee.setFirstName("Bob");
employee.setLastName("Smith");
manager.makePersistent(employee);
```

Updating JDO Objects To modify JDO objects within a transaction context, begin and commit a transactional object manually. A transactional object is an object that is subject to the transaction boundary. Use one of the following methods to obtain transactional objects:

- Using `getObjectById()`
- Executing a transactional-read query
- Using the OracleAS TopLink extended API `getTransactionalObject()`

OracleAS TopLink executes the transactional-read query when the `nontransactionalRead` flag of the current transaction is false. To obtain the current transaction from the `PersistenceManager`, call `currentTransaction()`.

Example 4–51 Update an Employee

This example illustrates how to add a new phone number to an employee object, modify the address, and increase the salary by 10 percent.

```
Transaction transaction = manager.currentTransaction();
if(!transaction.isActive()) {
```

```
transaction.begin();
}
// Get the transactional instance of the employee
Object id = manager.getTransactionalObjectId(employee);
Employee transactionalEmployee = manager.getObjectById(id, false);
transactionalEmployee.getAddress().setCity("Ottawa");
transactionalEmployee.setSalary((int) (employee.getSalary() * 1.1));
transactionalEmployee.addPhoneNumber(new PhoneNumber("fax", "613", "3213452"));

transaction.commit();
```

Deleting Persistent Objects To delete JDO objects, use either `deletePersistent()` or `deletePersistentAll()`. The objects need not be transactional. If you do not manually begin the transaction, then OracleAS TopLink begins and commits the transaction when you invoke either `deletePersistent ()` or `deletePersistentAll ()`.

Deleting objects using `deletePersistent()` or `deletePersistentAll()` is similar to deleting objects using a Unit of Work. When you delete an object, you also automatically delete its privately owned parts, because they cannot exist without their owner. At commit time, OracleAS TopLink generates SQL to delete the objects, taking database constraints into account.

When you delete an object, set references to the deleted object to null or remove them from the collection, and modify references to the object using its transactional instance. This ensures that the object model reflects the change.

Example 4–52 Deleting a Team Leader from a Project

```
Transaction transaction = manager.currentTransaction();
if(!transaction.isActive()) {
transaction.begin();
}
Object id = manager.getTransactionalObjectId(projectNumber);
Project transactionalProject = (Project) manager.getObjectById(id);
Employee transactionalEmployee = transactionalProject.getTeamLeader();
// Remove team leader from the project
transactionalProject.setTeamLeader(null);
// Remove owner that is the team leader from phone numbers
for(Enumeration enum = transactionalEmployee.getPhoneNumbers().elements();
enum.hasMoreElements();) {
     ((PhoneNumber) enum.nextElement()).setOwner(null);
}
manager.deletePersistent(transactionalEmployee);
```

```
transaction.commit();
```

Example 4–53 Deleting a Phone Number

```
Transaction transaction = manager.currentTransaction();
if(!transaction.isActive()) {
transaction.begin();
}
Object id = manager.getTransactionalObjectId(phoneNumber);
PhoneNumber transactionalPhoneNumber = (PhoneNumber) manager.getObjectById(id);
transactionalPhoneNumber.getOwner().getPhoneNumbers().remove(transactionalPhoneN
umber);
manager.deletePersistent(phoneNumber);
transaction.commit();
```

Obtaining Query OracleAS TopLink does not support the JDO Query language, but includes support within JDO for the more advanced OracleAS TopLink query framework.

For more information about the OracleAS TopLink query framework, see Chapter 6, "Queries".

A key difference is that the JDO query language requires returned results to be a collection of candidate JDO instances (either a `java.util.Collection`, or an Extent). Conversely, the return type in OracleAS TopLink depends on the type of query. For example, if you use a `ReadAllQuery`, the result is a `Vector`.

The following APIs support for the query factory:

- Standard API:

  ```
  newQuery();
  newQuery(Class persistentClass);
  ```

- OracleAS TopLink extended API:

  ```
  newQuery(Class persistentClass, Expression expressionFilter);
  ```

> **Note:** If you obtain Query from a different `newQuery()` API, this can result in a `JDOUserException`, or the creation of the query from the supported API.

You create a `ReadAllQuery` with the query instance by default.

Reference Table 4–19 and Table 4–20 summarize the most common public methods for the Query API and OracleAS TopLink extended API. For more information about the available methods for the Query API and OracleAS TopLink extended API, see the *Oracle Application Server TopLink API Reference*.

Table 4–19 Elements for Query API

Method Name	Description
`close()`	Releases resource to allow garbage collection
`currentTransaction()`	Specifies current transaction

`deletePersistent`
`(Object object)deletePersistentAll`
`(Collection objects)deletePersistentAll`
`(java.lang.Object[] objects)`
	Deletes objects

`evict(Object object)evictAll()evictAll` `(Collection objects)evictAll(Object[] objects)`	Marks objects as no longer needed in the cache
`getExtent(Class queryClass, boolean readSubclasses)`	Specifies extent
`getIgnoreCache()setIgnoreCache(boolean ignoreCache)`	Sets cache mode for queries. The default is set to ignore cache from the persistence manager factory.

`getObjectById`
`(Object object, boolean validate)`
`getTransactionalObjectId(Object object)`
	Obtains transactional state of object

`isClosed()`	Closes the `PersistenceManager` instance

`makePersistent(Object object)`
`makePersistentAll`
`(Collection objects)makePersistentAll`
`(Object[] objects)`
	Inserts persistent objects

Table 4–19 Elements for Query API (Cont.)

Method Name	Description
`makeTransactional (Object object)makeTransactionalAll (Collection objects)makeTransactionalAll (Object[] objects)`	Registers objects to Unit of Work, making them subject to transactional boundaries
`newQuery()newQuery(Class queryClass)`	Creates new query factory
`refresh(Object object)refreshAll()refreshAll (Collection objects)refreshAll(Object[] objects)`	Refreshes objects

Table 4–20 Elements for OracleAS TopLink Extended API

Method Name	Description
`getTransactionalObject(Object object)`	Obtains transactional object
`newQuery(Class queryClass, Expression expression)`	Creates query factory
`readAllObjects(Class domainClass)readAllObjects (Class domainClass)readObject (Class domainClass, Expression expression)`	Reads objects

JDOQuery

The `JDOQuery` class implements the `JDOQuery` interface. It defines the API to configure the query (filter, ordering, parameters, and variables) and to execute the query. OracleAS TopLink extends the query functionality to include the full OracleAS TopLink query framework.

For more information about the OracleAS TopLink query framework, see Chapter 6, "Queries".

Users can customize the query to use advanced features, such as batch reading, stored procedure calls, partial object reading, and query by example. OracleAS TopLink does not support the JDO query language, but users can employ either SQL or EJB QL in the `JDOQuery` interface.

Each `JDOQuery` instance is associated with an OracleAS TopLink query. To obtain a `JDOQuery` from the `PersistenceManager`, call a supported `newQuery` method. OracleAS TopLink creates a new `ReadAllQuery` and associates it with the query.

Call `asReadObjectQuery()`, `asReadAllQuery()`, or `asReportQuery` to set the JDO Query OracleAS TopLink query to a specific type.

Customizing the Query Using the OracleAS TopLink Query Framework The OracleAS TopLink query framework provides most of its functionality as a public API. To create a customized OracleAS TopLink query and associate it with the JDO Query, call the `setQuery()` method to build complex functionality into your queries.

Customized OracleAS TopLink queries give you the complete functionality of the OracleAS TopLink query framework. For example, use a `DirectReadQuery` with custom SQL to read the ID column of the employee.

Note: OracleAS TopLink extended APIs support a specific OracleAS TopLink query type. To avoid exceptions, match the API to the correct query type. See Table 4–21 for correct usage.

Example 4–54 Use a ReadAllQuery to Read All Employees Who Live in New York

```
Expression expression = new
ExpressionBuilder().get("address").get("city").equal("New York");
Query query = manager.newQuery(Employee.class, expression);
Vector employees = (Vector) query.execute();
```

Example 4–55 Use a ReadObjectQuery to Read the Employee Named Bob Smith

```
Expression exp1 = new ExpressionBuilder().get("firstName").equal("Bob");
Expression exp2 = new ExpressionBuilder().get("lastName").equal("Smith ");
JDOQuery jdoQuery = (JDOQuery) manager.newQuery(Employee.class);
jdoQuery.asReadObjectQuery();
jdoQuery.setFilter(exp1.and(exp2));
Employee employee = (Employee) jdoQuery.execute();
```

Example 4–56 Use a ReportQuery to Report Employee's Salary

```
JDOQuery jdoQuery = (JDOQuery) manager.newQuery(Employee.class);
jdoQuery.asReportQuery();
jdoQuery.addCount();
jdoQuery.addMinimum("min_salary",jdoQuery.getExpressionBuilder().get("salary"));
jdoQuery.addMaximum("max_salary",jdoQuery.getExpressionBuilder().get("salary"));
jdoQuery.addAverage("average_salary",jdoQuery.getExpressionBuilder().get("salary"));
// Return a vector of one DatabaseRow that contains reported info
Vector reportQueryResults = (Vector) jdoQuery.execute();
```

Example 4–57 Use a Customized DirectReadQuery to Read Employee 's id column

```
DirectReadQuery TopLinkQuery = new DirectReadQuery();
topLinkQuery.setSQLString("SELECT EMP_ID FROM EMPLOYEE");
JDOQuery jdoQuery = (JDOQuery) manager.newQuery();
jdoQuery.setQuery(topLinkQuery);
// Return a Vector of DatabaseRows that contain ids
Vector ids = (Vector)jdoQuery.execute(query);
```

Reference Table 4–21 and Table 4–22 summarize the most common public methods for the JDO Query API and OracleAS TopLink extended API. For more information about the available methods for the JDO Query API and OracleAS TopLink extended API, see the *Oracle Application Server TopLink API Reference*.

Table 4–21 Elements for JDO Query API

Method Name	Description
`close(Object queryResult)`	Closes cursor result
`declareParameters(String parameters)`	Declares query parameters
`execute()execute(Object arg1)execute (Object arg1, Object arg2)execute (Object arg1, Object arg2, Object arg3)executeWithArray (java.lang.Object[] arg1)executeWithMap(Map arg1)`	Executes query
`getIgnoreCache()setIgnoreCache(boolean ignoreCache)`	Sets cache mode for query result
`getPersistenceManager()`	`PersistenceManager`
`setClass(Class queryClass)`	`ReadObjectQuery, ReadAllQuery, ReportQuery`
`setOrdering(String ordering)`	`ReadAllQuery`

Table 4–22 Elements for OracleAS TopLink Extended JDO API

Method Name	Description
`asReadAllQuery()asReadObjectQuery()` `asReportQuery()`	Converts the query
`getQuery()setQuery(DatabaseQuery newQuery)`	Accesses the OracleAS TopLink query. The default is set to `ReadAllQuery`.

`acquireLocks()`
`acquireLocksWithoutWaiting()`
`addJoinedAttribute(String attributeName)`
`addJoinedAttribute(Expression attributeExpression)`
`addPartialAttribute(String attributeName)`
`addPartialAttribute(Expression attributeExpression)`
`checkCacheOnly()`
`dontAcquireLocks()`
`dontRefreshIdentityMapResult()`
`dontRefreshRemoteIdentityMapResult()`
`getExampleObject()`
`getExpressionBuilder()`
`setQueryByExampleFilter(Object exampleObject)`
`setQueryByExamplePolicy(QueryByExamplePolicy newPolicy)`
`setShouldRefreshIdentityMapResult(boolean shouldRefreshIdentityMapResult)`
`shouldRefreshIdentityMapResult()`
	`ReadObjectQuery, ReadAllQuery, ReportQuery`

`checkCacheByExactPrimaryKey()`
`checkCacheByPrimaryKey()`
`checkCacheThenDatabase()`
`conformResultsInUnitOfWork()`
`getReadObjectQuery()`
	`ReadObjectQuery`

Table 4–22 Elements for OracleAS TopLink Extended JDO API (Cont.)

Method Name	Description
addAscendingOrdering(String queryKeyName)	ReadAllQuery
addDescendingOrdering(String queryKeyName)	
addOrdering(Expression orderingExpression)	
addBatchReadAttribute(String attributeName)	
addBatchReadAttribute(Expression	
attributeExpression)	
addStandardDeviation(String itemName)	
addStandardDeviation(String itemName,Expression attributeExpression)	
addSum(String itemName)	
addSum(String itemName, Expression attributeExpression)	
addVariance(String itemName)	
addVariance(String itemName, Expression	
attributeExpression)	
getReadAllQuery()	
useCollectionClass(Class concreteClass)	
useCursoredStream()	
useCursoredStream(int initialReadSize, int pageSize)	
useCursoredStream(int initialReadSize, int pageSize, ValueReadQuery sizeQuery)	
useDistinct()useMapClass(Class concreteClass, String methodName)	
useScrollableCursor()	
useScrollableCursor(int pageSize)	

Table 4–22 Elements for OracleAS TopLink Extended JDO API (Cont.)

Method Name	Description
`addAttribute(String itemName)`	`Query arguments`
`addAttribute(String itemName, Expression attributeExpression)`	
`addAverage(String itemName)`	
`addAverage(String itemName, Expression attributeExpression)`	
`ddCount()`	
`addCount(String itemName)`	
`addCount(String itemName, Expression attributeExpression)`	
`addGrouping(String attributeName)`	
`addGrouping(Expression expression)`	
`addItem(String itemName, Expression attributeExpression)`	
`addMaximum(String itemName)`	
`addMaximum(String itemName, Expression attributeExpression)`	
`addMinimum(String itemName)`	
`addMinimum(String itemName, Expression attributeExpression)`	
`getReportQuery()`	

Table 4–22 Elements for OracleAS TopLink Extended JDO API (Cont.)

Method Name	Description
addArgument(String argumentName) bindAllParameters() cacheStatement() cascadeAllParts() cascadePrivateParts() dontBindAllParameters() dontCacheStatement() dontCascadeParts() dontCheckCache() dontMaintainCache() dontUseDistinct() getQueryTimeout()getReferenceClass() getSelectionCriteria() refreshIdentityMapResult() setCall(Call call) setEJBQLString(String ejbqlString) setFilter(Expression selectionCriteria) setQueryTimeout(int queryTimeout) setSQLString(String sqlString) setShouldBindAllParameters(boolean shouldBindAllParameters) setShouldCacheStatement(boolean shouldCacheStatement) setShouldMaintainCache(boolean shouldMaintainCache) shouldBindAllParameters() shouldCacheStatement() shouldCascadeAllParts() shouldCascadeParts() shouldCascadePrivateParts() shouldMaintainCache()	DatabaseQuery

JDOTransaction

The `JDOTransaction` class implements the `JDOTransaction` interface. It defines the basic begin, commit, and rollback APIs, and synchronization callbacks within the Unit of Work. It supports the optional nontransactional read JDO feature.

Read Modes Set the read mode of a JDO transaction by calling the `setNontransactionalRead()` method.

> **Note:** To avoid exceptions, do not change the read mode while the transaction is active.

Here are the available read modes:

- *Nontransactional Read:* Nontransactional reads provide data from the database, but do not attempt to update the database with changes at commit time. This transaction mode is the `PersistenceManagerFactory` default. Nontransactional reads support nested Units of Work.

  When you execute queries in nontransactional read mode, their results are not subject to the transactional boundary. To update objects from the query results, modify objects in their transactional instances.

  To enable nontransactional read mode, set `setNontransactionalRead()` to true.

- *Transactional Read:* Transactional reads provide data from the database and write any changes to the database at commit time. When you use transactional read, OracleAS TopLink uses the same Unit of Work for all data store interaction (begin, commit, rollback). Because this can cause the cache to grow large over time, use this mode only with short-lived PersistenceManager instances. Doing this allows garbage collection on the Unit of Work.

  When you execute queries in transactional read mode, the results are transactional instances, subject to the transactional boundary. You can update objects from the result of a query that is executed in transactional mode.

  Because you use the same Unit of Work in this mode, the transaction is always active. You must release it when you change the read mode from transactional read to nontransactional read.

Caution: Before you call the OracleAS TopLink extended API `release()` method, commit all changes to avoid losing the transaction.

To enable transactional read mode, set the `setNontransactionalRead()` flag to false.

Synchronization You can register a Synchronization listener with the transaction. The transaction notifies the listener when the transaction is complete. Doing this returns the `beforeCompletion` and `afterCompletion` methods when the precommit and post-commit events of the Unit of Work trigger.

Running the OracleAS TopLink JDO Example

OracleAS TopLink includes an example application that illustrates some of the JDO functionality. You can locate the example in `<ORACLE_HOME>\toplink\examples\foundation\jdo`.

5

Data Access

Managing and protecting data are key components of good application design. Oracle Application Server TopLink enables you to build your application around your choice of datasource and connection, and to customize data access functions to improve performance and security.

This chapter explores the ways you can configure OracleAS TopLink data access, and includes discussions on:

- Introduction to Data Access Concepts
- Database Platforms
- JDBC Connection Pools
- Database Login Information
- OracleAS TopLink Conversion Manager
- Performance
- Table Qualifier
- Locking Policy
- Using the OracleAS TopLink SDK
- OracleAS TopLink XML Support

Introduction to Data Access Concepts

In OracleAS TopLink applications, data access offers the functionality and features that enable you to manipulate data on a database and mapping a data source with the OracleAS TopLink Software Development Kit (SDK).

This section introduces some of the key concepts associated with OracleAS TopLink data access features.

JDBC Connections

Java Database Connectivity (JDBC) is an application programming interface (API) that gives Java applications access to a database. OracleAS TopLink applications rely on JDBC connections to read objects from, and write objects to, the database.

OracleAS TopLink applications use either individual JDBC connections or a JDBC connection pool, depending on the application architecture.

Individual JDBC Connections

An individual JDBC connection gives a single user access to the database for a single session. For example, a two-tier OracleAS TopLink architecture generally connects to the database using a database session and a single JDBC connection. OracleAS TopLink invokes the JDBC connection as part of the login for the database session in the `sessions.xml` file.

For more information about sessions, see Chapter 4, "Sessions".

JDBC Connection Pools

A JDBC connection pool is a collection of JDBC connections managed as a group. Most three-tier multiuser applications use connection pools.

JDBC connection pools enable you to configure connections for several users using less than a one-to-one ratio of connections to users, because the connections in the pool are reusable. For example, a two-tier application requires one JDBC connection for its one user. A three-tier application, conversely, can support several thousand users with a connection pool of only a few connections, depending on the application. The connection pool assigns connections to clients, retrieves the connections when clients complete their tasks, and reuses them for future database requests. The connection pool also queues database requests when requests outnumber the available connections.

OracleAS TopLink supports two types of connection pools: the default OracleAS TopLink internal connection pool and external connection pools.

Internal Pools Because of the multiuser nature of a server session, OracleAS TopLink establishes a connection pool for all server sessions by default. The pool includes several database connections that can be configured, and OracleAS TopLink manages the pool automatically.

External Pools OracleAS TopLink supports external connection pools. Applications that include an application server usually use an external connection pool, managed by a Java Transaction Architecture (JTA) device.

JTA

The JTA is a specification that enables your application to participate in a distributed transaction system. The system provides transaction management and connection pooling and enables your application to interact with multiple databases transparently.

OracleAS TopLink applications that use an application server often use JTA to manage database transactions.

Data Conversion

OracleAS TopLink applications store object attributes on a database. To enable this functionality, OracleAS TopLink must convert object attributes, which are Java types such as `STRING` and `INTEGER`, to database types, such as `VARCHAR` and `NUMERIC`. The OracleAS TopLink conversion manager manages these conversions and enables you to build custom conversion classes.

Database Platforms

OracleAS TopLink communicates with databases using Structured Query Language (SQL). Because each database platform uses its own variation on the basic SQL language, OracleAS TopLink must adjust the SQL it uses to communicate with the database to ensure that the application runs smoothly.

This section describes OracleAS TopLink support for:

- JDBC-SQL and Native SQL
- Custom Platforms

JDBC-SQL and Native SQL

By default, OracleAS TopLink accesses the database using JDBC-SQL and automatically performs the conversions between Java types and database types. OracleAS TopLink provides the conversions listed in Table 5–1 automatically.

Table 5–1 JDBC-SQL Conversion Types

Class	Oracle Type	DB2 Type	dBase Type	Sybase Type	Microsoft Access Type
java.lang.Boolean	NUMBER	SMALLINT	NUMBER	BIT default 0	SHORT
java.lang.Byte	NUMBER	SMALLINT	NUMBER	SMALLINT	SHORT
java.lang.Byte[]	LONG RAW	BLOB	BINARY	IMAGE	LONGBINARY
java.lang.Integer	NUMBER	INTEGER	NUMBER	INTEGER	LONG
java.lang.Long	NUMBER	INTEGER	NUMBER	NUMERIC	DOUBLE
java.lang.Float	NUMBER	FLOAT	NUMBER	FLOAT(16)	DOUBLE
java.lang.Double	NUMBER	FLOAT	NUMBER	FLOAT(32)	DOUBLE
java.lang.Short	NUMBER	SMALLINT	NUMBER	SMALLINT	SHORT
java.lang.String	VARCHAR2	VARCHAR	CHAR	VARCHAR	TEXT
java.lang.Character	CHAR	CHAR	CHAR	CHAR	TEXT
java.lang.Character[]	LONG	CLOB	MEMO	TEXT	LONGTEXT
java.math.BigDecimal	NUMBER	DECIMAL	NUMBER	NUMERIC	DOUBLE
java.math.BigInteger	NUMBER	DECIMAL	NUMBER	NUMERIC	DOUBLE
java.sql.Date	DATE	DATE	DATE	DATETIME	DATETIME
java.sql.Time	DATE	TIME	CHAR	DATETIME	DATETIME
java.sql.Timestamp	DATE	TIMESTAMP	CHAR	DATETIME	DATETIME

OracleAS TopLink provides the required customization by enabling you to specify your database platform. OracleAS TopLink provides specific support for the following platforms:

- Oracle databases
- IBM DB2
- IBM DBase

- IBM Cloudscape
- IBM Informix
- Microsoft Access
- Microsoft SQL Server
- Sybase SQL Server
- JDBC
- PointBase databases

Specify your database platform in the `login` element of the `sessions.xml` file or the login section of your Java project configuration file (`project.java`). If you set your database platform in the OracleAS TopLink Mapping Workbench, then the OracleAS TopLink Mapping Workbench and the OracleAS TopLink Sessions Editor manage the database platform configuration for you automatically.

Example 5–1 Specifying an Oracle Database Platform in the sessions.xml File

For clarity, the code that sets the platform class is bold in this example.

```
<session>
  ...
  <login>
    ...
    <platform-class>oracle.toplink.internal.databaseaccess.OraclePlatform</platform-class>
    ...
  </login>
...
</session>
```

Example 5–2 Specifying an Oracle Database Platform in Java

```
project.getLogin().useOracle();
```

Custom Platforms

You can specify a custom database platform for your OracleAS TopLink application. Custom platform support enables you to use a database when OracleAS TopLink has no predefined platform.

To enable custom database platform support, create a new platform class that extends one of the existing platform classes, and call the class at runtime by referencing it in the session configuration file (the `sessions.xml` file), as Example 5–1 illustrates.

JDBC Connection Pools

A JDBC connection pool is a collection of reusable database connections that service a single application. This section introduces the following topics and techniques for working with JDBC connection pools:

- Default Connection Pools
- External Connection Pools
- JDBC Datasources
- JTA

Default Connection Pools

OracleAS TopLink provides a default internal connection pool for sessions that use a server session for database access. The default settings are appropriate for most applications; however, you can modify the connection pool attributes in the `sessions.xml` file to tailor the pool to your needs. You can specify:

- The type of connection in the connection pool (read or write)
- The name of the connection pool
- The maximum number of database connections available in the connection pool
- The minimum number of database connections available in the connection pool

For complete information on specifying the internal connection pool in the `sessions.xml` file, see "connection-pool Element" on page 4-23.

External Connection Pools

With OracleAS TopLink you can use an external connection pool rather than the default internal pool, enabling you to leverage external transaction management systems such as JTA. This is common in applications that incorporate an application server.

To use an external connection pool, enable and specify it as follows:

- If your application uses EJB entity beans, modify the `toplink-ejb-jar.xml` file, using the elements described in Table 9–1, "login Elements" on page 9-8.

- If your application does not leverage EJB entity beans, configure the external connection pool in the `sessions.xml` file using the elements described in "connection-pool Element" on page 4-23.

JDBC Datasources

OracleAS TopLink uses a datasource to access your database information—your application does not need to be aware or maintain the connection information. OracleAS TopLink can access the database through a connection pool or a datasource. OracleAS TopLink JTA integration often uses a datasource.

You can configure a datasource as follows:

- If your application uses EJB entity beans, modify the `toplink-ejb-jar.xml` file, using the elements described in Table 9–1, "login Elements" on page 9-8.
- If your application does not leverage EJB entity beans, configure the datasource in the `sessions.xml` file `login` element, using the optional `data-source` element described in Table 4–3, "Basic Configuration Tags Within the Login Element" on page 4-13.

For more information about defined connection pools and datasources with EJB entity beans, see "Configuring the toplink-ejb-jar.xml File with the BEA WebLogic Server" on page 9-6.

Container-Managed Persistence and Datasources

OracleAS TopLink CMP applications can leverage datasources rather than connection pools. To use a datasource, configure Java Transaction Service (JTS) support. JTS is the specification that supports JTA.

To use a datasource, configure both a JTS and a non-JTS datasource in the `toplink-ejb-jar.xml` file. To configure the required sources, specify them in the `datasource` and `non-jts-data-source` tags in the `login` element. These tags correspond to JTS and non-JTS datasources respectively.

The values for these datasource tags correspond directly to the names of the datasources as defined in your J2EE container or application server. Following is an example of a partial `toplink-ejb-jar.xml` file listing, using datasources:

```
...
<datasource>myJtsDataSource</datasource>
<non-jts-data-source>myNonJtsDataSource</non-jts-data-source>
...
```

For more information, see "Configuring the toplink-ejb-jar.xml File with the BEA WebLogic Server" on page 9-6.

JTA

You can integrate your OracleAS TopLink application with a transaction service that complies with JTA, thereby enabling sessions to:

- Participate in distributed transactions
- Leverage existing connection pools
- Access several databases managed by the JTA system transparently

JTA is a Java 2 Enterprise Edition (J2EE) component.

For more information about leveraging JTA in your application, see "J2EE Integration" on page 7-44.

Database Login Information

Java applications that access a database log in to the database through a JDBC driver. Database logins generally require a valid user name and password. OracleAS TopLink applications store this login information in the `DatabaseLogin` class. All sessions must have a valid `DatabaseLogin` instance before logging in to the database.

This section describes:

- Creating a Login Object
- Specifying Driver Information
- Setting Login Parameters
- Database Login Advanced Features

Creating a Login Object

Your project configuration file (`project.xml` or `project.java`) must include a login object to enable database access. The most basic login mechanism creates an instance of `DatabaseLogin` through its default constructor, as follows:

```
Databaselogin login = new Databaselogin();
...
```

If you create the project in the OracleAS TopLink Mapping Workbench, OracleAS TopLink creates the login object for you automatically and enables you to access the login from your project instance. This ensures that the session uses login information set in the OracleAS TopLink Mapping Workbench (for example: sequencing information) and also prevents you from inadvertently overwriting the login information already included in the project.

You can also access the login in Java code, using the `getLogin() instance` method to return the project's login. This method returns an instance of `DatabaseLogin`, which you can either use directly or augment with additional information before logging in.

Specifying Driver Information

The `DatabaseLogin` class includes helper methods that set the driver class, driver Uniform Resource Locator (URL) prefix, and database information for common drivers. When you use helper methods, use the `setDatabaseURL()` method to set the database instance for the JDBC driver URL.

These helper methods also specify any additional settings required for that driver, such as binding byte arrays or using native SQL.

For example:

```
project.getLogin().useOracleThinJDBCDriver();
project.getLogin().setDatabaseURL("dbserver:1521:orcl");
```

Using the Sun Microsystems JDBC-ODBC Bridge

To use the Sun Microsystems JDBC-ODBC bridge, specify the ODBC datasource name by calling the `setDataSourceName()`.

Example 5–3 Using the Sun Microsystems JDBC-ODBC Bridge

```
project.getLogin().useJDBCODBCBridge();
project.getLogin().useOracle();
project.getLogin().setDataSourceName("Oracle");
```

In Example 5–3, OracleAS TopLink splits the URL into the driver and database calls. You can also use the `setConnectionString()` function to specify the URL in a single line of code.

Using a Different Driver

If you require a driver other than the Sun Microsystems JDBC-ODBC bridge, specify a different connection mechanism by calling the `setDriverClass()` and `setConnectionString()` methods.

For more information about the correct driver settings to use with these methods, see the driver documentation.

Example 5–4 Using an Alternative Driver

```
project.getLogin().setDriverClass(oracle.jdbc.driver.OracleDriver.class);
project.getLogin().setConnectionString("jdbc:oracle:thin:@dbserver:1521:orcl");
```

Setting Login Parameters

You can set several session properties as part of the login, including user information, database information, and JDBC driver information.

User Information

If a database requires user and password information, call the `setUserName()` and `setPassword()` methods after you specify the driver. Specify user and password information when you use the login object from a OracleAS TopLink Mapping Workbench project.

Example 5–5 Using setUserName() and setPassword()

```
project.getLogin().setUserName("userid");
project.getLogin().setPassword("password");
```

Database Information

You can specify properties such as the database name and the server name using the `setServerName()` and `setDatabaseName()` methods. The ODBC datasource Administrator for most JDBC-ODBC bridges usually sets these properties, but some drivers do require you to specify them explicitly.

Note that, because the database and server name properties are part of the database URL, most JDBC drivers do not require you to specify them explicitly and may fail if you do specify them.

Additional JDBC Properties

If your JDBC driver requires additional properties, use the `setProperty()` method to send these properties. Use caution when specifying properties, because, although some drivers require additional information, other drivers can fail if you specify properties that are not required. If you use the `setProperty()` method and the connection always fails, ensure that the specified properties are correct, complete, and required.

> **Note:** Do not set the login password directly using the `setProperty()` method, because OracleAS TopLink encrypts and decrypts the password. Use the `setPassword()` method instead.

Database Login Advanced Features

You can set the following options within your code, rather than through the OracleAS TopLink Mapping Workbench:

- Setting Sequencing at Login
- Setting Direct Connect Drivers
- Using JDBC 2.0 Datasources
- Using Custom Database Connections

There are several options you can set at login rather than through more conventional methods, such as through the OracleAS TopLink Mapping Workbench.

Setting Sequencing at Login

For most projects, you set sequencing in the OracleAS TopLink Mapping Workbench project. To configure sequencing in Java code, you can use any of the following methods:

- `setSequenceCounterFieldName()`
- `setSequenceNameFieldName()`
- `setSequencePreallocationSize()`
- `setSequenceTableName()`
- `useNativeSequencing()`

OracleAS TopLink supports native sequencing on Oracle databases, IBM Informix, Microsoft SQL Server, and Sybase SQL Server. Using native sequencing requires that you specify the database platform. Call the `useNativeSequencing()` method to configure your application to use native sequencing rather than a sequence table.

When you implement native sequencing, consider the following:

- The sequence preallocation size defaults to 1. If you use Sybase, Microsoft SQL Server or IBM Informix native sequencing, you cannot use preallocation and you cannot change the size.
- When using native sequencing with Oracle, specify the name of the sequence object (the object that generates the sequence numbers) for each descriptor. The sequence preallocation size must also match the increment on the sequence object.

Notes:

- Ensure you match the increment of the Oracle sequence and not the cache. The cache refers to the sequences cached on the database server; the increment refers to the number of sequences that can be cached on the database client.
- When you use sequencing or native sequencing, specify the sequence information in each descriptor that makes use of a generated ID.
- Use preallocation and native sequencing for Oracle databases.

Example 5–6 Configuring Oracle Native Sequencing in Java Code

```
project.getLogin().useOracle();
project.getLogin().useNativeSequencing();
project.getLogin().setSequencePreallocationSize(1);
```

Note: Using the `Project` class to create a `DatabaseLogin` instance automatically uses the sequencing information specified in the OracleAS TopLink Mapping Workbench.

For more information, see "Sequencing" on page 3-37.

Setting Direct Connect Drivers

By default, OracleAS TopLink loads a JDBC driver and connects to a database as follows:

- To load and initialize the class, OracleAS TopLink calls `java.lang.Class.forName()`.
- To obtain a connection, OracleAS TopLink calls `java.sql.DriverManager.getConnection()`.

Some drivers do not allow you to use the `java.sql.DriverManager` to connect to a database. To load these drivers, configure OracleAS TopLink to instantiate the drivers directly, by invoking the `DirectDriverConnect()` method.

Example 5–7 Using useDirectDriverConnect()

```
project.getLogin().useDirectDriverConnect("com.direct.connectionDriver",
  "jdbc:far:", "server");
```

Using JDBC 2.0 Datasources

The JDBC 2.0 specification recommends using a Java Naming and Directory Interface (JNDI) naming service to acquire a connection to a database. To use this feature, configure an instance of `oracle.toplink.jndi.JNDIConnector` and pass it to the project login object using the `setConnector()` method.

Example 5–8 Using JNDI

```
import oracle.toplink.sessions.*;
import oracle.toplink.jndi.*;
    javax.naming.Context context = new javax.naming.InitialContext();
    Connector connector = new JNDIConnector(context, "customerDB");
    project.getLogin().setConnector(connector);
```

Using Custom Database Connections

OracleAS TopLink allows you to develop your own class to obtain a connection to a database. The class must implement the `oracle.toplink.sessions.Connector` interface. This requires the class to implement the following methods:

- `java.sql.Connection connect(java.util.Properties properties)`: Receives a dictionary of properties (including the user name and password) and must return a valid connection to the database.
- `void toString(PrintWriter writer)`: Prints out any helpful information on the OracleAS TopLink log.

Implement the custom class, instantiate it, and then pass it to the project login object, using the `setConnector()` method.

Example 5–9 Using the oracle.toplink.sessions.Connector Interface

```
import oracle.toplink.sessions.*;
Connector connector = new MyConnector();
project.getLogin().setConnector(connector);
```

OracleAS TopLink Conversion Manager

OracleAS TopLink uses a class known as the `ConversionManager` to convert database types to Java types. This class, found in the `oracle.toplink.internal.helper` package, is the central location for type conversion and provides developers with a mechanism for using custom types in OracleAS TopLink.

This section describes:

- Creating Custom Types with the Conversion Manager
- Conversion Manager Class Loader
- Resolving Class Loader Exceptions

Creating Custom Types with the Conversion Manager

Use the conversion manager to create and use custom types in OracleAS TopLink.

To use custom types in OracleAS TopLink:

1. Use one of the following methods to create a subclass of the `ConversionManager`:
   - Overload the `public Object convertObject(Object sourceObject, Class javaClass)` method to call the conversion method you provide in the subclass for the custom type.
   - Delegate the conversion to the superclass.
2. Implement the `protected ClassX convertObjectToClassX(Object sourceObject) throws ConversionException` conversion method to convert incoming objects to the required class.

3. Assign the class to OracleAS TopLink in either of two ways:

   - Assign a custom conversion manager to the OracleAS TopLink session using the `(getSession().getPlatform().setConversionManager (ConversionManager))` platform.
   - Set the conversion manager singleton by calling the `setDefaultManager(ConversionManager)` static method on the conversion manager. This setting causes all OracleAS TopLink sessions created in the Java virtual machine (JVM) to use the custom conversion manager. See the `ConversionManager` class in the *Oracle Application Server TopLink API Reference* for examples.

Conversion Manager Class Loader

OracleAS TopLink provides a class loader within the conversion manager which enables the conversion manager to load classes from both a OracleAS TopLink Mapping Workbench project and the class library. The conversion manager uses the System class loader by default.

Resolving Class Loader Exceptions

In some cases, such as when OracleAS TopLink is deployed within an application server, you may want to use other class loaders for the deployed classes. Doing this can cause a `ClassNotFound` exception. To resolve this problem, use one of the following methods:

- Call the `public void setShouldUseClassLoaderFromCurrentThread(boolean useCurrentThread)` method on the default conversion manager before logging in any sessions. This method resolves the problem for most application servers and ensures that OracleAS TopLink uses the correct class loader.
- Set the default class loader to be the one that the application uses. For example, if you use the session manager, pass the class loader into the `getSession()` call to set the required class loader on the conversion manager.
- Call `public static void setDefaultLoader(ClassLoader classLoader)` on the conversion manager before any sessions are logged in, and pass in the class loader that contains the deployed classes.

Performance

You can use several techniques to improve data access performance for your application. This section discusses some of the more common approaches, including:

- Data Optimization
- Batch Writing
- Binding and Parameterized SQL
- Prepared Statement Caching

Data Optimization

By default, OracleAS TopLink optimizes data access by accessing the data from JDBC in the format the application requires. For example, OracleAS TopLink retrieves `longs` from JDBC instead of having the driver return a `BigDecimal` that OracleAS TopLink would then have to convert into a `long`.

OracleAS TopLink also retrieves dates as strings and converts directly to the date or `Calendar` type used by the application. Some older drivers do not convert data correctly. For example, earlier BEA WebLogic JDBC drivers cannot convert dates to strings in the correct format. If you use one of these drivers, disable data optimization.

> **Note:** The problems mentioned here may have been fixed in more recent versions of the drivers. See your vendor documentation for relevant updates.

Example 5–10 Disabling Data Optimization in Code

```
session.getLogin().dontOptimizeDataConversion() ;
```

Example 5–11 Disabling Data Optimization in the sessions.xml File

```
<login>
    ...
    <should_optimize_data_conversion>false</should-optimize-data-conversion>
</login>
```

Batch Writing

Batch writing can improve database performance by sending groups of `INSERT`, `UPDATE`, and `DELETE` statements to the database in a single transaction, rather than individually. OracleAS TopLink supports batch writing for selected databases and for JDBC 2.0 batch-compliant drivers.

To enable JDBC 2.0 batch writing, invoke the `useBatchWriting()` method on the login.

If you use a JDBC driver that does not support batch writing directly, you can still take advantage of batch writing, because OracleAS TopLink provides its own batch writing functionality. To enable OracleAS TopLink batch writing support, run the code in Example 5–12.

Example 5–12 Batch Writing

```
project.getLogin().useBatchWriting();
project.getLogin().dontUseJDBCBatchWriting();
```

For more information about batch writing, see Chapter 10, "Tuning for Performance" on page 10-1.

Binding and Parameterized SQL

By default, OracleAS TopLink prints data inlined into its generated SQL and does not use parameterized SQL. However, you can implement parameterized SQL to:

- Alleviate the limit imposed by some drivers on the size of the data to be printed.
- Cache prepared statements to improve performance.

OracleAS TopLink does not implement parameterized SQL because many JDBC drivers do not fully support parameter binding, and have size or type limits.

For more information about binding and binding size limits, see your database documentation.

If your driver supports parameter binding and also imposes a limit on the size of the printable results, use parameter binding to accommodate large binary data in one of the following ways:

- Call the `useByteArrayBinding()` method. This is a common method to accommodate large binary data.

- If you use a JDBC driver that is more efficient at reading large binary data through streams, call the `useStreamsForBinding()` method.
- Configure binding for large string data with the `useStringBinding()` method.

Example 5–13 Using Parameter Binding with Large Binary Data

```
project.getLogin().useByteArrayBinding();

project.getLogin().useStreamsForBinding();
project.getLogin().useStringBinding(50);
project.getLogin().bindAllParameters();
project.getLogin().cacheAllStatements();
project.getLogin().setStatementCacheSize(50);
```

Prepared Statement Caching

OracleAS TopLink enables you to cache JDBC prepared statements to improve query performance. Prepared statements improve database performance by reducing the number of times the database SQL engine parses and prepares a SQL call for a frequently called query.

To enable prepared statement caching, cache the statement and bind its parameters. You can do this at the query level or at the session level.

Prepared Statement Caching for a Query

To cache the prepared statement for an individual query, configure statement caching in the query definition before execution the query. You can do this either in Java code, or as part of the SQL for a named query in the OracleAS TopLink Mapping Workbench.

Example 5–14 Caching a Prepared Statement in Code for an Individual Query

```
// Add a query.
ExpressionBuilder builder = new ExpressionBuilder();
ReadAllQuery query = new ReadAllQuery(PhoneNumber.class, builder);

Expression exp = builder.get("id").equal(builder.getParameter("ID"));
query.setSelectionCriteria(exp.and(builder.get("areaCode").equal("613")));

query.addArgument("ID");

/* The following options force OracleAS TopLink to cache the prepared statement
```

```
and bind any arguments required by the query */
query.cacheStatement();
query.bindAllParameters();

descriptor.getQueryManager().addQuery("localNumbers", query);
```

Prepared Statement Caching for a Session

To cache all prepared statements for a session, edit the `sessions.xml` file in the OracleAS TopLink Sessions Editor, adding login options to bind all parameters and cache statements.

Example 5–15 Caching Prepared Statements in the sessions.xml File

```
<session>
    ...
    <login>
        ...
        <should-bind-all-parameters>true</should-bind-all-parameters>
        <should-cache-all-statements>true</should-cache-all-statements>
    </login>
    ....
</session>
```

Table Qualifier

A table qualifier affects the data in a table to which a user has access. You can use table qualifiers to manage data access in databases that support them, such as Oracle and IBM DB2. You can also use table qualifiers to fully-qualify the table names of tables that have a different creator.

OracleAS TopLink enables you to add a table qualifier to all table references in a given session. Use the `setTableQualifier()` method on your session login object to prepend a string to all tables accessed by the session.

Example 5–16 Adding a Table Qualifier

```
session.getLogin().setTableQualifier([QUALIFIER_STRING])
```

Locking Policy

A locking policy is an important component of any multi-user OracleAS TopLink application. When users share objects in an application, a locking policy ensures that two or more users do not attempt to modify the same object or its underlying data simultaneously.

OracleAS TopLink works with relational databases to provide support for several types of locking policy, including:

Optimistic Lock All users have read access to the object. When a user attempts to write a change, the application checks to ensure the object has not changed since the last read. OracleAS TopLink provides this locking policy.

Optimistic Read Lock As with optimistic lock, the optimistic read lock ensures that the object has not changed before writing a change. However, the optimistic read lock also forces a read of any related tables that contribute information to the object. OracleAS TopLink offers this locking policy.

Pessimistic Locking When a user accesses an object to update it, the database locks the object until the update is completed. No other user can read or update the object until the first user releases the lock. The database offers this locking type.

No Locking The application does not verify that data is current.

Note: Most OracleAS TopLink applications use either optimistic locking or optimistic read locking, because they are the safest and most efficient of these locking strategies.

Using Optimistic Locking

Optimistic locking, also known as write locking, allows unlimited read access to a given object, but allows a client to modify the object only if the object has not changed since the client last read it.

Optimistic locking checks an object's version at transaction commit time against the version read during the transaction. This check ensures that no other client modified the data after it was read by the current transaction. If this check detects stale data, the check raises an `OptimisticLockException`, and the commit fails.

Note: Using optimistic locking by itself does not protect against having different copies of the same object existing in multiple nodes. For more information, see "Optimistic Locking" in the *Oracle Application Server TopLink Mapping Workbench User's Guide.*

Set optimistic locking on the descriptor using one of two locking policies:

- *Version locking policies* enforce optimistic locking using a version field (or write lock field). OracleAS TopLink updates this field each time it modifies a record. Add a version field to the table for this purpose.
- *Field locking policies* enforce optimistic locking by preventing other processes from writing to the field until the current transaction commits. Field locking does not require additional fields in the table, but you must commit changes to the database using a Unit of Work to implement this type of policy.

For more information about locking policies, see "Two Different Locking Policies" on page 5-28.

Advantages and Disadvantages of Optimistic Locking

Here are the advantages of optimistic locking:

- It prevents users and applications from editing stale data.
- It notifies users of any locking violation immediately, when updating the object.
- It does not require you to lock up the database resource.
- It prevents database deadlocks.

However, optimistic locking cannot prevent applications from selecting and attempting to modify the same data. When two different processes modify data, the first one to commit the changes succeeds while the other process fails and receives an `OptimisticLockException`.

Advanced Optimistic Locking Policies

All OracleAS TopLink optimistic locking policies implement the `OptimisticLockingPolicy` interface. This interface includes several of methods that you can implement to customize the optimistic locking policy.

For more information about these methods, see the *Oracle Application Server TopLink API Reference*.

Optimistic Read Locking

Optimistic read lock is an advanced type of optimistic lock that enables you to force lock checking on objects that are not modified by the current transaction. Optimistic read lock also offers the option to increment the unchanged object version or leave the version unchanged.

For example, consider a transaction that updates a mortgage rate by multiplying the central bank prime rate by 1.25. The transaction executes an optimistic read lock on the central prime rate at commit time to ensure that the prime rate has not changed since the transaction began. Note that in this example, the transaction does not increment the version of the unchanged object (the central prime rate).

Example 5–17 Optimistic Read Lock with No Version Increment

```
try {
    UnitOfWork uow = session.acquireUnitOfWork();
    MortgageRate cloneMortgageRate = (MortgageRate)
      uow.registerObject(mortgageRate);
    CentralPrimeRate cloneCentralPrimeRate = (CentralPrimeRate)
      uow.registerObject(CentralPrimeRate);
    /* Change the Mortgage Rate */
    cloneMortgageRate.setRate(cloneCentralPrimeRate.getRate() * 1.25);
    /* Optimistic read lock check on Central prime rate with no version update*/
    uow.forceUpdateToVersionField(cloneCentralPrimeRate, false);
    uow.commit();
}(OptimisticLockException exception) {
    /* Refresh the out-of-date object */
    session.refreshObject(exception.getObject());
    /* Retry… */
}
```

Consider another example, in which an *invoice* thread calculates an invoice for a customer. If another thread (the *service* thread) adds a service to the same customer or modifies the current service, it must inform the *invoice* thread, which adds the changes to the invoice. This feature is available for objects that implement a version of field locking policy or timestamp locking policy. When you update an object that implements a version locking policy, the version value is incremented or set to the current timestamp.

For more information about field locking policies, see "Field Locking Policies" on page 5-29.

Example 5–18 Optimistic Read Lock with Version Increment

```
/* The following code represents the service thread. Notice that the thread
forces a version update. */
try {
    UnitOfWork uow = session.acquireUnitOfWork();
    Customer cloneCustomer = (Customer uow.registerObject(customer);
    Service cloneService = (Service uow.registerObject(service);
    /* Add a service to customer */
    cloneService.setCustomer(cloneCustomer);
    cloneCustomer.getServices().add(cloneService);
    /* Modify the customer version to inform other application that
      the customer has changed */
    uow.forceUpdateToVersionField(cloneCustomer, true);
    uow.commit();
}
catch {
    (OptimisticLockException exception) {
    /* Refresh out-of-date object */
    session.refreshObject(exception.getObject());
    /* Retry… */
}

/* The following code represents the invoice thread, and calculates a bill for
the customer. Notice that it does not force an update to the version */

try {
    UnitOfWork uow = session.acquireUnitOfWork();
    Customer cloneCustomer = (Customer) uow.registerObject(customer);
    Invoice cloneInvoice = (Invoice) uow.registerObject(new Invoice());
    cloneInvoice.setCustomer(cloneCustomer);
    /* Calculate services' charge */
    int total = 0;
    for(Enumeration enum = cloneCustomer.getServices().elements();
    enum.hasMoreElements();) {
    total += ((Service) enum.nextElement()).getCost();
    }
    cloneInvoice.setTotal(total);
    /* Force optimistic lock checking on the customer to guarantee a valid
      calculation */
    uow.forceUpdateToVersionField(cloneCustomer, false);
    uow.commit();
}
catch {
    (OptimisticLockException exception) {
```

```
    /* Refresh the customer and its privately owned parts */
    session.refreshObject(cloneCustomer);
    /* If the customer's services are not private owned then use a
    ReadObjectQuery to refresh all parts */
    ReadObjectQuery query = new ReadObjectQuery(customer);
    /* Refresh the cache with the query's result and cascade refreshing
      to all parts including customer's services */
    query.refreshIdentityMapResult();
    query.cascadeAllParts();
    /* Refresh from the database */
    query.dontCheckCache();
    session.executeQuery(query);
    /* Retry… */
}
```

When is an Object Considered Changed? The Unit of Work considers an object changed when you modify its direct-to-field or aggregate object mapping attribute. Adding, removing, or modifying objects related to the source object does not render the source object changed for the purposes of the Unit of Work.

Pessimistic Locking

Pessimistic locking locks objects when the transaction accesses them, before commit time, ensuring that only one client is editing the object at any given time.

Pessimistic locking detects locking violations at object read time. The OracleAS TopLink implementation of pessimistic locking uses database row-level locks, such that attempts to read a locked row either fail or are blocked until the row is unlocked, depending on the database.

Example 5–19 Pessimistic Locking with ReadObjectQuery

```
import oracle.toplink.sessions.*;
import oracle.toplink.queryframework.*;

...
UnitOfWork uow = session.acquireUnitOfWork();

ReadObjectQuery query = new ReadObjectQuery();
query.setReferenceClass(Employee.class);
query.acquireLocks();
Employee employee = (Employee) uow.executeQuery(query);

// Make changes to object
...
```

```
uow.commit();
...
```

Example 5–20 Pessimistic Locking with ReadAllQuery

```
import oracle.toplink.sessions.*;
import oracle.toplink.queryframework.*;

...
UnitOfWork uow = session.acquireUnitOfWork();
ReadAllQuery query = new ReadAllQuery();
query.setReferenceClass(Employee.class);
query.setSelectionCriteria(new ExpressionBuilder().get("salary").greaterThan(25000));
query.acquireLocks();
/* NOTE: the objects are registered when they are obtained by using Unit of Work. OracleAS
TopLink will update all the changes to registered objects when Unit of Work commit */
Vector employees = (Vector) uow.executeQuery(query);
    // Make changes to objects
    ...
    uow.commit();
...
```

Example 5–21 Pessimistic Locking with a Session Using ReadAllQuery

```
import oracle.toplink.sessions.*;
import oracle.toplink.sessions.queryframework.*;
...
// It must begin a transaction or the lock request will throw an exception
session.beginTransaction();
ReadAllQuery query = new ReadAllQuery();
query.setReferenceClass(Employee.class);
query.setSelectionCriteria(new ExpressionBuilder().get("salary").greaterThan(25000));
query.acquireLocks();
// or acquireLocksWithoutWaiting()
query.refreshIdentityMapResult();
Vector employees = (Vector) session.executeQuery(query);
// Make changes to objects
...
// Update objects to reflect changes
for (Enumeration enum = employees.elements();
    employees.hasMoreElements(); {
    session.updateObject(enum.nextElement());
}
session.commitTransaction();
...
```

Pessimistic Locking and the Cache

When you acquire a pessimistic lock on an object, you refresh the object in the session cache. This is different from an optimistic lock, which refreshes objects in the cache only after a successful commit. Because of this, and because it prevents other processes from reading locked objects, a pessimistic lock is not as efficient as an optimistic lock.

Note: OracleAS TopLink uses database row-level locking to implement pessimistic locking. Although this is the standard way of implementing pessimistic locking in the database, not all databases support row-level locking functionality. Consult your database documentation to see if your database supports row-level locking and the `SELECT ... FOR UPDATE [NO WAIT]` API.

Pessimistic Locking and Database Transactions

Because pessimistic locks exist for the duration of the current transaction, the associated database transaction remains open from the point of the first lock request until the transaction commits. When the transaction commits or rolls back, the database releases the locks.

The Unit of Work starts a database transaction automatically when it attempts to read the first object in its operations. If you are not using the Unit of Work, manually begin a transaction on the session.

WAIT and NO_WAIT Options

OracleAS TopLink offers two methods of locking, `WAIT` and `NO_WAIT`. These options determine how the transaction responds when it encounters a locked row. If you select the:

- The `WAIT` option, then the transaction waits until the database releases the lock on the object. It then obtains a lock on the object and continues.
- The `NO_WAIT` option, then OracleAS TopLink throws an exception when the transaction encounters a locked row.

Example 5–22 Pessimistic Locking with Wait for Lock

This example illustrates a pessimistic lock with the `WAIT` mode in the context of a Unit of Work.

```
import oracle.toplink.sessions.*;
```

```
import oracle.toplink.queryframework.*;
...
UnitOfWork uow = session.acquireUnitOfWork();
Employee employee = (Employee) uow.readObject(Employee.class);

/* Note: This will cause the Unit of Work to begin a transaction. In a
three-Tier model this will also cause the ClientSession to acquire its write
connection from the ServerSession's pool */
uow.refreshAndLockObject(employee, ObjectLevelReadQuery.LOCK);
// Make changes to object
...
uow.commit();
...
```

Example 5–23 Pessimistic Locking with No Wait for Lock

This example illustrates a pessimistic lock with the `No_Wait` mode in the context of a Unit of Work.

```
import oracle.toplink.sessions.*;
import oracle.toplink.queryframework.*;
import oracle.toplink.exceptions.*;
...
UnitOfWork uow = session.acquireUnitOfWork();
Employee employee = (Employee) uow.readObject(Employee.class);

try {
    employee = (Employee)
      uow.refreshAndLockObject(employee,ObjectLevelReadQuery.LOCK_NOWAIT);
}
catch (DatabaseException dbe) {
    // Some databases throw an exception instead of returning nothing.
    employee = null;
}
if (employee == null) {
    // Lock cannot be obtained
    uow.release();
    throw new Exception("Locking error.");
} else {
    // Make changes to object
    ...
    uow.commit();
}
...
```

Advantages of Pessimistic Locking

The following are the advantages of pessimistic locking:

- Pessimistic locking can prevent users and applications from editing data that is being or has been changed.
- Processes know immediately when a locking violation occurs, rather than after the transaction is complete.

Disadvantages of Pessimistic Locking

The following are the disadvantages of pessimistic locking:

- It is not fully supported by all databases.
- It consumes extra database resources.
- It requires OracleAS TopLink to maintain an open transaction and database lock for the duration of the transaction, which can lead to database deadlocks.
- It decreases the concurrency of connection pooling when using the server session, which affects the overall scalability of your application.

Reference

Table 5–2 summarizes the most common public methods for `Pessimistic Locking`. The Default column describes default settings of the descriptor element. For more information about the available methods for `Pessimistic Locking`, see the *Oracle Application Server TopLink API Reference*.

Table 5–2 Elements for Pessimistic Locking

Element	Default	Method Name
Lock mode (for ObjectLevelRead Query)	No lock	`acquireLocks()` `acquireLocksWithoutWaiting()`
Refresh and lock (for Session)	not applicable	`refreshAndLockObject(Object object, short lockMode)`

Two Different Locking Policies

A locking policy describes how you manage record locking on the database and track changed objects. OracleAS TopLink offers two different strategies for managing locking: field locking and timestamp locking.

Field Locking Policies

Field locking policies compare the current values of certain mapped fields with previous values. OracleAS TopLink support for field locking policies does not require any additional fields in the database. Field locking policy support includes:

- `AllFieldsLockingPolicy`
- `ChangedFieldsLockingPolicy`
- `SelectedFieldsLockingPolicy`

These policies require you to use a Unit of Work for database updates. Each policy handles its field comparisons in a specific way defined by the policy:

- When you update or delete an object under `AllFieldsLockingPolicy`, the Unit of Work checks all table fields that are part of the SQL `where` clause. If any values have changed since the object was read, the update or delete fails. This comparison is only on a per table basis. If you perform an update on an object mapped to multiple tables (including multiple table inheritance), only the changed table(s) appear in the `where` clause.
- When you update an object under `ChangedFieldsLockingPolicy`, the Unit of Work checks only the modified fields. This allows multiple clients to modify different parts of the same row without failure. Using this policy, a delete compares only on the primary key.
- When you update or delete an object under `SelectedFieldsLockingPolicy`, the Unit of Work compares a list of selected fields in the update statement.

When an update fails due to an optimistic locking violation, OracleAS TopLink raises an `OptimisticLockException`. Under most circumstances, the application handles this exception by refreshing the object and reapplying changes.

Version Locking Policies

OracleAS TopLink supports version locking policies through the `VersionLockingPolicy` interface and the `TimestampLockingPolicy` interface. Each of these policies requires an additional field in the database to operate:

- For `VersionLockingPolicy`, add a numeric field to the database.
- For `TimestampLockingPolicy`, add a timestamp field to the database.

OracleAS TopLink records the version as it reads an object from a table. When the client attempts to write the object, OracleAS TopLink compares the object version with the version in the table record. If the versions match, OracleAS TopLink writes the updated object to the table and updates the version of both the table record and the object. If the versions are different, the write fails and OracleAS TopLink raises an error.

These two version locking policies have different ways of writing the version fields back to the database:

- `VersionLockingPolicy` increments the value in the version field by one.
- `TimestampLockingPolicy` inserts a new timestamp into the row. The timestamp is configurable to get the time from the server or the local machine.

For either policy, you write the value of the write lock field in either the identity map or in a writable mapping within the object.

If you store the value in the identity map, you do not require an attribute mapping for the version field. However, if the application does map the field, the mappings must be read-only to allow OracleAS TopLink to control writing the fields.

Timestamp Versus Version Locking Policies When choosing a locking policy, consider the following:

- If you need absolute certainty for versioning, and especially if your database does not offer fine time granularity, implement the `VersionLockingPolicy`. This policy uses integers for field locking and guarantees that you recognize changes.
- If your database time offers a fine granularity, or if you need to know when an object was last updated, implement the `TimestampLockingPolicy`.

Using the OracleAS TopLink SDK

The OracleAS TopLink Software Development Kit (SDK) enables you to extend OracleAS TopLink to access objects stored on nonrelational data stores. To take advantage of the SDK, develop several classes that enable OracleAS TopLink to access your particular data store. You can take advantage of several OracleAS TopLink mappings and use several OracleAS TopLink customization features not used by applications that work with relational databases.

In OracleAS TopLink applications that address a relational database, a query works as follows:

1. The client application builds a query.
2. OracleAS TopLink converts the query search criteria into key-value pairs, formatted as a database row.

3. OracleAS TopLink uses the key-value pairs to build a call to the relational database.

In Step 3, OracleAS TopLink uses an internal mechanism to generate the calls, based on your chosen data repository. The SDK enables you replace the internal mechanism with one of your own design. This enables you to develop custom calls that address non-relational datasources.

There are four major steps to using the SDK:

- Define an accessor that holds a connection to your data store.
- Create the application calls that read data from and write data to your data store. These calls interact with your data store through the accessor and convert the data to and from OracleAS TopLink `DatabaseRows`.
- Build descriptors and mappings that map your object model to the `DatabaseRows`.
- Deploy the application using sessions.

Step One: Define an Accessor

OracleAS TopLink uses an accessor to maintain a connection to your data store. To define an accessor, create a subclass of `SDKAccessor`. The `SDKAccessor` is an implementation of the `Accessor` interface, which offers a minimal implementation, including:

- The protocol required by the `Accessor` interface
- Message logging
- Non-JTS transaction support
- Call execution

If you do not define your own accessor, the SDK creates an instance of `oracle.toplink.sdk.SDKAccessor` and uses it during execution.

Data Store Connection

When logging in, an OracleAS TopLink session uses your accessor to establish a connection to your data store by calling the `connect(DatabaseLogin, Session)` method.

The `DatabaseLogin` passed in holds several settings, including the user ID and password set by your application. As with regular database logins, you can store several user-defined properties in the `DatabaseLogin` that configure its connection. The API for this is:

```
void setProperty(String Object Value)
```

OracleAS TopLink occasionally queries the status of your accessor's connection to your data store by calling the `isConnected()` method. This method returns true if the accessor still has a connection. You can set your accessor to verify the viability of the connection. This verification is optional if you know your data store will not drop the connection.

If your accessor's connection times out or disconnects, your application can attempt to reconnect by calling the `reestablishConnection(Session)` method. Your application (rather than OracleAS TopLink) calls this method, which enables you to control when the application attempts to reconnect.

When logging out, an OracleAS TopLink session uses your accessor to disconnect from your data store by calling the `disconnect(Session)` method.

Call Execution

During execution of your application, the OracleAS TopLink session holds your accessor and uses it whenever you execute a call with the `executeCall(Call, DatabaseRow, Session)` method.

Transaction Processing

If you execute calls together within the context of a transaction, OracleAS TopLink indicates to your accessor that your connection must begin a transaction by calling the `beginTransaction(Session)` method. If any Exceptions occur during the execution of the calls contained within the transaction, OracleAS TopLink rolls back the transaction by calling `rollbackTransaction(Session)`. If all the calls execute successfully, OracleAS TopLink commits the transaction by calling `commitTransaction(Session)`.

Step Two: Create the Application Calls

OracleAS TopLink calls are the hooks OracleAS TopLink uses to call out to your code for reading and writing your nonrelational data. To write a call for the SDK, subclass `oracle.toplink.sdk.AbstractSDKCall` and implement the `execute(DatabaseRow, Accessor)` method.

The code for calls is specific to your particular data store. To see an example implementation of these calls, review the code for the XML calls in the package `oracle.toplink.xml`. "OracleAS TopLink XML Support" on page 5-56 also discusses these calls.

A minimum implementation requires the following calls for every persistent Class stored in a nonrelational data store:

- Read Object Call
- Read All Call
- Insert Call
- Update Call
- Delete Call
- Does Exist Call

Depending on the capabilities of your data store, you may need to implement the following custom calls:

- Named Session Call
- Named Descriptor Call

If you use OracleAS TopLink relationship mappings, implement the appropriate calls to read the reference object(s) for each mapping.

You can divide any individual call into multiple calls, and combine the resulting calls into a single query.

Input Database Row

Calls include the key-value pairs that define the query. OracleAS TopLink formats this information into an input database row that implements the `java.util.map` interface. The input database row can also hold nested database rows or nested direct values. This allows OracleAS TopLink to manipulate non-normalized, hierarchical data.

SDK Field Value Use `oracle.toplink.sdk.SDKFieldValue to manipulate` nested database rows and direct values. Within the OracleAS TopLinkSDK, any field in a database row can have a value that is an instance of SDK field value. An SDK field value can hold one or more nested database rows or direct values.

An SDK field value can also include a data type name indicating the type of elements held in the nested collection. The data store requirements for nested data elements determine whether the data type name is required.

Nested database rows can also themselves contain nested database rows, and there is no limit to the nesting.

Table 5–3 lists several examples in this chapter that illustrate the use of SDK field value.

Table 5–3 SDK Field Value Examples

Example of an SDK Field Value to	Reference
Read a single nested row	Example 5–33 on page 5-46
Write a single nested row	Example 5–34 on page 5-46
Read nested direct values	Example 5–37 on page 5-48
Write nested direct values	Example 5–38 on page 5-48
Read nested rows	Example 5–43 on page 5-53
Write nested rows	Example 5–44 on page 5-48

Read Object Call

A read object call reads the data required to build a single object for a specified primary key. OracleAS TopLink passes the search criteria to the `ReadObject` call as an input database row. The call returns a single database row for the specified object.

Read All Call

A read all call reads the data required to build a collection of *all* objects (instances) for a particular class. OracleAS TopLink passes an empty database row to the `ReadAll` call. The call returns a collection of all the database rows for the selected class.

Insert Call

An insert call inserts a newly created object on the appropriate data store. OracleAS TopLink passes values for all mapped fields for the inserted object as an input database row. The call returns a count of the number of rows inserted, generally one.

Update Call

An update call writes the data for a modified object to the appropriate data store. OracleAS TopLink passes the primary keys and values for all the mapped fields for the updated objects as an input database row. The call returns a count of the number of rows updated, generally one.

Delete Call

A delete call deletes the data from the data store based on primary key. OracleAS TopLink provides primary keys for the delete call as an input database row. The call returns a count of the number of rows deleted, generally one.

Does Exist Call

A does exist call checks for the existence of data for a specified primary key. This enables OracleAS TopLink to determine an insert or update call, depending on the result. OracleAS TopLink provides primary keys for the does exist call as an input database row. The call returns a null if the object does not exist on the data store, and a database row if the object does exist.

Custom Call

You can write a custom call to support other capabilities provided by your data store. Your custom calls can leverage parameter binding. Store custom calls as named queries in the OracleAS TopLink database session or in any OracleAS TopLink descriptor. Pass values to the calls as an input database row. The call returns whatever is appropriate for the containing query. Table 5–4 lists the query types and return values for Custom Calls.

Table 5–4 Query Types and Return Values for Custom Calls

Query	Return value
`DataModifyQuery`	Row count
`DeleteAllQuery`	Row count
`DeleteObjectQuery`	Row count
`InsertObjectQuery`	Row count
`UpdateObjectQuery`	Row count
`DataReadQuery`	Vector of database rows
`DirectReadQuery`	Vector of database rows
`ValueReadQuery`	Vector of database rows
`ReadAllQuery`	Vector of database rows
`ReadObjectQuery`	Database row

FieldTranslator

If the names of fields expected by your OracleAS TopLink descriptors and database mappings differ from those generated by your data store (for example: when dealing with aggregate objects), you can resolve the mismatch by:

- Subclassing the `oracle.toplink.sdk.AbstractSDKCall`. This enables you to use the `SDKFieldTranslator` class.
- Building the `SDKFieldTranslators` into your own calls.
- Creating your own mechanism for translating field names between OracleAS TopLink and your data store on a per-call basis.

Field Translator Interface The `oracle.toplink.sdk.FieldTranslator` interface defines a simple read and write protocol for translating the field names in a database row. The default implementation of the `oracle.toplink.sdk.DefaultFieldTranslator` interface performs no translations.

oracle.toplink.sdk.SimpleyFieldTranslator The `oracle.toplink.sdk.SimpleFieldTranslator` provides a mechanism for translating field names in a database row, either before the row is written to the data store or after the row is read from the data store. `SimpleFieldTranslator` also allows for wrapping another `FieldTranslator`, and for processing the read and write translations through the wrapped `FieldTranslator`. A `SimpleFieldTranslator` also translates the field names of any nested database rows contained in SDK field values.

Example 5–24 Building a SimpleFieldTranslator

```
/* Add translations for the first and last name field names. F_NAME on the data
store will be converted to FIRST_NAME for OracleAS TopLink, and vice versa.
Likewise for L_NAME and LAST_NAME. */
AbstractSDKCall call = new EmployeeCall();
SimpleFieldTranslator translator = new SimpleFieldTranslator();
translator.addReadTranslation("F_NAME", "FIRST_NAME");
translator.addReadTranslation("L_NAME", "LAST_NAME");
call.setFieldTranslator(translator);
```

`AbstractSDKCall` offers methods that enable you to perform the same operation, without building your own translator.

```
AbstractSDKCall call = new EmployeeCall();
    call.addReadTranslation("F_NAME", "FIRST_NAME");
```

```
call.addReadTranslation("L_NAME", "LAST_NAME");
```

If your calls are all subclasses of `AbstractSDKCall`, use the method in `SDKDescriptor` that sets the same field translations for all the calls in the `DescriptorQueryManager`, as follows:

```
descriptor.addReadTranslation("F_NAME", "FIRST_NAME");
descriptor.addReadTranslation("L_NAME", "LAST_NAME");
```

SDKDataStoreException

If your call encounters a problem while accessing your data store, it raises an `oracle.toplink.sdk.SDKDataStoreException`. This exception can hold an error code, a session, an internal exception, a database query, and an accessor. An exception handler can use this state to recover from the thrown exception or to provide useful information to the user or developer about the cause of the exception.

Step Three: Build Descriptors and Mappings

You can use your developed calls to define the descriptors and mappings. OracleAS TopLink can use these descriptors and mappings to read and write your objects rather than the normal OracleAS TopLink descriptors. Use a subclass of the descriptor, `oracle.toplink.sdk.SDKDescriptor`. This class provides support for mappings supplied by the SDK. The SDK supports most of the typical OracleAS TopLink mappings, as well as the mappings that provide access to non-normalized data.

SDK Descriptor

The SDK supports most of the properties of the standard descriptor, including:

- Basic Properties
- Descriptor Query Manager
- Sequence Numbers
- Inheritance

For more information about other properties, see "Other Supported Properties" on page 5-39 and "Unsupported properties" on page 5-39.

Basic Properties The code required to build a basic `SDKDescriptor` is almost identical to that used to build a normal descriptor.

Example 5–25 A Basic SDK Descriptor

```
SDKDescriptor descriptor = new SDKDescriptor();
descriptor.setJavaClass(Employee.class);
descriptor.setTableName("employee");
descriptor.setPrimaryKeyFieldName("id");
```

The Java class is required. The table name is usually required. How you store the data and translate the calls determines whether you allow multiple table names. OracleAS TopLink also requires the primary key field name, which OracleAS TopLink uses to maintain object identity.

Descriptor Query Manager The major difference between building an `SDKDescriptor` and building a standard descriptor is that you define *all* the custom queries for the descriptor's query manager.

Example 5–26 Building a Database Query for the Descriptor's Query Manager

```
ReadObjectQuery query = new ReadObjectQuery();
    query.setCall(new EmployeeReadCall());
    descriptor.getQueryManager().setReadObjectQuery(query);
```

`SDKDescriptor` has several convenience methods that simplify setting all these calls.

```
descriptor.setReadObjectCall(new EmployeeReadCall());
descriptor.setReadAllCall(new EmployeeReadAllCall());

descriptor.setInsertCall(new EmployeeInsertCall());
descriptor.setUpdateCall(new EmployeeUpdateCall());
descriptor.setDeleteCall(new EmployeeDeleteCall());

descriptor.setDoesExistCall(new EmployeeDoesExistCall());
```

You can also create custom calls to an `SDKDescriptor` that enable you to set query criteria at runtime.

Example 5–27 A Dynamic Query

```
// "LastName" is an argument for the call
descriptor.addReadAllCall("readByLastName", new EmployeesByLastNameCall(), "LastName");
// "Location" is an argument for the call
```

```
descriptor.addReadObjectCall("readByLocation", new EmployeeByLocationCall(), "Location");
```

Your application invokes custom calls at runtime and provides a parameter value through a database row. The call communicates with your data store and returns a database row with the appropriate data to build an instance of the returned object.

Sequence Numbers If your data store provides support for sequencing, you can configure your descriptor to use sequence numbers.

Example 5–28 Using Sequencing

```
descriptor.setSequenceNumberName("employee");
descriptor.setSequenceNumberFieldName("id");
```

To use sequencing, define several custom queries that query and update the sequence numbers.

For more information, see the *Oracle Application Server TopLink API Reference*.

Inheritance The `SDKDescriptor` supports OracleAS TopLink inheritance settings. If you define a single table in the root class descriptor, but do not define any additional tables in the subclass descriptors, calls build database rows for a single table, leaving out the fields that are not required for the particular subclass descriptor.

For more information, see "Inheritance" on page 3-47.

Other Supported Properties The `SDKDescriptor` supports most other descriptor properties without any special consideration, including:

- Interfaces
- Copy Policy
- Instantiation Policy
- Wrapper Policy
- Identity Maps
- Descriptor Events

Unsupported properties The OracleAS TopLink SDK does not support the following descriptor properties:

- Query Keys

- Optimistic Locking

Standard Mappings

The OracleAS TopLink SDK provides support for many of the database mappings in the base OracleAS TopLink class library, as well as hierarchical data mechanisms.

Direct Mappings The OracleAS TopLink SDK supports all the base OracleAS TopLink direct mappings:

- Direct-to-field mappings
- Type conversion mappings
- Object type mappings
- Serialized object mappings
- Transformation mappings

The only mapping that requires special consideration is the `SerializedObjectMapping`. Read calls that support descriptors with this type of mapping must return the data for the `SerializedObjectMapping` either as a byte array (`byte[]`) or as a hexadecimal string representation of a byte array. OracleAS TopLink passes the data for the `SerializedObjectMapping` to any Write call as a byte array (`byte[]`).

Relationship mappings The OracleAS TopLink SDK provides support for several of the base OracleAS TopLink relationship mappings. In addition, alternative mappings provide any functionality lost by unsupported mappings in the SDK.

Private relationships The OracleAS TopLink SDK offers full support for private relationships. When you write an object to the data store, OracleAS TopLink also writes its private objects. Likewise, when you remove an object, OracleAS TopLink also removes its private objects.

Because OracleAS TopLink invokes the appropriate calls to write and delete private objects, your calls do not need to be aware of private relationships. OracleAS TopLink acquires the appropriate call for a particular private object from the object's `DescriptorQueryManager`.

Indirection The OracleAS TopLink SDK provides full support for OracleAS TopLink indirection, including valueholder indirection, proxy indirection and transparent indirection.

For more information, see "Indirection" on page 3-27.

Because OracleAS TopLink invokes calls to read in reference objects when required, your calls do not need to be aware of indirection. OracleAS TopLink acquires the appropriate call for indirect relationships from the custom selection query from the relationship's mapping.

Container Policy The OracleAS TopLink SDK supports OracleAS TopLink container policies. A container policy allows you to specify the concrete class OracleAS TopLink uses to store query results.

Calls do not need to be aware of the container policy. For ease of development, specify your calls to use a `java.util.Vector` to return collections of database rows. OracleAS TopLink converts any vector of database rows into the appropriate collection (or map) of business objects. OracleAS TopLink determines the appropriate concrete container class by getting the container policy from the appropriate database query or database mapping.

Aggregate Object Mapping Although the limitations of the aggregate object mapping prevent the OracleAS TopLink SDK from supporting it, the SDK does provide nearly equivalent behavior. See "SDK Aggregate Object Mapping" on page 5-44.

One-To-One Mapping The OracleAS TopLink SDK supports one-to-one mapping. Provide the mapping with a custom selection query as follows:

```
ReadObjectQuery query = new ReadObjectQuery();
query.setCall(new ReadAddressForEmployeeCall());
mapping.setCustomSelectionQuery(query);
```

The Read call used for the custom selection query must be aware of whether the mapping uses either a source foreign key or a target foreign key. It must also know which fields hold the primary or foreign key values. Because the mapping contains this information, construct the call with the mapping as a parameter, as follows:

```
query.setCall(new ReadAddressForEmployeeCall(mapping));
```

Variable-One-To-One Mapping The OracleAS TopLink SDK supports variable one-to-one mapping. As with the one-to-one mapping, you must provide the mapping with a custom selection query.

Direct Collection Mapping The OracleAS TopLink SDK supports direct collection mapping. Use a direct collection mapping if your data store requires that you perform an additional query to fetch the direct values related to a given object. If your data store includes the direct values in a hierarchical fashion within the database row for a given object, use SDK direct collection mapping.

For more information about SDK direct collection mapping, see "SDK Direct Collection Mapping" on page 5-47.

Provide the direct collection mapping with several custom queries. Because the objects contained in a direct collection do not have a descriptor, provide the mapping with the queries that OracleAS TopLink uses to insert and delete the reference objects.

Example 5–29 Mappings and Custom Selection Queries for Direct Collection Mapping

```
DirectReadQuery readQuery = new DirectReadQuery();
readQuery.setCall(new ReadResponsibilitiesForEmployeeCall());
mapping.setCustomSelectionQuery(readQuery);

DataModifyQuery insertQuery = new DataModifyQuery();
insertQuery.setCall(new InsertResponsibilityForEmployeeCall());
mapping.setCustomInsertQuery(insertQuery);

DataModifyQuery deleteAllQuery = new DataModifyQuery();
deleteAllQuery.setCall(new DeleteResponsibilitiesForEmployeeCall());
mapping.setCustomDeleteAllQuery(deleteAllQuery);
```

The mapping does not need a custom update query because, if any of the reference objects change, OracleAS TopLink deletes and reinserts them.

The Read and Delete calls for this mapping must know which fields hold the primary key values. Because the mapping contains this information, construct the call with the mapping as a parameter, as follows:

```
readQuery.setCall(new ReadResponsibilitiesForEmployeeCall(mapping));
deleteAllQuery.setCall(new DeleteResponsibilitiesForEmployeeCall(mapping));
```

One-To-Many Mapping The OracleAS TopLink SDK supports one-to-many mapping. Use a one-to-many mapping if the reference objects have foreign keys to the source object (target foreign keys). However, if the foreign keys are forward-pointing (source foreign keys) and are included in a hierarchical fashion in the database row for a given object, use SDK aggregate object mapping instead.

For more information about SDK aggregate object mapping, see "SDK Aggregate Object Mapping" on page 5-44.

Example 5–30 Mappings and Custom Selection Queries for One-To-Many Mapping

```
ReadAllQuery readQuery = new ReadAllQuery();
readQuery.setCall(new ReadManagedEmployeesForEmployeeCall());
mapping.setCustomSelectionQuery(readQuery);
```

You can also provide the mapping with a custom `DeleteAll` query. If this query is present, OracleAS TopLink uses it to delete all components in the relationship with a single query. Without this query, OracleAS TopLink deletes components individually.

Example 5–31 Defining a Delete All Query

```
DeleteAllQuery deleteAllQuery = new DeleteAllQuery();
deleteAllQuery.setCall(new DeleteManagedEmployeesForEmployeeCall());
mapping.setCustomDeleteAllQuery(deleteAllQuery);
```

The `Read` and `Delete` calls for this mapping must know which fields hold the primary key values. Because the mapping contains this information, construct the call with the mapping as a parameter, as follows:

```
readQuery.setCall(new ReadManagedEmployeesForEmployeeCall(mapping));
deleteAllQuery.setCall(new DeleteManagedEmployeesForEmployeeCall(mapping));
```

Aggregate Collection Mapping The OracleAS TopLink SDK supports aggregate collection mapping. Aggregate collection mapping is similar to the one-to-many mapping but does not require a back reference mapping from each of the target objects to the source object.

As with the one-to-many mapping, supply the mapping with a custom selection query. You can also include a DeleteAll query.

Many-To-Many Mapping Because the many-to-many mapping depends on the relational implementation of many-to-many relationships, the OracleAS TopLink SDK does not support it.

Structure Mapping Because structure mapping depends on the object-relational data model, the OracleAS TopLink SDK does not support it. However, the SDK aggregate object mapping provides nearly identical functionality.

For more information, see "SDK Aggregate Object Mapping" on page 5-44.

Reference Mapping Because the reference mapping depends on the object-relational data model, the OracleAS TopLink SDK does not support it. However, OneToOne mapping provides nearly identical functionality.

For more information, see "One-To-One Mapping" on page 5-41).

Array Mapping Because the array mapping depends on the object-relational data model, the OracleAS TopLink SDK does not support it. However, SDK direct collection mapping provides nearly identical functionality.

For more information, see "SDK Direct Collection Mapping" on page 5-47.

Object Array Mapping Because the object array mapping depends on the object-relational data model, the OracleAS TopLink SDK does not support it. However, SDK direct collection mapping provides nearly identical functionality. For more information, see "SDK Direct Collection Mapping" on page 5-47.

Nested Table Mapping Because the nested table mapping depends on the object-relational data model, the OracleAS TopLink SDK does not support it. However, SDK object collection mapping provides nearly identical functionality.

For more information, see "SDK Object Collection Mapping" on page 5-51.

SDK Mappings

The OracleAS TopLink SDK provides four new mappings that support non-normalized, hierarchical data:

- SDK Aggregate Object Mapping
- SDK Direct Collection Mapping
- SDK Aggregate Collection Mapping
- SDK Object Collection Mapping

SDK Aggregate Object Mapping The SDK aggregate object mapping is similar to the standard aggregate object mapping, but differs as follows:

- All fields that the reference (aggregate) descriptor uses to build the aggregate object appear in a single, *nested* database row, not in the base database row. The base database row has a single field mapped to the aggregate object attribute that contains an SDK field value. This SDK field value holds the nested database row, and this nested database row contains all the fields needed by the reference descriptor to build an instance of the aggregate object.

- There is no need for field name translations. If necessary, the appropriate call can translate the field names when it converts data from the data store's native format to an OracleAS TopLink database row (and vice versa), as described in "FieldTranslator" on page 5-36.
- There is no need for the `isNullAllowed` flag. Because the fields used to build the aggregate object appear in a single field in the base database row, there is no need to specify how to handle null field values. If the attribute is null, then the field value in the base database row is also null. If the attribute contains an instance of the aggregate object with all null attributes, then the field value in the base database row is an SDK field value with a single, nested database row whose field values are all null.

The code to build an SDK aggregate object mapping is similar to that for the aggregate object mapping. Specify an attribute name, a reference class, and a field name.

Example 5–32 Building an SDK Aggregate Object Mapping

```
SDKAggregateObjectMapping mapping = new SDKAggregateObjectMapping();
mapping.setAttributeName("period");
mapping.setReferenceClass(EmploymentPeriod.class);
mapping.setFieldName("period");
descriptor.addMapping(mapping);
```

Because the data used to build the aggregate object appears nested within the base database row, a separate query is not necessary to fetch the data for the aggregate object. Table 5–5 illustrates an example of the values contained in a typical database row with data for an aggregate object.

Table 5–5 Field Names and Mappings for SDK Aggregate Object Mapping

Field Name	Field Value
`employee.id`	`1`
`employee.firstName`	`"Grace"`
`employee.lastName`	`"Hopper"`

Table 5–5 Field Names and Mappings for SDK Aggregate Object Mapping (Cont.)

Field Name	Field Value
`employee.period`	`SDKFieldValue elements=[ DatabaseRow(employmentPeriod.startDate="1943-01-01" employmentPeriod.endDate="1992-01-01" ) ] elementDataTypeName="employmentPeriod" isDirectCollection=false`

In the Example 5–33, an SDK aggregate object mapping maps the attribute `period` to the field `employee.period` and specifies the reference class as `EmploymentPeriod`. The value in the field, `employee.period`, is an SDK field value with a single, nested database row. The `EmploymentPeriod` descriptor uses this nested row to build the aggregate object.

The names of the fields in the nested database row must match those expected by the `EmploymentPeriod` descriptor.

Example 5–33 Reading for an SDK Aggregate Object Mapping

```
Integer id = (Integer) row.get("employee.id");
String firstName = (String) row.get("employee.firstName");
String lastName = (String) row.get("employee.lastName");

SDKFieldValue value = (SDKFieldValue) row.get("employee.period");
DatabaseRow nestedRow = (DatabaseRow) value.getElements().firstElement();
String startDate = (String) nestedRow.get("employmentPeriod.startDate");
String endDate = (String) nestedRow.get("employmentPeriod.endDate");
```

Example 5–34 Write Call that Supports SDK Aggregate Object Mapping

```
DatabaseRow row = new DatabaseRow();
row.put("employee.id", new Integer(1));
row.put("employee.firstName", "Grace");
row.put("employee.lastName", "Hopper");

DatabaseRow nestedRow = new DatabaseRow();
nestedRow.put("employmentPeriod.startDate", "1943-01-01");
nestedRow.put("employmentPeriod.endDate", "1992-01-01");
Vector elements = new Vector();
```

```
elements.addElement(nestedRow);

SDKFieldValue value = SDKFieldValue.forDatabaseRows(elements,"employmentPeriod");
row.put("employee.period", value);
```

SDK Direct Collection Mapping The SDK direct collection mapping is similar to the standard direct collection mapping because it represents a collection of objects that are not OracleAS TopLink-enabled (they are not associated with any OracleAS TopLink descriptors).

The SDK direct collection mapping differs from a direct collection mapping because the data representing the collection of objects appears nested within the base database row. As a result, a separate query to the data store is not necessary to read the data.

To build an SDK direct collection mapping, specify the attribute and the field names. Alternatively, if your data store requires you to indicate the data type name of each element in the direct collection, include the data type name instead.

Example 5–35 Building an SDK Direct Collection Mapping

```
SDKDirectCollectionMapping mapping = new SDKDirectCollectionMapping();
mapping.setAttributeName("responsibilitiesList");
mapping.setFieldName("responsibilities");
mapping.setElementDataTypeName("responsibility");
descriptor.addMapping(mapping);
```

The SDK direct collection mapping container policy enables you to specify the concrete implementation of the `Collection` interface that holds the direct collection, as follows:

```
mapping.useCollectionClass(Stack.class);
```

The SDK direct collection mapping also allows you to specify the class of objects in the direct collection or the database row. If possible, OracleAS TopLink converts the objects contained by the direct collection before setting the attribute in the object or passing the collection to your call.

Example 5–36 Specifying Object Types for an SDK Direct Collection Mapping

```
mapping.setAttributeElementClass(Class.class);
mapping.setFieldElementClass(String.class);
```

Because the data used to build the aggregate object appears nested within the base database row, a separate query is not necessary to fetch the data for the SDK direct collection mapping.

Table 5–6 illustrates examples of the values that appear in a typical database row with data for a direct collection.

Table 5–6 Field Names and Values for SDK Aggregate Object Mapping

Field Name	Field Value
`employee.id`	`1`
`employee.firstName`	`"Grace"`
`employee.lastName`	`"Hopper"`

`employee. responsibilities`	`SDKFieldValue`
	`elements=[`
	`"find bugs"`
	`"develop compilers"`
	`]`
	`elementDataTypeName="responsibility"`
	`isDirectCollection=true`

In Example 5–37, an SDK direct collection mapping maps the attribute `responsibilitiesList` to the field `employee.responsibilities`. The value in the field `employee.responsibilities` is an SDK field value that contains a collection of strings that make up the direct collection.

Example 5–37 Reading for an SDK Direct Collection Mapping

```
DatabaseRow row = new DatabaseRow();
row.put("employee.id", new Integer(1));
row.put("employee.firstName", "Grace");
row.put("employee.lastName", "Hopper");

Vector responsibilities = new Vector();
responsibilities.addElement("find bugs");
responsibilities.addElement("develop compilers");
SDKFieldValue value = SDKFieldValue.forDirectValues(responsibilities, "responsibility");
row.put("employee.responsibilities", value);
```

Example 5–38 Write Call that Supports an SDK Direct Collection Mapping

```
Integer id = (Integer) row.get("employee.id");
String firstName = (String) row.get("employee.firstName");
```

```
String lastName = (String) row.get("employee.lastName");

SDKFieldValue value = (SDKFieldValue) row.get("employee.responsibilities");
Vector responsibilities = value.getElements();
```

SDK Aggregate Collection Mapping The SDK aggregate collection mapping maps attributes that are collections of aggregate objects constructed from data contained in the base database row.

The data that the reference (aggregate) descriptor uses to build the aggregate collection appears in a collection of nested database rows, not in the base database row. The base database row has a single field mapped to the aggregate collection attribute that contains an SDK field value. This SDK field value holds the nested database rows, and the nested database rows each contain all the fields the reference descriptor requires to build a single element in the aggregate collection.

To build an SDK aggregate collection mapping, specify an attribute name, a reference class, and a field name.

Example 5–39 Building an SDK Aggregate Collection Mapping

```
SDKAggregateCollectionMapping mapping = new SDKAggregateCollectionMapping();
mapping.setAttributeName("phoneNumbers");
mapping.setReferenceClass(PhoneNumber.class);
mapping.setFieldName("phoneNumbers");
descriptor.addMapping(mapping);
```

The SDK aggregate collection mapping container policy enables you to specify the concrete implementation of the `Collection` interface that holds the direct collection.

```
mapping.useCollectionClass(Stack.class);
```

Because the data used to build the aggregate collection is already nested within the base database row, it does not require a separate query to fetch the data.

Table 5–7 illustrates examples of the values that appear in a typical database row with data for an aggregate collection.

Table 5–7 Field names and values for SDK Aggregate Collection Mapping

Field Name	Field Value
`employee.id`	1
`employee.firstName`	“Grace”

Table 5–7 Field names and values for SDK Aggregate Collection Mapping (Cont.)

Field Name	Field Value
`employee.lastName`	"Hopper"

`employee. phoneNumbers`	`SDKFieldValue`
	`elements=[`
	`DatabaseRow(`
	`phone.areaCode="888"`
	`phone.number="555-1212"`
	`phone.type="work"`
	`)`
	`DatabaseRow(`
	`phone.areaCode="800"`
	`phone.number="555-1212"`
	`phone.type="home"`
	`)aggregate collection mapping`
	`]`
	`elementDataTypeName="phone"`
	`isDirectCollection=false`

In Example 5–40, an SDK aggregate collection mapping maps the attribute `phoneNumbers` to the field `employee.phoneNumbers` and specifies the reference class as `phoneNumber`. The value in the field `employee.phoneNumbers` is an SDK field value with a collection of nested database rows. The `PhoneNumber` descriptor uses these nested rows to build the elements of the aggregate collection. The names of the fields in the nested database rows must match those expected by the `PhoneNumber` descriptor.

Example 5–40 Reading for an SDK Aggregate Collection Mapping

```
Integer id = (Integer) row.get("employee.id");
String firstName = (String) row.get("employee.firstName");
String lastName = (String) row.get("employee.lastName");

SDKFieldValue value = (SDKFieldValue) row.get("employee.phoneNumbers");
Enumeration enum = value.getElements().elements();
while (enum.hasMoreElements()) {DatabaseRow nestedRow = (DatabaseRow) enum.nextElement();
String areaCode = (String) nestedRow.get("phone.areaCode");
String number = (String) nestedRow.get("phone.number");
String type = (String) nestedRow.get("phone.type");
...
}
```

Example 5–41 Write Call that Supports an SDK Aggregate Collection Mapping

```
DatabaseRow row = new DatabaseRow();
row.put("employee.id", new Integer(1));

row.put("employee.firstName", "Grace");
row.put("employee.lastName", "Hopper");

Vector elements = new Vector();

DatabaseRow nestedRow1 = new DatabaseRow();
nestedRow1.put("phone.areaCode", "888");
nestedRow1.put("phone.number", "555-1212");
nestedRow1.put("phone.type", "work");
elements.addElement(nestedRow1);

Database nestedRow2 = new DatabaseRow();
nestedRow2.put("phone.areaCode", "800");
nestedRow2.put("phone.number", "555-1212");
nestedRow2.put("phone.type", "home");
elements.addElement(nestedRow2);

SDKFieldValue value = SDKFieldValue.forDatabaseRows(elements, "phone");
row.put("employee.phoneNumbers", value);
```

SDK Object Collection Mapping The SDK object collection mapping is similar to the standard one-to-many mapping—because both map a collection of target objects that use foreign keys to point to their primary keys. However, the foreign keys in SDK object collection mapping appear in the base database row that references the target objects' primary keys. This makes the foreign keys in an SDK object collection mapping forward-pointing, whereas the foreign keys in a one-to-many mapping are back-pointing.

All foreign keys appear in a collection of *nested* database rows, not in the base database row. The base database row includes a single field, mapped to the object collection attribute containing an SDK field value. This SDK field value holds the nested database rows, and these nested database rows each contain the fields required to build a foreign key to an element object's primary key.

The code to build an SDK object collection mapping is similar to that for the one-to-many mapping. Specify an attribute name, a reference class, a field name, and the source foreign and target key relationships.

If your data store requires you to indicate the data type name of each element in the collection of foreign keys, include the data type name. Alternatively, you can provide this information with your call. Build a custom selection query to read the reference objects contained in the collection.

Example 5–42 Building an SDK Object Collection Mapping

```
SDKObjectCollectionMapping mapping = new SDKObjectCollectionMapping();
mapping.setAttributeName("projects");
mapping.setReferenceClass(Project.class);
mapping.setFieldName("projects");
mapping.setSourceForeignKeyFieldName("projectId");
mapping.setReferenceDataTypeName("project");
descriptor.addMapping(mapping);
```

The SDK object collection mapping container policy allows you to specify the concrete implementation of the `Collection` interface that holds the collection of objects.

```
mapping.useCollectionClass(Stack.class);
```

Table 5–8 demonstrates an example of the values contained in a typical database row with data for a collection of foreign keys.

Table 5–8 Field Names and Values for SDK Object Collection Mapping

Field Name	Field Value
`employee.id`	1
`employee.firstName`	"Grace"
`employee.lastName`	"Hopper"

`employee.projects`	`SDKFieldValue`
	`elements=[`
	`DatabaseRow(`
	`project.projectId=42`
	`)`
	`DatabaseRow(`
	`project.projectId=17`
	`)`
	`]`
	`elementDataTypeName="project"`
	`isDirectCollection=false`

Example 5–43 illustrates an SDK object collection mapping that maps the attribute `projects` to the field `employee.projects` and specifies the reference class as `Project`. The value in the field `employee.projects` is an SDK field value with a collection of nested database rows.

Nested rows contain foreign keys that the mapping's custom selection query uses to read in the elements of the object collection. The field names in the nested database rows must match those expected by the custom selection query's call.

Example 5–43 Reading for an SDK Object Collection Mapping

```
DatabaseRow row = new DatabaseRow();
row.put("employee.id", new Integer(1));
row.put("employee.firstName", "Grace");
row.put("employee.lastName", "Hopper");

Vector elements = new Vector();

DatabaseRow nestedRow1 = new DatabaseRow();
nestedRow1.put("project.projectId", new Integer(42));
elements.addElement(nestedRow1);

DatabaseRow nestedRow2 = new DatabaseRow();
nestedRow2.put("project.projectId", new Integer(17));
elements.addElement(nestedRow2);

SDKFieldValue value = SDKFieldValue.forDatabaseRows(elements, "project");
row.put("employee.projects", value);
```

Example 5–44 Write Call that Supports an SDK Object Collection Mapping

```
Integer id = (Integer) row.get("employee.id");
String firstName = (String) row.get("employee.firstName");
String lastName = (String) row.get("employee.lastName");

SDKFieldValue value = (SDKFieldValue row.get("employee.projects");
Enumeration enum = value.getElements().elements();
while (enum.hasMoreElements(DatabaseRow nestedRow = (DatabaseRow)enum.nextElement();
Object projectId = nestedRow.get("project.projectId");
// do stuff with the foreign key
}
```

Step Four: Deploy the Application Using Sessions

After you develop your accessor and calls, and map your object model to your data store, you can configure and log in to a database session, following these steps:

- If necessary, build an instance of your custom platform.
- If necessary, build an instance of SDK login, with this custom platform.
- Build an OracleAS TopLink project with this SDK login, populating it with your descriptors.
- Acquire a session from this OracleAS TopLink project and log in.

  For more information about acquiring a session, see "Session Manager" on page 4-29.

SDK Platform and Sequencing

OracleAS TopLink uses the platform classes to isolate the database platform-specific implementations of two major activities:

- SQL generation
- Sequence number generation

Because the OracleAS TopLink SDK is generally unconcerned with SQL generation, you usually build a custom platform only if your data store provides a mechanism for generating sequence numbers. In this case, create your subclass, and override the appropriate methods for building the calls that read and update sequence numbers.

If you use sequence numbers and want OracleAS TopLink to manage them for you, create a subclass of `oracle.toplink.sdk.SDKPlatform`.

Use the `buildSelectSequenceCall()` method to build and call the sequence number `Read` call. OracleAS TopLink invokes this call to read the value of a specific sequence number. The database row in the call contains the `sequenceNameFieldName` (as set in the SDK login), and the field value is the name of the sequence number returned by the call.

The `buildUpdateSequenceCall()` method builds the sequence number Update call. OracleAS TopLink invokes this call to update the value of a specific sequence number. The database row in the call contains two fields:

- The first field name is the `sequenceNameFieldName` (as set in the SDK login); the field value is the name of the sequence number updated by the call.

- The second field name is the `sequenceCounterFieldName` (as set in the SDK login); the field value is the new value for the sequence number identified by the first field.

SDK Login

If you build a custom SDK platform, use it to construct and configure your SDK login.

```
SDKLogin login = new SDKLogin(new EmployeePlatform());
```

If you do not require a custom platform, use the default constructor for SDK login.

```
SDKLogin login = new SDKLogin();
```

If you use a custom accessor to maintain a connection to your data store, configure the login to use it. Doing this enables OracleAS TopLink to construct a new instance of your accessor when the application requires a connection to the data store. If you do not use a custom accessor, you do not need to set this property. In that case, the login uses the `SDKAccessor` class by default.

```
login.setAccessorClass(EmployeeAccessor.class);
```

You can then configure the values of the standard login properties.

```
login.setUserName("user");
login.setPassword("password");

login.setSequenceTableName("sequence");
login.setSequenceNameFieldName("name");
login.setSequenceCounterFieldName("count");
```

You can store non-OracleAS TopLink properties, in the login. Your custom accessor uses these properties when it connects to the data store.

```
login.setProperty("foo", aFoo);
Foo anotherFoo = (Foo) login.getProperty("foo");
```

OracleAS TopLink Project

Build your OracleAS TopLink project by creating an instance of `oracle.toplink.sessions.Project`, and passing it your login. You can then add your descriptors to the project.

Example 5–45 Instantiating the Project and Adding Descriptors

```
Project project = new Project(login);
project.addDescriptor(buildEmployeeDescriptor());
project.addDescriptor(buildAddressDescriptor());
project.addDescriptor(buildProjectDescriptor());
// etc.
```

Session

After you build your OracleAS TopLink project, obtain a database session (or server session) and log in.

Example 5–46 Obtaining a Session and Login

```
DatabaseSession session = project.createDatabaseSession();
session.login();
```

When you finish with the session, log out.

```
session.logout();
```

Unsupported Features

The OracleAS TopLink SDK does not offer support for the following regular OracleAS TopLink features:

- Expressions
- Pessimistic locking
- Cursored streams and scrollable cursors

OracleAS TopLink XML Support

OracleAS TopLink enables you to read and modify objects in XML files. Object-to-XML (O-X) mapping enables your application to deal exclusively with objects, rather than managing the intricacies of XML parsing and deconstruction. You can use OracleAS TopLink XML support to exchange data with other applications (for example: legacy applications or business partner applications).

This section describes:

- Getting Started

- Customizations
- Implementation Details
- XML File Accessor
- XML Call
- XMLTranslator Implementations
- XML Descriptor
- XML Platform
- XML File Login
- XML Schema Manager
- XML Accessor
- XML Translator
- XML Zip File Extension

The OracleAS TopLink implementation of XML support uses a file and directory paradigm to store information, as follows:

- OracleAS TopLink creates and uses a base directory, which is analogous to a relational database that contains a collection of related tables.
- Subdirectories are analogous to the table name in the relational model.
- Filenames are analogous to row within a table in the relational model.

Getting Started

The default XML extension is similar to a regular OracleAS TopLink project. Use the following steps to develop your application:

1. Configure your login using an `XMLFileLogin`.

```
XMLFileLogin login = new XMLFileLogin();
login.setBaseDirectoryName("C:\Employee Database");

// set up the sequences
login.setSequenceRootElementName("sequence");
login.setSequenceNameElementName("name");
login.setSequenceCounterElementName("count");

// create the directories if they don't already exist
```

```
login.createDirectoriesAsNeeded();
```

2. Build your project.

```
Project project = new Project(login);
project.addDescriptor(buildEmployeeDescriptor());
project.addDescriptor(buildAddressDescriptor());
project.addDescriptor(buildProjectDescriptor());
// etc.
```

3. Build your descriptors using `XMLDescriptors`.

```
XMLDescriptor descriptor = new XMLDescriptor();
descriptor.setJavaClass(Employee.class);
descriptor.setRootElementName("employee");
descriptor.setPrimaryKeyElementName("id");
descriptor.setSequenceNumberName("employee");
descriptor.setSequenceNumberElementName("id");
// etc.
```

4. Build your mappings. For the XML extension, `OneToOneMappings` and `SDKObjectCollectionMappings` require custom selection queries as follows:

```
// 1:1 mapping
OneToOneMapping addressMapping = new OneToOneMapping();
addressMapping.setAttributeName("address");
addressMapping.setReferenceClass(Address.class);
addressMapping.privateOwnedRelationship();
addressMapping.setForeignKeyFieldName("addressId");
// build the custom selection query
ReadObjectQuery addressQuery = new ReadObjectQuery();
addressQuery.setCall(new XMLReadCall(addressMapping));
addressMapping.setCustomSelectionQuery(addressQuery);
descriptor.addMapping(addressMapping);
// 1:n mapping

SDKObjectCollectionMapping projectsMapping = new
  SDKObjectCollectionMapping();
projectsMapping.setAttributeName("projects");
projectsMapping.setReferenceClass(Project.class);
projectsMapping.setFieldName("projects");
projectsMapping.setSourceForeignKeyFieldName("projectId");
projectsMapping.setReferenceDataTypeName("project");
// use convenience method to build the custom selection query
projectsMapping.setSelectionCall(new XMLReadAllCall(projectsMapping));
```

```
descriptor.addMapping(projectsMapping);
```

5. Build your database session and log in.

```
DatabaseSession session = project.createDatabaseSession();session.login();
```

6. Configure sequencing, if necessary.

```
(new XMLSchemaManager(session)).createSequences();
```

7. Run your application normally.

   For example:

```
Vector employees = session.readAllObjects(Employee.class);
Employee employee = (Employee) employees.firstElement();
UnitOfWork uow = session.acquireUnitOfWork();
Employee employeeClone = uow.registerObject(employee);
employeeClone.setSalary(employeeClone.getSalary() + 50);
uow.commit();
```

8. Log out when your session is complete.

```
session.logout();
```

Customizations

You can customize the OracleAS TopLink XML extension in two key ways, by modifying:

- Where and how you store the XML documents, by developing your own implementation of the `XMLAccessor` interface
- How XML documents translate into database rows, and the converse, by developing your own implementation of the `XMLTranslator` interface

Implementation Details

The package `oracle.toplink.xml` contains the classes that implement OracleAS TopLink support for O-X mapping. These classes represent a simple example of how to use the OracleAS TopLink SDK as "Using the OracleAS TopLink SDK" on page 5-30 describes.

The XML package defines its own set of interfaces, in addition to the SDK interfaces. You can use these interfaces to alter how you map your objects to XML documents, without re-implementing the entire SDK suite of interfaces and subclasses.

The XML extension includes the following implementations of the SDK interfaces and subclasses:

- XML File Accessor
- XML Call
- XML Descriptor
- XML Platform
- XML File Login
- XML Schema Manager

The XML extension also defines its own set of interfaces into which you can plug your own implementation classes, as follows:

- XML Accessor
- XML Translator
- XML Zip File Extension

These interfaces enable you to easily alter the way your objects map to XML documents.

XML File Accessor

The `XMLFileAccessor` is a subclass of the `SDKAccessor` that defines how the application stores XML documents in a native file system. As a subclass of SDK accessor, the XML file accessor does not have to implement any of the accessor protocol, although it does implement the `connect(DatabaseLogin, Session)` method.

The XML file accessor uses the standard SDK method of call execution and does not support transaction processing. This limitation is typical of native file systems.

XML Accessor Implementation

In addition to the `Accessor` interface, the XML file accessor implements the `XMLAccessor` interface. The `XMLAccessor` interface defines the protocol necessary to fetch streams of data for reading and writing XML documents. The XML file accessor implements this protocol by wrapping files in streams that can be used by the XML calls to read or write XML documents.

The `XMLAccessor` methods defined to fetch a stream (either a `java.io.Reader` or `java.ioWriter`) generally require three parameters:

- A root element name
- A database row
- A vector of DatabaseFields (the ordered primary key element names)

The XML file accessor resolves the values of these three parameters to a File. It wraps the file in a stream (either a `java.io.FileReader` or a `java.io.FileWriter`) and returns it to the XML call for processing.

The XML file accessor calculates the file name as follows:

- The configuration of the XML file login determines the base directory. The base directory is analogous to a relational database that contains a collection of related tables. If you do not specify a base directory name, then OracleAS TopLink uses the current working directory (for example: `C:\EmployeeDB`).
- The subdirectory has the same name as the XML root element name. The root element name is analogous to the table name in the relational model, meaning that all XML documents in the same directory have the same root element name (for example: `C:\EmployeeDB\employee`).
- The vector of DatabaseFields and the database row determine the file name root. The filename is analogous to a row within a table in the relational model. The vector indicates which fields in the database row make up the primary key. The values in these fields (which must all be strings) are concatenated together in the order in which they are listed in the vector. This composite string forms the root of the file name (for example: `C:\EmployeeDB\employee\1234`).
- The configuration of the XML file login determines the file name extension. The extension is optional. For example, you can assign an extension to associate the file with other applications. If you do not specify a file name extension, it defaults to `.xml` (for example: `C:\EmployeeDB\employee\1234.xml`).

Directory Creation

You can configure the XML file accessor to create directories automatically when required. To enable this, include the `createsDirectoriesAsNeeded` call, set to **TRUE**, in the XML file login.

The `createsDirectoriesAsNeeded` call causes the accessor to create directories as required, including the base directory. If you set this call to FALSE, the accessor throws an XML data store exception if it encounters a request for an XML document that resolves to a nonexistent directory.

The default for this setting is **FALSE**. Set it to **TRUE** to enable directory creation.

XML Call

The XML call and its subclasses are the layer between the OracleAS TopLink database queries call interface and the XML document accessing protocol provided by an XML accessor.

XML calls include two properties:

XML Stream Policy The `XMLStreamPolicy` is an interface that defines a protocol to fetch streams of data for reading and writing XML documents. XML calls use the implementation from the XML accessor stream policy. This implementation delegates every request for a stream to the XML accessor.

This policy enables you to override the default behavior on a per-call basis. For example, you can name a specific file in a call, rather than relying on the XML file accessor to resolve the required file name. XML file stream policy provides this behavior. To access it use the methods `XMLCall.setFile(File)` and `XMLCall.setFileName(String)`.

XML Translator `XMLCalls` use the `XMLTranslator` object to translate data between an XML document and an OracleAS TopLink database row. This pluggable interface enables you to modify the behavior of the XML calls. The XML calls default implementation of XML translator is `DefaultXMLtranslator`.

XMLTranslator Implementations

Several subclasses of XML call provide concrete implementations of call and SDK call. These classes differ in their implementations of the `Call.execute(DatabaseRow, Accessor)` method.

XML translator implementations offer object calls and data calls.

Object-Level Calls

Object-level calls enable you to call for objects from the datasource. All object calls other than `Read` calls require an association with a database query. OracleAS TopLink provides this automatically when you build a database query and configure it to use a custom call.

`Read` calls are an exception, because they are associated with a relationship mapping do not require an associated database query.

The following subclasses enable you to manipulate objects:

- XML Read Call
- XML Read All Call
- XML Insert Call
- XML Update Call
- XML Delete Call
- XML Does Exist Call

XML Read Call If an `XMLReadCall` includes a reference to a one-to-one mapping, it extracts the foreign key for the mapping's relationship from the database row passed in to the `execute(DatabaseRow, Accessor)` method. If there is no mapping, the XML read call extracts the primary key for the query's associated descriptor from the database row.

In either case, XML read call then uses the resulting key to find the appropriate XML document.

XML Read All Call If the `XMLReadAllCall` includes a reference to an SDK object collection mapping, it extracts the foreign keys for the mapping's relationship from the database row passed in to the `execute(DatabaseRow, Accessor)` method. It then uses the foreign keys to find the appropriate XML documents.

If no mapping is present, the XML read all call determines the root element name for the query's associated descriptor and returns all the DatasebaseRows for that root element name.

XML Insert Call An `XMLInsertCall` takes the database row passed in to the `execute(DatabaseRow, Accessor)` method and uses the primary key to find the appropriate XML document stream. It then takes the modify row from the associated `ModifyQuery`, converts it to an XML document, and writes it out.

If the XML document exists, XML insert call raises an XML data store exception.

XML Update Call An `XMLUpdateCall` takes the database row passed in to the `execute(DatabaseRow, Accessor)` method and uses the primary key to find the appropriate XML document stream. It then takes the modify row from the associated `ModifyQuery`, converts it to an XML document, and writes it out.

If the XML document does not exist, XML update call raises an XML data store exception.

XML Delete Call An `XMLDeleteCall` takes the database row passed in to the `execute (DatabaseRow, Accessor)` method and uses the primary key to find the appropriate XML document stream. It then deletes this stream.

If the XML document exists, the call returns a row count of one. If not, the call returns a row count of zero.

XML Does Exist Call An XML does exist call takes the database row passed in to the `execute(DatabaseRow, Accessor)` method and uses the primary key to find the appropriate XML document stream. If the document exists, OracleAS TopLink converts it to a database row to verify the object's existence. If the object does not exist, OracleAS TopLink returns a null.

Data Calls

Data calls enable you to retrieve data, rather than objects, from the datasource. Because XML data calls are not associated with a database query, they require a root element name and a set of ordered primary key element names. Pass these settings, along with the appropriate database row, to the XML stream policy at runtime. OracleAS TopLink uses this information to determine the appropriate XML document stream.

Example 5–47 A Typical Data Call

```
XMLDataReadCall call = new XMLDataReadCall();
call.setRootElementName("employee");
call.setPrimaryKeyElementName("id");
```

The following subclasses provide data call functionality:

- XML Data Read Call
- XML Data Insert Call
- XML Data Update Call
- XML Data Delete Call

XML Data Read Call An XML data read call takes the database row passed in to the `execute (DatabaseRow, Accessor)` method and uses the primary key to find the appropriate XML document stream, and converts the stream to a database row. OracleAS TopLink returns the database row in a vector to ensure a consistent result object.

If the XML data read call does not include a primary key element, it performs a simple read-all for all the XML documents, with the specified root element name. XML data read call converts these and returns them as a vector of database rows.

You can further configure XML data read calls to specify the fields to return and their types.

Example 5–48 An XML Data Read Call

```
XMLDataReadCall call = new XMLDataReadCall();
call.setRootElementName("employee");
call.setPrimaryKeyElementName("id");
call.setResultElementName("salary");
call.setResultElementType(java.math.BigDecimal.class);
```

XML Data Insert Call An XML data insert call takes the database row passed in to the `execute(DatabaseRow, Accessor)` method and uses the primary key to find the appropriate XML document stream. It then converts that row to an XML document, and writes it out.

If the XML document already exists, XML data insert call raises an XML data store exception.

XML Data Update Call An XML data update call takes the database row passed in to the `execute(DatabaseRow, Accessor)` method and uses the primary key to find the appropriate XML document stream. It then converts that row to an XML document and writes it out.

If the XML document does not already exist, XML data update call raises an XML data store exception.

XML Data Delete Call An XML data delete call takes the database row passed in to the `execute(DatabaseRow, Accessor)` method and uses the primary key to find the appropriate XML document stream. It then deletes this stream.

If the XML document already exists, the call returns a row count of one. If not, the call returns a row count of zero.

XML Descriptor

An `XMLDescriptor` is a subclass of the `SDKDescriptor` that:

- Automatically initializes its query manager with a set of default database queries, configured to use the appropriate XML calls. If you use OracleAS TopLink's default support for XML documents, no further modification of these calls is required.
- Adds methods named in accordance with XML concepts rather than relational concepts. The `setRootElementName(String)` method replaces the `setTableName(String)` method, `setPrimaryKeyElementName(String)` replaces `setPrimaryKeyFieldName(String)`, and so on.

XML Platform

XML platform is a subclass of SDK platform that implements the methods required to support sequence numbers: `buildSelectSequenceCall()` and `buildUpdateSequenceCall()`. These methods build and return the XML data calls that allow OracleAS TopLink to use sequence numbers maintained in XML documents.

To set the root element name for these XML documents, and the names of the elements used to hold the sequence name and sequence counter, specify these elements through the XML file login.

XML File Login

XML file login is a subclass of SDK login that allows you to configure the XML file accessor and XML platform. Use the XML file login to configure the following settings:

- The base directory name for the XML files. This is the directory under which you store the root element name subdirectories.

  For more information about file name resolution, see "XML File Accessor" on page 5-60. The default is the current working directory.

  ```
  login.setBaseDirectoryName("C:\Employee Database");
  ```

- The file name extension for the XML files. The default is `.xml`.

  ```
  login.setFileExtension(".xml");
  ```

- Whether directories for the XML files should be created as needed. The default is **False**.

  ```
  login.setCreatesDirectoriesAsNeeded(true);
  ```

- Sequence number settings.

```
login.setSequenceRootElementName("sequence");
login.setSequenceNameElementName("name");
login.setSequenceCounterElementName("count");
```

XML Schema Manager

XML schema manager is a subclass of SDK schema manager. It provides support for building the XML-based sequences required by your OracleAS TopLink database session. After you build your OracleAS TopLink project, use it to create a database session. Then you can log in and create the required sequences with the XML schema manager.

Example 5–49 Using XML Schema Manager for Sequencing

```
DatabaseSession session = project.createDatabaseSession();
session.login();
SchemaManager manager = new XMLSchemaManager(session);
manager.createSequences();
```

XML Accessor

`XMLAccessor` is an interface that extends the `oracle.toplink.internal.databaseaccess.Accessor` interface. It is the default interface that XML calls use to access streams for a given XML document.

To store XML documents in a non-native file system, provide a custom implementation of this interface. For example, to access your XML documents with a messaging service such the Java Message Service (JMS), you can develop an implementation of XML accessor that translates the method calls into JMS calls.

If you build a custom accessor, configure an XML login to use it.

Example 5–50 Using a Custom Accessor with XML Login

```
XMLLogin login = new XMLLogin();
login.setAccessorClass(XMLJMSAccessor.class);
login.setUserName("user");
login.setPassword("password");
// etc.
```

XML Translator

XML calls use the `XMLTranslator` interface to manipulate XML documents stored in a non-native file system. Each XML call has its own XML translator. The default XML translator is an instance of `DefaultXMLTranslator`, but you can replace the `DefaultXMLTranslator` with your own custom implementation.

XML translator defines the following protocol:

- The `read(java.io.Reader)` method takes a `Reader` that streams over an XML document, converts that document into a database row, and returns that database row.
- The `write(java.io.Writer, DatabaseRow)` method takes a database row, converts it into an XML document, and writes that document out on the `Writer`.

Default XML Translator

As the default XML translator for XML calls, `DefaultXMLTranslator` performs translations between database rows and XML documents. To enable translations with the default XML translator:

- All fields in a database row must have the same table name or default XML translator raises an `XMLDataStoreException`. The table name is the root element name of the XML document.

```
<?xml version="1.0"?>
<employee>
<!-- field values will go here -->
</employee>
```

- Each field in the database row maps to an XML element. The field name is the element name, and the field value is the element content.

```
<?xml version="1.0"?>
<employee>
    <id>1</id>
    <firstName>Grace</firstName>
    <lastName>Hopper</lastName>
</employee>
```

- Any field in the database row with a value of null maps to an empty XML element with an attribute named null whose value is **TRUE**.

```
<managedEmployees null="true"/>
```

- If the value of a field in the database row is an SDK field value, default XML translator converts the elements of the SDK field value into nested XML elements. If the elements of the SDK field value are also database rows, default XML translator translates these recursively, using the same set of translations.

The `DefaultXMLTranslator` delegates the translation to two other classes:

- The `DatabaseRowToXMLTranslator` builds an XML document from a database row and writes it onto a stream.
- The `XMLToDatabaseRowTranslator` reads the XML document from a stream and builds a database row.

XML Zip File Extension

The XML zip file extension is an enhancement to the XML implementation of the SDK. This extension adds the flexibility of maintaining the XML data store in archive files rather than in the directory/file structure of the standard XML data store. The format is similar to the standard XML data store; however, archive files replace directories in representing tables. The archive contains XML documents that map back to a database row in the same manner as if you stored them in a directory.

Using the Zip File Extension

To use the XML Zip file extension, configure your XML login to use the XML zip file accessor.

```
XMLLogin login = new XMLLogin();
login.setAccessorClass(XMLZipFileAccessor.class);
```

Configure Direct File Access With Zip File Extension

To access an XML document within an archive file, the call must know both the archive file location and the name of the XML document entry within the archive. Therefore, the `setFileName()` message sent to an XML call must include both the archive file and the XML document entry name, as follows:

```
XMLReadCall call = new XMLReadCall();
call.setFileName("C:/Employee DataStore/employee.zip", "1.xml");
```

Implementation Details

Zip file support requires the following two packages, stored in the package `oracle.toplink.xml.zip`:

- The XML zip file accessor extends the XML file accessor. It offers the same functionality as the XML file accessor, but supports an XML zip file stream policy rather than the XML file stream policy
- The XML zip file stream policy manages the XML archive files, and returns streams for reading and writing from individual archive entries. It does not provide additional functionality over its XML counterpart, the XML file stream policy, other than managing the added complication of getting read and write streams from an archive file.

6 Queries

Queries are a key element to any Oracle Application Server TopLink application, because they enable OracleAS TopLink to manage persistent data on the database. The query framework that OracleAS TopLink provides gives you the flexibility you need to manage the complex persistence requirements of enterprise applications.

The OracleAS TopLink query framework offers the following key features:

- A rich set of query types that enable you to query for objects, object summaries, and data
- Flexible search criteria, including support for query by example, stored procedures, OracleAS TopLink expressions, Structured Query Language (SQL) and Enterprise JavaBean Query Language (EJB QL)
- Configuration options that enable you to customize query execution, and optimize query performance

To define OracleAS TopLink queries, use the OracleAS TopLink Mapping Workbench, the OracleAS TopLink API, or in the case of entity beans, EJB Finders.

This chapter introduces OracleAS TopLink queries, and includes discussions on:

- Oracle Extension Support
- Advanced Querying
- EJB Finders
- Exception Handling

Introduction to Query Concepts

Queries are the cornerstone of OracleAS TopLink applications. Queries enable you to retrieve information or objects from the database, modify or delete those objects, and create new objects on the database.

The following concepts are key to understanding OracleAS TopLink queries:

- Query Types
- Query Components
- Query Configuration Options
- Query Development Options

Query Types

The type of query you build determines the type of result set the query returns. You can build:

- Object queries that return an object or objects
- Summary queries that return partial information about an object or objects
- Data queries that return raw data
- Object write queries that modify the objects in the database

Object Queries

Object queries, the most common query type in an OracleAS TopLink application, enable you to search a database for persistent objects. OracleAS TopLink offers two object query mechanisms: a `readObject` query that searches the database for a single object that matches the search criteria, and a `readAll` query that searches for all matching objects.

Object queries search for objects rather than data. For example, a query to find all employees over the age of 40 searches for objects—the employees.

Summary Queries

Summary queries enable you to search for partial information about objects that match your search criteria. There are two types of summary queries:

- Report queries return data from the database tables that represents a portion of the available information. To build a report query, you specify the search criteria and the information you require about the objects in the result set.

Report queries search for information about objects rather than the objects themselves. For example, you can create a report query to discover the average age of all employees in your company. The report query is not interested in the specific objects (the employees), but rather, summary information about them (their average age).

For more information, see "ReportQuery" on page 6-73.

- Partial object queries retrieve partially populated objects from the database rather than complete objects. You do not cache partial objects, nor can you modify them.

  Applications frequently use partial object queries to compile a list for further selection. For example, a query to find the names and addresses of all employees over the age of 40 returns a list of data (the names and addresses) that partially represents objects (the employees). A common next step is to present this list so the user can select the required object or objects from the list.

  For more information, see "Partial Object Reading" on page 6-47.

Data Queries

Data queries enable you to query data fields directly from the database tables rather than objects. Data queries represent a common approach to working with unmapped data, such as foreign keys and object version fields.

Object Write Queries

Object write queries enable you to modify data and objects directly on the database. You can use write queries to insert and update objects on the database. Write queries are useful when you manage simple, nonbusiness object data that have no relationships, such as user preferences.

For more information about write queries, see "Query Objects and Write Operations" on page 6-67.

To avoid concurrency issues when you write more complex data to the database, use the Unit of Work.

For more information, see "Unit of Work Basics" on page 7-12.

Query Components

Query components are the mechanisms with which you build your query. These components include:

- Advanced query mechanisms, such as query by example, OracleAS TopLink expressions, and database stored procedures
- Query languages and syntaxes, such as SQL and EJB QL

OracleAS TopLink Expressions

The OracleAS TopLink expression framework is a querying syntax. Expressions enable you to specify search criteria in a query, based on the object model. They provide support for standard boolean operators, such as AND, OR, and NOT and support many database functions and operators.

You can create expressions in the OracleAS TopLink Mapping Workbench or in the OracleAS TopLink API.

For more information, see "Expressions" on page 6-12.

Query by Example

Limited in complexity, query by example is an intuitive way to express a query. To specify a query by example, provide sample instances of the persistent objects to query, and specify the fields and values that define the query. You can use any valid constructor to create an example object.

For more information, see "Query by Example" on page 6-34.

Stored Procedures

A stored procedure is a function, such as Procedural Language/Structured Query Language (PLSQL) statement or Java code, written on the database. Stored procedures enable you to execute logic and access data on the database server.

For more information, see "Stored Procedure Calls" on page 6-29.

EJB QL

EJB QL presents queries from an object model perspective, enabling users to declare queries using the attributes of each abstract entity bean in the object model. EJB QL includes path expressions that enable navigation over relationships defined for entity beans and dependent objects.

OracleAS TopLink enables you to use EJB QL to define both queries that return Java objects and finders that return EJBs.

For more information, see "EJB QL" on page 6-31.

Custom SQL

SQL is a standard query language enables you to request information from a database. The use of a native query language such as SQL is complex, but it offers advantages unavailable with other querying options.

For more information, see "Custom SQL" on page 6-27.

Query Configuration Options

OracleAS TopLink queries offer several configuration options to customize query execution, cache usage, and performance.

Query Execution Options

The following query execution options enable you to optimize the way you collect and present query results.

Ordering You can specify an order for the results of a query.

For more information, see "Ordering for Read All Queries" on page 6-44.

Collection Types By default, a query that returns a collection of objects presents the objects in a vector. You can specify that the collection be returned in any collection class that implements the `Collection` or `Map` interface (for example: `HashMap`).

For more information, see "Collection Classes" on page 6-46.

Maximum Rows You can set a maximum row size on any read query to limit the size of the result set. Use this to manage queries that can return an excessive number of objects.

For more information, see "Maximum Rows Returned" on page 6-47.

Timeouts You can set the maximum amount of time that OracleAS TopLink waits for results from a query. This forces a hung or lengthy query to abort after the specified time has elapsed.

For more information, see "Query Timeout" on page 6-47.

Query and the Cache

When you execute a query, OracleAS TopLink retrieves the information from either the database or the OracleAS TopLink session cache. You can configure the way queries use the OracleAS TopLink cache to optimize performance.

Refresh Refresh the cache to update all objects in the cache with information from the database. This ensures that all objects in the cache are current.

For more information, see "Refresh" on page 6-66.

In-Memory Querying An in-memory query is a query that is run against the shared session cache. Careful configuration of in-memory querying improves performance, but not all queries benefit from in-memory querying. For example, queries for individual objects based on primary keys generally see performance gains from in-memory querying; queries based on non-primary keys are less likely to benefit.

By default, queries that look for a single object based on primary keys attempt to retrieve the required object from the cache first and then search the database if the object is not in the cache. All other query types search the database first, by default. You can specify whether a given query runs against the in-memory cache, the database, or both.

For more information, see "In-Memory Query Cache Usage" on page 6-62.

Caching Results By default, OracleAS TopLink stores query results in the session cache enabling OracleAS TopLink to execute the query repeatedly, without accessing the database. This is useful when you execute queries that run against static data.

In that it does not know how many objects it is looking. by default a read all query always goes to the database. However if the object already exists in the cache, time is saved by not having to build a new object from the row.

For more information, see "Caching Query Results" on page 6-67.

Holding Results in the Query You can configure a query to maintain an internal cache of the objects returned by the query. This internal cache is disabled by default.

For more information, see "Cache Results In Query Objects" on page 6-76.

Performance

OracleAS TopLink offers several query options to improve performance, including the following:

- *Binding and Parameterized SQL:* Enables you to create and store queries that are complete except for one or more search parameters. To enhance query performance, invoke the query, and *bind* parameters to the query. This can improve query performance.

  For more information about binding and parameterized SQL, see "Binding and Parameterized SQL" on page 5-17.

- *Batch and Join Reading:* To optimize database reads, OracleAS TopLink supports both batch and join reading. When you use these techniques, you dramatically decrease the number of times you access the database during a read operation, especially when your result set contains a large number of objects.

  For more information about batch and join reading, see "Query Object Performance Options" on page 6-70.

- *Partial Object Reading:* Partial object queries enable you to retrieve partially populated objects from the database rather than complete objects.

  For more information about partial object reading, see "Partial Object Reading" on page 6-47.

- *Java Streams:* Enable you to retrieve data from the database in cursored Java streams. A cursored stream allows you to view a collection in manageable increments rather than as a complete collection. This is useful when you have a large result set.

  For more information about Java streams, see "Java Streams" on page 6-60.

- *Scrollable Cursors:* Retrieves the result set from a query on a row-by-row basis. This is useful when you want to operate on the rows individually.

  For more information about scrollable cursors, see "Cursors and Streams" on page 6-81.

Unit of Work

Queries that write to the database are often executed within a Unit of Work. You can also execute read queries within a Unit of Work, although reading the database this way is not common. There are two key configuration options available when you query within the Unit of Work:

- *Registering Results:* When you execute a read query within a Unit of Work, the Unit of Work registers the objects in the result set and returns clones to the Unit of Work cache. If you do not need to modify any of the returned objects, consider executing your query through a regular session.

  For more information about read queries within the Unit of Work, see "Reading and Querying Objects with the Unit of Work" on page 7-9.

- *Conform Results to Unit of Work:* The OracleAS TopLink conforming feature enables you to query against your relative logical or transaction view of the database. By default queries are executed on the database. If you have uncommitted changes, this can pose a problem in a Unit of Work. Uncommitted changes not yet written to the database cannot influence which result set gets returned.

  For more information, see "Conforming Results (UnitOfWork)" on page 6-64.

Query Development Options

There are two ways to build OracleAS TopLink queries: you can use the OracleAS TopLink Mapping Workbench, or you can build them in code using the OracleAS TopLink API.

Building Queries with the OracleAS TopLink Mapping Workbench

The OracleAS TopLink Mapping Workbench **Query** tab supports OracleAS TopLink expressions, EJB QL queries and finders, and custom SQL queries and finders.

For more information, see "Specifying Named Queries and Finders" in the *Oracle Application Server TopLink Mapping Workbench User's Guide.*

Building Queries in Java

As with the OracleAS TopLink Mapping Workbench, the OracleAS TopLink query API supports OracleAS TopLink expressions, EJB QL queries and finders, and custom SQL queries and finders. However, if you require more options than are offered by these selection criteria types, you can create queries using the OracleAS TopLink query API to leverage OracleAS TopLink support for query by example and stored procedures.

For more information about the OracleAS TopLink query API, see the *Oracle Application Server TopLink API Reference.*

Using Predefined Queries

An effective way to implement queries is to build predefined queries that you store as part of the project descriptor file. OracleAS TopLink loads the queries into the application at runtime.

OracleAS TopLink supports the following predefined queries:

- *Named Queries* are defined in the session and called by name from the session. You can create named queries with the OracleAS TopLink Mapping Workbench or in Java code.
- *Redirect Queries* allow you to define the query implementation in code as a static method. When you invoke the query, the call redirects to the specified static method. The query can include any arbitrary parameters (or none at all), packaged into a vector and passed to the redirect method.

For more information, see "Predefined Queries" on page 6-48.

Using Named Queries

Named queries are complete, self-contained queries stored in the project descriptor file. Using named queries improves your application performance because it reduces the resources required to run a query.

Building Named Queries with the OracleAS TopLink Mapping Workbench You can create queries in the OracleAS TopLink Mapping Workbench using the OracleAS TopLink Mapping Workbench **Query** tab. The queries you build in the **Query** tab become part of the OracleAS TopLink project: OracleAS TopLink exports them automatically when you create deployment files from the project.

For more information, see "Specifying Named Queries and Finders" in the *Oracle Application Server TopLink Mapping Workbench User's Guide.*

Building Named Queries in Java The OracleAS TopLink query API enables you to build queries outside of the OracleAS TopLink Mapping Workbench. However, unlike queries built in the OracleAS TopLink Mapping Workbench, OracleAS TopLink does not include these queries automatically in your OracleAS TopLink application. Instead, add them to the application manually, using after load methods to amend the project descriptor.

For more information about after load methods, see "Customizing OracleAS TopLink Descriptors with Amendment Methods" on page 3-82.

Using Redirect Queries

Although most OracleAS TopLink queries search for objects directly, a redirect query generally invokes a method that exists on another class and waits for the results of the remote query. Redirect queries enable you to build and use complex operations, including operations that might not otherwise be possible within the query framework.

For more information, see "Redirect Queries" on page 6-52.

Building EJB Finders

An EJB finder is a query as defined by the EJB specification. It returns EJBs, collections, and enumerations. The difference between a finder and a query is that queries return Java objects, but finders return EJBs. The OracleAS TopLink query framework enables you to create and execute complex finders that retrieve entity beans.

Finders contain finder methods that define search criteria. The work involved in creating these methods depends on whether you are building container-managed persistence (CMP) bean finders or bean managed persistence (BMP) bean finders:

- CMP finders require the developer to define the finder API method signature on the bean `Home` interface. The CMP provider generates the actual code mechanisms for the finder from the API definition.
- BMP finders require the developer to provide the code required to execute the finder methods.

In either case, you define finders in the `Home` interface of the bean.

For more information, see "EJB Finders" on page 6-85.

Query Keys

A query key is an alias that OracleAS TopLink expressions use to relate to the descriptors and mappings for a given class. The query key is generally the name of an attribute of the class.

For example, consider a database table that includes a column called `F_NAME` that represents the attribute `firstName` in the class. Both represent the concept of an object's first name. OracleAS TopLink expressions use a query key to relate the two when you query on the database using the `firstName` as a selection criteria.

By default, OracleAS TopLink builds a query key in a descriptor for each attribute you map and automatically creates query keys for all mapped attributes of a class.

The default name of the query key is the same as the name of the mapping.You can add additional query keys for nonmapped or duplicate purpose fields, either in Java code or using the OracleAS TopLink Mapping Workbench.

For more information, see "Working with Query Keys" in the *Oracle Application Server TopLink Mapping Workbench User's Guide.*

Query Building Basics

OracleAS TopLink supports several options for creating queries, including:

- Expressions
- Custom SQL
- Stored Procedure Calls
- EJB QL
- Query by Example

Expressions

OracleAS TopLink expressions enable you to specify query search criteria based on the object model. OracleAS TopLink translates the resulting query into SQL and converts the results of the query into objects. OracleAS TopLink provides two public classes to support expression:

- The *Expression* class represents an expression, which can be anything from a simple constant to a complex clause with boolean logic. The developer can manipulate, group, and integrate expressions in several ways.
- The *ExpressionBuilder* class is the factory for constructing new expressions.

Accessing Methods in Expressions

The OracleAS TopLink expression framework provides methods through the following classes:

- The `Expression` class provides most general functions, such as `toUpperCase`.
- The `ExpressionMath` class supplies mathematical methods.

The following code examples illustrate the two classes. Example 6–1 uses the `Expression` class, while Example 6–2 uses the `ExpressionMath` class.

Example 6–1 Using the Expression Class

```
expressionBuilder.get("lastName").equal("Smith");
```

Example 6–2 Using the ExpressionMath Class

```
ExpressionMath.abs(ExpressionMath.subtract(emp.get("salary"),
emp.get("spouse").get("salary")).greaterThan(10000)
```

This division of functionality enables OracleAS TopLink expressions to provide similar mathematical functionality to the Java class, `java.lang.Math`, but keeps both the `Expression` and `ExpressionMath` classes from becoming unnecessarily complex.

Expression Components

A simple expression normally consists of three parts:

- The *attribute*, which represents a mapped attribute or query key of the persistent class
- The *operator*, which is an expression method that implements boolean logic, such as `GreaterThan`, `Equal`, or `Like`
- The *constant* or *comparison*, which refers to the value used to select the object

In the following code fragment:

```
expressionBuilder.get("lastName").equal("Smith");
```

- The attribute is `lastName`.
- The operator is `equal()`.
- The constant is the string “`Smith`”.

The `expressionBuilder` substitutes for the object or objects to be read from the database. In this example, `expressionBuilder` represents employees.

Expressions Compared to SQL Expressions offer the following advantages over SQL when you access a database:

- Expressions are easier to maintain because the database is abstracted.

- Changes to descriptors or database tables do not affect the querying structures in the application.
- Expressions enhance readability by standardizing the `Query` interface so that it looks similar to traditional Java calling conventions. For example, the Java code required to get the street name from the Address object of the Employee class looks like this:

  ```
  emp.getAddress().getStreet().equals("Meadowlands");
  ```

  The expression to get the same information is similar:

  ```
  emp.get("address").get("street").equal("Meadowlands");
  ```

- Expressions allow read queries to transparently query between two classes that share a relationship. If these classes are stored in multiple tables in the database, OracleAS TopLink automatically generates the appropriate join statements to return information from both tables.
- Expressions simplify complex operations. For example, the following Java code retrieves all Employees that live on "Meadowlands" whose salary is greater than 10,000:

  ```
  ExpressionBuilder emp = new ExpressionBuilder();
  Expression exp = emp.get("address").get("street").equal("Meadowlands");
  Vector employees = session.readAllObjects(Employee.class,
    exp.and(emp.get("salary").greaterThan(10000)));
  ```

  OracleAS TopLink automatically generates the appropriate SQL from that code:

  ```
  SELECT t0.VERSION, t0.ADDR_ID, t0.F_NAME, t0.EMP_ID, t0.L_NAME, t0.MANAGER_
  ID, t0.END_DATE, t0.START_DATE, t0.GENDER, t0.START_TIME, t0.END_
  TIME,t0.SALARY FROM EMPLOYEE t0, ADDRESS t1 WHERE (((t1.STREET =
  'Meadowlands')AND (t0.SALARY > 10000)) AND (t1.ADDRESS_ID = t0.ADDR_ID))
  ```

Boolean Logic Expressions use standard boolean operators, such as `AND`, `OR`, and `NOT` and you can combine multiple expressions to form more complex expressions. For example, the following code fragment queries for projects managed by a selected person, with a budget greater than or equal to 1,000,000.

```
ExpressionBuilder project = new ExpressionBuilder();
Expression hasRightLeader, bigBudget, complex;
Employee selectedEmp = someWindow.getSelectedEmployee();
hasRightLeader = project.get("teamLeader").equal(selectedEmp);
bigBudget = project.get("budget").greaterThanEqual(1000000);
```

```
complex = hasRightLeader.and(bigBudget);
Vector projects = session.readAllObjects(Project.class, complex);
```

Database Functions OracleAS TopLink supports the following database functions and operators:

- `like()`
- `notLike()`
- `toUpperCase()`
- `toLowerCase()`
- `toDate()`
- `rightPad()`

Database functions allow you to define more flexible queries. For example, the following code fragment matches several last names, including “SMART”, “Smith”, and “Smothers”:

```
emp.get("lastName").toUpperCase().like("SM%")
```

You access most functions through methods such as `toUpperCase` on the `Expression` class.

Mathematical Functions Mathematical functions are available through the `ExpressionMath` class. Mathematical function support in expressions is similar to the support provided by the Java class `java.lang.Math`.

For example:

```
ExpressionMath.abs(ExpressionMath.subtract(emp.get("salary"),emp.get("spouse")
  .get("salary")).greaterThan(10000)
```

Platform and User Defined Functions You can use expressions to implement database functions that OracleAS TopLink does not support directly. For simple functions, use the `getFunction()` operation, which the argument is the name of a function. For example, consider the following expression, which calls a function called `VacationCredit` on the database:

```
emp.get("lastName").getFunction("VacationCredit").greaterThan(42)
```

This expression produces the following SQL:

```
SELECT . . . WHERE VacationCredit(EMP.LASTNAME) > 42
```

You can also create more complex functions and add them to OracleAS TopLink. See "Platform and User-Defined Functions" on page 6-22.

Expressions for One-to-One and Aggregate Object Relationships Expressions can include an attribute that has a one-to-one relationship with another persistent class. A one-to-one relation translates naturally into a SQL join that returns a single row.

For example, the following code fragment accesses fields from an employee's address:

```
emp.get("address").get("country").like("S%")
```

This example corresponds to joining the EMPLOYEE table to the ADDRESS table, based on the `address` foreign key, and checking for the country name. You can nest these relationships infinitely, so it is possible to ask for complex information as follows:

```
project.get("teamLeader").get("manager").get("manager").get("address").get("street")
```

Expressions for Complex Relationships You can query against complex relationships, such as one-to-many, many-to-many, direct collection, and aggregate collection relationships. Expressions for these types of relationships are more complex to build, because the relationships do not map directly to joins that yield a single row per object.

To query across a one-to-many or many-to-many relationship, use the `anyOf` operation. As its name suggests, this operation supports queries that return all items on the "many" side of the relationship that satisfy the query criteria. For example, consider the following code fragment:

```
emp.anyOf("managedEmployees").get("salary").lessThan(10000);
```

This code returns employees who manage at least one employee (through a one-to-many relationship) with a salary below $10,000. You can query across a many-to-many relationship using a similar strategy:

```
emp.anyOf("projects").equal(someProject)
```

OracleAS TopLink translates these queries to SQL, and SQL joins the relevant tables using a `DISTINCT` clause to remove duplicates.

For example:

```
SELECT DISTINCT . . . FROM EMP t1, EMP t2 WHERE
t2.MANAGER_ID = t1.EMP_ID AND t2.SALARY < 10000
```

Creating Expressions with the Expression Builder

To create `Expression` objects, use the `get()` method or its related methods on an `Expression` or `ExpressionBuilder`. The `ExpressionBuilder` acts as a stand-in for the objects you query. To construct a query, send messages to the `ExpressionBuilder` that correspond to the attributes of the objects. We recommend that you name `ExpressionBuilder` objects according to the type of objects against which you do a query.

> **Note:** An instance of `ExpressionBuilder` is specific to a particular query. Do not attempt to build another query using an existing builder, because it still contains information related to the first query.

Example 6–3 A Simple Expression Builder Expression

This example uses the query key `lastName` to reference the field name `L_NAME`.

```
Expression expression = new ExpressionBuilder().get("lastName").equal("Young");
```

Example 6–4 An Expression Using the and() Method

```
ExpressionBuilder emp = new ExpressionBuilder();
    Expression exp1, exp2;
    exp1 = emp.get("firstName").equal("Ken");
    exp2 = emp.get("lastName").equal("Young");
    return exp1.and(exp2);
```

Example 6–5 An Expression Using the notLike() Method

```
Expression expression = new ExpressionBuilder().get("lastName").notLike("%ung");
```

Using Multiple Expressions

Expressions support subqueries (SQL subselects) and parallel selects. To create a subquery, use a single expression builder. With parallel selects, use multiple expression builders when you define a single query. This enables you to specify joins for unrelated objects at the object level.

Subselects and Subqueries Some queries compare the results of other, contained queries (or subqueries). SQL supports this comparison through subselects. OracleAS TopLink expressions provide subqueries to support subselects.

Subqueries enable you to define sophisticated expressions that query on aggregated values (`counts`, `min`, `max`) and unrelated objects (`exists`, `in`, `comparisons`). To obtain a subquery, pass an instance of a report query to any expression comparison operation, or use the `subQuery` operation on expression builder. The subquery is not required to have the same reference class as the parent query, and it must use its own expression builder.

You can nest subqueries, or use them in parallel. Subqueries can also make use of custom SQL.

For expression comparison operations that accept a single value (`equal`, `greaterThan`, `lessThan`), the subquery result must return a single value. For expression comparison operations that accept a set of values (`in`, `exists`), the subquery result must return a set of values.

Example 6–6 A Subquery Expression Using a Comparison and Count Operation

This example searches for all employees with more than 5 managed employees.

```
ExpressionBuilder emp = new ExpressionBuilder();
ExpressionBuilder managedEmp = new ExpressionBuilder();
ReportQuery subQuery =new ReportQuery(Employee.class, managedEmp);
subQuery.addCount();
subQuery.setSelectionCriteria(managedEmp.get("manager") .equal(emp));
Expression exp = emp.subQuery(subQuery).greaterThan(5);
```

Example 6–7 A Subquery Expression Using a Comparison and Max Operation

This example searches for the employee with the highest salary in the city of Ottawa.

```
ExpressionBuilder emp = new ExpressionBuilder();
ExpressionBuilder ottawaEmp = new ExpressionBuilder();
ReportQuery subQuery = new ReportQuery(Employee.class, ottawaEmp);
```

```
subQuery.addMax("salary");
subQuery.setSelectionCriteria(ottawaEmp.get("address").get("city").equal("Ottawa"));
Expression exp =
  emp.get("salary").equal(subQuery).and(emp.get("address").get("city").equal("Ottawa"));
```

Example 6–8 A Subquery Expression Using a Not Exists Operation

This example searches for all employees that have no projects.

```
ExpressionBuilder emp = new ExpressionBuilder();
ExpressionBuilder proj = new ExpressionBuilder();
ReportQuery subQuery = new ReportQuery(Project.class, proj);
subQuery.addAttribute("id");
subQuery.setSelectionCriteria(proj.equal(emp.anyOf("projects"));
Expression exp = emp.notExists(subQuery);
```

Parallel Expressions Parallel expressions enable you to compare unrelated objects. Parallel expressions require multiple expression builders, but do not require the use of report queries. Each expression must have its own expression builder, and you must use the constructor for expression builder that takes a `class` as an argument. The class does not have to be the same for the parallel expressions, and you can create multiple parallel expressions in a single query.

Only one of the expression builders is considered the primary expression builder for the query. This primary builder makes use of the zero argument expression constructor, and OracleAS TopLink obtains its class from the query.

Example 6–9 A Parallel Expression on Two Independent Employees

This example queries all employees with the same last name as another employee of different gender, and accounts for the possibility that returned results can be a spouse.

```
ExpressionBuilder emp = new ExpressionBuilder();
ExpressionBuilder spouse = new ExpressionBuilder(Employee.class);
Expression exp = emp.get("lastName").equal(spouse.get("lastName"))
  .and(emp.get("gender").notEqual(spouse.get("gender"));
```

Parameterized Expressions and Finders

A relationship mapping differs from a regular query because it retrieves data for many different objects. To enable you to specify these queries, supply arguments when you execute the query. Use the `getParameter()` and `getField()` methods to acquire values for the arguments.

A parameterized expression executes searches and comparisons based on variables instead of constants. This approach enables you to build expressions that retrieve context-sensitive information. This technique is useful when you:

- Customize mappings
- Create reusable queries
- Define EJB finders

Parameterized expressions require that the relationship mapping know how to retrieve an object or collection of objects based on its current context. For example, a one-to-one mapping from `Employee` to `Address` must query the database for an address based on foreign key information from the `Employee` table. Each mapping contains a query that OracleAS TopLink constructs automatically based on the information provided in the mapping. To specify expressions yourself, use the mapping customization mechanisms. For more information about the mapping customization mechanisms, see the *Oracle Application Server TopLink Mapping Workbench User's Guide.*

Expression getParameter() The `getParameter()` method returns an expression that becomes a parameter in the query. This method enables you to create a query that employs user input as the search criteria. The parameter must be either the fully qualified name of the field from a descriptor's row, or a generic name for the argument.

Parameters you construct this way are global to the current query, so you can send this message to any expression object.

Example 6–10 Using Expression getParameter() and getField()

```
ExpressionBuilder address = new ExpressionBuilder();
Expression exp = address.getField
  ("ADDRESS.EMP_ID").equal(address.getParameter("EMPLOYEE.EMP_ID"));
exp = exp.and(address.getField("ADDRESS.TYPE").equal(null));
```

Expression getField() The `getField()` method returns an expression that represents a database field with the given name. Use the `Expression getField()` method to construct the selection criteria for a mapping. The argument is the fully qualified name of the required field. Because fields are not global to the current query, you must send this method to an expression that represents the table from which this field is derived. See also "Data Queries" on page 6-23.

Example 6–11 The Use of a Parameterized Expression in a Mapping

This example obtains a simple one-to-many mapping from class `PolicyHolder` to `Policy` using a nondefault selection criteria. The SSN field of the POLICY table is a foreign key to the SSN field of the HOLDER table.

```
OneToManyMapping mapping = new OneToManyMapping();
mapping.setAttributeName("policies");
mapping.setGetMethodName("getPolicies");
mapping.setSetMethodName("setPolicies");
mapping.setReferenceClass(Policy.class);

// Build a custom expression here rather than using the defaults
ExpressionBuilder policy = new ExpressionBuilder();
mapping.setSelectionCriteria(policy.getField("POLICY.SSN")).equal(policy.
  getParameter("HOLDER.SSN")));
```

Example 6–12 A Parameterized Expression in a Custom Query

This example uses an employee's first name to demonstrate how to use a custom query to find the employee.

```
ExpressionBuilder emp = new ExpressionBuilder();
Expression firstNameExpression;
firstNameExpression = emp.get("firstName").equal(emp.getParameter("firstName"));
ReadObjectQuery query = new ReadObjectQuery();
query.setReferenceClass(Employee.class);
query.setSelectionCriteria(firstNameExpression);
query.addArgument("firstName");
Vector v = new Vector();
v.addElement("Sarah");
Employee e = (Employee) session.executeQuery(query, v);
```

Example 6–13 Nested Parameterized Expressions

This example demonstrates how to use a custom query to find all employees that live in the same city as a given employee.

```
ExpressionBuilder emp = new ExpressionBuilder();
Expression addressExpression;
addressExpression =
```

```
emp.get("address").get("city").equal(emp.getParameter("employee").get("address")
.get("city"));
ReadObjectQuery query = new ReadObjectQuery(Employee.class);
query.setName("findByCity");
query.setReferenceClass(Employee.class);
query.setSelectionCriteria(addressExpression);
query.addArgument("employee");
Vector v = new Vector();
v.addElement(employee);
Employee e = (Employee) session.executeQuery(query, v);
```

Platform and User-Defined Functions

Different databases sometimes implement the same functions in different ways. For example, an argument that specifies that data returns in ascending order might be `ASC` or `ASCENDING`. To manage differences, OracleAS TopLink recognizes functions and other operators that vary according to the relational database.

Although most platform-specific operators exist in OracleAS TopLink, use the `ExpressionOperator` class to add your own.

An `ExpressionOperator` has a selector and a vector of strings:

- The selector is the identifier (*id*) by which users refer to the function.
- The strings are the constant strings used in printing the function. When printed, the strings alternate with the function arguments.

You can also specify whether the operator is prefix or postfix. In a prefix operator, the first constant string prints before the first argument; in a postfix, it prints afterwards.

Example 6–14 Creating a New Expression Operator—The toUpperCase Operator

```
ExpressionOperator toUpper = new ExpressionOperator();
toUpper.setSelector();
Vector v = new Vector();
v.addElement("UPPER(");
v.addElement(")");
toUpper.printAs(v);
toUpper.bePrefix();
toUpper.setNodeClass(FunctionExpression.class);

// To add this operator for all database
ExpressionOperator.addOperator(toUpper);
```

```
// To add to a specific platform
DatabasePlatform platform = session.getLogin().getPlatform();
platform.addOperator(toUpper);
```

Example 6–15 Accessing a User-Defined Function

This example illustrates the `getFunction()` method, called with a vector of arguments.

```
ReadObjectQuery query = new ReadObjectQuery(Employee.class);
expression functionExpression = new
  ExpressionBuilder().get("firstName").getFunction(ExpressionOperator.toUpper).
  equal("BOB");
query.setSelectionCriteria(functionExpression);
session.executeQuery(query);
```

Data Queries

You can use expressions to retrieve data rather than objects. This is a common approach when you work with unmapped information in the database, such as foreign keys and version fields.

Expressions that query for objects generally refer to object attributes, which may in turn refer to other objects. Data expressions refer to tables and their fields. You can combine data expressions and object expressions within a single query. OracleAS TopLink provides two main operators for expressions that query for data: `getField()`, and `getTable()`.

getField() The `getField()` operator enables you to retrieve data from either an unmapped table or an unmapped field from an object. In either case, the field must be part of a table represented by that object's class; otherwise, OracleAS TopLink raises an exception when you execute the query.

You can also use the `getField()` operator to retrieve the foreign key information for an object.

Example 6–16 Using getField Against an Object

```
builder.getField("[FIELD_NAME]").greaterThan("[ARGUMENT]");
```

getTable() The `getTable()` operator returns an expression that represents an unmapped table in the database. This expression provides a context from which to retrieve an unmapped field when you use the `getField()` operator.

Example 6–17 Using getTable() and getField() Together

```
builder.getTable("[TABLE_NAME]").getField("[FIELD_NAME]").equal("[ARGUMENT]");
```

A common use for the `getTable()` and `getField()` operators is to retrieve information from a link table (or reference table) that supports a many-to-many relationship. Example 6–18 reads a many-to-many relationship that uses a link table and also checks an additional field in the link table. This code combines an object query with a data query, using the employee's manager as the basis for the data query. It also features parameterization for the project ID.

Example 6–18 Using a Data Query Against a Link Table

```
ExpressionBuilder emp = new ExpressionBuilder();
Expression manager = emp.get("manager");
Expression linkTable = manager.getTable("PROJ_EMP");
Expression empToLink = emp.getField("EMPLOYEE
  .EMP_ID").equal(linkTable.getField("PROJ_EMP.EMP_ID");
Expression projToLink = linkTable.getField("PROJ_EMP
  .PROJ_ID").equal(emp.getParameter("PROJECT.PROJ_ID"));
Expression extra = linkTable.getField("PROJ_EMP.TYPE").equal("W");
query.setSelectionCriteria((empToLink.and(projToLink)).and(extra));
```

Query Keys

A query key is an alias for a field name. Instead of referring to a field using a DBMS-specific field name such as `F_NAME`, query keys allow OracleAS TopLink expressions to refer to the field using class attribute names such as `firstName`. This offers the following advantages:

- Query keys enhance code readability when you define OracleAS TopLink expressions.
- Query keys increase portability by making code independent of the database schema. If you rename a field, you can redefine the query key without changing any code that references it.
- Unlike interface descriptors that only define common query keys shared by their implementors, aliased fields can have different names in each of the implementor tables.

For more information about query keys with the OracleAS TopLink Mapping Workbench, see "Working with Query Keys," in the *Oracle Application Server TopLink Mapping Workbench User's Guide.*

Automatically-Generated Query Keys OracleAS TopLink defines direct query keys for all direct mappings and has a special query key type for each mapping. You can use query keys to access fields that do not have direct mappings associated with them, such as the version field used for optimistic locking or the type field used for inheritance.

Example 6–19 Automatically-Generated Query Key in the OracleAS TopLink Expression Framework

```
Vector employees = session.readAllObjects(Employee.class,
  new ExpressionBuilder().get("firstName").equal("Bob"));
```

Relationship Query Keys OracleAS TopLink supports and defines query keys for relationship mappings. You can use query keys to join across a relationship. One-to-one query keys define a joining relationship. To access query keys for relationship mappings, use the `get()` method in expressions.

Example 6–20 One-to-One Query Key

The following code example illustrates how to use a one-to-one query key within the OracleAS TopLink expression framework.

```
ExpressionBuilder employee = new ExpressionBuilder();
Vector employees = session.readAllObjects(Employee.class,
  employee.get("address").get("city").equal("Ottawa"));
```

To access one-to-many and many-to-many query keys that define a distinct join across a collection relationship, use the `anyOf()` method in expressions.

If no mapping exists for the relationship, you can also define relationship query keys manually. Relationship query keys are not supported directly by the OracleAS TopLink Mapping Workbench. To define a relationship query key, specify and write an amendment method, and use the `addQueryKey()` message to register the query keys.

Example 6–21 Defining One-to-One Query Key Example

The following code defines a one-to-one query key.

```
/* Static amendment method in Address class, addresses do not know their owners
```

```
in the object-model, however you can still query on their owner if a
user-defined query key is defined */
public static void addToDescriptor(Descriptor descriptor)
{
OneToOneQueryKey ownerQueryKey = new OneToOneQueryKey();
ownerQueryKey.setName("owner");
ownerQueryKey.setReferenceClass(Employee.class);
ExpressionBuilder builder = new ExpressionBuilder();
ownerQueryKey.setJoinCriteria(builder.getField("EMPLOYEE.ADDRESS_
ID").equal(builder.getParameter("ADDRESS.ADDRESS_ID")));
descriptor.addQueryKey(ownerQueryKey);
}
```

Reference

Table 6–1 and Table 6–2 summarize the most common public methods for `ExpressionBuilder` and `Expression`. For more information about the available methods for `ExpressionBuilder` and `Expression`, see the *Oracle Application Server TopLink API Reference*.

Table 6–1 Elements for Expression Builder

Element	**Method Name**
Constructors	`ExpressionBuilder()` `ExpressionBuilder(Class aClass)`

Expression creation methods	`get(String queryKeyName)`
	`getAllowingNull(String queryKeyName)`
	`anyOf(String queryKeyName)`
	`anyOfAllowingNone(String queryKeyName)`
	`getField(String fieldName)`
	`in(ReportQuery subQuery)`

Table 6–2 Elements for Expression

Element	**Method Name**
Constructors	Never use the `Expression` constructors. Always use an `ExpressionBuilder` to create a new expression.

Table 6–2 Elements for Expression (Cont.)

Element	Method Name

Expression operators	`equal(Object object)`
	`notEqual(Object object)`
	`greaterThan(Object object)`
	`lessThan(Object object)`
	`isNull()`
	`notNull()`

Logical operators	`and(Expression theExpression)`
	`not()`
	`or(Expression theExpression)`

Key word searching	`equalsIgnoreCase(String theValue)` `likeIgnoreCase(String theValue)`
Aggregate functions (for use with report query)	`minimum()` `maximum()`

Relationship operators	`anyOf(String queryKeyName)`
	`anyOfAllowingNone(String queryKeyName)`
	`get(String queryKeyName)`
	`getAllowingNull(String queryKeyName)`
	`getField(String fieldName)`

Custom SQL

The expression framework enables you to define complex queries at the object level. If your application requires a more complex query, use SQL or stored procedure calls to create custom database operations.

For more information about stored procedure calls, see "Stored Procedure Calls" on page 6-29.

SQL Queries

You can provide a SQL string to any query instead of an expression, but the SQL string must return all data required to build an instance of the queried class. The SQL string can be a complex SQL query or a stored procedure call.

You can invoke SQL queries through the session read methods or through a read query instance.

Example 6–22 A Session Read Object Call Query With Custom SQL

```
Employee employee = (Employee) session.readObjectCall(Employee.class), new
SQLCall("SELECT * FROM EMPLOYEE WHERE EMP_ID = 44");
```

Example 6–23 A Session Method with Custom SQL

This example queries user and time information.

```
Vector rows = session.executeSelectingCall(new SQLCall("SELECT USER, SYSDATE FROM DUAL"));
```

SQL Data Queries

OracleAS TopLink offers the following data-level queries to read or modify data (but not objects) in the database:

- `DataReadQuery`: for reading rows of data
- `DirectReadQuery`: for reading a single column of data
- `ValueReadQuery`: for reading a single value of data
- `DataModifyQuery`: for modifying data

Example 6–24 A Direct Read Query with SQL

This example uses SQL to read all employee IDs.

```
DirectReadQuery query = new DirectReadQuery();
query.setSQLString("SELECT EMP_ID FROM EMPLOYEE");
Vector ids = (Vector) session.executeQuery(query);
```

Example 6–25 A Data Modify Query with SQL

This example uses SQL to switch the database.

```
DataModifyQuery query = new DataModifyQuery();
query.setSQLString("USE SALESDATABASE");
session.executeQuery(query);
```

Stored Procedure Calls

You can provide a `StoredProcedureCall` object to any query instead of an expression or SQL string, but the procedure must return all data required to build an instance of the class you query.

Example 6–26 A Read All Query With a Stored Procedure

```
ReadAllQuery readAllQuery = new ReadAllQuery();
call = new StoredProcedureCall();
call.setProcedureName("Read_All_Employees");
call.useNamedCursorOutputAsResultSet("RESULT_CURSOR");
readAllQuery.setCall(call);
Vector employees = (Vector) session.executeQuery(readAllQuery);
```

Output Parameters

The `StoredProcedureCall` object allows you to use output parameters. Output parameters enable the stored procedure to return additional information. You can use output parameters to define a `readObjectQuery` if they return the fields required to build the object.

Note: Not all databases support the use of output parameters to return data. However, because these databases generally support returning result sets from stored procedures, they do not require output parameters.

Example 6–27 Stored Procedure Call with an Output Parameter

```
StoredProcedureCall call = new StoredProcedureCall();
call.setProcedureName("CHECK_VALID_POSTAL_CODE");
call.addNamedArgument("POSTAL_CODE");
call.addNamedOutputArgument("IS_VALID");
ValueReadQuery query = new ValueReadQuery();
query.setCall(call);
query.addArgument("POSTAL_CODE");
Vector parameters = new Vector();
parameters.addElement("L5J1H5");
Number isValid = (Number) session.executeQuery(query,parameters);
```

Cursor Output Parameters

Oracle databases use output parameters rather than result sets to return data from stored procedures. Cursored output parameters enable you to retrieve the result set in a cursored stream rather than as a single result set. When you use the Oracle JDBC drivers, configure a `StoredProcedureCall` object to pass a cursor to OracleAS TopLink as a normal result set.

Example 6–28 Stored Procedure with a Cursored Output Parameter

```
StoredProcedureCall call = new StoredProcedureCall();
call.setProcedureName("READ_ALL_EMPLOYEES");
call.useNamedCursorOutputAsResultSet("RESULT_CURSOR");
ReadAllQuery query = new ReadAllQuery();
query.setReferenceClass(Employee.class);
query.setCall(call);
Vector employees = (Vector) Session.executequery(Query);
```

For more information about cursored streams, see "Java Streams" on page 6-60.

Output Parameter Event

OracleAS TopLink manages output parameter events for databases that support them. For example, if a stored procedure returns an error code that indicates that the application wants to check for an error condition, OracleAS TopLink raises the session event `OutputParametersDetected` to allow the application to process the output parameters.

Example 6–29 Stored Procedure with Reset Set and Output Parameter Error Code

```
StoredProcedureCall call = new StoredProcedureCall();
call.setProcedureName("READ_EMPLOYEE");
call.addNamedArgument("EMP_ID");
call.addNamedOutputArgument("ERROR_CODE");
ReadObjectQuery query = new ReadObjectQuery();
query.setCall(call);
query.addArgument("EMP_ID");
ErrorCodeListener listener = new ErrorCodeListener();
session.getEventManager().addListener(listener);
Vector args = new Vector();
args.addElement(new Integer(44));
Employee employee = (Employee) session.executeQuery(query, args);
```

Reference

Table 6–3 summarizes the most common public methods for the `StoredProcedureCall`. For more information about the available methods for the `StoredProcedureCall`, see the *Oracle Application Server TopLink API Reference*.

Table 6–3 Elements for Stored Procedure Call

Element	Method Name
Selection specification	`setProcedureName(String name)`

Input parameters	`addNamedArgument(String name)`
	`addNamedArgument(String dbName, String javaName)`
	`addNamedArgumentValue(String dbName, Object value)`
	`addUnnamedArgument(String javaName)`
	`addUnnamedArgumentValue(Object value)`

Input/Output parameters	`addNamedInOutputArgument(String name)`
	`addNamedInOutputArgument(String dbName, String javaName, String javaName, Class type)`
	`addNamedInOutputArgumentValue(String dbName, Object value, String javaName, Class type)`
	`public void addUnnamedInOutputArgument(String inArgumentFieldName, String outArgumentFieldName, Class type)`
	`public void addUnnamedInOutputArgumentValue(Object inArgumentValue, String outArgumentFieldName, Class type)`

Output parameters	`addNamedOutputArgument(String name)`
	`addNamedOutputArgument(String dbName, String javaName)`
	`addNamedOutputArgument(String dbName, String javaName, Class javaType)`
	`addUnnamedOutputArgument(String javaName)`
	`public void addunnamedOutputArgument(String argumentFieldName, Class type)`

Cursor output parameters	`useNamedCursorOutputAsResultSet(String argumentName)` `useUnnamedCursorOutputAsResultSet()`

EJB QL

EJB QL is a query language that is similar to SQL, but differs because it presents queries from an object model perspective and includes path expressions that enable navigation over the relationships defined for entity beans and dependent objects. Although EJB QL is usually associated with Enterprise JavaBeans (EJBs), OracleAS

TopLink enables you to use EJB QL with regular Java objects as well. In OracleAS TopLink, EJB QL enables users to declare queries, using the attributes of each abstract entity bean in the object model. This offers the following advantages:

- You do not need to know the database structure (tables, fields).
- You can use relationships in a query to provide navigation from attribute to attribute.
- You can construct queries using the attributes of the entity beans instead of using database tables and fields.
- EJB QL queries are portable because they are database-independent.
- You can use `SELECT` to specify the query reference class (the class or entity bean you are querying against).

Using EJB QL with OracleAS TopLink

OracleAS TopLink support for EJB QL enables you to:

- Add EJB QL queries to descriptors in the OracleAS TopLink Mapping Workbench.
- Build and use EJB QL dynamically at runtime, using a `ReadQuery` or the OracleAS TopLink session.

For more information about EJB QL queries with the OracleAS TopLink Mapping Workbench, see the *Oracle Application Server TopLink Mapping Workbench User's Guide.*

ReadAllQuery

The basic API for a `ReadAll` query with EJB QL is as follows:

```
setEJBQLString("...")
```

Provide either a `SELECT` clause or a reference class, and execute the query normally.

Example 6–30 A Simple ReadAllQuery Using EJB QL

```
ReadAllQuery theQuery = new ReadAllQuery();
theQuery.setReferenceClass(EmployeeBean.class);
theQuery.setEJBQLString("SELECT OBJECT(emp) FROM EmployeeBean emp");
…
Vector returnedObjects = (Vector)aSession.executeQuery(theQuery);
```

Example 6–31 A Simple ReadAllQuery Using EJB QL and Passing Arguments

This example defines the query similarly to Example 6–30, but creates, fills, and passes a vector of arguments to the `executeQuery` method.

```
// First define the query
ReadAllQuery theQuery = new ReadAllQuery();
theQuery.setReferenceClass(EmployeeBean.class);
theQuery.setEJBQLString("SELECT OBJECT(emp) FROM EmployeeBean emp WHERE
emp.firstName = ?1");
...
// Next define the Arguments
Vector theArguments = new Vector();
theArguments.add("Bob");
...
// Finally execute the query passing in the arguments
Vector returnedObjects = (Vector)aSession.executeQuery(theQuery, theArguments);
```

Session

You can execute EJB QL directly against the session. This returns a vector of the objects specified by the reference class. The basic API is:

```
aSession.readAllObjects(<ReferenceClass>, <EJBQLCall>)
/* <ReferenceClass> is the return class type and <EJBQLCall> is the EJBQL string
to be executed */
// Call ReadAllObjects on a session.
Vector theObjects = (Vector)aSession.readAllObjects(EmployeeBean.class, new
EJBQLCall( "SELECT OBJECT (emp) from EmployeeBean emp");
```

EJB QL Limitations

OracleAS TopLink supports all the EJB QL specification with the following exceptions:

- arithmetic functions
- `LOCATE`
- `ESCAPE`
- `IS [NOT] EMPTY`
- `[NOT] MEMBER [OF]`

Query by Example

Query by example enables you to specify queries when you provide sample instances of the persistent objects to be queried.

To define a query by example, provide a `ReadObjectQuery` or a `ReadAllQuery` with a sample persistent object instance and an optional query by example policy. The sample instance contains the data to query, and the query by example policy contains optional configuration settings, such as the operators to use and the attributes to consider or ignore.

Note: Query by example is not available for EJB 2.0 beans.

Defining a Sample Instance

Query by example enables you to query on any attribute that uses a direct mapping or a one-to-one relationship (including those with nesting). It does not support other relationship mapping types.

By default, OracleAS TopLink ignores attributes in the sample instance that contain zero (0), empty strings, and `FALSE`. To modify the list of values, see "Defining a Query by Example Policy" on page 6-35. You can use any valid constructor to create a sample instance or example object. Set only the attributes on which you base the query; set all other attributes to null.

Query by example uses the `AND` operator to tie the attribute comparisons together.

Example 6–32 Using Query by Example

This example queries the employee Bob Smith.

```
ReadObjectQuery query = new ReadObjectQuery();
Employee employee = new Employee();
employee.setFirstName("Bob");
employee.setLastName("Smith");
query.setExampleObject(employee);

Employee result = (Employee) session.executeQuery(query);
```

Example 6–33 Using Query by Example

This example queries across the employee’s address.

```
ReadAllQuery query = new ReadAllQuery();
```

```
Employee employee = new Employee();
Address address = new Address();
address.setCity("Ottawa");
employee.setAddress(address);
query.setExampleObject(employee);

Vector results = (Vector) session.executeQuery(query);
```

Defining a Query by Example Policy

OracleAS TopLink support for query by example includes a query by example policy. You can edit the policy to modify query by example default behavior. You can modify the policy to:

- Use `LIKE` or other operations to compare attributes. By default, query by example allows only `EQUALS`.
- Modify the set of values query by example ignores (the IGNORE set). The default ignored values are zero (0), empty strings, and `FALSE`.
- Force query by example to consider attribute values, even if the value is in the IGNORE set.
- Use `isNull` or `notNull` for attribute values.

To specify a query by example policy, include an instance of `QueryByExamplePolicy` with the query.

Example 6–34 Query by Example Policy Using Like

This example uses `like` for Strings and includes only objects whose salary is greater than zero.

```
ReadAllQuery query = new ReadAllQuery();
Employee employee = new Employee();
employee.setFirstName("B%");
employee.setLastName("S%");
employee.setSalary(0);
query.setExampleObject(employee);
/* Query by example policy section adds like and greaterThan */
QueryByExamplePolicy policy = new QueryByExamplePolicy();
policy.addSpecialOperation(String.class, "like");
policy.addSpecialOperation(Integer.class, "greaterThan");
policy.alwaysIncludeAttribute(Employee.class, "salary");
query.setQueryByExamplePolicy(policy);
Vector results = (Vector) session.executeQuery(query);
```

Example 6–35 Query by Example Policy Using Key Words

This example uses key words for Strings and ignores -1.

```
ReadAllQuery query = new ReadAllQuery();
Employee employee = new Employee();
employee.setFirstName("bob joe fred");
employee.setLastName("smith mc mac");
employee.setSalary(-1);
query.setExampleObject(employee);
/* Query by example policy section */
QueryByExamplePolicy policy = new QueryByExamplePolicy();
policy.addSpecialOperation(String.class, "containsAnyKeyWords");
policy.excludeValue(-1);
query.setQueryByExamplePolicy(policy);
Vector results = (Vector) session.executeQuery(query);
```

Combining Query by Example with Expressions

To create more complex query by example queries, combine query by example with OracleAS TopLink expressions.

Example 6–36 Combining Query by Example with Expressions

```
ReadAllQuery query = new ReadAllQuery();
Employee employee = new Employee();
employee.setFirstName("Bob");
employee.setLastName("Smith");
query.setExampleObject(employee);
/* This section specifies the expression */
ExpressionBuilder builder = new ExpressionBuilder();
query.setSelectionCriteria(builder.get("salary").between(100000,200000);
Vector results = (Vector) session.executeQuery(query);
```

Reference

Table 6–4 summarizes the most common public methods for `QueryByExample`. For more information about the available methods, see the *Oracle Application Server TopLink API Reference*.

Table 6–4 Elements for Query By Example Policy

Element	**Method Name**
Special operations	`addSpecialOperation(Class theClass, String operation)`
Forced inclusion	`alwaysIncludeAttribute(java.lang.Class exampleClass, java.lang.String attributeName)`
	`includeAllValues()`
Attribute exclusion	`excludeValue(Object value)` `excludeDefaultPrimitiveValues()`
Null equality	`setShouldUseEqualityForNulls(boolean flag)`

Executing Queries

OracleAS TopLink provides several options to execute queries, including:

- Session Queries
- Query Objects
- Predefined Queries
- Queries Defined with the OracleAS TopLink Mapping Workbench
- Query Managers

Session Queries

The `Session` class and its subclasses (including `DatabaseSession` and `UnitOfWork`) provide methods to read, create, modify, and delete objects stored in a database. These methods, known as query methods, enable you to create queries against the object model. Session queries are easy to use and are flexible enough to perform most database operations.

The `DatabaseSession` class provides direct support to read and modify the database by offering read, write, insert, update, and delete operations.

The `UnitOfWork` class also provides methods to modify data. The Unit of Work is a safer approach to data modification than the `DatabaseSession` methods, because it isolates changes until they are complete. Whenever possible, use the Unit of Work to write or update rather than the write, insert, update, and delete methods available in the database session.

For more information, see "Unit of Work Basics" on page 7-12.

Reading Objects from the Database

The session provides the following methods to access the database:

- The `readObject()` method reads a single object from the database. Use this method with a primary key when looking for a specific object.
- The `readAllObjects()` method reads multiple objects from the database. Use this method to return a group of objects that match the selection criteria.
- The `refreshObject()` method refreshes objects in the cache with data from the database.

Read Operation The `readObject()` method retrieves a single object from the database. The application must specify the class of object to read. If no object matches the criteria is found, null is returned.

For example, the basic read operation is:

```
session.readObject(MyDomainObject.class);
```

This example returns the first instance of `MyDomainObject` found in the table used for `MyDomainObject`. OracleAS TopLink provides the `Expression` class to specify querying parameters for a specific object.

When you search for a single, specific object using a primary key, the `readObject()` method is more efficient than the `readAllObjects()` method because `readObject()` can find an instance in the cache without accessing database. Because a `readAllObjects()` operation does not know how many objects match the criteria, it always searches the database to find matching objects, even if it finds matching objects in the cache.

Example 6–37 readObject() Using an Expression

```
import oracle.toplink.sessions.*;
import oracle.toplink.expressions.*;

/* Use an expression to read in the Employee whose last name is Smith. Create an
expression using the Expression Builder and use it as the selection criterion of
the search */
Employee employee = (Employee) session.readObject(Employee.class, new
ExpressionBuilder().get("lastName").equal("Smith"));
```

Read All Operation The `readAllObjects()` method retrieves a `Vector` of objects from the database and does not order the returned objects. If the query does not find any matching objects, it returns an empty `Vector`.

Specify the class for the query. You can also include an expression to define more complex search criteria, as illustrated in Example 6–38.

Example 6–38 readAllObjects() Using an Expression

```
// Returns a Vector of employees whose employee salary > 10000
Vector employees = session.readAllObjects(Employee.class,new
ExpressionBuilder.get("salary").greaterThan(10000));
```

Refresh Operation The `refreshObject()` method causes OracleAS TopLink to update the object in memory with data from the database. This operation refreshes any privately owned objects as well.

Note: A privately owned object is one that cannot exist without its parent, or source object.

Writing Objects to the Database

The Unit of Work provides the safest mechanism for writing objects in most OracleAS TopLink applications. However, when you can safely write directly to the database (for example: in a single-user or a two-tier application), session methods are the most efficient database writing tool. Database session provides the following methods to write to a database:

- `writeObject()`
- `writeAllObjects()`
- `insertObject()`
- `updateObject()`
- `deleteObject()`

Writing a Single Object to the Database When you invoke the `writeObject()` method, the method performs a *does-exist* check to determine whether an object exists. If the object exists, `writeObject()` updates the object; if it does not exist, `writeObject()` inserts a new object.

The `writeObject()` method writes privately owned objects in the correct order to maintain referential integrity.

Call the `writeObject()` method when you cannot verify that an object exists on the database.

Example 6–39 Writing a Single Object Using writeObject()

```
//Create an instance of employee and write it to the database
Employee susan = new Employee();
susan.setName("Susan");
...
//Initialize the susan object with all other instance variables
session.writeObject(susan);
```

Writing All Objects to the Database You can call the `writeAllObjects()` method to write multiple objects to the database. The `writeAllObjects()` method performs the same *does-exist* check as the `writeObject()` method and then performs the appropriate insert or update operations.

Example 6–40 Writing Several Objects Using writeAllObjects()

```
// Read a Vector of all the current employees in the database.
Vector employees = (Vector) session.readAllObjects(Employee.class);
...//Modify any employee data as necessary
//Create a new employee and add it to the list of employees
Employee susan = new Employee();
...
//Initialize the new instance of employee
employees.add(susan);
/* Write all employees to the database. The new instance of susan which is not
currently in the database will be inserted. All the other employees which are
currently stored in the database will be updated */
session.writeAllObjects(employees);
```

Adding New Objects to the Database The `insertObject()` method creates a new object on the database, but does not perform the *does-exist* check before it attempts the insert operation. The `insertObject()` method is more efficient than the `writeObject()` method if you are certain that the object does not yet exist on the database. If the object does exist, the database throws an exception when you execute the `insertObject()` call.

Modifying Existing Objects in the Database The `updateObject()` method updates existing objects in the database, but does not perform the *does-exist* check before it attempts the update operation. The `updateObject()` is more efficient than the `writeObject()`method if you are certain that the object does exist in the database. If the object does not exist, the database throws an exception when you execute the `updateObject()` call.

Deleting Objects in the Database To delete an OracleAS TopLink object from the database, read the object from the database and then call the `deleteObject()` method. This method deletes both the specified object and any privately owned data.

Query Objects

Query objects are the standard devices OracleAS TopLink uses to interact with the database. They support database commands such as create, read, update, and delete, and accept search criteria specified in several ways, including OracleAS TopLink expressions.

OracleAS TopLink provides you with direct access to query objects, which support more complex queries than the session query API. You can build custom query objects to improve application performance or to support complex queries. Use the custom query object classes you create with the session or a descriptor's query manager to:

- Create new query operations.
- Create named queries registered with the session.
- Customize the session's default database operations, such as `readObject()` and `writeObject()`.

The OracleAS TopLink Mapping Workbench provides graphical tools to create query objects. Although this section discusses query objects in the context of Java code, we recommend that you create query objects in the OracleAS TopLink Mapping Workbench.

Query Object Components

OracleAS TopLink uses query objects to store information about a database query. A complete query object stores information about:

- The query type, specified by the query object class
- The class that the query accesses (the reference class)

- The query execution, which can be through SQL, a database call or an OracleAS TopLink expression

Creating a Query Object

The following steps illustrate how to create a query object in Java code.

Step 1 Specify the query type to initialize the query object.

To execute a query, select one of the following query object classes:

- `ReadAllQuery`: reads a collection of objects
- `ReadObjectQuery`: reads a single object
- `ReportQuery`: reads information about objects
- `DeleteObjectQuery`: removes an object from the database
- `InsertObjectQuery`: inserts new objects into the database
- `UpdateObjectQuery`: updates existing objects
- `WriteObjectQuery`: writes an object to the database, either with an insert (for new objects) or an update (for existing objects)

To execute SQL expressions, use the following query object classes:

- `ValueReadQuery`: returns a single data value
- `DirectReadQuery`: returns a collection of column values; can be used for direct collection queries
- `DataReadQuery`: executes a SQL SELECT, returns a collection of database row (map) objects
- `DataModifyQuery`: executes a non-selecting SQL string

Step 2 Set the reference class.

The reference class specifies the class against which the query runs. Use the `setReferenceClass()` call to select a searchable class.

Step 3 For read queries, configure the query for execution.

To specify how a query executes, call one of the following the methods:

- `setSelectionCriteria()`: passes an expression to the query object
- `setSQLString()`: passes a SQL string

- `setCall()`: passes a database call

This setting is optional. If you do not specify read criteria, a `ReadAllQuery` returns every object of the reference class in the database, and a `ReadObjectQuery` returns the first object it encounters.

Step 4 Add query arguments.

You can pass arguments to the query object by calling `addArgument()` in addition to the `executeQuery()` method. Arguments describe the objects for the query to return.

Step 5 Register the query object with the session.

After initialization, use the `addQuery()` method to register the query object with the session. Name the query when you register it. The session then manages the query for you. This enables you to call the query by name.

Registering the query object with the session is optional. If you do not register the query object, specify the entire query every time you execute it, or manage it manually outside of the session.

Step 6 Execute the query.

To execute the query, use the `executeQuery()` call to call the object by name. As required, provide values for any defined arguments.

Read Query Object Examples

Although query objects support writing to a database, reading is their most common use. This section provides several examples that illustrate the use of query objects for reading the database.

Example 6–41 illustrates a simple read query. It uses an OracleAS TopLink expression, but does not use its own arguments for the query. Instead, it relies on the search parameters the expression provides. This example builds the expression within its code, but does not register the query with the session.

Example 6–41 A Simple ReadAllQuery

```
// This example returns a Vector of employees whose employee ID is > 100.

// Initialize the query object by specifying the query type
ReadAllQuery query = new ReadAllQuery();

//Set the reference class for the query.
```

```
query.setReferenceClass(Employee.class);

/* Configure the query execution. Because this example uses an expression, it
uses the setSelectionCriteria call */
query.setSelectionCriteria(new ExpressionBuilder.get("id").greaterThan(100));

// Execute the query
Vector employees = (Vector) session.executeQuery(query);
```

Example 6–42 illustrates a complex `readObject` query that uses all available configuration options.

Example 6–42 A Named Read Query with Two Arguments

```
// Define two expressions that map to the first and last name of the employee.
ExpressionBuilder emp = new ExpressionBuilder();
Expression firstNameExpression =
emp.get("firstName").equal(emp.getParameter("firstName"));
Expression lastNameExpression = emp.get("lastName").equal(emp.getParameter("lastName"));

//Initialize the query object by specifying the query type
ReadObjectQuery query = new ReadObjectQuery();
//Set the reference class for the query.
query.setReferenceClass(Employee.class);
/* Configure the query execution. Because this example uses an expression, it uses the
setSelectionCriteria call */
query.setSelectionCriteria(firstNameExpression.and(lastNameExpression));
//Specify the required arguments for the query.
query.addArgument("firstName");
query.addArgument("lastName");

// Add the query to the session.
session.addQuery("getEmployeeWithName", query);

/* Execute the query by referencing its name and providing values for the specified
arguments */
Employee employee = (Employee) session.executeQuery("getEmployeeWithName","Bob","Smith");
```

Specialized Query Object Options

In addition to the query object configuration options discussed in "Creating a Query Object" on page 6-42, several more specialized options are available for customizing query objects

Ordering for Read All Queries Ordering is a common option for query objects. To order the collection of objects returned from a `ReadAllQuery`, use the `addOrdering()`, `addAscendingOrdering()`, or `addDescendingOrdering()` methods. You can apply order based on attribute names or query keys and expressions.

Example 6–43 A Query with Simple Ordering

```
// Retrieves objects ordered by lastName then firstName in Ascending Order
ReadAllQuery query = new ReadAllQuery();
query.setReferenceClass(Employee.class);
query.addAscendingOrdering ("lastName");
query.addAscendingOrdering ("firstName");
Vector employees = (Vector) session.executeQuery(query);
```

Example 6–44 A Query with Complex Ordering

```
/* Retrieves objects ordered by Street Address, descending case-insensitive
order of Cities, and manager's Last Name */
ReadAllQuery query = new ReadAllQuery();
query.setReferenceClass(Employee.class);
ExpressionBuilder emp = new ExpressionBuilder();
query.addOrdering (emp.getAllowingNull("address").get("street"));
query.addOrdering
(emp.getAllowingNull("address").get("city").toUpperCase().descending());
query.addOrdering(emp.getAllowingNull("manager").get("lastName"));
Vector employees = (Vector) session.executeQuery(query);
```

Note the use of `getAllowingNull`, which creates an outer join for the address and manager relationships. This ensures that employees without an address or manager still appear in the list.

For more information, see "Join Reading" on page 6-72.

Parameterized SQL in Query Objects To enable the parameterized SQL on individual queries, use the `bindAllParameters()` and `cacheStatement()` methods. This causes OracleAS TopLink to use a prepared statement, binding all SQL parameters and caching the prepared statement. When you re-execute this query, you avoid the SQL preparation, which improves performance.

For more information, see Chapter 10, "Tuning for Performance" on page 10-1.

Note: Do not use OracleAS TopLink's internal statement caching with an external connection pool.

Example 6–45 A Simple Read Query Object with Parameterized SQL

```
ReadObjectQuery query = new ReadObjectQuery(Employee.class);
query.setShouldBindAllParameters(true);
query.setShouldCacheStatement(true);
```

Collection Classes By default, a `ReadAllQuery` returns its result objects in a vector. You can configure the query to return the results in any collection class that implements the `Collection` or `Map` interface.

Example 6–46 Specifying the Collection Class for a Collection

```
ReadAllQuery query = new ReadAllQuery(Employee.class);
query.useCollectionClass(LinkedList.class);
LinkedList employees = (LinkedList) getSession().executeQuery(query);
```

Example 6–47 Specifying the Collection Class for a Map

```
ReadAllQuery query = new ReadAllQuery(Employee.class);
query.useMapClass(HashMap.class, "getFirstName");
HashMap employees = (HashMap) getSession().executeQuery(query);
```

For more information about interfaces, see "Working with Interfaces" in the *Oracle Application Server TopLink Mapping Workbench User's Guide.*

Using Cursoring for a ReadAllQuery The `ReadAllQuery` class includes methods for cursored stream and scrollable cursor support. If you expect the result set to be large, streams and cursors enable you to handle the result sets more efficiently.

For more information, see "Cursors and Streams" on page 6-81.

Query Optimization

OracleAS TopLink supports both joins and batch reads to optimize database reads. When your query reads a large number of objects, these techniques dramatically decrease the number of times you need to access the database during a read operation. Use the `addJoinedAttribute()` and `addBatchReadAttribute()` methods to configure query optimization.

For more information, see "Query Object Performance Options" on page 6-70, and Chapter 10, "Tuning for Performance" on page 10-1.

Other options to optimize queries include the `setMaxRows()` method and partial object reading.

Maximum Rows Returned You can limit a query to a specified maximum number of rows. Use this feature to avoid queries that can return an excessive number of objects.

To specify a maximum number of rows, use the `setMaxRows` method, and pass an integer that represents the maximum number of rows for the query.

Example 6–48 Setting the Maximum Returned Object Size

```
ReadAllQuery query = new ReadAllQuery();
query.setReferenceClass(Employee.class);
query.setMaxRows(5);
Vector employees = (Vector) session.executeQuery(query);
```

The `setMaxRows` method limits the number of rows the query returns, but does not enable you to acquire more records after the initial result set. If you want to browse the result set in fixed increments, use either cursors or cursored streams.

For more information, see "Java Streams" on page 6-60.

Partial Object Reading OracleAS TopLink enables you to query for partial objects. For example, you can create a read query that returns a subset of an object's attributes, rather than the entire object. This option improves read performance when the full object is not required. For example, use partial object reading to create a list of objects from which the client chooses the required object.

When you use partial object reading, be aware that:

- You cannot cache or edit partial objects.
- OracleAS TopLink does not automatically include primary key information in a partially populated object. If you need primary key information (for example: if you want to re-query or edit the object) specify it as one of the required attributes.

Use the `addPartialAttribute()` method to configure partial object reading.

For more information, see "Query Object Performance Options" on page 6-70, and Chapter 10, "Tuning for Performance".

Query Timeout You can implement a timeout for query objects. This enables you to automatically abort a hung or lengthy query after the specified time elapses. OracleAS TopLink throws a `DatabaseException` after the timeout.

To specify a timeout, implement the `setQueryTimeout()` call and pass the timeout interval as an integer representing the number of seconds before timeout occurs.

Example 6–49 Timeout on Query Objects

```
// Create the appropriate query and set timeout limits
ReadAllQuery query = new ReadAllQuery();
query.setReferenceClass(Employee.class);
query.setQueryTimeout(2);
try{
    Vector employees = (Vector)session.executeQuery(query);
} catch (DatabaseException ex) {
    // timeout occurs
}
```

Predefined Queries

Predefined queries enable you to create efficient, reusable queries. OracleAS TopLink creates predefined queries and registers them with a session or descriptor when the application starts. You can then retrieve the queries by name and execute them.

The most common way to create a predefined query is to register the query to a descriptor by specifying an amendment method with the OracleAS TopLink Mapping Workbench for an *after load* event.

Predefined queries improve the performance of frequently called queries because when you create a query, it is saved and reused as required. Each time you use a query, you create three or more objects that OracleAS TopLink uses to build the SQL statement. If you use predefined queries, OracleAS TopLink creates these objects only once, at binding time. OracleAS TopLink stores the queries as SQL statements in the descriptor and makes them available for the duration of the session.

In addition to performance improvements, predefined queries add structure to a querying framework and give you more options for reading query structure from alternative sources, such as XML.

Named Queries

Named queries improve application performance, because they reduce the resources required to run a query.

The `readAllObjects(Class c, Expression e)` creates a `ReadAllQuery`, which builds the other objects it needs to perform its task. After the you execute the `readEmployeesMatchingLastName` method, the `query`, `expression`, `expressionBuilder`, and any other related objects become *garbage*. Each time you call this method, OracleAS TopLink creates these related objects again, uses them once, and then discards them.

The use of named queries eliminates this behavior. To configure named queries, use a descriptor amendment method. This creates named queries when you open a database session.

Example 6–50 Named Query in the Descriptor File

```
public class MyTopLinkManager {
// some code that manages sessions, login, etc…
…
// This method is called by front end when needing to query on last names
   public Vector readEmployeesMatchingLastName(String theName) {
      ExpressionBuilder eBuilder = new ExpressionBuilder();
      Expression exp = eBuilder.get("lastName").like(theName+"%");
      return session.readAllObjects(Employee.class, exp);
   }
}
```

Use and Reuse OracleAS TopLink stores named queries by name on a per descriptor basis. When the application needs a query, it calls the named query and passes the required arguments. Because OracleAS TopLink builds the query when it opens the database session, the query is immediately available. In addition, the query is named and bound to a descriptor, so it is reusable.

The first time you execute a named query, OracleAS TopLink calculates the core SQL based on your database platform and schema. OracleAS TopLink caches this information and reuses it if you reuse the query.

Centralized Query Management OracleAS TopLink creates and registers named queries in a centralized location, usually your descriptor amendment method. Storing all queries in one location facilitates the reuse of queries and simplifies query maintenance.

When Not To Use Named Queries Rarely used queries may be more efficient when built on an as-needed basis. If you seldom use a given query, it may not be worthwhile to build and store that query when you invoke a session.

Named Finders

A named finder is an OracleAS TopLink query registered with an EJB container under a specific name. When using named finders, the `find` method on the `Home` interface must correspond to the name of an OracleAS TopLink query registered with the container. To implement and register the query with the container, use an OracleAS TopLink descriptor amendment method or session amendment class.

Example 6–51 A Named Finder

```
/* The named finder in this example uses an OracleAS TopLink query named
findCustomersInCity */
public Enumeration findCustomersInCity(String City)throws FinderException,
RemoteException;
```

Before you build and implement the `findCustomersInCity` finder shown in Example 6–51, define the corresponding named query, and register it with the project descriptor. To build the named query, employ:

- OracleAS TopLink Mapping Workbench Using EJB QL, SQL, or Expressions
- Java Code Using the OracleAS TopLink Expression Framework
- OracleAS TopLink Expression Framework
- Generic Named Finder

OracleAS TopLink Mapping Workbench Using EJB QL, SQL, or Expressions Use EJB QL, SQL, or the OracleAS TopLink expression framework in the OracleAS TopLink Mapping Workbench to:

- Define the query in the OracleAS TopLink Mapping Workbench. Specify the query in the bean descriptor's **Queries** tab.
- Add the query to the descriptor in a user-defined method.

Java Code Using the OracleAS TopLink Expression Framework Use the OracleAS TopLink expression framework to add the query employing a user defined method. Define these methods in one of the following ways:

- Use the OracleAS TopLink Mapping Workbench to specify a descriptor amendment method on the bean descriptor (see Example 6–52).
- Add a `preLogin` method to a session event listener class. Specify the session event listener classes using the `event-listener-class` element in the `toplink-ejb-jar.xml` descriptor (see Example 6–53).

Example 6–52 Define an Amendment Method

```
/* This example defines the findCustomersInCity query in the amendment method
of the descriptor */
public static void amendment(Descriptor descriptor) {
// create a query...

descriptor.getQueryManager().addQuery("findCustomersInCity", query);
```

Example 6–53 Define a Pre-Login Event

```
/* This example defines the findCustomersInCity query in the preLogin method
of a session event listener class and specifies the session event listener class
in the toplink-ejb-jar.xml deployment descriptor */
public void preLogin(SessionEvent event) {
// create a query...
event.getSession().getDescriptor(Customer.class).getQueryManager().addQuery("fin
dCustomersInCity", query);
}
```

OracleAS TopLink Expression Framework To use the OracleAS TopLink expression framework, define the finder in the OracleAS TopLink Mapping Workbench to specify the finder as a query object. Set the reference class to the name of the bean against which you run the query.

For more information, see "Query Objects" on page 6-41.

If you build your finder in code, use the `builder.getParameter()` call to retrieve the arguments defined in the query. Use the arguments for comparison, combining them with various predicates and operators, such as `equal()`, `like()`, and `anyOf()`.

Example 6–54 Using the OracleAS TopLink Expression Framework and Java Code

```
public static void addCustomerFinders(Descriptor descriptor) {
/* This code supports the query, Enumeration findCustomersInCity(String aCity)
Since this finder returns an Enumeration, it requires a ReadAllQuery. The finder
is a "NAMED" finder that is registered with the QueryManager */
//1 Define the query.
ReadAllQuery query = new ReadAllQuery();
query.setName("findCustomersInCity");
query.addArgument("aCity");
query.setReferenceClass(CustomerBean.class);
//2 Use an expression
```

```
ExpressionBuilder builder = new ExpressionBuilder();
query.setSelectionCriteria
builder.get("city").like(builder.getParameter("aCity"));
/*3 You can set options on the query, such as  query.refreshIdentityMapResult();
*/
//4 Register the query with the querymanager.
descriptor.getQueryManager().addQuery("findCustomersInCity",query);
}
```

Generic Named Finder You can use a named query without the need to provide the matching implementation on the `Home` interface. To do this, use the *Generic Named* finder provided by OracleAS TopLink. This finder takes the name of the named query and a vector of arguments as parameters.

Example 6–55 The Generic Named Finder

```
public Enumeration findAllByNamedQuery(String queryName, Vector arguments)
throws RemoteException, FinderException;
```

For more information about finders, see "EJB Finders" on page 6-85.

Redirect Queries

To perform complex operations, you can combine query redirectors with the OracleAS TopLink query framework. To create a redirector, implement the `oracle.toplink.queryframework.QueryRedirector` interface. The query mechanism executes the `Object invokeQuery(DatabaseQuery query, DatabaseRow arguments, Session session)` method and waits for the results.

OracleAS TopLink provides one pre-implemented redirector, the `MethodBasedQueryRedirector` method. To use this redirector, create a static `invoke method` on a class, and use the `setMethodName(String)` call to specify the method to invoke.

Example 6–56 Redirect Query

```
ReadObjectQuery query = new ReadObjectQuery(Employee.class);
query.setName("findEmployeeByAnEmployee");
query.addArgument("employee");

MethodBaseQueryRedirector redirector = new
MethodBaseQueryRedirector(QueryRedirectorTest.class,
"findEmployeeByAnEmployee");
query.setRedirector(redirector);
```

```
Descriptor descriptor = getSession().getDescriptor(query.getReferenceClass());
descriptor.getQueryManager().addQuery(query.getName(), query);

Vector arguments = new Vector();
arguments.addElement(employee);
objectFromDatabase = getSession().executeQuery(query,arguments);

public class QueryRedirectorTest{
public static Object findEmployeeByAnEmployee(DatabaseQuery query,
oracle.toplink.publicinterface.DatabaseRow arguments,
oracle.toplink.sessions.Session session) {
    ((ReadObjectQuery) query).setSelectionObject(arguments.get("employee"));
    return session.executeQuery(query);
}
}
```

EJBs and Redirect Finders

Redirect finders enable you to specify a finder in which the implementation is defined as a static method on an arbitrary helper class. When you invoke the finder, it redirects the call to the specified static method.

The finder can have any arbitrary parameters. If the finder includes parameters, OracleAS TopLink packages them into a vector and passes them to the redirect method.

Advantages Because you define the redirect finder implementation independently from the bean that invokes it, you can build the redirect finder to accept any type and number of parameters. This enables you to create a generic redirect finder that accepts several different parameters and return types, depending on input parameters.

A common strategy for using redirect finders is to create a generic finder that:

- Includes logic to perform several of tasks
- Reads the first passed parameter to identify the type of finder requested and select the appropriate logic

The redirect method contains the logic required to extract the relevant data from the parameters and uses it to construct an OracleAS TopLink query.

Disadvantages Redirect finders are complex and can be difficult to configure. They also require an extra helper method to define the query.

To create a redirect finder:

1. Declare the finder in the `ejb-jar.xml` file, and leave the `ejb-ql` tag empty.
2. Declare the finder on the `Home` interface, the `localHome` interface, or both, as required.
3. Create an amendment method.

   For more information, see "Customizing OracleAS TopLink Descriptors with Amendment Methods" on page 3-82.
4. Start the OracleAS TopLink Mapping Workbench.
5. Choose **Advanced Properties** > **After Load** from the menu for the bean.
6. Specify the class and name of the static method to enable the amendment method for the descriptor.

The amendment method then adds a query to the descriptor's query manager, as follows:

```
ReadAllQuery query = new ReadAllQuery();
query.setRedirector(new MethodBaseQueryRedirector (examples.ejb.cmp20.advanced.
  FinderDefinitionHelper.class,"findAllEmployeesByStreetName"));
descriptor.getQueryManager().addQuery ("findAllEmployeesByStreetName", query);
```

The redirect method must return either a single entity bean (object) or a vector. Here are the possible method signatures:

```
public static Object redirectedQuery2(oracle.toplink.sessions.Sessions, Vector args)
```

and

```
public static Vector redirectedQuery4(oracle.toplink.sessions.Sessions, Vector args)
```

When you implement the query method, ensure that the method returns the correct type. For methods that return more than one bean, set the return type to `java.util.Vector`. OracleAS TopLink converts this result to `java.util.Enumeration` (or Collection) if required.

Note: The redirect method also interprets an OracleAS TopLink session as a parameter. For more information about an OracleAS TopLink session, see Chapter 4, "Sessions".

At runtime, the client invokes the finder from the entity bean home and packages the arguments into the `args` vector in order of appearance from the finder method signature. The client passes the vector to the redirect finder, which uses them to execute an OracleAS TopLink expression.

Example 6–57 A Simple Redirect Query Implementation

```
public class RedirectorTest {
    private Session session;
    private Project project;
    public static void main(String args[]) {

       RedirectorTest test = new RedirectorTest();

       test.login();

            try {
            // Create the arguments to be used in the query
                Vector arguments = new Vector(1);
                arguments.add("Smith");

            // Run the query
                Object o = test.getSession()
                .executeQuery(test.redirectorExample(), arguments);
                o.toString();
            }
            catch (Exception e) {
                System.out.println("Exception caught -> " + e);
                e.printStackTrace();
            }
    }

    public ReadAllQuery redirectorExample() {

       // Create a redirector
       MethodBasedQueryRedirector redirector = new MethodBasedQueryRedirector();

       // Set the class containgin the public static method
       redirector.setMethodClass(RedirectorTest.class);

       // Set the name of the method to be run
       redirector.setMethodName("findEmployeeByLastName");

       // Create a query and add the redirector created above
       ReadAllQuery readAllQuery = new ReadAllQuery(Employee.class);
       readAllQuery.setRedirector(redirector);
       readAllQuery.addArgument("lastName");
```

```
        return readAllQuery;
    }
    //Call the static method
    public static Object findEmployeeByLastName(oracle.toplink.sessions
      .Session
    session, Vector arguments) {

        // Create a query
        ReadAllQuery raq = new ReadAllQuery();
        raq.setReferenceClass(Employee.class);
        raq.addArgument("lastName");

        // Create the selection criteria
        ExpressionBuilder employee = new ExpressionBuilder();
        Expression whereClause =
        employee.get("lastName").equal(arguments.firstElement());

        // Set the selection criteria
        raq.setSelectionCriteria(whereClause);

        return (Vector)session.executeQuery(raq, arguments);
    }
[...]
}
```

Queries Defined with the OracleAS TopLink Mapping Workbench

You can define several types of queries with the OracleAS TopLink Mapping Workbench, including custom SQL queries and named queries (which you can build using OracleAS TopLink expressions, EJB QL, or SQL).

For more information about the features and options available to create queries with the OracleAS TopLink Mapping Workbench, see "Understanding Descriptors," in the *Oracle Application Server TopLink Mapping Workbench User's Guide.*

Query Managers

A *query manager* is a descriptor-owned object that controls descriptor access to the database. The query manager generates its own SQL to access the database in a transparent manner.

You can modify the query manager to do the following:

- Customize the Default Query Methods

- Define Additional Join Expressions
- Customize the Existence Check

Customize the Default Query Methods

Query managers generate SQL for five database actions:

- Insert
- Update
- Delete
- Read
- Read all

The OracleAS TopLink `session` class provides default query objects to perform these database functions. However, you can also use the query manager to provide custom query objects or SQL strings to perform these functions.

For example, to replace the OracleAS TopLink `readObject` function with a stored procedure call, specify the replacement code in the OracleAS TopLink Mapping Workbench. If you use a Sybase database, the stored procedure call to read an object looks like this:

```
EXEC PROC Read_Employee(@EMP_ID = 4653)
```

To implement this replacement code, add the following string to read the object:

```
EXEC PROC Read_Employee(@EMP_ID = #EMP_ID)
```

In the deployed project, the query manager substitutes the code you specified for the `readObject` call in any queries that include this call.

For more information about customizing default query methods in the OracleAS TopLink Mapping Workbench, see "Custom SQL Queries" in the *Oracle Application Server TopLink Mapping Workbench User's Guide.*

Customize the Default Query Methods in Java Code To customize the query manager database access methods in Java code, use the `getQueryManager()` method to invoke the query manager. To change the default database access queries, use an amendment method listed in Table 6–5.

Table 6–5 Query Manager Methods for Database Access

To Change the Default	Use This Query Manager Method
Delete call using a query	`setDeleteQuery (DeleteObjectQuery query)`
Delete call using SQL	`setDeleteSQLString (String sqlString)`
Insert call using a query	`setInsertQuery (InsertObjectQuery query)`
Insert call using SQL	`setInsertSQLString (String sqlString)`
ReadAll call using a query	`setReadAllQuery (ReadAllQuery query)`
ReadAll call using SQL	`setReadAllSQLString (String sqlString)`
ReadObject call using a query	`setReadObjectQuery (ReadObjectQuery query)`
ReadObject call using SQL	`setReadObjectSQLString (String sqlString)`
Update call using a query	`setUpdateQuery (UpdateObjectQuery query)`
Update call using SQL	`setUpdateSQLString (String sqlString)`

Note: When you customize the update function for an application that uses optimistic locking, the custom update string must not write the object if the row version field has changed since the initial object was read. In addition, it must increment the version field if it writes the object successfully.

For example:

```
update Employee set F_NAME = #F_NAME, VERSION = VERSION + 1
where (EMP_ID = #EMP_ID) AND (VERSION = #VERSION)
```

The update string must also maintain the row count of the database.

Define Additional Join Expressions

You can set the query manager to automatically append an expression to every query it performs on a class. For example, you can add an expression that filters the database for the valid instances of a given class.

Use this to:

- Filter logically deleted objects.
- Enable two independent classes to share a single table without inheritance.
- Filter historical versions of objects.

The query manager provides the `setAdditionalJoinExpression()` and the `setMultipleTableJoinExpression()` methods for this purpose.

Example 6–58 Registering a Query that Includes a Join Expression

```
/* The join expression in this example filters invalid instances of employee
from the query */
public static void addToDescriptor(Descriptor descriptor)
{
    ExpressionBuilder builder = new ExpressionBuilder();
    descriptor.getQueryManager().setAdditionalJoinExpression((builder.getField("EMP.STATUS
      ").notEqual("DELETED")).and(builder.getField("EMP.STATUS").notEqual("HISTORICAL")));

}
```

Customize the Existence Check

When OracleAS TopLink writes an object to the database, OracleAS TopLink runs an existence check to determine whether to perform an insert or an update.

The query manager enables you to substitute custom logic for the existence check.

For more information on how to implement a custom existence check, see "Specifying Identity Mapping" in the *Oracle Application Server TopLink Mapping Workbench User's Guide.*

Use the following `DescriptorQueryManager` methods to modify the default existence checking:

```
checkCacheForDoesExist()
assumeExistenceForDoesExist()
assumeNonExistenceForDoesExist()
checkDatabaseForDoesExist()
setDoesExistQuery(DoesExistQuery)
setDoesExistSQLString(String)
```

Query Results

Queries can return different types of data, including:

- Objects
- Collections
- Java Streams
- Report Query Results

Queries can also return EJBs in systems that use EJB finders.

For more information, see "EJB Finders" on page 6-85.

Objects

OracleAS TopLink queries generally return Java objects as their result set. OracleAS TopLink queries can return

- Entire objects, with data and methods intact
- Partial objects (see "Partial Attribute Reading" on page 6-76)
- Vectors of objects
- Collections of objects (see "Collections" on page 6-60)

Collections

A collection is a group of Java objects related by a collection class that implements a `Collection` or `Map` interface. By default, `ReadAll` queries return results in a vector, but you can acquire the results in any collection class that implements the `Collection` or `Map` interface.

For more information on implementing `Collection` or `Map` interfaces, see the *Oracle Application Server TopLink Mapping Workbench User's Guide.*

Java Streams

A stream is a view of a collection, which can be a file, a device, or a `Vector`. A stream provides access to the collection, one element at a time in sequence. This makes it possible to implement stream classes in which the stream does not contain all the objects of a collection at the same time.

When a query is likely to generate a large result set, you can implement streams to improve performance.

For more information about streams, including advanced usage, see "Cursors and Streams" on page 6-81.

Report Query Results

Report query provides developers with a way to access information or data from a set of objects and their related objects. Report query supports database reporting functions and features. Although the report query returns data (not objects), it does enable you to query the returned data and specify it at the object level.

For more information, see "ReportQuery" on page 6-73.

Queries and the Cache

OracleAS TopLink caches objects written to and read from the database to maintain object identity. The sequence in which a query checks the cache and database affects query performance. By default, primary key queries check the cache before accessing the database, and all queries check the cache before rebuilding an object from its row.

Note: You can override the default behavior in the caching policy configuration information in the OracleAS TopLink descriptor. For more information, see "Explicit Query Refreshes" on page 8-13.

This section illustrates ways to manipulate the query-cache relationship, including:

- Cache Usage
- Disabling the Identity Map Cache Update During a Read Query
- Refresh
- Caching Query Results

Cache Usage

OracleAS TopLink maintains a client-side cache to reduce the number of reads required from the database.

Cache and the Database

The cache in an OracleAS TopLink application holds objects that have already been read from or written to the database. Use of the cache in an OracleAS TopLink application reduces the number of accesses to the database. Because accessing the database is a time-intensive and resource-intensive act, an effective caching strategy is important to the efficiency of your application.

For more information about configuring and using the cache, see Chapter 8, "Cache".

In-Memory Query Cache Usage

In-memory querying enables you to perform queries on the cache rather than the database. In-memory querying supports the following relationships:

- One-to-one
- One-to-many
- Many-to-many
- Aggregate collection
- Direct collection

Note: By default, the relationships themselves must be in memory for in-memory traversal to work. Ensure that you trigger all valueholders to enable in-memory querying to work across relationships.

You can configure in-memory query cache usage at the query level for both `readObject` and `readAll` queries. OracleAS TopLink supports the following in-memory query features:

- `checkCacheByPrimaryKey()`: The default setting; if a read object query contains an expression that compares at least the primary key, you can obtain a cache hit if you process the expression against the objects in memory.
- `checkCacheByExactPrimaryKey()`: If a read object query contains an expression where the primary key is the only comparison, you can obtain a cache hit if you process the expression against the object in memory.
- `checkCacheThenDatabase()`: You can configure any read object query to check the cache completely before you resort to accessing the database.
- `checkCacheOnly()`: You can configure any read all query to check only the cache and return the result from the cache without accessing the database.
- `conformResultsInUnitOfWork()`: You can configure any read object or read all query within the context of a Unit of Work to conform the results with the

changes to the object made within that Unit of Work. This includes new objects, deleted objects and changed objects.

Table 6–6 identifies the in-memory queries options OracleAS TopLink supports.

Table 6–6 In-Memory Queries OracleAS TopLink Supports

Type	Query OracleAS TopLink Supports

Comparators	`equal(..)`
	`notEqual(..)`
	`like(..) (with JDK 1.4 only)`
	`lessThan(..)`
	`lessThanOrEqual(..)`
	`greaterThan(..)`
	`greaterThanOrEqual(..)`
	`between(...)`
	`notBetween(...)`
	`isNull()`
	`notNull()`
	`in(...)`

Logical operators	`or(..)` `and(..)`

Joining	`get(..)`
	`getAllowingNull(..)`
	`anyOf(..)`
	`anyOfAllowingNone(..)`

Handling Exceptions Resulting from In-Memory Queries In-memory queries fail for several reasons, the most common of which are:

- The query expression is too complex to execute in memory.
- There are untriggered valueholders in which indirection is used. All object models that use indirection must first trigger valueholders before they conform on the relevant objects.

OracleAS TopLink provides a mechanism to handle indirection exceptions. To specify how the application must handle these exceptions, use `InMemoryQueryIndirectionPolicy` class:

- `Should throw indirection exception`: The default setting; it is the only setting that throws indirection exceptions.

- `Should trigger indirection`: Triggers all valueholders to eliminate the problem.
- `Should ignore exception return conformed`: Returns conforming if an untriggered valueholder are encountered.
- `Should ignore exception return not conformed`: Returns not conforming if an untriggered valueholder is encountered.

Note: When you build new applications, consider throwing all conform exceptions. This provides more detailed feedback for unsuccessful in-memory queries.

Conforming Results (UnitOfWork) You can conform query results in the Unit of Work across one-to-many, as well as a combination of one-to-one and one-to-many relationships. The following is an example of a query across two levels of relationships, one-to-many and one-to-one.

```
Expression exp =
  bldr.anyOf("managedEmployees").get("address").get("city").equal("Perth");
```

Note: When relationships in an in-memory query use indirection, trigger all valueholders to ensure that the objects are available in the cache.

Exceptions thrown by the conform feature are masked by default. However, OracleAS TopLink includes an API that allows exceptions to be thrown rather than masked. The API is: `uow.setShouldThrowConformExceptions(ARGUMENT)`.

`ARGUMENT` is an integer with one of the following values:

- Do not throw conform exceptions (default)
- Throw all conform exceptions

For more information, see "Validating a Unit of Work" on page 7-41.

Cache and the Primary Key

When a query searches for a single object by primary key, OracleAS TopLink extracts the primary key from the query and attempts to return the object from the cache without accessing the database. If the object is not in the cache, the query executes against the database, builds the resulting object(s), and places it in the identity map.

If the query is based on a non-primary key selection criteria or is a `readAll` query, the query executes against the database (unless you have selected the `checkCacheOnly()` option). The query matches primary keys from the result set to objects in the cache and returns the cached objects, if any, in the result set.

If an object is not in the cache, OracleAS TopLink builds the object. If the query is a refreshing query, OracleAS TopLink updates the contents of any objects with the results from the query. Use Object identity (==) if you properly configure and use an identity map.

Clients can refresh objects when they want to ensure that they have the latest data at a particular time.

Disabling the Identity Map Cache Update During a Read Query

To disable the identity map cache update, which is normally performed by a read query, call the `dontMaintainCache()` method. This improves the query performance when you read objects that are not needed later by the application.

Example 6–59 Disabling the Identity Map Cache Update

This example demonstrates how code reads Employee objects from the database and writes the information to a flat file.

```
// Reads objects from the employee table and writes them to an employee file.
void writeEmployeeTableToFile(String filename, Session session)
{
    Vector employeeObjects;
    ReadAllQuery query = new ReadAllQuery();
    query.setReferenceClass(Employee.class);
    query.setSelectionCriteria(new
    ExpressionBuilder.get("id").greaterThan(100));
    query.dontMaintainCache();
    Vector employees = (Vector) session.executeQuery(query);
    // Write all the employee data to a file.
    Employee.writeToFile(filename, employees);
}
```

Refresh

You can refresh objects in the cache to ensure that they are current with the database while preserving object identity.

Object Refresh

To refresh objects in the cache with the data in the database, call the `session.refreshObject()` method or the `readObjectQuery.setShouldRefreshIdentityMapResult(true)` method.

Cascading Object Refresh

You can control the depth at which a refresh updates objects and their related objects. There are three options:

- *CascadePrivateParts*: Default refresh behavior. Refreshes the local level object and objects that are referenced in privately owned, nonindirect, relationships.
- *CascadeNone*: Refreshes only the first level of the object, but does not refresh related objects.
- *CascadeAll*: Refreshes the entire object tree, stopping when it either reaches the leaf objects or when it encounters untriggered indirection in the tree.

Refreshing the Identity Map Cache During a Read Query

Include the `refreshIdentityMapResult()` method in a query to force an identity map refresh with the results of the query.

Example 6–60 Refreshing the Result of a Query in the Identity Map Cache During a Read Query

```
ReadObjectQuery query = new ReadObjectQuery();
query.setReferenceClass(Employee.class);
query.setSelectionCriteria(new
ExpressionBuilder().get("lastName").equal("Smith"));
query.refreshIdentityMapResult();
Employee employee = (Employee) session.executeQuery(query);
```

The `refreshIdentityMapResult()` method refreshes the object's attributes, but not the attributes of its privately owned parts. However, under most circumstances, refresh an object's privately owned parts and other related objects to ensure consistency with the database.

To refresh privately owned or related parts, use the following methods:

- `cascadePrivateParts()`: refreshes all privately owned objects
- `cascadeAllParts()`: refreshes all related objects

Example 6–61 Using the cascadePrivateParts Method

```
ReadAllQuery query = new ReadAllQuery();
query.setReferenceClass(Employee.class);
query.refreshIdentityMapResult();
query.cascadePrivateParts();
Vector employees = (Vector) session.executeQuery(query);
```

Note: If the object is in the session cache, you can also use the `refreshObject()` method to refresh an object and its privately owned parts.

Caching Query Results

When an application executes a query, you can store the results of that query in the cache. This is useful for frequently executed queries that run against static data. Caching the results also ensures that the query returns the same results for a given period of time (for example: within the scope of a particular transaction) and then refreshes the data later if required.

Query Objects and Write Operations

Although OracleAS TopLink applications most often perform database write operations through a Unit of Work, you can also write to the database with query objects. This section describes some of the more common strategies for using write queries and includes discussions on:

- Write Query Overview
- Non-Cascading Write Queries
- Disabling the Identity Map Cache During a Write Query
- Using Query Objects to Customize the Default Database Operations

Write Query Overview

To execute a write query, use a `WriteObjectQuery` instance instead of using the `writeObject()` method of the session. Likewise, substitute `DeleteObjectQuery`, `UpdateObjectQuery` and `InsertObjectQuery` objects for their respective `Session` methods.

Example 6–62 Using a WriteObjectQuery Object

```
WriteObjectQuery writeQuery = new WriteObjectQuery();
writeQuery.setObject(domainObject);
session.executeQuery(writeQuery);
```

Example 6–63 Using Other Write Query Objects with Similar Syntax

```
InsertObjectQuery insertQuery= new InsertObjectQuery();
insertQuery.setObject(domainObject);
session.executeQuery(insertQuery);

/* When you use UpdateObjectQuery without a Unit of Work, UpdateObjectQuery
writes all direct attributes to the database */
UpdateObjectQuery updateQuery= new UpdateObjectQuery();
updateQuery.setObject(domainObject2);
session.executeQuery(updateQuery);

DeleteObjectQuery deleteQuery = new DeleteObjectQuery();
deleteQuery.setObject(domainObject2);
session.executeQuery(deleteQuery);
```

Non-Cascading Write Queries

When you execute a write query, it writes both the object and its privately owned parts to the database by default. To build write queries that do not update privately owned parts, include the `dontCascadeParts()` method in your query definition.

Use this method to:

- Increase performance when you know that only the object's direct attributes have changed.
- Resolve referential integrity dependencies when you write large groups of new, independent objects.

Note: Because the Unit of Work resolves referential integrity internally, this method is not required if you use the Unit of Work to write to the database.

Example 6–64 Performing a Non-Cascading Write Query

```
// theEmployee is an existing employee read from the database.
Employee.setFirstName("Bob");
UpdateObjectQuery query = new UpdateObjectQuery();
query.setObject(Employee);
query.dontCascadeParts();
session.executeQuery(query);
```

Disabling the Identity Map Cache During a Write Query

When you write objects to the database, OracleAS TopLink copies them to the session cache by default. To disable this behavior within a query, call the `dontMaintainCache()` method within the query. This improves query performance when you insert objects into the database, but must only be used on objects that will not be required later by the application.

Example 6–65 Disabling the Identity Map Cache During a Write Query

This code reads all the objects from a flat file and writes new copies of the objects into a table.

```
// Reads objects from an employee file and writes them to the employee table.
void createEmployeeTable(String filename, Session session)
{
   Iterator iterator;
   Employee employee;
   // Read the employee data file.
   List employees = Employee.parseFromFile(filename);
   Iterator iterator = employees.iterator();
   while (iterator.hasNext()) {
      Employee employee = (Employee) iterator.next();
      InsertObjectQuery query = new InsertObjectQuery();
      query.setObject(employee);
      query.dontMaintainCache();
      session.executeQuery(query);
}
```

Caution: Disable the identity map only when object identity is unimportant in subsequent operations.

Using Query Objects to Customize the Default Database Operations

OracleAS TopLink provides default querying behavior for each of the read and write operations that is sufficient for most applications. In addition, applications can define their own custom queries where required:

- If the custom query is specific to a persistent class, register it with the descriptor of that class.

  For more information, see "Query Managers" on page 6-56.

- If the custom query is global for the project rather than specific to a particular class, register it with the session. Execute registered queries by calling one of the `executeQuery()` methods of `DatabaseSession` or `UnitOfWork`.

Query Object Performance Options

Several optimizations are available that improve the performance of your queries, including:

- Batch Reading
- Join Reading
- ReportQuery
- Partial Attribute Reading
- Cache Results In Query Objects

For more information about improving the performance of your application and information on how to optimize queries, see Chapter 10, "Tuning for Performance".

Batch Reading

Batch reading propagates query selection criteria through an object's relationship attribute mappings. You can also nest batch reads down through complex object graphs. This significantly reduces the number of required SQL select statements and improves database access efficiency.

For example, in reading n employees and their related projects, OracleAS TopLink may require n + 1 selects. All employees are read at once, but the projects of each are read individually. With batch reading all related projects can also be read with one select by using the original selection criteria, for a total of only 2 selects.

To implement batch reading, use one of the following methods:

- To add the batch read attribute to a query, use the `query.addBatchReadAttribute(Expression anExpression)` API.

  For example:

  ```
  …
  ReadAllQuery raq = new ReadAllQuery(Trade.class);
  ExpressionBuilder tradeBuilder = raq.getBuilder();
  …
  Expression batchReadProduct = tradeBuilder.get("product");
  readAllQuery.addBatchReadAttribute(batchReadProduct);
  Expression batchReadPricingDetails = batchReadProduct.get("pricingDetails");
  readAllQuery.addBatchReadAttribute(batchReadPricingDetails);
  …
  ```

- Add batch reading at the mapping level for a descriptor. Use either the OracleAS TopLink Mapping Workbench or a descriptor amendment method to add the `setUsesBatchReading()` API on the descriptor's relationship mappings.

  For example:

  ```
  public static void amendTradeDescriptor(Descriptor theDescriptor) {
      OneToOneMapping productOneToOneMapping =
        theDescriptor.getMappingForAttributeName("product");
      productOneToOneMapping.setUsesBatchReading(true);
  }
  ```

You can combine batch reading and indirection to provide controlled reading of object attributes. For example, if you have one-to-one backpointer relationship attributes, you can defer backpointer instantiation until the end of the query, when all parent and owning objects are instantiated. This prevents unnecessary database access and optimizes OracleAS TopLink cache use.

Guidelines for Implementing Batch Reading

Consider the following guidelines when you implement batch reading:

- Use batch reading for processes that read in objects and all their related objects.
- Do not enable batch reading for both sides of a bidirectional relationship.

- Avoid nested batch reads, because they result in multiple joins on the database that can slow query execution.

For more information, see "Reading Case 2: Batch Reading Objects" on page 10-12.

Join Reading

When OracleAS TopLink queries, it can use joins to check values from other objects or other tables that represent parts of the same object. Although this works well under most circumstances, it can cause problems when you query against a one-to-one relationship in which one side of the relationship is not present.

For example, `Employee` objects may have an `Address` object, but if the `Address` is unknown, it is null at the object level and has a null foreign key at the database level. When you attempt a read that traverses the relationship, missing objects cause the query to return unexpected results. Consider the expression:

```
(emp.get("firstName").equal("Steve")).or(emp.get("address"). get("city").equal("Ottawa"))
```

In this case, employees with no address do not appear in the result set, regardless of their first name. Although not obvious at the object level, this behavior is fundamental to the nature of relational databases.

Outer joins rectify this problem in the databases that support them. In this example, the use of an outer join provides the expected result: all employees named Steve appear in the result set, even if their address is unknown.

To implement an outer join, use `getAllowingNull()` rather than `get()`, and `anyOfAllowingNone()` rather than `anyOf()`.

For example:

```
(emp.get("firstName").equal("Steve")).or
  (emp.getAllowingNull ("address").get("city").equal("Ottawa"))
```

Support and syntax for outer joins vary widely between databases and database drivers. OracleAS TopLink supports outer joins for Oracle databases, IBM DB2, SQL Anywhere, Microsoft Access, Microsoft SQL Server, Sybase SQL Server, and the JDBC outer join syntax. Of these, only Oracle supports the outer join semantics in `or` clauses.

You can also use outer joins with ordering.

For more information, see "Ordering for Read All Queries" on page 6-44.

Join reading enables you to read data from a one-to-one mapping in conjunction with data from the original query. Join reading is available only for one-to-one mappings. To implement join reading, use either of the following methods:

- To add the joined attribute to the query at the query level, use the `Query.addJoinedAttribute(Expression anExpression)` API.

  For example:

  ```
  …
  ReadAllQuery raq = new ReadAllQuery(Trade.class);
  ExpressionBuilder tradeBuilder = raq.getBuilder();
  …
  Expression portfolio = tradeBuilder.get("portfolio");
  readAllQuery.addJoinedAttribute(portfolio);
  …
  ```

- Use the OracleAS TopLink Mapping Workbench or a descriptor amendment method to invoke the `setUsesJoining()` API on the `OneToOneMapping` class, as follows:

  ```
  public static void amendTradeDescriptor(Descriptor theDescriptor) {
      OneToOneMapping portfolioOneToOneMapping =
        theDescriptor.getMappingForAttributeName("portfolio");
      portfolioOneToOneMapping.setUsesJoining(true);
  }
  ```

For more information about joins as a performance tool, see Chapter 10, "Tuning for Performance".

ReportQuery

Report query enables you to retrieve data from a set of objects and their related objects. Report query supports database reporting functions and features. Although the report query returns data rather than objects, it still enables you to query and specify the data at the object level.

The `ReportQuery` API returns a collection of `ReportQueryResult` objects, similar in structure and behavior to a `DatabaseRow` or a `Map`.

Report query allows you to:

- Specify a subset of the object's attributes and its related object's attributes, which allows you to query for lightweight information.

- Build complex object-level expressions for the selection criteria and ordering criteria.
- Use database aggregation functions, such as `SUM`, `MIN`, `MAX`, `AVG`, and `COUNT`.
- Use expressions to group data.
- Request primary key attributes with each `ReportQueryResult`. This makes it easy to request the real object from a lightweight result.

> **Note:** OracleAS TopLink report queries do not support multiple references to the same attribute in a single result set.

Example 6–66 Querying Reporting Information on Employees

This example reports the total and average salaries for Canadian employees grouped by their city.

```
ExpressionBuilder emp = new ExpressionBuilder();
ReportQuery query = new ReportQuery(emp);
query.setReferenceClass(Employee.class);
query.addMaximum("max-salary", emp.get("salary"));
query.addAverage("average-salary", emp.get("salary"));
query.addAttribute("city", emp.get("address").get("city"));

query.setSelectionCriteria(emp.get("address").get("country").equal("Canada"));
query.addOrdering(emp.get("address").get("city"));
query.addGrouping(emp.get("address").get("city"));
Vector reports = (Vector) session.executeQuery(query);
```

Table 6–7 summarizes the most common public methods for `ReportQuery`. For more information about the available methods for the `ReportQuery`, see the *Oracle Application Server TopLink API Reference*.

Table 6–7 Elements for Report Query

Element	Default	Method Name
Adding items to select	Nothing selected	addAttribute**(String itemName)** addAttribute**(String itemName, Expression attributeExpression)** addAverage**(String itemName)** addAverage**(String itemName, Expression attributeExpression)** addMaximum**(String itemName)** addMaximum**(String itemName, Expression attributeExpression)** addMinimum**(String itemName)** addMinimum**(String itemName, Expression attributeExpression)** addSum**(String itemName)** addSum**(String itemName, Expression attributeExpression)** addStandardDeviation**(String itemName)** addStandardDeviation**(String itemName, Expression attributeExpression)** addVariance**(String itemName)** addVariance**(String itemName, Expression attributeExpression)** addCount**()** addCount**(String itemName)** addCount**(String itemName, Expression attributeExpression)** addItem**(String itemName, Expression attributeExpression)** addFunctionItem**(String itemName, Expression attributeExpression, String functionName)**
Group by	Not grouped	addGrouping**(String attributeName)** addGrouping**(Expression expression)**
Retrieving primary keys	Not retrieved	retrievePrimaryKeys**()** dontRetrievePrimaryKeys**()** setShouldRetrievePrimaryKeys**(boolean shouldRetrievePrimaryKeys)**

Note: Because `ReportQuery` inherits from `ReadAllQuery`, it also supports most `ReadAllQuery` properties.

Partial Attribute Reading

You can query for parts of objects rather than complete objects. For example, you can build a read query that returns a subset of an object's attributes rather than the entire object. This improves database read performance when you do not require the complete object.

To configure partial object reading, use the `addPartialAttribute()` method. For more information, see "Partial Object Reading" on page 10-10.

Consider the following when you use partial object reading:

- You cannot edit or cache partial objects.
- OracleAS TopLink does not automatically include primary key information in a partial object. If you need primary key information (for example: if you want to re-query or edit the object) specify it as a required attribute.

Cache Results In Query Objects

Query objects maintain an internal cache of the objects previously returned by the query. This improves query performance and ensures that the query always returns the same objects.

The internal cache is disabled by default. To enable it, use the `cacheQueryResults()` method in the query.

Example 6–67 Using the Internal Query Object Cache

```
ReadObjectQuery query = new ReadObjectQuery();
query.setReferenceClass(Employee.class);
query.cacheQueryResults();

// The query object reads from the database the first time you invoke it.
Employee employee = (Employee) session.executeQuery(query);

/* On this second call to execute the query, the query object does not read from
the database, but reads from the query object’s internal cache instead */
Employee employee = (Employee) session.executeQuery(query);
```

Oracle Extension Support

OracleAS TopLink supports the following Oracle enterprise enhancements for Oracle databases:

- Oracle Hints and the OracleAS TopLink Query Framework

- Hierarchical Queries

Oracle Hints and the OracleAS TopLink Query Framework

Oracle Hints is an Oracle database feature through which a developer makes decisions usually reserved for the optimizer. Developers use hints to specify things such as join order for a join statement, or the optimization approach of a SQL call.

The OracleAS TopLink query framework supports Oracle Hints with the following API:

```
addHintString("/*[hints or comments]*/");
```

OracleAS TopLink adds the hint to the SQL string as a comment immediately following a `SELECT`, `UPDATE`, `INSERT`, or `DELETE` statement.

To add hints to a read query:

1. Create a `ReadObjectQuery` or a `ReadAllQuery`
2. Set the selection criteria.
3. Add hints as needed.

For example, the following code uses the *`FULL`* hint (which explicitly chooses a full table scan for the specified table):

```
// This line sets up the query
ReadObjectQuery query = new ReadObjectQuery(Employee.class);
query.setSelectionCritera(new ExpressionBuilder().get("id").equal(new Integer(1));
// This line adds the hint
query.addHintString("/*+ FULL */" );
```

This code generates the following SQL:

```
SELECT /*+ FULL */ FROM EMPLOYEE WHERE ID=1
```

To add hints to `WRITE`, `INSERT`, `UPDATE`, and `DELETE`, create custom queries for these operations in the OracleAS TopLink query framework, then specify hints as required.

For more information about the available hints, see the Oracle database documentation.

Hierarchical Queries

Hierarchical Queries is an Oracle database mechanism that enables you to select database rows based on hierarchical order. For example, you can design a query that reads the row of a given employee, followed by the rows of people the employee manages, followed by their managed employees, and so on.

To create a hierarchical query, use the `setHierarchicalQueryClause()` method. This method takes three parameters, as follows:

```
setHierarchicalQueryClause(StartWith, ConnectBy, OrderSibling)
```

This expression requires all three parameters, as follows:

StartWith Parameter The `StartWith` parameter in the expression specifies the first object in the hierarchy. This parameter mirrors the Oracle database `START WITH` clause.

To include a `StartWith` parameter, build an expression to specify the appropriate object, and pass it as a parameter in the `setHierarchicalQueryClause()` method. If you do not specify the root object for the hierarchy, set this value to `NULL`.

ConnectBy Parameter The `ConnectBy` parameter specifies the relationship that creates the hierarchy. This parameter mirrors the Oracle database `CONNECT BY` clause.

Build an expression to specify the `ConnectBy` parameter, and pass it as a parameter in the `setHierarchicalQueryClause()` method. Because this parameter defines the nature of the hierarchy, it is required for the `setHierarchicalQueryClause()` implementation.

OrderSibling Parameter The `OrderSibling` parameter in the expression specifies the order in which the query returns sibling objects in the hierarchy. This parameter mirrors the Oracle database `ORDER SIBLINGS` clause.

To include an `OrderSibling` parameter, define a vector, and to include the order criteria, use the `addElement()` call. Pass the vector as the third parameter in the `setHierarchicalQueryClause()` method. If you do not specify an order, set this value to `NULL`.

Example 6–68 Hierarchical Query

```
ReadAllQuery raq = new ReadAllQuery(Employee.class);
// Specify a START WITH expression
```

```
Expression startExpr = expressionBuilder.get("id").equal(new Integer(1));
// Specifies a CONNECT BY expression
Expression connectBy = expressionBuilder.get("managedEmployees");
//Specifies an ORDER SIBLINGS BY vector
Vector order = new Vector();
order.addElement(expressionBuilder.get("lastName"));
order.addElement(expressionBuilder.get("firstName"));
raq.setHierarchicalQueryClause(startExpr, connectBy, order);
Vector employees = uow.executeQuery(raq);
```

This code generates the following SQL:

```
SELECT * FROM EMPLOYEE START WITH ID=1 CONNECT BY PRIOR ID=MANAGER_ID ORDER
SIBLINGS BY LAST_NAME, FIRST_NAME
```

Advanced Querying

OracleAS TopLink offers several advanced mechanisms and techniques that enhance your queries. This section describes the following:

- Creating Additional Query Keys
- Querying on Interfaces
- Querying on an Inheritance Hierarchy
- Cursors and Streams
- Querying Across Variable One-to-One Mappings

Creating Additional Query Keys

A query key is an alias for a field name. Instead of referring to a field using a DBMS-specific field name such as `F_NAME`, query keys allow OracleAS TopLink expressions to refer to the field using Java attribute names, such as `firstName`.

For more information about Query Keys, see "Query Keys" on page 6-24.

You can implement query keys either with the OracleAS TopLink Mapping Workbench or in Java.

For more information about implementing query keys with the OracleAS TopLink Mapping Workbench, see "Working with Query Keys" in the *Oracle Application Server TopLink Mapping Workbench User's Guide.*

Implementing Query Keys in Java

To add and register query keys with a descriptor, implement the following methods:

- `addQueryKey()`: method of the `Descriptor` class for regular query keys
- `addDirectQueryKey()`: method for one-to-one query keys that specifies the name of the query key and the name of the table field
- `addAbstractQueryKey()`: method for abstract query keys

Example 6–69 Implementing a One-to-One Query Key

```
// Add a query key for the foreign key field using the direct method
descriptor.addDirectQueryKey("managerId", "MANAGER_ID");

// The same query key can also be added through the add method
DirectQueryKey directQueryKey = new DirectQueryKey();
directQueryKey.setName("managerId");
directQueryKey.setFieldName("MANAGER_ID");
descriptor.addQueryKey(directQueryKey);

/* Add a one-to-one query key for the large project that the employee is a
leader of (this assumes only one project) */
OneToOneQueryKey projectQueryKey = new OneToOneQueryKey();
projectQueryKey.setName("managedLargeProject");
projectQueryKey.setReferenceClass(LargeProject.class);
ExpressionBuilder builder = new ExpressionBuilder();
projectQueryKey.setJoinCriteria(builder.getField("PROJECT.LEADER_
ID").equal(builder.getParameter("EMPLOYEE.EMP_ID")));
descriptor.addQueryKey(projectQueryKey);
```

Example 6–70 Implementing a One-to-Many Query Key

```
/* Implements keys for the projects where the employee manages multiple projects
*/
OneToManyQueryKey projectsQueryKey = new OneToManyQueryKey();
projectsQueryKey.setName("managedProjects");
projectsQueryKey.setReferenceClass(Project.class);
ExpressionBuilder builder = new ExpressionBuilder();
projectsQueryKey.setJoinCriteria(builder.getField("PROJECT.LEADER_
ID").equal(builder.getParameter("EMPLOYEE.EMP_ID")));
descriptor.addQueryKey(projectsQueryKey);
// Next define the mappings.
...
```

Example 6–71 Implementing a Many-to-Many Query Key

```
ManyToManyQueryKey key = new ManyToManyQueryKey();
key.setName("myAs");
key.setReferenceClass(A.class);
ExpressionBuilder builder = new ExpressionBuilder();
Expression exp = builder.getField("AB_JOIN.B_
ID").equal(builder.getParameter("B.ID" ));
Expression exp1 = builder.getField("AB_JOIN.A_
ID").equal(builder.getField("A.ID") );
key.setJoinCriteria(exp.and(exp1));
descriptor.addQueryKey(key);
```

Querying on Interfaces

When you define descriptors for an interface to enable querying, OracleAS TopLink supports querying on an interface, as follows:

- If there is only a single implementor of the interface, the query returns an instance of the concrete class.
- If there are multiple implementors of the interfaces, the query returns instances of all implementing classes.

Querying on an Inheritance Hierarchy

When you query on a class that is part of an inheritance hierarchy, the session checks the descriptor to determine the type of the class:

- If you configure the descriptor to read subclasses (the default configuration), the query returns instances of the class and its subclasses.
- If you configure the descriptor to not read subclasses, the query returns only instances of the queried class, but no instances of the subclasses.
- If neither of these conditions apply, the class is a leaf class and does not have any subclasses. The query returns instances of the queried class.

Cursors and Streams

Cursors and streams are related mechanisms that enable you to work with large result sets efficiently.

Cursors and Java Iterators

The OracleAS TopLink scrollable cursor enables you to scroll through a result set from the database without reading the whole result set in a single database read. The `ScrollableCursor` class implements the Java `ListIterator` interface to allow for direct and relative access within the stream. Scrollable cursors also enable you to scroll forward and backward through the stream.

Traversing Data with Scrollable Cursors Several methods enable you to navigate data with a scrollable cursor:

- `relative(int i)`: advances the row number in relation to the current row by one row
- `absolute(int i)`: places the cursor at an absolute row position, 1 being the first row

Several strategies are available for traversing data with cursors. For example, to start at the end of the data set and work toward the first record:

1. Call the `afterLast()` method to place the cursor after the last row in the result set.
2. Use the `hasPrevious()` method to determine whether there is a record above the current record. This method returns FALSE when you reach the final record in the data set.
3. If the `hasPrevious()` method returns TRUE, call the `previous()` method to move the cursor to the row above the current row and read that object.

These are common methods for data traversal, but they are not the only available methods. For more information about the available methods, see the *Oracle Application Server TopLink API Reference*.

To use the `ScrollableCursor` object, the JDBC driver must be compatible with JDBC 2.0 specifications.

Example 6–72 Traversing with a Scrollable Cursor

```
ReadAllQuery query = new ReadAllQuery();
query.setReferenceClass(Employee.class);
query.useScrollableCursor();
ScrollableCursor cursor = (ScrollableCursor) session.executeQuery(query);

while (cursor.hasNext()) {
    System.out.println(cursor.next().toString());
}
```

```
cursor.close();
```

Java Streams

Java streams enable you to retrieve query results as individual records or groups of records, which can result in a performance increase. You can use streams to build efficient OracleAS TopLink queries, especially when the queries are likely to generate large result sets.

Cursored Stream Support Cursored streams combine the iterative ability of the `ScrollableCursor` interface with OracleAS TopLink support for streams. The result is the ability to read back a query result set from the database in manageable subsets, and to scroll through the result set stream.

The `useCursoredStream()` method of the `ReadAllQuery` class provides cursored stream support.

Example 6–73 Cursored Streams

```
CursoredStream stream;
ReadAllQuery query = new ReadAllQuery();
query.setReferenceClass(Employee.class);
query.useCursoredStream();
stream = (CursoredStream) session.executeQuery(query);
```

The query returns an instance of `CursoredStream` rather than a `Vector`, which can be a more efficient approach. For example, consider the following two code examples. Example 6–74 returns a `Vector` that contains all employee objects. If ACME has 10,000 employees, the `Vector` contains references to 10,000 `Employee` objects.

Example 6–74 Using a Vector

```
ReadAllQuery query = new ReadAllQuery();
query.setReferenceClass(Employee.class);
Enumeration employeeEnumeration;

Vector employees = (Vector) session.executeQuery(query);
employeeEnumeration = employee.elements();

while (employeeEnumeration.hasMoreElements())
{
Employee employee = (Employee) employeeEnumeration.nextElement();
```

```
employee.doSomeWork();
}
```

Example returns a `CursoredStream` instance rather than a `Vector`. The `CursoredStream` collection appears to contain all 10,000 objects, but initially contains a reference only to the first 10 `Employee` objects. It retrieves the remaining objects in the collection as they are needed. In many cases, the application never needs to read all the objects.

The following approach results in a significant performance increase:

```
ReadAllQuery query = new ReadAllQuery();
query.setReferenceClass(Employee.class);
query.useCursoredStream();

CursoredStream stream = (CursoredStream) session.executeQuery(query);
while (! stream.atEnd())
{
    Employee employee = (Employee) stream.read();
    employee.doSomeWork();
    stream.releasePrevious();
}
stream.close();
```

Note: The `releasePrevious()` message is optional. This releases any previously read objects and frees system memory. Even though released objects are removed from the cursored stream storage, they remain in the identity map.

Optimizing Streams

To optimize `CursoredStream` performance, provide a *threshold* and *page size* to the `useCursoredStream(Threshold, PageSize)` method, as follows:

- The threshold specifies the number of objects to read into the stream initially. The default threshold is 10.
- The page size specifies the number of objects to read into the stream after the initial group of objects. This occurs after the threshold number of objects is read. Although larger page sizes result in faster overall performance, they introduce delays into the application when OracleAS TopLink loads each page. The default page size is 5.

When you execute a batch-type operation, use the `dontMaintainCache()` option with a cursored stream. A batch operation performs simple operations on large numbers of objects and then discards the objects. Cursored streams create the required objects only as needed, and the `dontMaintainCache()` option ensures that these transient objects are not cached.

Querying Across Variable One-to-One Mappings

OracleAS TopLink does not provide a method to directly query against variable one-to-one mappings. To query against this type of mapping, combine OracleAS TopLink `DirectQueryKeys` and OracleAS TopLink `ReportQueries` to create query selection criteria for classes that implement the interface, as follows:

1. Create two `DirectQueryKeys` to query for the possible implementors of the interface.
   - The first `DirectQueryKey` is for the class indicator field for the variable one-to-one mapping.
   - The second `DirectQueryKey` is for the foreign key to the class or table that implements the interface.
2. Create a `subSelect` statement for each concrete class that implements the interface included in the query selection criteria.
3. Implement a `ReportQuery`.

Example 6–75 Creating DirectQueryKeys

```
/*The DirectQueryKeys as generated in the OracleAS TopLink project java source
code from the OracleAS TopLink Mapping Workbench */
…
descriptor.addDirectQueryKey("locationTypeCode","DEALLOCATION.DEALLOCATIONOBJECTTYPE");
descriptor.addDirectQueryKey("locationTypeId","DEALLOCATION.DEALLOCATIONOBJECTID");
```

EJB Finders

The OracleAS TopLink query framework enables you to construct *finders*, which are queries that retrieve entity beans. This section describes OracleAS TopLink support for finders, and includes discussions on the following topics and techniques:

- Defining Finders in OracleAS TopLink
- ejb-jar.xml Finder Options

- Call Finders
- Expression Finders
- EJB QL Finders
- SQL Finders
- Dynamic Finders
- ReadAll Finders
- Choosing the Best Finder Type for Your Query
- ejbSelect
- Advanced Finder Options

Defining Finders in OracleAS TopLink

To define a finder method for an entity bean that uses the OracleAS TopLink query framework, follow these steps:

1. Declare the finder in the `ejb-jar.xml` file.
2. Define the finder method.
   - For EJB 1.1 beans, define the method on the entity bean's remote interface.
   - For EJB 2.0 beans, define the method on the entity bean's `remoteHome` or `localHome` interface.
3. Use the OracleAS TopLink Mapping Workbench to change any options on finders.
4. If required, create an implementation for the query. Some query options require a query definition in code on a helper class, but most common queries do not.

When you use OracleAS TopLink CMP, define finder methods on the bean’s `Home` interface, not in the entity bean itself. OracleAS TopLink CMP provides this functionality and offers several strategies to create and customize finders. The EJB container and OracleAS TopLink automatically generate the implementation.

ejb-jar.xml Finder Options

The `ejb-jar.xml` file contains a project’s EJB entity bean information, including definitions for any finders used for the beans. To create and maintain the `ejb-jar.xml` file, use either a text editor or the OracleAS TopLink Sessions Editor.

entity tag

The `entity` tag encapsulates a definition for an EJB entity bean. Each bean has its own `entity` tag that contains several other tags that define bean functionality, including bean finders.

Example 6–76 illustrates the structure of a typical finder defined within the `ejb-jar.xml` file.

Example 6–76   A Simple Finder Within the ejb-jar.xml File

```
<entity>...
    <query>
        <query-method>
            <method-name>findLargeAccounts</method-name>
                <method-params>
                <method-param>double</method-param>
            </method-params>
        </query-method>
    <ejb-ql><![CDATA[SELECT OBJECT(account) FROM AccountBean account WHERE
       account.balance > ?1]]></ejb-ql>
    </query>
...
</entity>
```

query Section The `entity` tag contains zero or more `query` elements. Each `query` tag corresponds to a finder method defined on the bean's home or local `Home` interface.

> **Note:** You can share a single query between both `Home` interfaces, as follows:
>
> - Define the same finder (same name, return type, and parameters) on both `Home` interfaces.
> - Include a single `query` element in the `ejb-jar.xml` file.

Here are the elements defined in the `query` section of the `ejb-jar.xml` file:

- `description` (optional): Provides a description of the finder.
- `query-method`: Specifies the method for a finder or `ejbSelect` query.
- `method-name`: Specifies the name of a finder or select method in the entity bean implementation class.
- `method-params`: Contains a list of the fully-qualified Java type names of the method parameters.

- `method-param`: Contains the fully-qualified Java type name of a method parameter.
- `result-type-mapping` (optional): Specifies how to map an abstract schema type returned by a query for an `ejbSelect` method. You can map the type to an `EJBLocalObject` or `EJBObject` type. Valid values are Local or Remote
- `ejb-ql`: Used for all EJB QL finders. It contains the EJB QL query string that defines the finder or `ejbSelect` query. Leave this element empty for non-EJB QL finders.

Call Finders

Call finders enable you to create queries dynamically and generate the queries at runtime rather than deployment time. Call finders pass an OracleAS TopLink `SQLCall` or `StoredProcedureCall` as a parameter and return an `Enumeration`.

Creating Call Finders

OracleAS TopLink provides the implementation for Call finders. To use this feature in a bean, add the following finder definition to the `Home` interface of your bean.

```
public Enumeration findAll(Call call) throws RemoteException, FinderException;
```

Executing a Call Finder

When you execute a Call finder, OracleAS TopLink creates the call on the client using the OracleAS TopLink interface `oracle.toplink.queryframework.Call`. This call has three implementors: `EJBQLCall`, `SQLCall` and `StoredProcedureCall`.

Example 6–77 Executing a Call Finder (Select Statement)

```
{
    SQLCall call = new SQLCall();
    call.setSQLString("SELECT * FROM EMPLOYEE");
    Enumeration employees = getEmployeeHome().findAll(call);
}
```

Example 6–78 Executing a Call Finder (Stored Procedure)

```
{
    StoredProcedureCall call = new StoredProcedureCall();
    call.setProcedureName("READ_ALL_EMPLOYEES");
```

```
    Enumeration employees = getEmployeeHome().findAll(call);
}
```

Expression Finders

To define finder query logic, use OracleAS TopLink expressions. Expression finders support dynamic queries that you generate at runtime rather than deployment time. To use an expression finder, pass the expression as a parameter to a finder that returns an `Enumeration`.

Example 6–79 Executing an Expression Finder

```
{
    Expression expression = new
    ExpressionBuilder().get("firstName").like("J%");
    Enumeration employees =
    getEmployeeHome().findAll(expression);
}
```

EJB QL Finders

EJB QL is the standard query language defined in the EJB 2.0 specification. OracleAS TopLink supports EJB QL for both EJB 1.1 and EJB 2.0 beans. EJB QL finders enable you to specify an EJB QL string as the implementation of the query.

EJB QL offers several advantages:

- It is the EJB 2.0 standard for queries.
- You can use it to construct most queries.
- You can implement dependent object queries with EJB QL.

The disadvantage of EJB QL is that it is difficult to use when you construct complex queries.

To create an EJB QL finder under EJB 1.1:

1. Declare the finder on the `remote` interface.
2. Start the OracleAS TopLink Mapping Workbench.
3. Go to the **Queries** > **Finders** > **Named Queries** tab for the bean.

4. Add a finder and give it a name that matches the method name you declared in Step 1.
5. Set the required parameters.
6. Set **Query Format** to EJB QL, and enter the EJB QL query in the **Query String** field.

To create an EJB QL finder under EJB 2.0:

1. Declare the finder on either the `localHome` or the `remoteHome` interface.
2. Start the OracleAS TopLink Mapping Workbench.
3. Re-import the `ejb-jar.xml` file to synchronize the project to the file.

   The OracleAS TopLink Mapping Workbench synchronizes changes between the project and the `ejb-jar.xml` file.

The following is an example of a simple EJB QL query that requires one parameter. In this example, the question mark ("?") in`?1` specifies a parameter.

```
SELECT OBJECT(employee) FROM Employee employee WHERE (employee.name =?1)
```

To create an EJB QL finder for a CMP bean:

1. Declare the finder in the `ejb-jar.xml` file, and enter the EJB QL string in the `ejb-ql` tag.
2. Declare the finder on the `Home` interface, the `localHome` interface, or both, as required.
3. Start the OracleAS TopLink Mapping Workbench.
4. Specify the `ejb-jar.xml` file location and choose **File** > **Updated Project** from the `ejb-jar.xml` file to read in the finders.
5. Go to the **Queries** > **Finders** > **Named Queries** tab for the bean.
6. Add a finder, and give it the same name as the finder you declared on your bean's home. Then add any required parameters.
7. Select and configure the finder.

The following is an example of a simple EJB QL query that requires one parameter. In this example, the question mark ("?") in`?1` specifies a parameter.

```
SELECT OBJECT(employee) FROM Employee employee WHERE (employee.name =?1)
```

ReadAll Query and EJB QL

To execute a query normally, you supply either a reference class or a `SELECT` clause.

The basic API for a `ReadAll` query with EJB QL is:

```
ReadAllQuery setEJBQLString("...")
```

Example 6–80 ReadAllQuery Using EJB QL

```
ReadAllQuery theQuery = new ReadAllQuery();
theQuery.setReferenceClass(EmployeeBean.class);
theQuery.setEJBQLString("SELECT OBJECT(emp) FROM EmployeeBean emp");
…
Vector returnedObjects = (Vector)aSession.executeQuery(theQuery);
```

Example 6–81 ReadAllQuery Using EJB QL and Passing Arguments

This code creates, populates, and passes a vector of arguments into the `executeQuery` method

```
// First define the query
ReadAllQuery theQuery = new ReadAllQuery();
theQuery.setReferenceClass(EmployeeBean.class);
theQuery.setEJBQLString("SELECT OBJECT(emp) FROM EmployeeBean emp WHERE
emp.firstName = ?1");
theQuery.addArgument("1");
...
// Next define the Arguments
Vector theArguments = new Vector();
theArguments.add("Bob");
...
// Finally execute the query passing in the arguments
Vector returnedObjects = (Vector)aSession.executeQuery(theQuery, theArguments);
```

EJB QL Session Queries

When you execute EJB QL directly against the session, it returns a vector of the objects specified by the reference class. The basic API is as follows:

```
aSession.readAllObjects(<ReferenceClass>, <EJBQLCall>)
```

Example 6–82 EJB QL Session Query

```
/* <EJBQLCall> is the EJBQL string to be executed and <ReferenceClass> is the
return class type */
```

```
// Call ReadAllObjects on a session.
Vector theObjects = (Vector)aSession.readAllObjects(EmployeeBean.class, new
EJBQLCall( "SELECT OBJECT (emp) from EmployeeBean emp));
```

SQL Finders

You can use custom SQL code to specify finder logic. SQL enables you to implement logic that might not be possible to express with OracleAS TopLink expressions or EJB QL.

To create a SQL finder:

1. Declare the finder in the `ejb-jar.xml` file, and leave the `ejb-ql` tag empty.
2. Start the OracleAS TopLink Mapping Workbench.
3. Specify the `ejb-jar.xml` file location and choose **File** > **Updated Project** from the `ejb-jar.xml` file to read in the finders.
4. Go the **Queries** > **Named Queries** tab for the bean.
5. Select the finder, check the SQL radio button, and enter the SQL string.
6. Configure the finder.

The following is an example of a simple SQL finder that requires one parameter. In this example, the hash character, '#', is used to bind the argument `projectName` within the SQL string.

```
SELECT * FROM EJB_PROJECT WHERE (PROJ_NAME = #projectName)
```

Dynamic Finders

OracleAS TopLink provides several predefined finders you can use to execute dynamic queries, in which the logic is determined by the user at runtime. The OracleAS TopLink runtime reserves the names for these finders; they cannot be reused for other finders.

The predefined finders are:

```
EJBObject findOneByEJBQL(String ejbql, Vector args)
Collection findManyByEJBQL(String ejbql, Vector args)
EJBObject findOneBySQL(String sql, Vector args)
Collection findManyBySQL(String sql, Vector args)
EJBObject findOneByQuery(DatabaseQuery query, Vector args)
```

```
Collection findManyByQuery(DatabaseQuery query, Vector args)
```

Note: With EJB 2.0, if the finder is located on a local home, replace `EJBObject` with `EJBLocalObject` in finders that contain `findOneby`.

You can also use each of these finders without a vector of arguments. For example, `EJBObject findOneByEJBQL(String ejbql)` is a valid dynamic finder, but you must replace the return type of `EJBObject` with your bean's component interface.

To create a dynamic finder:

1. Declare the finder in the `ejb-jar.xml` file, and leave the `ejb-ql` tag empty.
2. Declare the finder on the `Home` interface, the `localHome` interface, or both, as required.
3. Start the OracleAS TopLink Mapping Workbench.
4. Specify the `ejb-jar.xml` file location and choose **File** > **Updated** Project from the `ejb-jar.xml` file to read in the finders.
5. Go to the **Queries** > **Named Queries** tab for the bean.
6. Select and configure the finder.

Notes: If the advanced query options in "Advanced Finder Options" on page 6-96 are not required, you need only complete steps 1 and 2.

Do not configure any query options for the `findOneByQuery` and `findManyByQuery` dynamic finders, because the client creates the query at runtime and passes it as a parameter to the finder. Set any required system options on that query.

Using findAll In addition to the preceding dynamic finder, OracleAS TopLink provides a default `findAll` query that returns all the beans of a given type. As with other dynamic finders, the OracleAS TopLink runtime reserves the name `findAll`.

For more information about defining and configuring the finder, see "To create a dynamic finder:" on page 6-93.

Using findByPrimaryKey OracleAS TopLink creates the `findByPrimaryKey` finder to a bean class when the class initializes. You can configure the `findByPrimaryKey` finder with the various OracleAS TopLink query options.

Because the EJB 2.0 specification requires the container to implement the `findByPrimaryKey` call on each bean `Home` interface, do not delete this finder from a bean.

ReadAll Finders

ReadAll finders enable you to create dynamic queries that you generate at runtime rather than deployment time. To use a ReadAll finder, pass an OracleAS TopLink `ReadAllQuery` as a parameter to a finder that returns an `Enumeration`.

Creating READALL Finders

OracleAS TopLink provides an implementation for ReadAll finders. To use this feature in a bean, add the following finder definition to the `Home` interface of your bean.

```
public Enumeration findAll(ReadAllQuery query) throws RemoteException,
FinderException;
```

To execute a ReadAll finder, create the query on the client.

Example 6–83 A ReadAll Finder

```
{
    ReadAllQuery query = new ReadAllQuery(Employee.class);
    query.addJoinedAttribute("address");
    Enumeration employees = getEmployeeHome().findAll(query);
}
```

Choosing the Best Finder Type for Your Query

To optimize performance, choose the finder type that best suits your needs.

Using the OracleAS TopLink Expression Framework

Using OracleAS TopLink expressions offers the following advantages:

- Version controlled standardized queries to Java code
- Ability to simplify most complex operations
- A more complete set of querying features than is available through EJB QL

Because expressions enable you to specify finder search criteria based on the object model, they are frequently the best choice for constructing your finders.

For more information about implementing finders using OracleAS TopLink expressions, see "Expression Finders" on page 6-89.

Using Redirect Finders

Redirect finders enable you to implement a finder that is defined on an arbitrary helper class as a static method. When you invoke the finder, OracleAS TopLink redirects the call to the specified static method.

Redirect queries are complex and require an extra helper method to define the query. However, because they support complex logic, they are often the best choice when you need to implement logic unrelated to the bean on which the redirect method is called.

Using SQL

Using SQL to define a finder offers the following advantages:

- SQL enables you to implement logic that cannot be expressed when you use EJB QL or the OracleAS TopLink expression framework.
- It allows for the use of a stored procedure instead of OracleAS TopLink generated SQL.
- There may be cases in which custom SQL will improve performance.

SQL finders also have the following disadvantages:

- Writing complex custom SQL statements requires a significant maintenance effort if the database tables change.
- The hard coded SQL limits portability to other databases.
- No validation is performed on the SQL String. Errors in the SQL will not be detected until runtime.
- The use of SQL for a function other than `SELECT` may result in unpredictable errors.

ejbSelect

The ejbSelect method is a query method intended for internal use within an entity bean instance. Specified on the abstract bean itself, the `ejbSelect` method is not directly exposed to the client in the home or component interface. Defined as abstract, each bean can include zero or more such methods.

Select methods have the following characteristics:

- The method name must have `ejbSelect` as its prefix.
- It must be declared as public.
- It must be declared as abstract.
- The `throws` clause must specify the `javax.ejb.FinderException`, although it may also specify application-specific exceptions as well.
- Under EJB 2.0, the `result-type-mapping` tag in the `ejb-jar.xml` file determines the return type for `ejbSelects`. Set the flag to `Remote` to return `EJBObjects`; set it to `Local`, to return `EJBLocalObjects`.

The format for an `ejbSelect` method definition looks like this:

```
public abstract type ejbSelect<METHOD>(...);
```

The `ejbSelect` query return type is not restricted to the entity bean type on which the `ejbSelect` is invoked. Instead, it can return any type corresponding to a container-managed relationship or container-managed field.

Although the select method is not based on the identity of the entity bean instance on which it is invoked, it can use the primary key of an entity bean as an argument. This creates a query that is logically scoped to a particular entity bean instance.

To create an ejbSelect:

1. Declare the `ejbSelect` in the `ejb-jar.xml` file, enter the EJB QL string in the `ejb-ql` tag, and specify the return type in the `result-type-mapping` tag (if required).
2. Declare the `ejbSelect` on the abstract bean class.
3. Start the OracleAS TopLink Mapping Workbench.
4. Specify the `ejb-jar.xml` file location, and choose **File** > **Updated Project** from the `ejb-jar.xml` file to read in the finders.
5. Go the **Queries** > **Named Queries** tab for the bean.
6. Select and configure the `ejbSelect` query.

Advanced Finder Options

The default finder configuration is appropriate for most applications. However, finders also allow for several advanced configuration options.

Caching Options

You can apply various configurations to the underlying query to achieve the correct caching behavior for the application. There are several ways to control the caching options for queries. For most queries, you can set caching options with the OracleAS TopLink Mapping Workbench. For more information, see “Caching objects” in the *Oracle Application Server TopLink Mapping Workbench User’s Guide.*

You can set the caching options on a per-finder basis. Table 6–8 lists the valid values.

Table 6–8 Finder Caching Options

This Setting . . .	Causes Finders to . . .	When the Search Involves a Finder That . . .
`ConformResultsInUnitOfWork` (default)	Check the Unit of Work cache before querying the session cache or the database. The finder's results always conform to uncommitted new objects, deleted objects, and changed objects.	Returns either a single bean or a collection
`DoNotCheckCache`	Query the database, bypassing the OracleAS TopLink internal caches.	Returns either a single bean or a collection
`CheckCacheByExactPrimaryKey`	Check the session cache for the object.	Contains only a primary key, and returns a single bean
`CheckCacheByPrimaryKey`	Check the session cache for the object.	Contains a primary key (and may contain other search parameters), and returns a single bean
`CheckCacheThenDatabase`	Search the session cache before accessing the database	Returns a single bean
`CheckCacheOnly`	Search against the session cache, but not the database.	Returns either a single bean or a collection

For more information about the OracleAS TopLink queries as well as the OracleAS TopLink Unit of Work and how it integrates with JTS, see Chapter 7, "Transactions".

Note: To apply caching options to finders with manually created (`findOneByQuery`, `findManyByQuery`) queries, use the OracleAS TopLink API.

Disable Cache for Returned Finder Results

By default, OracleAS TopLink adds all returned objects to the session cache. However, if you know the set of returned objects is very large, and you want to avoid the expense of storing these objects, you can disable this behavior. To override the default configuration, implement the `dontMaintainCache()` call on the query, or disable returned object caching for the query in the OracleAS TopLink Mapping Workbench.

For more information about disabling caching for returned finder results, see the *Oracle Application Server TopLink Mapping Workbench User's Guide.*

Refreshing Finder Results

A finder may return information from the database for an object whose primary key is already in the cache. When set to true, the Refresh Cache option (in the OracleAS TopLink Mapping Workbench) causes the query to refresh the object's nonprimary key attributes with the returned information. This occurs on `findByPrimaryKey` finders as well as all expression and SQL finders for the bean.

If you build a query in Java code, you can set this option by including the `refreshIdentityMapResult()` method. This method automatically cascades changes to privately owned parts of the beans. If you require different behavior, configure the query using a dynamic finder instead.

Caution: When you invoke this option from within a transaction, the refresh overwrites object attributes, including any that have not yet been written to the database.

If your application includes an `OptimisticLock` field, use the refresh cache option in conjunction with the `onlyRefreshCacheIfNewerVersion()` option. This ensures that the application refreshes objects in the cache only if the version of the object in the database is newer than the version in the cache.

For finders that have no refresh cache setting, the `onlyRefreshCacheIfNewerVersion()` method has no effect.

Managing Large Result Sets with Cursored Streams

Large result sets can be resource intensive to collect and process. To give the client more control over the returned results, configure OracleAS TopLink finders to use cursors. This combines OracleAS TopLink's `CursoredStream` with the ability of the database to cursor data, and breaks up the result set into smaller, more manageable pieces.

The behavior of a finder including a cursored stream differs from other finder as follows:

- Only the elements requested by the client are sent to the client.
- Nothing is cached on the client in the `CursoredEnumerator`.
- If you use the transactional attribute REQUIRED for your entity bean, wrap all reads in a `UserTransaction begin()` and `commit()` to ensure that reads beyond the first page of the cursor have a transaction in which to work.

Building the Query You can configure any finder that returns a `java.util.Enumeration` (under EJB 1.1) or a `java.util.Collection` (under EJB 2.0) to use a cursor. When you create the query for the finder, add the `useCursoredStream()` option to enable cursoring.

Example 6–84 Cursored Stream in a Finder

```
ReadAllQuery raq = new ReadAllQuery();
ExpressionBuilder bldr = new ExpressionBuilder();
raq.setReferenceClass(ProjectBean.class);
raq.useCursoredStream();
raq.addArgument("projectName");
raq.setSelectionCriteria(bldr.get("name").
like(bldr.getParameter("projectName")));
descriptor.getQueryManager().addQuery ("findByNameCursored", query);
```

Executing the Finder from the Client in EJB 1.1 OracleAS TopLink offers additional elements for traversing finder results. These elements include:

- `hasMoreElements()`: Returns a boolean indicating whether there are any more elements in the result set.
- `nextElement()`: Returns the next available element.

- `nextElements(int count)`: Retrieves a `Vector` of at most **`count`** elements from the available results, depending on how many elements remain in the result set.
- `close()`: Closes the cursor on the server. The client must send this message, or the database connection does not close.

Example 6–85 illustrates client-code executing a cursored finder.

Example 6–85 Cursored Finder Under EJB 1.1

```
import oracle.toplink.ejb.cmpwls11. CursoredEnumerator;
//... other imports as necessary
getTransaction().begin();
CursoredEnumerator cursoredEnumerator = (CursoredEnumerator)getProjectHome()
 .findByNameCursored("proj%");

Vector projects = new Vector();
for (int index = 0; index < 50; i++) {
Project project = (Project)cursoredEnumerator.nextElement();
projects.addElement(project);
}
// Rest all at once ...
Vector projects2 = cursoredEnumerator.nextElements(50);
cursoredEnumerator.close();
getTransaction().commit();
```

Executing the Finder from the Client in EJB 2.0 As with EJB 1.1, OracleAS TopLink offers additional elements for traversing finder results under EJB 2.0. These elements include:

- `isEmpty()`: As with `java.util.Collection`, `isEmpty()` returns a boolean indicating whether the `Collection` is empty.
- `size()`: As with `java.util.Collection`, `size()` returns an integer indicating the number of elements in the `Collection`.
- `iterator()`: As with `java.util.Collection`, `iterator()` returns a `java.util.Iterator` for enumerating the elements in the `Collection`.

OracleAS TopLink also offers an extended protocol for `oracle.toplink.ejb.cmp.wls.CursoredIterator` (based on `java.util.Iterator`):

- `close()`: Closes the cursor on the server. The client must send this message to close the database connection.

- `hasNext()`: Returns a boolean indicating whether any more elements are in the result set.
- `next()`: Returns the next available element.
- `next(int count)`: Retrieves a `Vector` of at most count elements from the available results, depending on how many elements remain in the result set.

Example 6–86 illustrates client code executing a cursored finder.

Example 6–86 Cursored Finder Under EJB 2.0

```
//import both CursoredCollection and CursoredIterator
import oracle.toplink.ejb.cmp.wls.*;
//... other imports as necessary
getTransaction().begin();
CursoredIterator cursoredIterator = (CursoredIterator)
getProjectHome().findByNameCursored("proj%").iterator();
Vector projects = new Vector();
for (int index = 0; index < 50; i++) {
Project project = (Project)cursoredIterator.next();
projects.addElement(project);
}
// Rest all at once ...
Vector projects2 = cursoredIterator.next(50);
cursoredIterator.close();
getTransaction().commit();
```

Exception Handling

Most exceptions in queries are database exceptions, resulting from a failure in the database operation. Write operations can also throw an `OptimisticLockException` on a write, update, or delete operation in applications that use optimistic locking. To catch these exceptions, execute all database operations within a *try-catch* block.

```
{
    try {
        Vector employees = session.readAllObjects(Employee.class);
        }
    catch (DatabaseException exception) {
        // Handle exception
        }
}
```

For more information about exceptions in a OracleAS TopLink application, see Appendix C, "Error Codes and Messages".

7 Transactions

A database transaction is a set of operations (create, read, update, or delete) that either succeed or fail as a single operation. The database discards, or *rolls back*, unsuccessful transactions, leaving the database in its original state.

In OracleAS TopLink, transactions are encapsulated by the Unit of Work object. Using the Unit of Work, you can transactionally modify objects directly or by way of a Java 2 Enterprise Edition (J2EE) external transaction controller such as the Java Transaction API (JTA).

This chapter explains how to use the OracleAS TopLink Unit of Work, including:

- Introduction to Transaction Concepts
- Understanding the Unit of Work
- Unit of Work Basics
- Advanced Unit of Work
- J2EE Integration

Introduction to Transaction Concepts

This section describes generic database transaction concepts and how they apply to the OracleAS TopLink Unit of Work.

Database Transactions

Transactions execute in their own *context*, or logical space, isolated from other transactions and database operations.

The transaction context is *demarcated*; that is, it has a defined structure that includes:

- A *begin* point, where the operations within the transaction begin. At this point, the transaction begins to execute its operations.
- A *commit* point, where the operations are complete and the transaction attempts to formalize changes on the database.

The degree to which concurrent (parallel) transactions on the same data are allowed to interact is determined by the level of *transaction isolation* configured. ANSI/SQL defines four levels of database transaction isolation as shown in Table 7–1. Each offers a trade-off between performance and resistance from the following unwanted behaviors:

- dirty read: a transaction reads uncommitted data written by a concurrent transaction.
- non-repeatable read: a transaction re-reads data and finds it has been modified by some other transaction that committed after the initial read.
- phantom read: a transaction re-executes a query and the returned data has changed due to some other transaction that committed after the initial read.

Table 7–1 Transaction Isolation Levels

Transaction Isolation Level	Dirty Read	Non-repeatable Read	Phantom Read
Read Uncommitted	Yes	Yes	Yes
Read Committed	No	Yes	Yes
Repeatable Read	No	No	Yes
Serializeable	No	No	No

As a transaction is committed, the database maintains a log of all changes to the data. If all operations in the transaction succeed, the database allows the changes; if any part of the transaction fails, the database uses the log to roll back the changes.

OracleAS TopLink Unit of Work Transactions

In OracleAS TopLink, transactions are encapsulated by the Unit of Work object. Like any transaction, a Unit of Work transaction provides:

- Transaction Context
- Transaction Demarcation
- Transaction Isolation

Transaction Context

Unit of Work operations occur within a Unit of Work *context*, isolated from the database until commit time. The Unit of Work executes changes on copies, or *clones*, of objects in its own internal cache, and if successful, applies changes to objects in the database and the session cache.

Transaction Demarcation

If your application is a stand-alone OracleAS TopLink application, your application demarcates transactions using the Unit of Work.

If your application includes a J2EE container that provides container-managed transactions, you can configure OracleAS TopLink to integrate with the container's transaction demarcation. The Unit of Work supports:

- JTA Transaction Demarcation
- CMP Transaction Demarcation

JTA Transaction Demarcation J2EE containers use JTA to manage transactions in the application. If your application includes a J2EE container, the Unit of Work executes as part of an external JTA transaction. The Unit of Work still manages its own internal operations, but relies on the external transaction to commit changes to the database. The Unit of Work waits for the external transaction to commit successfully before writing changes back to the session cache.

Note that because the transaction happens outside of the Unit of Work context and is controlled by the JTA, errors can be more difficult to diagnose and fix.

For more information, see "J2EE Integration" on page 7-44.

CMP Transaction Demarcation Entity beans that use container-managed persistence can participate in either *client-demarcated* or *container-demarcated* transactions. They can demarcate transactions with the `javax.transaction.UserTransaction` interface. OracleAS TopLink automatically wraps invocations on entity beans in container transactions based on the *transaction attributes* in the EJB deployment descriptor. For more information about transactions with EJBs, see the EJB specification and your J2EE container documentation.

In transactions involving EJBs, OracleAS TopLink waits until the transaction begins its two-stage commit process before updating the database. This allows for:

- SQL optimizations that ensure only changed data is written to the data store
- Proper ordering of updates to allow for database constraints

Transaction Isolation

OracleAS TopLink `DatabaseLogin` API allows you to set the transaction isolation level used when you open a connection to a database:

```
databaseLogin.setTransactionIsolation(DatabaseLogin.TRANSACTION_SERIALIZABLE);
```

However, the Unit of Work does not participate in database transaction isolation. Because the Unit of Work may execute queries outside the database transaction, the database has no control over the data and its visibility outside the transaction.

To maintain transaction isolation, each Unit of Work instance operates on its own copy (clone) of affected objects (see "Clones and the Unit of Work" on page 7-7). Multiple reads to the same object return the same clone and the clone's state is from when it was first accessed (registered).

Optimistic locking, optimistic read locking, or pessimistic locking can be used to ensure concurrency (see "Locking Policy" on page 5-20).

The Unit of Work method `ShouldAlwaysConformResultsInUnitOfWork` allows querying to be done on object changes within a Unit of Work (see "Using Conforming Queries and Descriptors" on page 7-35).

Changes are committed to the database only when the Unit of Work `commit` method is called (either directly or by way of an external transaction controller.)

Understanding the Unit of Work

This section describes:

Unit of Work Benefits

The OracleAS TopLink Unit of Work simplifies transactions and improves transactional performance. It is the preferred method of writing to a database in OracleAS TopLink because it:

- sends a minimal amount of SQL to the database during the commit by updating only the exact changes down to the field level
- reduces database traffic by isolating transaction operations in their own memory space
- optimizes cache synchronization, in applications that use multiple caches, by passing change sets (rather than objects) between caches
- isolates object modifications in their own transaction space to allow parallel transactions on the same objects
- ensures referential integrity and minimizes deadlocks by automatically maintaining SQL ordering
- orders database inserts, updates, and deletes to maintain referential integrity for mapped objects
- resolves bidirectional references automatically
- frees the application from tracking or recording its changes
- simplifies persistence with *persistence by reachability* (see "Associations: New Source to Existing Target Object" on page 7-18)

Unit of Work Life Cycle

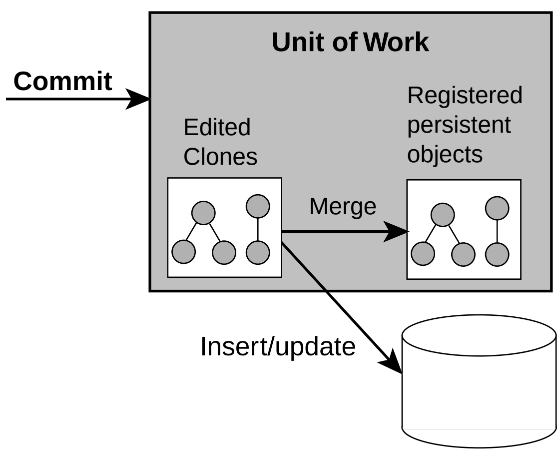
The Unit of Work is used as follows:

1. Client application acquires a Unit of Work from a session object.
2. Client application queries OracleAS TopLink to obtain the cache objects it wants to modify and then registers the cache objects with the Unit of Work.
3. When the first object is registered, the Unit of Work starts its transaction.

   As each object is registered, the Unit of Work accesses the object from the Session cache or database and creates a backup clone and working clone (see "Clones and the Unit of Work" on page 7-7).

   The Unit of Work returns the working clone to the client application.
4. Client application modifies the working clones.
5. Client application (or external transaction controller) commits the transaction (see "Commit and Rollback" on page 7-9).

Figure 7–1 The Life Cycle of a Unit of Work



Example 7–1 shows the life cycle in code.

Example 7–1 Unit of Work Life Cycle

```
// The application reads a set of objects from the database.
Vector employees = session.readAllObjects(Employee.class);

// The application specifies an employee to edit.
. . .
Employee employee = (Employee) employees.elementAt(index);

try {
    // Acquire a Unit of Work from the session.
    UnitOfWork uow = session.acquireUnitOfWork();
    // Register the object that is to be changed. Unit of Work returns a clone
    // of the object and makes a backup copy of the original employee
    Employee employeeClone = (Employee)uow.registerObject(employee);
    // We make changes to the employee clone by adding a new phoneNumber.
    // If a new object is referred to by a clone, it does not have to be
    // registered. Unit of Work determines it is a new object at commit time.
    PhoneNumber newPhoneNumber = new PhoneNumber("cell","212","765-9002");
    employeeClone.addPhoneNumber(newPhoneNumber);
    // We commit the transaction: Unit of Work compares the employeeClone with
    // the backup copy of the employee, begins a transaction, and updates the
    // database with the changes. If successful, the transaction is committed
    // and the changes in employeeClone are merged into employee. If there is an
    // error updating the database, the transaction is rolled back and the
    // changes are not merged into the original employee object.
    uow.commit();
} catch (DatabaseException ex) {
    // If the commit fails, the database is not changed. The Unit of Work should
    // be thrown away and application-specific action taken.
}
// After the commit, the Unit of Work is no longer valid. Do not use further.
```

Clones and the Unit of Work

The Unit of Work maintains two copies of the original objects registered with it:

- working clones
- backup clones

After you change the working clones and the transaction is committed, the Unit of Work compares the working copy clones to the backup copy clones, and writes any changes to the database. The Unit of Work uses clones to allow parallel Units of Work (see "Nested and Parallel Units of Work" on page 7-8) to exist, a requirement in multi-user three-tier applications.

The OracleAS TopLink cloning process is efficient in that it clones only the mapped attributes of registered objects, and stops at indirection objects unless you trigger the indirection. For more information, see "Indirection" on page 3-27.

You can customize the cloning process using the descriptor's copy policy. For more information, see "Descriptor Copy Policy" on page 3-87.

Never use a clone after committing the Unit of Work that the clone is from (even if the transaction fails and rolls-back). A clone is a working copy used during a transaction and as soon as the transaction is committed (successful or not), the clone must not be used. Accessing an uninstantiated clone value holder after a Unit of Work commit will raise an exception. The only time you can use a clone after a successful commit is when you use the advanced API described in "Resuming a Unit of Work After Commit" on page 7-39.

Nested and Parallel Units of Work

You can use OracleAS TopLink to create a:

- Nested Unit of Work
- Parallel Unit of Work

For information and examples on using nested and parallel Units of Work, see "Using a Nested or Parallel Unit of Work" on page 7-40.

Nested Unit of Work

You can nest a Unit of Work (the *child*) within another Unit of Work (the *parent*). A nested Unit of Work does not commit changes to the database. Instead, it passes its changes to the parent Unit of Work, and the parent attempts to commit the changes at commit time. Nesting Units of Work enables you to break a large transaction into smaller isolated transactions, and ensures that:

- Changes from each nested Unit of Work commit or fail as a group.
- Failure of a nested Unit of Work does not affect the commit or rollback of other operations in the parent Unit of Work.
- Changes are presented to the database as a single transaction.

Parallel Unit of Work

You can modify the same objects in multiple Unit of Work instances in parallel because the Unit of Work manipulates copies of objects. OracleAS TopLink resolves any concurrency issues when the Units of Work commit.

Reading and Querying Objects with the Unit of Work

A Unit of Work is a `Session`, and as such, offers the same set of database access methods as a regular session.

When called from a Unit of Work, these methods access the objects in the Unit of Work, register the selected objects automatically, and return clones.

Although this makes it unnecessary for you to call the `registerObject` and `registerAllObjects` methods, be aware of the restrictions on registering objects described in "Creating an Object" on page 7-13 and "Associations: New Source to Existing Target Object" on page 7-18.

Reading Objects with the Unit of Work

As with regular sessions, you use the `readObject` and `readAllObjects` methods to read objects from the database.

Querying Objects with the Unit of Work

You can execute queries in a Unit of Work with the `executeQuery` method.

Note: Because a Unit of Work manages changes to existing objects and the creation of new objects, modifying queries such as `InsertObjectQuery` or `UpdateObjectQuery` are not necessary and therefore are not supported by the Unit of Work.

Commit and Rollback

When a Unit of Work transaction is committed, it either succeeds or fails and rolls back. A commit can be initiated by your application or a J2EE container.

Commit

At commit time, the Unit of Work compares the working clones and backup clones to calculate the change set (that is, to determine the minimum changes required). Changes include updates to or deletion of existing objects, and the creation of new objects. The Unit of Work then begins a database transaction, and attempts to write the changes to the database. If all changes commit successfully on the database, the Unit of Work merges the changed objects into the session cache. If any of the changes fail on the database, the Unit of Work rolls back any changes on the database, and does not merge changes into the session cache.

The Unit of Work calculates commit order using foreign key information from one-to-one and one-to-many mappings. If you encounter constraint problems during commit, verify your mapping definitions. The order in which you register objects with the `registerObject` method does not affect the commit order.

Commit and JTA When your application uses JTA, the Unit of Work commit behaves differently than in a non-JTA application. In most cases, the Unit of Work attaches itself to an external transaction. If no transaction exists, the Unit of Work creates a transaction. This distinction affects commit behavior as follows:

- *If the Unit of Work attaches to an existing transaction*, the Unit of Work ignores the `commit` call. The transaction commits the Unit of Work when the entire external transaction is complete.
- *If the Unit of Work starts the external transaction*, the transaction treats the Unit of Work `commit` call as a request to commit the external transaction. The external transaction then calls its own commit code on the database.

In either case, only the external transaction can call `commit` on the database because it owns the database connection.

For more information, see "J2EE Integration" on page 7-44.

Rollback

A Unit of Work commit must succeed or fail as a unit. Failure in writing changes to the database causes the Unit of Work to roll back the database to its previous state. Nothing changes in the database, and the Unit of Work does not merge changes into the session cache.

Rollback and JTA In a JTA environment, the Unit of Work does not own the database connection. In this case, the Unit of Work sends the rollback call to the external transaction rather than the database, and the external transaction treats the rollback call as a request to roll the transaction back.

For more information, see "J2EE Integration" on page 7-44.

Primary Keys

You cannot modify the primary key attribute of an object in a Unit of Work. This is an unsupported operation and doing so will result in unexpected behaviour (exceptions and/or database corruption).

To replace one instance of an object with unique constraints with another, see "Using the Unit of Work setShouldPerformDeletesFirst Method" on page 7-42.

Example Object Model and Schema

Throughout this chapter, the following object model and schema is used in the examples provided. The example object model appears in Figure 7–2 and the example entity-relationship (data model) diagram appears in Figure 7–3.

Figure 7–2 Example Object Model

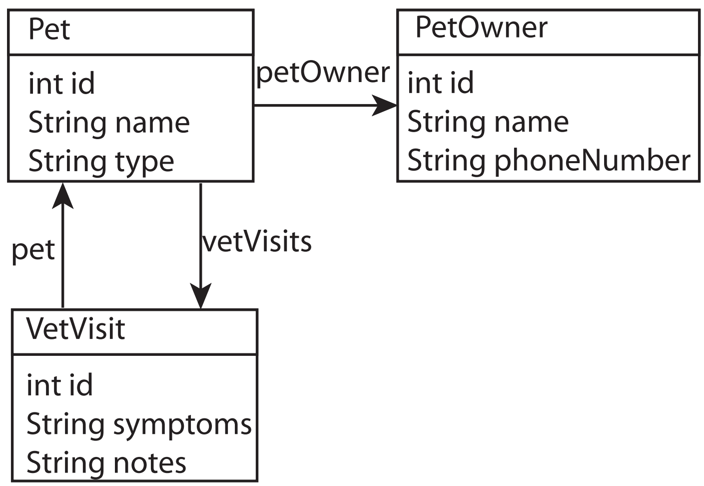
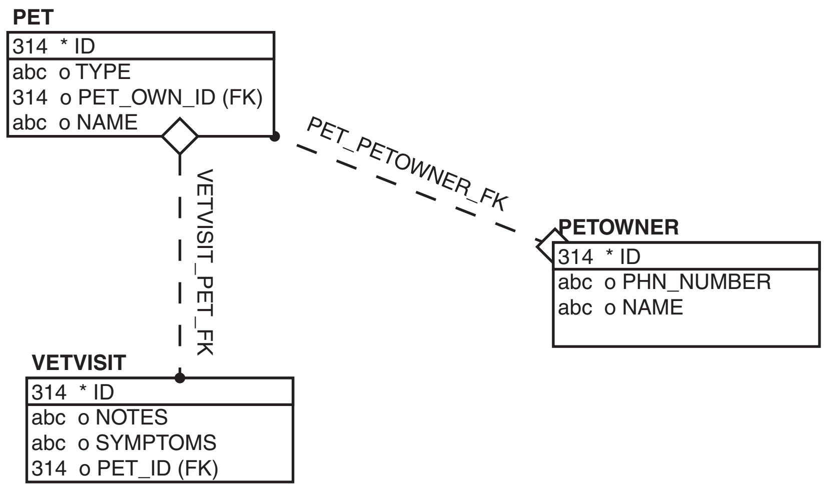


Figure 7–3 Example Data Model



Unit of Work Basics

This section explores the essential Unit of Work API calls most commonly used throughout the development cycle:

- Acquiring a Unit of Work
- Creating an Object
- Modifying an Object
- Associations: New Target to Existing Source Object
- Associations: New Source to Existing Target Object
- Associations: Existing Source to Existing Target Object
- Deleting Objects

For more information about the available methods for the `UnitOfWork`, see "Advanced Unit of Work" on page 7-23 and the *Oracle Application Server TopLink API Reference*.

Acquiring a Unit of Work

This example shows how to acquire a Unit of Work from a client session object.

```
Server server =
    (Server) SessionManager.getManager().getSession(
        sessionName, MyServerSession.class.getClassLoader()
    );
Session session = (Session) server.acquireClientSession();
UnitOfWork uow = session.acquireUnitOfWork();
```

You can acquire a Unit of Work from any session type. Note that you do not need to create a new session and login before every transaction.

The Unit of Work is valid until the `commit` or `release` method is called. After a commit or release, a Unit of Work is not valid even if the transaction fails and is rolled back.

A Unit of Work remains valid after the `commitAndResume` method is called as described in "Resuming a Unit of Work After Commit" on page 7-39.

When using a Unit of Work with JTA, you can also use the advanced API `getActiveUnitOfWork` method as described in "J2EE Integration" on page 7-44.

Creating an Object

When you create new objects in the Unit of Work, use the `registerObject` method to ensure that the Unit of Work writes the objects to the database at commit time.

The Unit of Work calculates commit order using foreign key information from one-to-one and one-to-many mappings. If you encounter constraint problems during commit, verify your mapping definitions. The order in which you register objects with the `registerObject` method does not affect the commit order.

Example 7–2 and Example 7–3 show how to create and persist a simple object (without relationships) using the clone returned by the Unit of Work `registerObject` method.

Example 7–2 Creating an Object: Preferred Method

```
UnitOfWork uow = session.acquireUnitOfWork();
    Pet pet = new Pet();
    Pet petClone = (Pet)uow.registerObject(pet);
    petClone.setId(100);
    petClone.setName("Fluffy");
    petClone.setType("Cat");
```

```
uow.commit();
```

Example 7–3 shows a common alternative:

Example 7–3 Creating an Object: Alternative Method

```
UnitOfWork uow = session.acquireUnitOfWork();
    Pet pet = new Pet();
    pet.setId(100);
    pet.setName("Fluffy");
    pet.setType("Cat");
    uow.registerObject(pet);
uow.commit();
```

Both approaches produce the following SQL:

```
INSERT INTO PET (ID, NAME, TYPE, PET_OWN_ID) VALUES (100, 'Fluffy', 'Cat', NULL)
```

Example 7–2 is preferred: it gets you into the pattern of working with clones and provides the most flexibility for future code changes. Working with combinations of new objects and clones can lead to confusion and unwanted results.

Modifying an Object

In Example 7–4, a `Pet` is read prior to a Unit of Work: the variable `pet` is the cache copy for that `Pet`. Inside of the Unit of Work, we must register the cache copy to get a working copy. We then modify the working copy and commit the Unit of Work.

Example 7–4 Modifying an Object

```
// Read in any pet.
Pet pet = (Pet)session.readObject(Pet.class);
UnitOfWork uow = session.acquireUnitOfWork();
    Pet petClone = (Pet) uow.registerObject(pet);
    petClone.setName("Furry");
uow.commit();
```

In Example 7–5, we take advantage of the fact that you can query through a Unit of Work and get back clones, saving the registration step. However, the drawback is that we do not have a handle to the cache copy.

If we wanted to do something with the updated `Pet` after commit, we would have to query the session to get it (remember that after a Unit of Work is committed, its clones are invalid and must not be used).

Example 7–5 Modifying an Object: Skipping the Registration Step

```
UnitOfWork uow = session.acquireUnitOfWork();
    Pet petClone = (Pet) uow.readObject(Pet.class);
    petClone.setName("Furry");
uow.commit();
```

Both approaches produce the following SQL:

```
UPDATE PET SET NAME = 'Furry' WHERE (ID = 100)
```

Take care when querying through a Unit of Work. All objects read in the query are registered in the Unit of Work and therefore will be checked for changes at commit time. Rather than do a `ReadAllQuery` through a Unit of Work, it is better for performance to design your application to do the `ReadAllQuery` through a session and then only register in a Unit of Work the objects that need to be changed.

Associations: New Target to Existing Source Object

There are two ways to associate a new target object with an existing source object with 1-many and 1-1 relationships:

- Associating without Reference to the Cache Object
- Associating with Reference to the Cache Object

Deciding which approach to use depends on whether or not your code requires a reference to the cache copy of the new object after the Unit of Work is committed and on how adaptable to change you want your code to be.

Associating without Reference to the Cache Object

Example 7–6 shows the first way of associating a new target with an existing source.

Example 7–6 Associating without Reference to the Cache Object

```
UnitOfWork uow = session.acquireUnitOfWork();
    Pet petClone = (Pet)uow.readObject(Pet.class);

    PetOwner petOwner = new PetOwner();
    petOwner.setId(400);
    petOwner.setName("Donald Smith");
    petOwner.setPhoneNumber("555-1212");

    VetVisit vetVisit = new VetVisit();
    vetVisit.setId(500);
```

```
    vetVisit.setNotes("Pet was shedding a lot.");
    vetVisit.setSymptoms("Pet in good health.");
    vetVisit.setPet(petClone);

    petClone.setPetOwner(petOwner);
    petClone.getVetVisits().addElement(vetVisit);
uow.commit();
```

This executes the proper SQL:

```
INSERT INTO PETOWNER (ID, NAME, PHN_NBR) VALUES (400, 'Donald Smith',
'555-1212')
UPDATE PET SET PET_OWN_ID = 400 WHERE (ID = 100)
INSERT INTO VETVISIT (ID, NOTES, SYMPTOMS, PET_ID) VALUES (500, 'Pet was
shedding a lot.', 'Pet in good health.', 100)
```

When associating new objects to existing objects, the Unit of Work treats the new object as if it was a clone. That is, after the commit:

```
petOwner != session.readObject(petOwner)
```

For a more detailed discussion of this fact, see "Using registerNewObject" on page 7-28).

Therefore, after the Unit of Work commit, the variables `vetVisit` and `petOwner` no longer point to their respective cache objects: they point at working copy clones.

If you need the cache object after the Unit of Work commit, you must query for it or create the association with a reference to the cache object (as described in "Associating with Reference to the Cache Object" on page 7-16).

Associating with Reference to the Cache Object

Example 7–7 shows how to associate a new target with an existing source with reference to the cache object.

Example 7–7 Associating with Reference to the Cache Object

```
UnitOfWork uow = session.acquireUnitOfWork();
    Pet petClone = (Pet)uow.readObject(Pet.class);

    PetOwner petOwner = new PetOwner();
    PetOwner petOwnerClone = (PetOwner)uow.registerObject(petOwner);
    petOwnerClone.setId(400);
    petOwnerClone.setName("Donald Smith");
    petOwnerClone.setPhoneNumber("555-1212");
```

```
    VetVisit vetVisit = new VetVisit();
    VetVisit vetVisitClone = (VetVisit)uow.registerObject(vetVisit);
    vetVisitClone.setId(500);
    vetVisitClone.setNotes("Pet was shedding a lot.");
    vetVisitClone.setSymptoms("Pet in good health.");
    vetVisitClone.setPet(petClone);

    petClone.setPetOwner(petOwnerClone);
    petClone.getVetVisits().addElement(vetVisitClone);
uow.commit();
```

Now, after the Unit of Work commit:

```
petOwner == session.readObject(petOwner)
```

This means that we have a handle to the cache copy after the commit, rather than a clone.

Example 7–8 shows another way to add a new object in a Unit of Work when a bidirectional relationship exists.

Example 7–8 Resolving Issues When Adding New Objects

```
// Get an employee read from the parent session of the Unit of Work.
Employee manager = (Employee)session.readObject(Employee.class);

// Acquire a Unit of Work.
UnitOfWork uow = session.acquireUnitOfWork();

// Register the manager to get its clone
Employee managerClone = (Employee)uow.registerObject(manager);

// Create a new employee
Employee newEmployee = new Employee();
newEmployee.setFirstName("Spike");
newEmployee.setLastName("Robertson");

/* INCORRECT: Do not associate the new employee with the original manager. This
will cause a QueryException when OracleAS TopLink detects this error during
commit. */
//newEmployee.setManager(manager);

/* CORRECT: Associate the new object with the clone. Note that in this example,
the setManager method is maintaining the bidirectional managedEmployees
```

```
relationship and adding the new employee to its managedEmployees. At commit
time, the Unit of Work will detect that this is a new object and will take the
appropriate action. */
newEmployee.setManager(managerClone);

/* INCORRECT: Do not register the newEmployee: this will create two copies and
cause a QueryException when OracleAS TopLink detects this error during commit.*/
//uow.registerObject(newEmployee);

/* CORRECT:
In the above setManager call, if the managerClone's managedEmployees was not
maintained by the setManager method, then you should call registerObject before
the new employee is related to the manager. If in doubt, you could use the
registerNewObject method to ensure that the newEmployee is registered in the
Unit of Work. The registerNewObject method registers the object, but does not
make a clone. */
uow.registerNewObject(newEmployee);

// Commit the Unit of Work
uow.commit();
```

Associations: New Source to Existing Target Object

This section describes how to associate a new source object with an existing target object with 1-many and 1-1 relationships.

OracleAS TopLink follows all relationships of all registered objects (deeply) in a Unit of Work to calculate what is new and what has changed. This is known as *persistence by reachablity*. In "Associations: New Target to Existing Source Object" on page 7-15, we saw that when you associate a new target with an existing source, you can choose to register the object or not. If you do not register the new object, it is still reachable from the source object (which is a clone, hence it is registered). However, when you need to associate a new source object with an existing target, you must register the new object. If you do not register the new object, then it is not reachable in the Unit of Work and OracleAS TopLink will not write it to the database.

For example, imagine we want to create a new `Pet` and associate it with an existing `PetOwner`. The code shown in Example 7–9 will accomplish this:

Example 7–9 Associating a New Source to an Existing Target Object

```
UnitOfWork uow = session.acquireUnitOfWork();
    PetOwner existingPetOwnerClone =
        (PetOwner)uow.readObject(PetOwner.class);
```

```
    Pet newPet = new Pet();
    Pet newPetClone = (Pet)uow.registerObject(newPet);
    newPetClone.setId(900);
    newPetClone.setType("Lizzard");
    newPetClone.setName("Larry");
    newPetClone.setPetOwner(existingPetOwnerClone);
uow.commit();
```

This generates the proper SQL:

```
INSERT INTO PET (ID, NAME, TYPE, PET_OWN_ID) VALUES (900, 'Larry', 'Lizzard',
400)
```

In this situation, you should register the new object and work with the working copy of the new object. If you associate the new object with the `PetOwner` clone without registering, it will not be written to the database. If you are in a situation where you want to associate the `PetOwner` clone with the new `Pet` object, use the advanced API `registerNewObject` as described in "Using registerNewObject" on page 7-28.

If you fail to register the clone and accidentally associate the cache version of the existing object with the new object, then OracleAS TopLink will generate an error which states that you have associated the cache version of an object ("from a parent session") with a clone from this Unit of Work. You must work with working copies in units of work.

Associations: Existing Source to Existing Target Object

This section explains how to associate an existing source object with an existing target object with 1-many and 1-1 relationships.

As shown in Example 7–10, associating existing objects with each other in a Unit of Work is as simple as associating objects in Java. Just remember to only work with working copies of the objects.

Example 7–10 Associating an Existing Source to Existing Target Object

```
// Associate all VetVisits in the database to a Pet from the database
UnitOfWork uow = session.acquireUnitOfWork();
    Pet existingPetClone = (Pet)uow.readObject(Pet.class);
    Vector allVetVisitClones;
    allVetVisitClones = (Vector)uow.readAllObjects(VetVisit.class);
    Enumeration enum = allVetVisitClones.elements();
    while(enum.hasMoreElements()) {
```

```
        VetVisit vetVisitClone =(VetVisit)enum.nextElement();
        existingPetClone.getVetVisits().addElement(vetVisitClone);
        vetVisitClone.setPet(existingPetClone);
    };
uow.commit();
```

The most common error when associating existing objects is failing to work with the working copies. If you accidentally associate a cache version of an object with a working copy you will get an error at commit time indicating that you associated an object from a parent session (the cache version) with a clone from this Unit of Work.

Example 7–11 shows another example of associating an existing source to an existing target object.

Example 7–11 Associating Existing Objects

```
// Get an employee read from the parent session of the Unit of Work.
Employee employee = (Employee)session.readObject(Employee.class)

// Acquire a Unit of Work.
UnitOfWork uow = session.acquireUnitOfWork();
Project project = (Project) uow.readObject(Project.class);

/* When associating an existing object (read from the session) with a clone, we
must make sure we register the existing object and assign its clone into a Unit
of Work. */

/* INCORRECT: Cannot associate an existing object with a Unit of Work clone. A
QueryException will be thrown. */
//project.setTeamLeader(employee);

/* CORRECT: Instead register the existing object then associate the clone. */
Employee employeeClone = (Employee)uow.registerObject(employee);
project.setTeamLeader(employeeClone);
uow.commit();
```

Deleting Objects

To delete objects in a Unit of Work, use the `deleteObject` or `deleteAllObjects` method. When you delete an object that is not already registered in the Unit of Work, the Unit of Work registers the object automatically.

When you delete an object, OracleAS TopLink deletes the object’s privately owned parts, because those parts cannot exist without the owning object. At commit time, the Unit of Work generates SQL to delete the objects, taking database constraints into account.

When you delete an object, you must take your object model into account. You may need to set references to the deleted object to null (for an example, see "Using privateOwnedRelationship" on page 7-21).

This section explains how to delete objects with a Unit of Work, including:

- Using privateOwnedRelationship
- Explicitly Deleting from the Database
- Understanding the Order in which Objects are Deleted

Using privateOwnedRelationship

Relational databases do not have garbage collection like a Java Virtual Machine (JVM) does. To delete an object in Java you just de-reference the object. To delete a row in a relational database you must explicitly delete it. Rather than tediously manage when to delete data in the relational database, use the mapping attribute `privateOwnedRelationship` to make OracleAS TopLink manage the garbage collection in the relational database for you.

As shown in Example 7–12, when you create a mapping using Java, use its `privateOwnedRelationship` method to tell OracleAS TopLink that the referenced object is privately owned: that is, the referenced object cannot exist without the parent object.

Example 7–12 Specifying a Mapping as Privately Owned

```
OneToOneMapping petOwnerMapping = new OneToOneMapping();
petOwnerMapping.setAttributeName("petOwner");
petOwnerMapping.setReferenceClass(com.top.uowprimer.model.PetOwner.class);
petOwnerMapping.privateOwnedRelationship();
petOwnerMapping.addForeignKeyFieldName("PET.PET_OWN_ID", "PETOWNER.ID");
descriptor.addMapping(petOwnerMapping);
```

When you create a mapping using the Mapping Workbench, you can select the **Private Owned** check box under the **General** tab.

When you tell OracleAS TopLink that a relationship is private owned, you are telling it two things:

- if the source of a private owned relationship is deleted, then delete the target.
- if you de-reference a target from a source, then delete the target.

Do not configure private owned relationships to objects that might be shared. An object should not be the target in more than one relationship if it is the target in a private owned relationship.

The exception to this rule is the case when you have a many-to-many relationship in which a relation object is mapped to a relation table and is referenced through a one-to-many relationship by both the source and target. In this case, if the one-to-many mapping is configured as privately owned, then when you delete the source, all the association objects will be deleted.

Consider the example shown in Example 7–13.

Example 7–13 Private Owned Relationships

```
// If the Pet-PetOwner relationship is privateOwned
// then the PetOwner will be deleted at uow.commit()
// otherwise, just the foreign key from PET to PETOWNER will
// be set to null.  The same is true for VetVisit.
UnitOfWork uow = session.acquireUnitOfWork();
    Pet petClone = (Pet)uow.readObject(Pet.class);
    petClone.setPetOwner(null);
    VetVisit vvClone =
        (VetVisit)petClone.getVetVisits().firstElement();
    vvClone.setPet(null);
    petClone.getVetVisits().removeElement(vvClone);
uow.commit();
```

If the relationships from `Pet` to `PetOwner` and from `Pet` to `VetVisit` are not private owned, this code produces the following SQL:

```
UPDATE PET SET PET_OWN_ID = NULL WHERE (ID = 150)
UPDATE VETVISIT SET PET_ID = NULL WHERE (ID = 350)
```

If the relationships are private owned, this code produces the following SQL:

```
UPDATE PET SET PET_OWN_ID = NULL WHERE (ID = 150)
UPDATE VETVISIT SET PET_ID = NULL WHERE (ID = 350)
DELETE FROM VETVISIT WHERE (ID = 350)
DELETE FROM PETOWNER WHERE (ID = 250)
```

Explicitly Deleting from the Database

If there are cases where you have objects that will not be garbage collected through private owned relationships (especially root objects in your object model) then you can explicitly tell OracleAS TopLink to delete the row representing the object using the `deleteObject` API. For example:

Example 7–14 Explicitly Deleting

```
UnitOfWork uow = session.acquireUnitOfWork();
    pet petClone = (Pet)uow.readObject(Pet.class);
    uow.deleteObject(petClone);
uow.commit();
```

The above code generates the following SQL:

```
DELETE FROM PET WHERE (ID = 100)
```

Understanding the Order in which Objects are Deleted

The Unit of Work does not track changes or the order of operations. It is intended to isolate you from having to modify your objects in the order the database requires.

By default, at commit time, the Unit of Work orders all inserts and updates using the constraints defined by your schema. After all inserts and updates are done, the Unit of Work will issue the necessary delete operations.

Constraints are inferred from one-to-one and one-to-many mappings. If you have no such mappings, you can add additional constraint knowledge to OracleAS TopLink as described in "Controlling the Order of Deletes" on page 7-42.

Advanced Unit of Work

This section explores more advanced Unit of Work API calls and techniques most commonly used later in the development cycle, including:

- Troubleshooting a Unit of Work
- Creating and Registering an Object in One Step
- Using registerNewObject
- Using registerAllObjects
- Using Registration and Existence Checking
- Working with Aggregates
- Unregistering Working Clones
- Declaring Read-Only Classes
- Using Conforming Queries and Descriptors
- Merging Changes in Working Copy Clones

- Resuming a Unit of Work After Commit
- Reverting a Unit of Work
- Using a Nested or Parallel Unit of Work
- Using a Unit of Work with Custom SQL
- Validating a Unit of Work
- Controlling the Order of Deletes
- Improving Unit of Work Performance

For more information about integrating the Unit of Work with J2EE and external transaction controllers, see "J2EE Integration" on page 7-44.

For more information about the available methods for the `UnitOfWork`, see the *Oracle Application Server TopLink API Reference*.

Troubleshooting a Unit of Work

This section examines common Unit of Work problems and debugging techniques, including:

- Avoiding the Use of Post-commit Clones
- Determining Whether or not an Object is the Cache Object
- Dumping the Contents of a Unit of Work
- Handling Exceptions

Avoiding the Use of Post-commit Clones

A common Unit of Work error is holding on to clones after commit. Typically the clones are stored in a static variable and the developer incorrectly thinks that this object is the cache copy. This leads to problems when another Unit of Work makes changes to the object and what the developer thinks is the cache copy is not updated (because a Unit of Work only updates the cache copy, not old clones).

Consider the error in Example 7–15. In this example we get a handle to the cache copy of a Pet and store it in the static CACHE_PET. We get a handle to a working copy and store it in the static CLONE_PET. In a future Unit of Work, the Pet is changed.

Developers who incorrectly store global references to clones from units of work often expect them to be updated when the cache object is changed in a future Unit of Work. Only the cache copy is updated.

Example 7–15 Incorrect Use of Handle to Clone

```
//Read a Pet from the database, store in static
CACHE_PET = (Pet)session.readObject(Pet.class);

//Put a clone in a static.  This is a bad idea and is a common error
UnitOfWork uow = session.acquireUnitOfWork();
    CLONE_PET = (Pet)uow.readObject(Pet.class);
    CLONE_PET.setName("Hairy");
uow.commit();
//Later, the pet is changed again
UnitOfWork anotherUow = session.acquireUnitOfWork();
    Pet petClone = (Pet)anotherUow.registerObject(CACHE_PET);
    petClone.setName("Fuzzy");
anotherUow.commit();

// If you incorrectly stored the clone in a static and thought it should be
// updated when it's later changed, you would be wrong: only the cache copy is
// updated; NOT OLD CLONES.
System.out.println("CACHE_PET is" + CACHE_PET);
System.out.println("CLONE_PET is" + CLONE_PET);
```

The two System.out calls produce the following output:

```
CACHE_PET isPet type Cat named Fuzzy id:100
CLONE_PET isPet type Cat named Hairy id:100
```

Determining Whether or not an Object is the Cache Object

In "Modifying an Object" on page 7-14, we noted that it is possible to read any particular instance of a class by executing:

```
session.readObject(Class);
```

There is also a `readObject` method that takes an object as an argument: this method is equivalent to doing a `ReadObjectQuery` on the primary key of the object passed in. For example, the following:

```
session.readObject(pet);
```

Is equivalent to the following:

```
ReadObjectQuery query = new ReadObjectQuery();
query.setReferenceClass(Pet.class);
ExpressionBuilder builder = new ExpressionBuilder();
Expression exp = builder.get("id").equal(pet.getId());
query.setSelectionCriteria(exp);
```

```
session.executeQuery(query);
```

Also note that primary key based queries, by default, will return what is in the cache without going to the database.

Given this, we have a very quick and simple method for accessing the cache copy of an object as shown in Example 7–16.

Example 7–16 Testing if an Object is the Cache Object

```
//Here is a test to see if an object is the cache copy
boolean cached = CACHE_PET == session.readObject(CACHE_PET);
boolean cloned = CLONE_PET == session.readObject(CLONE_PET);
System.out.println("Is CACHE_PET the Cache copy of the object: " + cached);
System.out.println("Is CLONE_PET the Cache copy of the object: " + cloned);
```

This code produces the following output:

```
Is CACHE_PET the Cache copy of the object: true
Is CLONE_PET the Cache copy of the object: false
```

Dumping the Contents of a Unit of Work

The Unit of Work has several debugging methods to help you analyze performance or track down problems with your code. The most useful is `printRegisteredObjects` which prints all the information about objects known in the Unit of Work. Use this method to see how many objects are registered and to make sure objects you are working on are registered.

To use this method, you must have log messages enabled for the session that the Unit of Work is from. Session log messages are disabled by default. To enable log messages, use the session `logMessages` method. To disable log messages, use the session `dontLogMessages` method as shown in Example 7–17.

Example 7–17 Dumping the Contents of a Unit of Work

```
session.logMessages(); // enable log messages
UnitOfWork uow = session.acquireUnitOfWork();
    Pet petClone = (Pet)uow.readObject(Pet.class);
    petClone.setName("Mop Top");

    Pet pet2 = new Pet();
    pet2.setId(200);
    pet2.setName("Sparky");
    pet2.setType("Dog");
    uow.registerObject(pet2);
```

```
    uow.printRegisteredObjects();
uow.commit();
session.dontLogMessages(); // disable log messages
```

This example produces the following output:

```
UnitOfWork identity hashcode: 32373
Deleted Objects:

All Registered Clones:
    Key: [100] Identity Hash Code:13901  Object: Pet type Cat named Mop Top
id:100
    Key: [200] Identity Hash Code:16010  Object: Pet type Dog named Sparky
id:200

New Objects:
    Key: [200] Identity Hash Code:16010  Object: Pet type Dog named Sparky
id:200
```

Handling Exceptions

OracleAS TopLink exceptions are instances of `RuntimeException`, which means that methods that throw them do not have to be placed in a try-catch statement.

However, the Unit of Work `commit` method is one that should be called within a `try-catch` statement to deal with problems that may arise.

Example 7–18 shows one way to handle Unit of Work exceptions:

Example 7–18 Handling Unit of Work Commit Exceptions

```
UnitOfWork uow = session.acquireUnitOfWork();
Pet petClone = (Pet)uow.registerObject(newPet);
petClone.setName("Assume this name is too long for a database constraint");
// Assume that the name argument violates a length constraint on the database.
// This will cause a DatabaseException on commit.
try {
    uow.commit();
} catch (TopLinkException tle) {
    System.out.println("There was an exception: " + tle);
}
```

This code produces the following output:

```
There was an exception: EXCEPTION [ORACLEAS TOPLINK-6004]:
```

```
oracle.toplink.exceptions.DatabaseException
```

Catching exceptions at commit time is mandatory if you are using optimistic locking because the exception raised is the indication that there was an optimistic locking problem. Optimistic locking allows all users to access a given object, even if it is currently in use in a transaction or Unit of Work. When the Unit of Work attempts to change the object, the database checks to ensure that the object has not changed since it was initially read by the Unit of Work. If the object has changed, the database raises an exception, and the Unit of Work rolls back the transaction.

For more information, see "Locking Policy" on page 5-20.

Creating and Registering an Object in One Step

Example shows how to use the Unit of Work `newInstance` method to create a new `Pet` object, register it with the Unit of Work, and return a clone, all in one step. If you are using a factory pattern to create your objects (and specified this in the builder), the `newInstance` method will use the appropriate factory.

Example 7–19 Creating and Registering an Object in One Step

```
UnitOfWork uow = session.acquireUnitOfWork();
    Pet petClone = (Pet)uow.newInstance(Pet.class);
    petClone.setId(100);
    petClone.setName("Fluffy");
    petClone.setType("Cat");
uow.commit();
```

Using registerNewObject

This example examines how to use the `registerNewObject` method, including:

- Registering a New Object with registerNewObject
- Associating New Objects with One Another

Registering a New Object with registerNewObject

The `registerNewObject` method registers a new object as if it was a clone. At commit time, the Unit of Work creates another instance of the object to be the cache version of that object.

Use `registerNewObject` in situations where:

- You do not need a handle to the cache version of the object after the commit and you do not want to work with clones of new objects.
- You must pass a clone into the constructor of a new object and then need to register the new object.

Example 7–20 shows how to register a new object with the `registerNewObject` method:

Example 7–20 Registering a New Object with the registerNewObject Method

```
UnitOfWork uow = session.acquireUnitOfWork();
    PetOwner existingPetOwnerClone =
        PetOwner)uow.readObject(PetOwner.class);

    Pet newPet = new Pet();
    newPet.setId(900);
    newPet.setType("Lizzard");
    newPet.setName("Larry");
    newPet.setPetOwner(existingPetOwnerClone);

    uow.registerNewObject(newPet);
uow.commit();
```

By using `registerNewObject`, the variable `newPet` should not be used after the Unit of Work is committed. The new object is the clone and if you need the cache version of the object, you need to query for it. If you needed a handle to the cache version of the `Pet` after the Unit of Work has committed, then you should use the first approach described in "Associations: New Source to Existing Target Object" on page 7-18. In that example, the variable `newPet` is the cache version after the Unit of Work is committed.

Associating New Objects with One Another

At commit time, OracleAS TopLink can determine if an object is new or not. In "Associations: New Target to Existing Source Object" on page 7-15, we saw that if a new object is reachable from a clone, you do not need to register it. OracleAS TopLink effectively does a `registerNewObject` to all new objects it can reach from registered objects.

When working with new objects, remember the following rules:

- Only reachable or registered objects will be persisted.
- New objects or objects that have been registered with `registerNewObject` are considered to be working copies in the Unit of Work.
- If you call `registerObject` with a new object, the result is the clone and the argument is considered the cache version.

Example 7–21 shows how to associate new objects with the `registerNewObject` method:

Example 7–21 Associating New Objects with the registerNewObject Method

```
UnitOfWork uow = session.acquireUnitOfWork();
    Pet newPet = new Pet();
    newPet.setId(150);
    newPet.setType("Horse");
    newPet.setName("Ed");

    PetOwner newPetOwner = new PetOwner();
    newPetOwner.setId(250);
    newPetOwner.setName("George");
    newPetOwner.setPhoneNumber("555-9999");

    VetVisit newVetVisit = new VetVisit();
    newVetVisit.setId(350);
    newVetVisit.setNotes("Talks a lot");
    newVetVisit.setSymptoms("Sore throat");

    newPet.getVetVisits().addElement(newVetVisit);
    newVetVisit.setPet(newPet);
    newPet.setPetOwner(newPetOwner);

    uow.registerNewObject(newPet);
uow.commit();
```

However, after the Unit of Work, the variables `newPet`, `newPetOwner`, and `newVetVisit` should not be used since they were technically copies from the Unit of Work.

If we needed a handle to the cache version of these business objects we could query for them or we could have done the Unit of Work as shown in Example 7–22.

Example 7–22 Associating New Objects with the newObjectMethod and Retaining a Handle to the Cache Objects

```
UnitOfWork uow = session.acquireUnitOfWork();
    Pet newPet = new Pet();
    Pet newPetClone = (Pet)uow.registerObject(newPet);
    newPetClone.setId(150);
    newPetClone.setType("Horse");
    newPetClone.setName("Ed");

    PetOwner newPetOwner = new PetOwner();
    PetOwner newPetOwnerClone =
        (PetOwner)uow.registerObject(newPetOwner);
    newPetOwnerClone.setId(250);
    newPetOwnerClone.setName("George");
    newPetOwnerClone.setPhoneNumber("555-9999");

    VetVisit newVetVisit = new VetVisit();
    VetVisit newVetVisitClone =
        (VetVisit)uow.registerObject(newVetVisit);
    newVetVisitClone.setId(350);
    newVetVisitClone.setNotes("Talks a lot");
    newVetVisitClone.setSymptoms("Sore throat");

    newPetClone.getVetVisits().addElement(newVetVisitClone);
    newVetVisitClone.setPet(newPetClone);
    newPetClone.setPetOwner(newPetOwnerClone);
uow.commit();
```

Using registerAllObjects

The `registerAllObjects` method takes a `Collection` of objects as an argument and returns a `Collection` of clones. This allows you to register many objects at once as shown in Example 7–23:

Example 7–23 Using registerAllObjects

```
UnitOfWork uow = session.acquireUnitOfWork();
    Collection toRegister = new Vector(2);
    VetVisit vv1 = new VetVisit();
    vv1.setId(70);
    vv1.setNotes("May have flu");
    vv1.setSymptoms("High temperature");
    toRegister.add(vv1);
```

```
    VetVisit vv2 = new VetVisit();
    vv2.setId(71);
    vv2.setNotes("May have flu");
    vv2.setSymptoms("Sick to stomach");
    toRegister.add(vv2);

    uow.registerAllObjects(toRegister);
uow.commit();
```

Using Registration and Existence Checking

When OracleAS TopLink writes an object to the database, OracleAS TopLink runs an existence check to determine whether to perform an insert or an update. You can specify the default existence checking policy for a project as a whole or on a per-descriptor basis. By default, OracleAS TopLink uses the check cache existence checking policy. If you use any existence checking policy other than check cache, then you can use the way you register your objects to your advantage to reduce the time it takes OracleAS TopLink to register an object.

This section explains how to use one of the following existence checking policies to accelerate object registration:

- Check Database
- Assume Existence
- Assume Non-existence

Check Database

If your existence checking policy is check database then OracleAS TopLink will check the database for existence for all objects registered in a Unit of Work. However, if you know that an object is new or existing, rather than use the basic `registerObject` method, you can use `registerNewObject` or `registerExistingObject` to bypass the existence check. OracleAS TopLink will not check the database for existence on objects that you have registered with these methods. It will automatically do an insert if `registerNewObject` is called or an update if `registerExistingObject` is called.

Assume Existence

If your existence checking policy is assume existence then all objects registered in a Unit of Work are assumed to exist and OracleAS TopLink will always do an update to the database on all registered objects, even new objects that you registered with `registerObject`. However, if you use the `registerNewObject` method on the new object, OracleAS TopLink knows to do an insert in the database even though the existence checking policy says assume existence.

Assume Non-existence

If your existence checking policy is assume non-existence then all objects registered in a Unit of Work are assumed to be new and OracleAS TopLink will always do an insert to the database, even on objects read from the database. However, if you use the `registerExistingObject` method on existing objects, OracleAS TopLink knows to do an update to the database.

Working with Aggregates

Aggregate mapped objects should never be registered in an OracleAS TopLink Unit of Work (in fact, you will get an exception if you try). Aggregate cloning and registration is automatic based on the owner of the aggregate object. In other words, if you register the owner of an aggregate, the aggregate is automatically cloned. When you get a working copy of an aggregate owner, its aggregate is also a working copy.

The bottom line of working with aggregates is you should always use an aggregate within the context of its owner:

- If you get an aggregate from a working copy owner, then the aggregate is a working copy.
- If you get an aggregate from a cache version owner then the aggregate is the cache version.

Unregistering Working Clones

The Unit of Work `unregisterObject` method allows you to unregister a previously registered object from a Unit of Work. An unregistered object will be ignored in the Unit of Work and any uncommitted changes made to the object up to that point will be discarded.

In general, this method is rarely used. It can be useful if you create a new object, but then decide to delete it in the same Unit of Work (which is also not recommended).

Declaring Read-Only Classes

You can declare a class as read-only within the context of a Unit of Work. Clones are neither created nor merged for such classes, thus improving performance. Such classes are ineligible for changes in the Unit of Work.

When a Unit of Work registers an object, it traverses and registers the entire object tree. If the Unit of Work encounters a read-only class, it does not traverse that branch of the tree and does not register objects referenced by the read-only class, so those classes are ineligible for changes in the Unit of Work.

Setting Read-Only Classes for a Single Unit of Work

For example, suppose class A owns a class B and class C extends class B. You acquire a Unit of Work in which you know only instances of A will change: you know that no class B's will be changed. Before registering an instance of B, use:

```
myUnitofWork.addReadOnlyClass(B.class);
```

Then you can proceed with your transaction: registering A objects, modifying their working copies, and committing the Unit of Work.

At commit time, the Unit of Work will not have to compare backup copy clones with the working copy clones for instances of class B (even if instances were registered explicitly or implicitly). This can improve Unit of Work performance if the object tree is very large.

Note that if you register an instance of class C, the Unit of Work does not create or merge clones for this object; any changes made to your C are not be persisted because C extends B and B was identified as read-only.

To identify multiple classes as read only, add them to a Vector and use:

```
myUnitOfWork.addReadOnlyClasses(myVectorOfClasses);
```

Note that a nested Unit of Work inherits the set of read-only classes from the parent Unit of Work. For more information on using a nested Unit of Work, see "Using a Nested or Parallel Unit of Work" on page 7-40.

Setting Read-Only Classes for All Units of Work

To establish a default set of read-only classes for all Units of Work, use the project method `setDefaultReadOnlyClasses(Vector)`. After you call this method, all new Units of Work include the `Vector` of read-only classes.

Read-Only Descriptors

When you declare a class as read-only, the read-only flag extends to its descriptors. You can flag a descriptor as read-only at development time, using either Java code or the OracleAS TopLink Mapping Workbench. This option improves performance by excluding the read-only descriptors from Unit of Work registration and editing.

To flag descriptors as read-only in Java code, call the `setReadOnly` method on the descriptor as follows:

```
descriptor.setReadOnly();
```

To flag a descriptor as read-only in the OracleAS TopLink Mapping Workbench, select the **Read Only** check box for a specific descriptor.

For more information, see "Working with Descriptors," in the *Oracle Application Server TopLink Mapping Workbench User's Guide.*

Using Conforming Queries and Descriptors

This section explains how to include new, changed, or deleted objects in queries within a Unit of Work prior to commit, including:

- Using Conforming Queries
- Conforming Query Alternatives
- Using Conforming Descriptors

Using Conforming Queries

Because queries are executed on the database, querying though a Unit of Work will not, by default, include new, uncommitted, objects in a Unit of Work. The Unit of Work will not spend time executing your query against new, uncommitted, objects in the Unit of Work unless you explicitly tell it to.

Assume that a single `Pet` of type `Cat` already exists on the database. Examine the code shown in Example 7–24.

Example 7–24 Using Conforming Queries

```
UnitOfWork uow = session.acquireUnitOfWork();
    Pet pet2 = new Pet();
    Pet petClone = (Pet)uow.registerObject(pet2);
    petClone.setId(200);
    petClone.setType("Cat");
    petClone.setName("Mouser");

    ReadAllQuery readAllCats = new ReadAllQuery();
    readAllCats.setReferenceClass(Pet.class);
    ExpressionBuilder builder = new ExpressionBuilder();
    Expression catExp = builder.get("type").equal("Cat");
    readAllCats.setSelectionCriteria(catExp);
```

```
    Vector allCats = (Vector)uow.executeQuery(readAllCats);

    System.out.println("All 'Cats' read through UOW are: " + allCats);
uow.commit();
```

This produces the following output:

```
All 'Cats' read through UOW are: [Pet type Cat named Fluffy id:100]
```

If you tell the query `readAllCats` to include new objects:

```
readAllCats.conformResultsInUnitOfWork();
```

The output would be:

```
All 'Cats' read through UOW are: [Pet type Cat named Fluffy id:100, Pet type Cat
named Mouser id:200]
```

Bear in mind that conforming will impact performance. Before you use conforming, make sure that it is actually necessary. For example, consider the alternative described in "Conforming Query Alternatives" on page 7-36.

Conforming Query Alternatives

Sometimes, you need to provide other code modules with access to new objects created in a Unit of Work. Conforming can be used to provide this access. However, the following alternative is significantly more efficient.

Somewhere a Unit of Work is acquired from a Session and is passed to multiple modules for portions of the requisite processing:

```
UnitOfWork uow = session.acquireUnitOfWork();
```

In the module that creates the new employee:

```
Pet newPet = new Pet();
Pet newPetClone = (Pet)uow.registerObject(newPet);
uow.setProperty("NEW PET", newPet);
```

In other modules where newPet needs to be accessed for further modification, it can simply be extracted from the Unit of Work's properties:

```
Pet newPet = (Pet) uow.getProperty("NEW PET");
newPet.setType("Dog");
```

Conforming queries are ideal if you are not sure if an object has been created yet or the criteria is dynamic.

However, for situations where the quantity of objects is finite and well known, this simple and more efficient solution is a very practical alternative.

Using Conforming Descriptors

OracleAS TopLink's support for conforming queries in the Unit of Work can be specified in the descriptors.

You can flag a descriptor directly to always conform results in the Unit of Work so that all queries performed on this descriptor conform its results in the Unit of Work by default. You can specify this either within code or from the OracleAS TopLink Mapping Workbench.

You can flag descriptors to always conform in the Unit of Work by calling the method on the descriptor as follows:

```
descriptor.setShouldAlwaysConformResultsInUnitOfWork(true);
```

To set this flag in the OracleAS TopLink Mapping Workbench, select the **Conform Results in Unit Of Work** check box for a descriptor.

Merging Changes in Working Copy Clones

In a three-tier application, the client and server exchange objects using a serialization mechanism such as RMI or CORBA.

When the client changes an object and returns it to the server, you cannot register this serialized object into a Unit of Work directly.

On the server, you must register the original object in a Unit of Work and then use the Unit of Work methods listed in Table 7–2 to merge serialized object changes into the working copy clone. Each method takes the serialized object as an argument.

Table 7–2 Unit of Work Merge Methods

Method	Purpose	Used When
`mergeClone`	Merges the serialized object and all its privately owned parts (excluding references from it to independent objects) into the working copy clone.	The client edits the object but not its relationships, or marks its independent relationships as transient.

Table 7–2 Unit of Work Merge Methods

Method	Purpose	Used When
`mergeCloneWithReferences`	Merges the serialized object and all its privately owned parts (including references from it to independent objects) into the working copy clone.	The client edits the object and the targets of its relationships and has not marked any attributes as transient.
`shallowMergeClone`	Merges only serialized object changes to attributes mapped with direct mappings into the working copy clone.	The client only edits the object's direct attributes or has marked all of the object's relationships as transient.
`deepMergeClone`	Merges the serialized object and everything connected to it (the entire object tree where the serialized object is the root) into the working copy clone.	Use with caution: if two different copies of an object are in the same traversal, it will merge one set of changes over the other. You should not have any transient attributes in any of your related objects.

Note that if your three-tier client is sufficiently complex, consider using the TopLink remote session (see "Remote Session" on page 4-58). It automatically handles merging and allows you to use a Unit of Work on the client.

You can merge clones with both existing and new objects. Because they do not appear in the cache and may not have a primary key, you can merge new objects only once within a Unit of Work. If you need to merge a new object more than once, call the Unit of Work `setShouldNewObjectsBeCached` method, and ensure that the object has a valid primary key; you can then register the object.

Example 7–25 shows one way to update the original object with the changes contained in the corresponding serialized object (`rmiClone`) received from a client.

Example 7–25 Merging a Serialized Object

```
update(Object original, Object rmiClone)
{
  original = uow.registerObject(original);
```

```
  uow.mergeCloneWithRefereneces(rmiClone);
  uow.commit();
}
```

Resuming a Unit of Work After Commit

At commit time, a Unit of Work and its contents expire: you must not use the Unit of Work nor its clones even if the transaction failed and rolled back.

However, OracleAS TopLink offers API that enables you to continue working with a Unit of Work and its clones:

- `commitAndResume`: commits the Unit of Work, but does not invalidate it or its clones
- `commitAndResumeOnFailure`: commits the Unit of Work. If the commit succeeds, the Unit of Work expires. However, if the commit fails, this method does not invalidate the Unit of Work or its clones. This method enables the user to modify the registered objects in a failed Unit of Work and retry the commit.

Example 7–26 shows how to use the `commitAndResume` method:

Example 7–26   Using the commitAndResume Method

```
UnitOfWork uow = session.acquireUnitOfWork();
    PetOwner petOwnerClone =
        (PetOwner)uow.readObject(PetOwner.class);
    petOwnerClone.setName("Mrs. Newowner");
    uow.commitAndResume();
    petOwnerClone.setPhoneNumber("KL5-7721");
uow.commit();
```

The `commitAndResume` call produces the SQL:

```
UPDATE PETOWNER SET NAME = 'Mrs. Newowner' WHERE (ID = 400)
```

And then the `commit` call produces the SQL:

```
UPDATE PETOWNER SET PHN_NBR = 'KL5-7721' WHERE (ID = 400)
```

Reverting a Unit of Work

Under certain circumstances, you may want to abandon some or all changes to clones in a Unit of Work, but not abandon the Unit of Work itself. The following options exist for reverting all or part of the Unit of Work:

- `revertObject`: abandons changes to a specific working copy clone in the Unit of Work
- `revertAndResume`: uses the backup copy clones to restore all clones to their original states, deregister any new objects, and reinstate any deleted objects

Using a Nested or Parallel Unit of Work

You can use a Unit of Work within another Unit of Work (nesting) or you can use two or more Units of Work with the same objects in parallel.

Parallel Unit of Work

To start multiple Units of Work that operate in parallel, call the `acquireUnitOfWork` method multiple times on the session. The Units of Work operate independently of one another and maintain their own cache.

Nested Unit of Work

To nest Units of Work, call the `acquireUnitOfWork` method on the parent Unit of Work. This creates a child Unit of Work with its own cache. If a child Unit of Work commits, it updates the parent Unit of Work rather than the database. If the parent does not commit, the changes made to the child are not written to the database.

OracleAS TopLink does not update the database or the cache until the outermost Unit of Work is committed. You must commit or release the child Unit of Work before you can commit its parent.

Working copies from one Unit of Work are not valid in another Units of Work: not even between an inner and outer Unit of Work. You must register objects at all levels of a Unit of Work where they are used.

Example 7–27 shows how to use nested Units of Work:

Example 7–27 Using Nested Units of Work

```
UnitOfWork outerUOW = session.acquireUnitOfWork();
    Pet outerPetClone = (Pet)outerUOW.readObject(Pet.class);

    UnitOfWork innerUOWa = outerUOW.acquireUnitOfWork();
```

```
        Pet innerPetCloneA =
            (Pet)innerUOWa.registerObject(outerPetClone);
        innerPetCloneA.setName("Muffy");
    innerUOWa.commit();

    UnitOfWork innerUOWb = outerUOW.acquireUnitOfWork();
        Pet innerPetCloneB =
            (Pet)innerUOWb.registerObject(outerPetClone);
        innerPetCloneB.setName("Duffy");
    innerUOWb.commit();
outerUOW.commit();
```

Using a Unit of Work with Custom SQL

You can add custom SQL to a Unit of Work at any time by calling the Unit of Work `executeNonSelectingCall` method as shown in Example 7–28.

Example 7–28 Using the executeNonSelectingCall Method

```
uow.executeNonSelectingCall(new SQLCall(mySqlString));
```

Validating a Unit of Work

The Unit of Work validates object references at commit time. If an object registered in a Unit of Work references other unregistered objects, this violates object transaction isolation, and causes OracleAS TopLink validation to raise an exception.

Although referencing unregistered objects from a registered object can corrupt the session cache, there are applications in which you want to disable validation. OracleAS TopLink offers API to toggle validation, as follows:

- `dontPerformValidation`: disables validation
- `performFullValidation`: enables validation

Validating the Unit of Work Before Commit

If the Unit of Work detects an error when merging changes into the session cache, it throws a `QueryException`. Although this exception specifies the invalid object and the reason it is invalid, it may still be difficult to determine the cause of the problem.

In this case, you can use the `validateObjectSpace` method to test registered objects and provide the full stack of traversed objects. This may help you more easily find the problem. You can call this method at any time on a Unit of Work.

Controlling the Order of Deletes

"Deleting Objects" on page 7-20 explained that OracleAS TopLink always properly orders the SQL based on the mappings and foreign keys in your object model and schema. You can control the order of deletes by:

- Using the Unit of Work setShouldPerformDeletesFirst Method
- Using the Descriptor addConstraintDependencies Method

Using the Unit of Work setShouldPerformDeletesFirst Method

It is possible to tell the Unit of Work to issue deletes before inserts and updates by calling the Unit of Work `setShouldPerformDeletesFirst` method.

By default, OracleAS TopLink does inserts and updates first to ensure that referential integrity is maintained.

If you are replacing an object with unique constraints by deleting it and inserting a replacement, if the insert occurs before the delete, you may raise a constraint violation. In this case, you may need to call `setShouldPerformDeletesFirst` so that the delete is performed before the insert.

Using the Descriptor addConstraintDependencies Method

The constraints used by OracleAS TopLink to determine delete order are inferred from one-to-one and one-to-many mappings. If you do not have such mappings, you can add constraint knowledge to OracleAS TopLink using the descriptor `addConstraintDependencies(Class)` method.

For example, suppose you have a composition of objects: A contains B (one-to-many, privately owned) and B has a one-to-one, non-private relationship with C. You want to delete A (and in doing so the included B's) but before deleting the B's, for some of them (not all) you want to delete the associated object C.

There are two possible solutions:

- Using deleteAllObjects without addConstraintDependencies
- Using deleteAllObjects with addConstraintDependencies

Using deleteAllObjects without addConstraintDependencies

In the first option, we do not use privately-owned on the one-to-many (A to B) relationship. When deleting an A, we make sure to delete all of it's B's as well as any C instances. For example:

```
uow.deleteObject(existingA);
```

```
uow.deleteAllObjects(existingA.getBs());
// delete one of the C's
uow.deleteObject(((B) existingA.getBs().get(1)).getC());
```

This option produces the following SQL:

```
DELETE FROM B WHERE (ID = 2)
DELETE FROM B WHERE (ID = 1)
DELETE FROM A WHERE (ID = 1)
DELETE FROM C WHERE (ID = 1)
```

Using deleteAllObjects with addConstraintDependencies

In the second option, we keep the one-to-many (A to B) relationship privately owned and add a constraint dependency from A to C. For example:

```
session.getDescriptor(A.class).addConstraintDependencies(C.class);
```

Now the delete code would be:

```
uow.deleteObject(existingA);
uow.deleteAllObjects(existingA.getBs());
// delete one of the C's
uow.deleteObject(((B) existingA.getBs().get(1)).getC());
```

This option produces the following SQL:

```
DELETE FROM B WHERE (A = 1)
DELETE FROM A WHERE (ID = 1)
DELETE FROM C WHERE (ID = 1)
```

In both cases, the B is deleted before A and C. The main difference is that the second option will generate fewer SQL statements as it knows that it is deleting the entire set of B's related from A.

Improving Unit of Work Performance

For best performance when using a Unit of Work, consider the following tips:

- Register objects with a Unit of Work only if objects are eligible for change. If you register objects that will not change, the Unit of Work needlessly clones and processes those objects.
- Avoid the cost of existence checking when you are registering a new or existing object (see "Using Registration and Existence Checking" on page 7-32).

- Avoid the cost of change set calculation on a class you know will not change by telling the Unit of Work that the class is read-only (see "Declaring Read-Only Classes" on page 7-33).
- Avoid the cost of change set calculation on an object read by a ReadAllQuery in a Unit of Work that you do not intend to change by unregistering the object (see "Unregistering Working Clones" on page 7-33).
- Before using conforming queries, be sure that it is necessary. For alternatives, see "Using Conforming Queries and Descriptors" on page 7-35.

J2EE Integration

OracleAS TopLink J2EE integration provides support for external datasources and external transaction controllers. Together, these features provide support for JTA. This enables you to incorporate external container support into your application, and to use JTA transactions.

This section describes:

- External Connection Pooling
- External Transaction Controllers

External Connection Pooling

For most non-J2EE applications OracleAS TopLink provides an internal connection or pool of connections. However, most J2EE applications use external connection pooling offered by the J2EE Container JTA `DataSource`. For J2EE applications OracleAS TopLink integrates with the J2EE Container connection pooling.

When to Use External Connection Pools

External connection pools enable your OracleAS TopLink application to:

- Integrate into a J2EE-enabled system.
- Integrate with JTA transactions (JTA transactions require a JTA enabled `DataSource`).
- Leverage a shared connection pool in which multiple applications use the same `DataSource`.
- Use a `DataSource` configured and managed directly on the server.

- Leverage a datasource that is accessible only through the `DataSource` interface.

Configure OracleAS TopLink to use the built-in JTA integration support to take advantage of these benefits. Without JTA, external connection pools generally offer benefits only if transactions in an OracleAS TopLink application are independent of each other and any other transactions in the system. In that case, the complexities of an OracleAS TopLink connection or connection pool are unnecessary.

Configuring an External Connection Pool in sessions.xml

To configure the use of an external connection pool in the sessions.xml file:

1. Configure the `DataSource` on the server.
2. Add the following elements to the `login` tag in the `sessions.xml` file to specify a `DataSource` and the use of an external connection pool:

```
<data-source>jdbc/MyApplicationDS</data-source>
<uses-external-connection-pool>true</uses-external-connection-pool>
```

Configuring an External Connection Pool in Java

To configure the use of an external connection pool in Java:

1. Configure the `DataSource` on the server.
2. Configure the `Login` to specify a `DataSource` and the use of an external connection pool:

```
login.setConnector(
    new JNDIConnector(new InitialContext(), "jdbc/MyApplicationDS")
);
login.setUsesExternalConnectionPooling(true);
```

External Transaction Controllers

A transaction controller is an OracleAS TopLink class that synchronizes the session cache with the data on the database. The transaction controller manages messages and callbacks from the J2EE transaction. On commit, the transaction controller executes the Unit of Work SQL on the database, and merges changed objects into the OracleAS TopLink session cache. Because JTA transaction controllers require a JTA-enabled `DataSource`, configure an external transaction controller and enable OracleAS TopLink external connection pool support.

OracleAS TopLink provides transaction controllers for container-specific support, as well as a generic controllers that can be used for other specification-conforming servers.

Table 7–3 lists the custom external transaction controllers OracleAS TopLink provides.

Table 7–3 OracleAS TopLink Custom External Transaction Controllers

Application Server or J2EE Container	OracleAS TopLink External Transaction Controller
Oracle Application Server Containers for J2EE	`oracle.toplink.jts.oracle9i.Oracle9iJTSExternalTransactionController`
IBM WebSphere 3.5	`oracle.toplink.jts.was.WebSphereJTSExternalTransactionController`
IBM WebSphere 4.0	`oracle.toplink.jts.was.JTSExternalTransactionController_4_0`
IBM WebSphere 5.0	`oracle.toplink.jts.was.JTSExternalTransactionController_5_0`
BEA WebLogic	`oracle.toplink.jts.wls.WebLogicJTSExternalTransactionController`
Other JTA Container	`oracle.toplink.jts.JTSExternalTransactionController`

Configuring an External Transaction Controller in sessions.xml

To configure the use of an external transaction controller in the sessions.xml file:

1. Configure a JTA-enabled `DataSource` on the server.

   For more information, see the J2EE container documentation.

2. Add the following elements to the `login` tag in the `sessions.xml` file to specify a `DataSource`, the use of an external transaction controller, and the use of an external connection pool:

   ```
   <data-source>jdbc/MyApplicationDS</data-source>
   <uses-external-transaction-controller>
       true
   </uses-external-transaction-controller>
   <uses-external-connection-pool>true</uses-external-connection-pool>
   ```

3. Specify an external transaction controller class in the `sessions.xml` file.

   For example:

   ```
   <external-transaction-controller-class>
       oracle.toplink.jts.oracle9i.Oracle9iJTSExternalTransactionController
   </external-transaction-controller-class>
   ```

Configuring an External Transaction Controller in Java

To configure the use of an external transaction controller in Java:

1. Configure a JTA-enabled `DataSource` on the server.

   For more information, see the J2EE container documentation.

2. Configure the `Login` to specify a `DataSource`, the use of an external transaction controller, and the use of an external connection pool:

```
login.setConnector(
    new JNDIConnector(new InitialContext(), "jdbc/MyApplicationDS")
);
login.setUsesExternalTransactionController(true);
login.setUsesExternalConnectionPooling(true);
```

3. Configure the session to use a particular instance of ExternalTransactionController:

```
serverSession.setExternalTransactionController(
    new Oracle9iJTSExternalTransactionController()
);
```

Acquiring a Unit of Work in a JTA Environment

You use a Unit of Work to write to a database even in a JTA environment. To ensure that only one Unit of Work is associated with a given transaction, use the `getActiveUnitOfWork` method to acquire a Unit of Work as shown in Example 7–29.

Note: Although there are other ways to write to a database through a JTA external controller, using `getActiveUnitOfWork` method is the safest approach to database updates under JTA.

The `getActiveUnitOfWork` method searches for an existing external transaction:

- If there is an active external transaction and a Unit of Work is already associated with it, return this Unit of Work.
- If there is an active external transaction with no associated Unit of Work, then acquire a new Unit of Work, associate it with the transaction and return it.
- If there is no active external transaction in progress, return null.

If a non-null Unit of Work is returned, use it exactly as you would in a non-JTA environment: the only exception is that you do not call the `commit` method (see "Using a Unit of Work When an External Transaction Exists" on page 7-48).

If a null Unit of Work is returned, start an external transaction either explicitly through the `UserTransaction` interface, or by acquiring a new Unit of Work using the `acquireUnitOfWork` method on the client session (see "Using a Unit of Work When No External Transaction Exists" on page 7-49).

Example 7–29 Using a Unit of Work in a JTA Transaction

```
boolean shouldCommit = false;
// Read in any pet.
Pet pet = (Pet)clientSession.readObject(Pet.class);
UnitOfWork uow = clientSession.getActiveUnitOfWork();
    if (uow == null) {
        uow = clientSession.acquireUnitOfWork(); // Start external transaction
        shouldCommit = true;
    }
    Pet petClone = (Pet) uow.registerObject(pet);
    petClone.setName("Furry");
    if (shouldCommit) {
        uow.commit(); // Ask external transaction controller to commit
    }
```

Using a Unit of Work When an External Transaction Exists

When `getActiveUnitOfWork` returns a non-null Unit of Work, you are associated with an existing external transaction. Use the Unit of Work as usual.

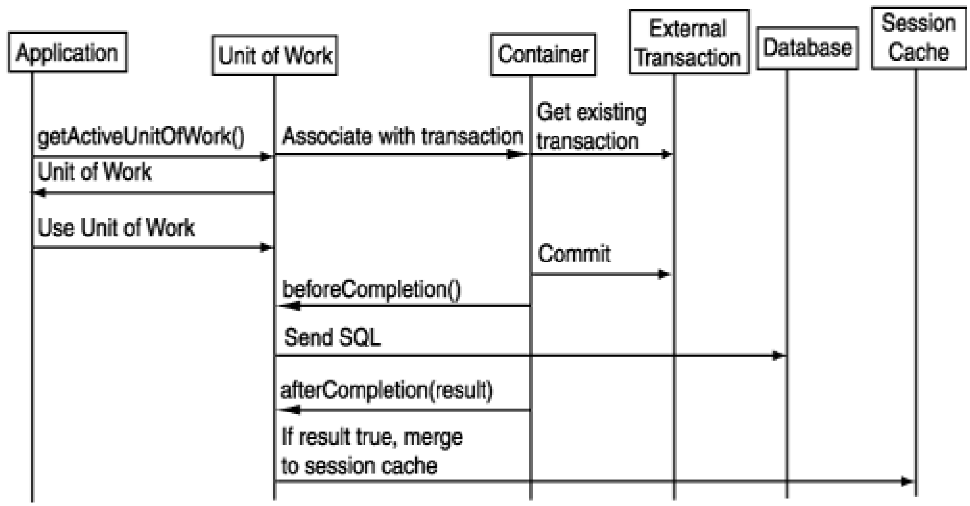
As the external transaction was not started by the Unit of Work, issuing a `commit` on it will not cause the JTA transaction to be committed. The Unit of Work will defer to the application or container that began the transaction. When the external transaction does get committed by the container, OracleAS TopLink receives sychronization callbacks at key points during the commit.

The Unit of Work sends the required SQL to the database when it receives the `beforeCompletion` call back.

The Unit of Work uses the boolean argument received from the `afterCompletion` call back to determine if the commit was successful (true) or not (false).

If the commit was successful, the Unit of Work merges changes to the session cache. If the commit was unsuccessful, the Unit of Work discards the changes.

Figure 7–4 Unit of Work When an External Transaction Exists



Using a Unit of Work When No External Transaction Exists

When `getActiveUnitOfWork` returns a null Unit of Work, there is no existing external transaction. You must start a new external transaction.

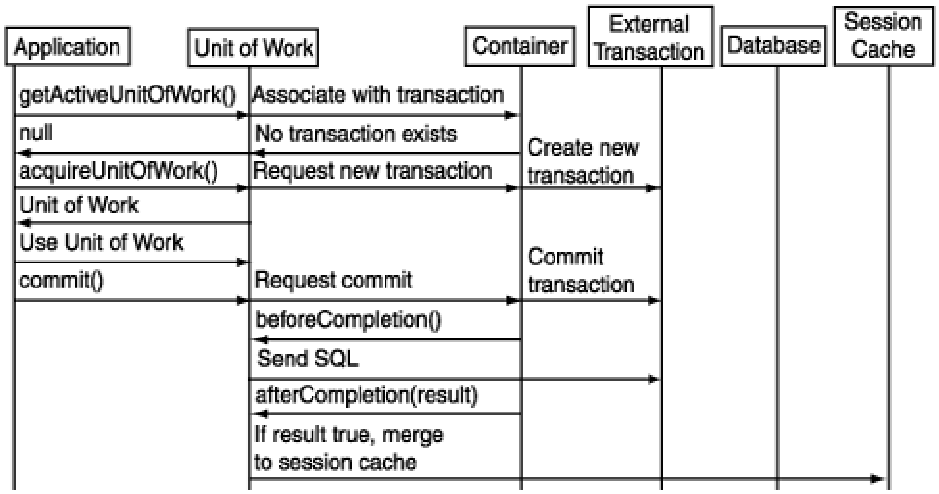
Do this either by starting an external transaction explicitly using the `UserTransaction` interface, or by acquiring a new Unit of Work using the `acquireUnitOfWork` method on the server session.

Use the Unit of Work as usual.

Once the modifications to registered objects are complete, you must commit the transaction either explicitly through the `UserTransaction` interface or by calling the Unit of Work `commit` method.

The transaction synchronization callbacks are then invoked on OracleAS TopLink and the database updates and cache merge occurs based upon those callbacks.

Figure 7–5 Unit of Work When No External Transaction Exists



8

Cache

A cache is a repository that stores recently used objects for an application. Holding objects in the cache helps you minimize database access, and improves application performance.

Oracle Application Server TopLink uses two object caches: the *session cache* maintains objects retrieved from and written to the database; and the *Unit of Work cache* holds objects while they participate in transactions. These caches maintain objects based on class and primary key values.

This chapter explores cache use, and discusses the following topics:

- Introduction to Cache Concepts
- Cache Locking and Isolation
- Configuring the Cache
- Distributed Cache Synchronization
- Remote Command Manager

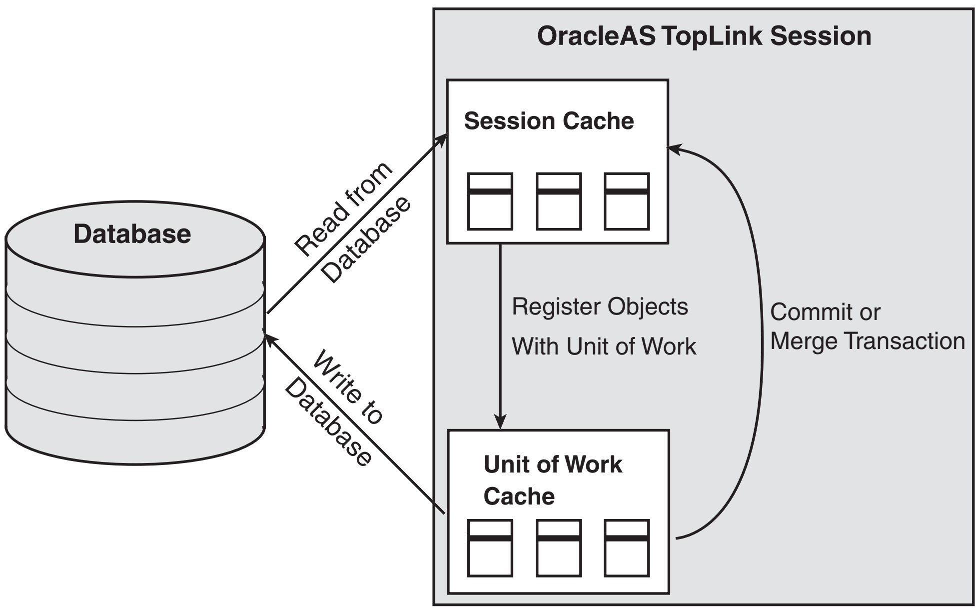
Introduction to Cache Concepts

The cache is a key OracleAS TopLink component. You use the cache to improve application performance and manage user access to the database. This section introduces concepts that help you optimize the way your application uses its caches.

Cache Architecture

The session cache and the Unit of Work cache work together with the database connection to manage objects in an OracleAS TopLink application. The object life cycle relies on these three mechanisms.

Figure 8–1 Object Life Cycle and the OracleAS TopLink Caches



Session Cache

The session cache is a shared cache that services clients attached to a given database session. When you read data from or write data to the database, OracleAS TopLink saves a copy in the session cache and provides that data to all other processes in the session.

OracleAS TopLink adds objects to the cache from:

- The database, when OracleAS TopLink executes a database read
- The Unit of Work cache, when a Unit of Work successfully commits a transaction

You can configure queries to search the cache for existing data.If the data exist in the cache, rather than perform a database read, OracleAS TopLink returns the cached data.

For more information about query cache usage, see "In-Memory Query Cache Usage" on page 6-62.

Unit of Work Cache

The Unit of Work cache services operations within the Unit of Work. It maintains and isolates objects, and writes changed or new objects to the session cache after the Unit of Work commits changes to the database.

Stale Data

Stale data is an artifact of caching in which an object is not the most recent version. To avoid stale data, implement an appropriate cache locking strategy.

Cache Locking

Cache locking regulates when processes read or write an object. Depending on how you configure it, cache locking determines whether a process can read or write an object that is in use with another process. Cache locking also enables you to manage stale data issues.

Distributed Cache Synchronization

When you deploy your OracleAS TopLink application in a cluster, the cluster generally includes several caches. Because each cache services a different application, this raises the possibility that changes from one application may not appear in the other applications in the cluster.

Distributed cache synchronization reduces the occurrence of stale data across the caches in the system. When an object changes in one cache, distributed cache synchronization enables you to update the other caches in the cluster to replace stale data.

For more information about distributed cache synchronization, see "Distributed Cache Synchronization" on page 8-6.

Cluster

An OracleAS TopLink cluster is a collection of servers that:

- Are connected by a local area network (LAN).
- Use OracleAS TopLink to provide the cooperation infrastructure between the servers.

Discovery

Discovery occurs when servers in a cluster learn of other servers in the cluster. Discovery uses a multicast protocol to monitor sessions as they join and leave the OracleAS TopLink cluster.

Message Transport

A message transport is the messaging protocol servers in a cluster use to send and receive messages. OracleAS TopLink uses a transport protocol to exchange object updates between cooperating sessions.

Name Service

A name service enables you to search for objects on remote caches. OracleAS TopLink cache synchronization uses a name service when it looks up connections to other sessions in the OracleAS TopLink cluster.

If you use RMI as a transport, the RMI Registry provides lookup capabilities. In most other cases, the Java Naming and Directory Interface (JNDI) provides lookup functionality.

Propagation Modes

The propagation mode determines when a client regains control after it propagates object changes. OracleAS TopLink supports synchronous and asynchronous propagation modes.

Synchronous Update Mode When you propagate updates synchronously, OracleAS TopLink prevents the committing client from performing other tasks until the remote merge process is complete.

Asynchronous Update Mode In asynchronous mode, OracleAS TopLink creates separate threads to propagate changes to remote servers. OracleAS TopLink returns control to the client immediately after the local commit, whether or not the changes merge successfully on the remote servers. This offers superior performance for applications that are somewhat tolerant of stale data.

Cache Locking and Isolation

By default, OracleAS TopLink optimizes concurrency to minimize cache locking during reads or writes. Use the default OracleAS TopLink isolation level unless you have a very specific reason to change it.

Use the following application programming interface (API) on `Databaselogin` to change the OracleAS TopLink isolation level:

```
login.setCacheTransactionIsolation(int cacheTransactionIsolation)
```

The available settings for `cacheTransactionIsolation` are:

- `ConcurrentReadWrite`: the default; it allows concurrent object read and write
- `SynchronizedWrite`: allows only a single Unit of Work to merge into the cache at once
- `SynchronizedReadOnWrite`: does not allow reading or other Unit of Work merge while a Unit of Work is merging

Configuring the Cache

A well-managed cache makes your application more efficient. There are very few cases in which you turn the cache off entirely, as the cache reduces database access, and is an important part of managing object identity.

To make the most of your cache strategy and to minimize your application's exposure to stale data, we recommend the following:

Configure the cache on a per-class basis If other applications can modify the data used by a particular class, use a weaker style of cache for the class. For example, the `SoftCacheWeakIdentityMap` or `WeakIdentityMap` minimizes the length of time the cache maintains a de-referenced object.

For more information about configuring cache usage on a per-class basis, see "Working with Identity Maps" in the *Oracle Application Server TopLink Mapping Workbench User's Guide.*

Note: If your application reaches a low system memory condition frequently enough or if your platform's JVM treats weak and soft references the same, the objects in the sub-cache may be garbage collected so often that you will not benefit from the performance improvement provided by the sub-cache. If this is the case, Oracle recommends that you use the `HardCacheWeakIdentityMap`. It is identical to the `SoftCacheWeakIdentityMap` except that it uses hard references in the sub-cache. This guarantees that your application will benefit from the performance improvement provided by the sub-cache.

Force a cache refresh when required on a per-query basis Any query can include a flag that forces a cache refresh to the database.

For more information about configuring cache refresh on a per-query basis, see "Refresh" on page 6-66.

Distributed Cache Synchronization

The need to maintain up-to-date data for all applications is a key design challenge for building a distributed application environment. The difficulty of this increases as the number of servers within an environment increases. OracleAS TopLink provides a distributed cache synchronization feature that ensures data in applications remains current.

Cache synchronization in no way eliminates the need for an effective locking policy. However, it does reduce the number of optimistic lock exceptions encountered in a distributed architecture, and decreases the number of failed or repeated transactions in an application.

OracleAS TopLink provides cache synchronization at the session level. This ensures that object updates associated with a given session propagate to the caches on all other servers in the cluster.

This section describes:

- Configuring Cache Synchronization in the sessions.xml File
- Explicit Query Refreshes

Configuring Cache Synchronization in the sessions.xml File

Because each application server approaches caching differently, you must configure cache synchronization to work effectively within the distributed system.

For more information about choosing cache configuration options, see the application server or J2EE container documentation.

To enable and configure cache synchronization in the `sessions.xml` file, specify the `cache-synchronization-manager` element, and configure the required sub-elements.

Example 8–1 illustrates how to configure cache synchronization in the `sessions.xml` file for a session that:

- Runs in Oracle Application Server Containers for J2EE
- Uses the default discovery settings
- Uses JNDI to look up the remote objects
- Distributes the changes using RMI

Example 8–1 Configuring Cache Synchronization in the sessions.xml File

```
<cache-synchronization-manager>
    <clustering-service>
        oracle.toplink.remote.rmi.RMIJNDIClusteringService
    </clustering-service>
    <jndi-user-name>userName</jndi-user-name>
    <jndi-password>password</jndi-password>
    <naming-service-initial-context-factory-name>
        oracle.com.evermind.server.rmi.RMIInitialContextFactory
    </naming-service-initial-context-factory-name>
    <naming-service-url>ormi://hostname:23791/appName</naming-service-url>
</cache-synchronization-manager>
```

The configuration in Example 8–1 includes the name, password, context factory class, and URL. Oracle Application Server Containers for J2EE requires all four sub-elements to enable name lookup on remote hosts. Other servers require different values for the context factory and URL.

For Oracle Application Server Containers for J2EE, the URL element includes the `ormi://` protocol, the local host name and RMI server port, and the name of the application in which the OracleAS TopLink session is deployed.

Clustering Service

The `clustering-service` element specifies the name service and transport combination used to communicate changes. Choose the combination that works best with your application. Your choices are:

- `oracle.toplink.remote.rmi.RMIJNDIClusteringService`: uses JNDI to look up remote sessions, and RMI point-to-point connections to propagate changes between sessions.
- `oracle.toplink.remote.rmi.RMIClusteringService`: uses RMIRegistry to look up sessions, and RMI point-to-point connections to propagate changes between sessions.
- `oracle.toplink.remote.ejb.EJBJNDIClusteringService`: uses JNDI to look up session beans that propagate changes between sessions.
- `oracle.toplink.remote.corba.CORBAJNDIClusteringService`: uses JNDI to look up sessions, and CORBA point-to-point connections to propagate changes between sessions.
- `oracle.toplink.remote.corba.JMSClusteringService`: uses JNDI to look up Java Message Service (JMS) topics that propagate changes between sessions.

Example 8–2 Configuring a Clustering Service in the sessions.xml File

```
<cache-synchronization-manager>
    <clustering-service>
        oracle.toplink.remote.rmi.RMIJNDIClusteringService
    </clustering-service>
    ...
</cache-synchronization-manager>
```

Discovery

Discovery occurs when servers in a cluster learn of other servers in the cluster and uses a multicast protocol to monitor sessions as they join and leave the OracleAS TopLink cluster. If you are running OracleAS TopLink with other Oracle Application Server 10*g* components, ensure the port you select does not conflict with other components. If OracleAS TopLink's default discovery configuration conflicts with settings for other services on the same host, you can override the discovery settings.

You can configure discovery to use specific optional multicast socket options, including:

- `multicast-port`: overrides the default multicast port used for discovery (default is 6018)
- `multicast-group-address`: overrides the default multicast group used for discovery (default is 226.18.6.18)
- `packet-time-to-live`: overrides the default time-to-live (TTL) setting for discovery multicast socket (default is 2)

Example 8–3 Configuring Discovery in the sessions.xml File

```
<cache-synchronization-manager>
    <clustering-service> … </clustering-service>
    <multicast-port>6020</multicast-port>
    <multicast-group-address>228.1.2.3</multicast-group-address>
    <packet-time-to-live>3</packet-time-to-live>
    ...
</cache-synchronization-manager>
```

> **Note:** When you select JMS as the transport mechanism in the `clustering-service` element, OracleAS TopLink ignores the discovery setting.

Name Service

A name service enables you to search for objects on remote caches. JNDI provides the name service for most applications, and offers the following optional elements to customize JNDI support in your application:

- `jndi-user-name`: user name value assigned to `Context.SECURITY_PRINCIPAL` property when looking up names in JNDI
- `jndi-password`: password value assigned to `Context.SECURITY_CREDENTIALS` property when looking up names in JNDI
- `naming-service-initial-context-factory-name`: the class used when creating initial context instances to use for looking up in JNDI
- `naming-service-url` - the URL to use when looking up through the naming service (value assigned to `Context.PROVIDER_URL` property in JNDI)

Not all servers require all four optional elements.

Example 8–4 Configuring JNDI Name Service in the sessions.xml File for WLS

```
<cache-synchronization-manager>
    <clustering-service> … </clustering-service>
    <naming-service-initial-context-factory-name>
        weblogic.jndi.WLInitialContextFactory
    </naming-service-initial-context-factory-name>
    <naming-service-url>t3://hostName:7001</naming-service-url>
</cache-synchronization-manager>
```

Using the Java Message Service

The JMS API is a protocol for communication that provides asynchronous communication between components in a distributed computing environment. Because OracleAS TopLink integrates with the JMS publish/subscribe mechanism, use JMS to improve the scalability of your cache synchronization.

For more information about the JMS API, see the JMS specification at

`http://java.sun.com/products/jms`

Preparing to use JMS You must configure a JMS service in the environment before OracleAS TopLink can leverage the service. To enable the service:

1. Configure a JMS connection factory and note the name. OracleAS TopLink uses the factory name to look-up the factory.
2. Configure a JMS topic and note the name. OracleAS TopLink uses the topic name to look-up the topic.
3. Configure the OracleAS TopLink `sessions.xml` file to use the factory and topic names.
4. Start the JMS service.

For more information on how to complete steps 1, 2, and 4, see the JMS service provider documentation.

For more information on how to complete step 3, see "Configuring JMS in sessions.xml" on page 8-11.

Example 8–5 illustrates a `jms.xml` configuration file for Oracle Application Server Containers for J2EE. Note that the host and port of the topic connection factory is the host and port of the JMS server hosting the topic, and not the host or port of the local JMS server.

Example 8–5 Example of the Oracle Application Server Containers for J2EE jms.xml File

```
<jms-server port="9128">
    <topic name="MyCacheSyncTopic" location="jms/MyCacheSyncTopic"/>
    <topic-connection-factory
      host="micky"
      port="9127"
      name="Cache Sync Topic Factory"
      location="jms/MyTopicFactory"
      password="password"
      username="admin"/>
    <log>
        <file path="../log/jms.log"/>
    </log>
</jms-server>
```

Configuring JMS in sessions.xml To configure JMS in the `sessions.xml` file, use the following optional elements:

- `jms-topic-connection-factory-name`: the JNDI name to use when looking up the connection factory for the JMS topic
- `jms-topic-name`: the JNDI name to use when looking up the JMS topic.

> **Note:** These elements are exclusive to JMS use only. Do not apply these elements when you when use a service other than JMS.

Example 8–6 JMS Entries in the sessions.xml File

```
<cache-synchronization-manager>
    <clustering-service>
        oracle.toplink.remote.jms.JMSClusteringService
    </clustering-service>
    <jms-topic-connection-factory-name>
        jms/MyTopicFactory
    </jms-topic-connection-factory-name>
    <jms-topic-name>jms/MyCacheSyncTopic</jms-topic-name>
    ...
</cache-synchronization-manager>
```

Note that JMS neither requires nor makes use of discovery.

Configuring JMS for Oracle Application Server Containers for J2EE When you use JMS in Oracle Application Server Containers for J2EE, set the naming service URL to the hostname of the JMS server hosting the topic. Example 8–7 illustrates this for an OracleAS TopLink session running in Oracle Application Server Containers for J2EE using JMS.

Example 8–7 Configuring OracleAS TopLink with JMS for Oracle Application Server Containers for J2EE

```
<cache-synchronization-manager>
    <clustering-service>
        oracle.toplink.remote.jms.JMSClusteringService
    </clustering-service>
    <jndi-user-name>admin</jndi-user-name>
    <jndi-password>password</jndi-password>
    <jms-topic-connection-factory-name>
        jms/MyTopicFactory
    </jms-topic-connection-factory-name>
    <jms-topic-name>jms/MyCacheSyncTopic</jms-topic-name>
    <naming-service-initial-context-factory-name>
        oracle.com.evermind.server.rmi.RMIInitialContextFactory
    </naming-service-initial-context-factory-name>
    <naming-service-url>ormi://micky</naming-service-url>
</cache-synchronization-manager>
```

Synchronous and Asynchronous Propagation

The Cache Synchronization Manager enables you to specify the propagation mode for your OracleAS TopLink application:

- If you send changes synchronously, the current transaction does not commit until OracleAS TopLink sends changes successfully to the other sessions in the system.
- If you send changes asynchronously, the transaction commits without waiting for OracleAS TopLink to propagate changes.

The optional `is-asynchronous` element controls the propagation mode, regardless of the transport used. By default, propagation occurs asynchronously.

Example 8–8 Configuring Propagation Mode

```
<cache-synchronization-manager>
    <clustering-service>...</clustering-service>
    <is-asynchronous>false</is-asynchronous>
```

```
    ...
</cache-synchronization-manager>
```

Error Handling

You can define error handlers to respond to raised exceptions. The `should-remove-connection-on-error` element (an optional sub-element of `cache-synchronization-manager`) specifies whether a connection to another session is discarded if an error occurs while sending an update. By default, OracleAS TopLink discards connections when errors occur.

Example 8–9 Configuring Error Handling

```
<cache-synchronization-manager>
    <clustering-service>...</clustering-service>
    <should-remove-connection-on-error>false</should-remove-connection-on-error>
    ...
</cache-synchronization-manager>
```

Explicit Query Refreshes

Some distributed systems require only a small number of objects to be consistent across the servers in the system. Conversely, other systems require that several specific objects must always be guaranteed to be up-to-date, regardless of the cost. If you build such a system, you can explicitly refresh selected objects from the database at appropriate intervals without incurring the full cost of distributed cache synchronization.

To implement this type of strategy:

1. Configure a set of queries that refresh the required objects.
2. Establish an appropriate refresh policy.
3. Invoke the queries as required to refresh the objects.

Refresh Policy

When you execute a query, if the required objects are in the cache, OracleAS TopLink returns the cached objects without checking the database for a more recent version. This reduces the number of objects that OracleAS TopLink must build from database results, and is optimal for non-clustered environments. However, this may not always be the best strategy for a clustered environment.

To override this behavior, set a refresh policy that specifies that the objects from the database always take precedence over objects in the cache. This updates the cached objects with the data from the database.

You can implement this type of refresh policy on each OracleAS TopLink descriptor, or just on certain queries, depending upon the nature of the application.

For more information about setting the refresh policy for a descriptor, see "Setting Descriptor Information," in the *Oracle Application Server TopLink Mapping Workbench User's Guide.*

For more information about setting the refresh policy for a query, see "Refresh" on page 6-66.

Note: Refreshing does not prevent phantom reads from occurring. See "Refreshing Finder Results" on page 6-98.

EJB Finders and Refresh Policy

When you invoke a `findByPrimaryKey` finder, if the object exists in the cache, OracleAS TopLink returns that copy. This is the default behavior, regardless of the refresh policy. To force a database query, you can configure the query to refresh by setting `refreshIdentityMapResult()` on it.

For more information about caching options, see "Caching Options" on page 6-97.

Remote Command Manager

The Remote Command Manager (RCM) enables OracleAS TopLink to send synchronization messages across the network to non-OracleAS TopLink applications. This feature is separate from the standard cache synchronization feature.

When you build a distributed system that includes both OracleAS TopLink and non-OracleAS TopLink applications, use the RCM in place of regular cache synchronization. Do not use RCM and regular OracleAS TopLink cache synchronization concurrently.

This section discusses the RCM, and offers information on:

- RCM Implementation Requirements
- RCM Structure
- RCM Channels

- Configuring the RCM
- Error Handling
- Guidelines for Using RCM
- Custom Remote Commands

RCM Implementation Requirements

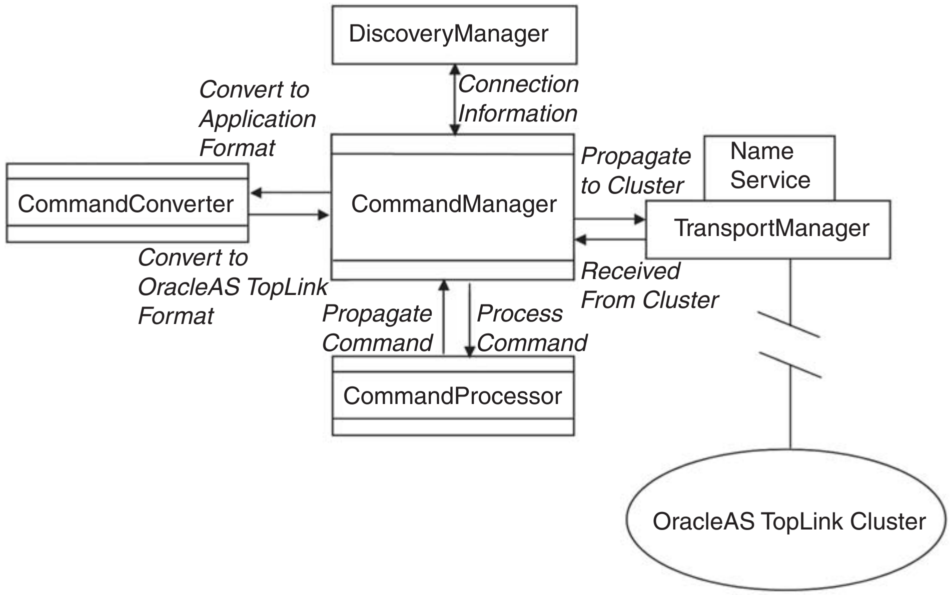
To enable RCM in a distributed system, enable RCM for all OracleAS TopLink sessions in the system. In addition, non-OracleAS TopLink applications must meet the following criteria to participate in cache synchronization through the RCM:

- The application must be a Java application or include a Java component.
- It must have access to a local JNDI service that supports remote access.
- The `toplink.jar` must be included in the application class path.
- You must configure the RCM in Java code for the application, and include the appropriate converter and processor components.

RCM Structure

The RCM is both modular and pluggable. Figure 8–2 illustrates the components of the RCM.

Figure 8–2 Remote Command Manager Components



RCM components include:

CommandManager The `CommandManager` is the central point of control for the system.

DiscoveryManager The `DiscoveryManager` dynamically maintains the membership of the OracleAS TopLink cluster.

TransportManager The `TransportManager` manages the transport level of the message exchange.

CommandProcessor The `CommandProcessor` interface sits between the RCM and the application. It is the main integration point for non-OracleAS TopLink applications.

CommandConverter An implementation of the `CommandConverter` translates commands between OracleAS TopLink and non-OracleAS TopLink applications. Regular OracleAS TopLink sessions do not require a `CommandConverter` implementation, because they do not require conversion.

Transmitting Commands From OracleAS TopLink with RCM

The process of initiating and transmitting commands from an OracleAS TopLink application is as follows:

1. Invoke the `getCommandManager()` accessor on the session to obtain a `CommandManager` interface.
2. Invoke the `CommandManager.propagateCommand(command)` method to initiate commands from the OracleAS TopLink session. Pass the command to be remotely executed as the `command` argument.
3. The `TransportManager` transmits the command to other members of the cluster.
4. If the receiving application is:
   - An OracleAS TopLink application, the OracleAS TopLink session executes the command.
   - A non-OracleAS TopLink application (or an application that does not use an OracleAS TopLink session), the application must provide implementation classes for the `CommandProcessor` and `CommandConverter` interfaces.

Using Commands on a Non-OracleAS TopLink Application

To send remote commands to the cluster, non-OracleAS TopLink applications invoke the `CommandManager.propagateCommand(command)` method. The application must provide a `CommandConverter` interface to convert the application-specific command format to an OracleAS TopLink Command object.

Likewise, when a non-OracleAS TopLink application receives an OracleAS TopLink command, it must implement a converter to translate the command for the `CommandManager`. To execute the command, a non-OracleAS TopLink application invokes the `processCommand(command)` method.

RCM Channels

The RCM passes remote commands along virtual channels. The RCM assigns each subscribing service a channel on which to send and receive commands, and all services assigned to a particular channel send (or publish) commands to that channel. Services also act as subscribers to their assigned channel, receiving all the commands published to that channel by other services.

You can assign any number of channels in the system without performance penalty, but any given service may only publish and subscribe to a single channel. You cannot reassign channels dynamically, or while discovery is active.

If you do not set a channel name, RCM assigns a default channel when you add services to the cluster. For example, if you do not set a channel name for any service instance you add to the system, all services subscribe to the same, default channel.

Configuring the RCM

Use the RCM API to configure the RCM. For OracleAS TopLink applications, create the cluster as part of the session initialization (for example: use a session `PreLogin` event when the session is initialized from the `sessions.xml` file). Note that neither the OracleAS TopLink Sessions Editor nor the `sessions.xml` file directly support RCM configuration.

The logical OracleAS TopLink cluster includes any number of OracleAS TopLink session-based applications, and non-OracleAS TopLink applications. Bind non-OracleAS TopLink applications in with OracleAS TopLink code to enable them to access the OracleAS TopLink commands.

Configuring the RCM for OracleAS TopLink Applications

To configure applications that use OracleAS TopLink sessions for RCM:

1. Create a Remote Command Manager implementation instance for the `CommandManager` interface. Pass the session as the `CommandProcessor` argument.

   For example:

   ```
   CommandManager rcm = new RemoteCommandManager(session);
   ```

2. To enable change set propagation between sessions, set the propagating option to true:

   ```
   session.setShouldPropagateChanges(true);
   ```

3. Set the URL that other RCM servers use to look up JNDI names in this Java virtual machine (JVM). For example, for Oracle Application Server Containers for J2EE, the URL can appear as follows:

   ```
   rcm.setUrl(“ormi://myHostname:23791/myDeployedApplication”);
   ```

   For a WebLogic Server, the URL can appear as follows:

```
rcm.setUrl("t3://myHostname:7001");
```

4. If you use Oracle Application Server Containers for J2EE, set a valid user and password. This enables the RCM services to look up remote names in JNDI. The user and password combination must be valid on all servers that participate in RCM.

   For example:

   ```
   rcm.getTransportManager().setUserName("admin");
   rcm.getTransportManager().setPassword("password");
   ```

5. If you are using WebLogic Server, leave the remote context properties empty.

   For example:

   ```
   rcm.getTransportManager().setRemoteContextProperties(
      new java.util.Hashtable());
   ```

6. (Optional) Set the `DiscoveryManager` parameters to custom multicast socket settings for your environment.

   For example:

   ```
   rcm.getDiscoveryManager().setMulticastGroupAddress("226.1.2.3");
   rcm.getDiscoveryManager().setMulticastPort(3122);
   ```

7. (Optional) Set the logical channel to assign a channel for the service.

   For example:

   ```
   rcm.setChannel("MyChannel");
   ```

8. (Optional) Set other RCM properties to customize the application.

   For example:

   ```
   rcm.setShouldPropagateAsynchronously(false);
   rcm.setShouldRemoveConnectionOnError(true);
   ```

Example 8–10 Enabling RCM on Oracle Application Server Containers for J2EE

```
CommandManager rcm = new RemoteCommandManager(session);
rcm.setUrl("ormi://ferengi:23791/orderEntryApp");
rcm.getTransportManager().setUserName("admin");
rcm.getTransportManager().setPassword("password");
session.setShouldPropagateChanges(true);
```

Example 8–11 Enabling RCM on the BEA WebLogic Server

```
CommandManager rcm = new RemoteCommandManager(session);
rcm.setUrl("t3://ferengi:7001");
rcm.getTransportManager().setRemoteContextProperties(
   new java.util.Hashtable());
session.setShouldPropagateChanges(true);
```

Configuring RCM for Non-OracleAS TopLink Applications

To configure RCM on applications that do not use the OracleAS TopLink sessions:

1. Create an application class to implement the `CommandProcessor` interface.

   For example:

   ```
   CommandProcessor processor = new ApplicationCommandProcessor();
   ```

1. Create a Remote Command Manager implementation instance for the `CommandManager` interface. Pass the session as the `CommandProcessor` argument:

   ```
   CommandManager rcm = new RemoteCommandManager(processor);
   ```

2. Create an application class to implement the `CommandConverter` interface and set an instance of the implementation class on the `CommandManager`.

   For example:

   ```
   CommandConverter converter = new ApplicationCommandConverter();
   rcm.setCommandConverter(converter);
   ```

3. If you are using WebLogic Server, leave the remote context properties empty.

   For example:

   ```
   rcm.getTransportManager().setRemoteContextProperties(
      new java.util.Hashtable());
   ```

4. (Optional) Set the `DiscoveryManager` parameters to custom multicast socket settings for your environment.

   For example:

   ```
   rcm.getDiscoveryManager().setMulticastGroupAddress("226.1.2.3");
   rcm.getDiscoveryManager().setMulticastPort(3122);
   ```

5. (Optional) Set the logical channel to assign a channel for the service.

   For example:

   ```
   rcm.setChannel("MyChannel");
   ```

6. (Optional) Set other RCM properties to customize the application.

   For example:

   ```
   rcm.setShouldPropagateAsynchronously(false);
   rcm.setShouldRemoveConnectionOnError(true);
   ```

7. Start the RCM service:

   ```
   rcm.initialize();
   ```

Example 8–12 Enabling RCM for a Non-OracleAS TopLink Application Using JNDI on Oracle Application Server Containers for J2EE

```
CommandManager rcm = new RemoteCommandManager(
   new ApplicationCommandProcessor());
rcm.setCommandConverter(
   new ApplicationCommandConverter());
rcm.setUrl("ormi://ferengi:23791/orderEntryApp");
rcm.getTransportManager().setUserName("admin");
rcm.getTransportManager().setPassword("password");
rcm.initialize();
```

Example 8–13 Enabling RCM for a Non-OracleAS TopLink Application Using JNDI on WebLogic Server

```
CommandManager rcm = new RemoteCommandManager(
   new ApplicationCommandProcessor());
rcm.setCommandConverter(
   new ApplicationCommandConverter());
rcm.setUrl("t3://ferengi:7001");
rcm.getTransportManager().setRemoteContextProperties(
   new java.util.Hashtable());
rcm.initialize();
```

Error Handling

Propagated commands often execute on multiple subscribing services. The subscribing services only return results to the publishing server if the command fails. The propagation mode affects error handling when a subscribing node reports a failure:

- In synchronous mode, the first remote command execution that fails raises a `RemoteCommandException` on the publishing service. The publishing service stops command propagation.
- In asynchronous mode, every server that fails raises a `RemoteCommandException` on the publishing service. Because the threads are asynchronous to the publishing server's thread, the exceptions are not raised within the context of the calling thread.

You can choose to catch and handle exceptions explicitly. The `CommandProcessor` interface includes the `handleException()` method for this purpose. Implement this method to catch exceptions thrown from a remote command service. For OracleAS TopLink applications, you can specify an exception handler on the session to handle the exception.

Raised exceptions are either:

- `CommunicationException`: thrown when a transport-level communications error occurs
- `RemoteCommandException`: thrown when any other problem occurs.

Guidelines for Using RCM

When you use RCM, consider the following:

- When you run Oracle Application Server Containers for J2EE, include the `-userThreads` command line option when you start the server. This enables the `DiscoveryManager` to initialize as a separate thread.
- When you deploy a single archive (for example: an EAR file) to multiple servers, implement one of the following on the host-specific Java code that configures the URL:
  - Use the java.net.InetAddress methods.
  - Define a system property on the command line to pass in the hostname or URL used by the RCM service.

- No transaction context is associated with remote command execution. The `CommandProcessor` interface must initiate its own transactions, and provide clean-up functionality in the case of failure.
- OracleAS TopLink applications must hook a server session as the `CommandProcessor` interface to an RCM. Do not use other types of sessions.

Custom Remote Commands

To create additional custom commands, extend the `oracle.toplink.remotecommand.Command` class, and implement the `executeWithSession(Session)` method. If the `CommandProcessor` interface is an OracleAS TopLink session, this method executes when the service executes.

You can pass instances of these commands to the `propagateCommand()` method, and publish them for execution on the remote services.

9

Packaging for Deployment

With your Oracle Application Server TopLink application built, you are ready to package and deploy the project to your enterprise. This chapter discusses:

- Introduction to Packaging and Deployment Concepts
- Creating OracleAS TopLink Deployment Files
- Packaging an OracleAS TopLink Application
- Hot Deployment of EJBs

This chapter discusses packaging and deployment from an OracleAS TopLink perspective. However, if you deploy your application to a J2EE container, you must configure elements of your application to enable OracleAS TopLink container support.

For more information, see also Appendix B, "Configuring OracleAS TopLink for J2EE Containers".

Introduction to Packaging and Deployment Concepts

This chapter introduces a basic approach to packaging that offers consistency across your projects, and the flexibility to work with projects of all kinds.

OracleAS TopLink Approach to Deployment

The OracleAS TopLink approach to deployment involves packaging application files into a single file, such as a Java archive (JAR) file, or an enterprise archive (EAR) file. This approach enables you to create clean and self-contained deployments that do not require significant file management.

After you create these files, you deploy the project.

OracleAS TopLink in an Enterprise Application

As an integral part of the enterprise application, OracleAS TopLink provides persistence and object-to-relational mapping functions. In most cases, the client does not interact with OracleAS TopLink directly; instead, clients access a client application that passes requests to OracleAS TopLink. As a result, there are two important steps to OracleAS TopLink deployment: make the packaged OracleAS TopLink application available; and add code to the client application to invoke OracleAS TopLink.

Road to Deployment

The goal of deployment is to provide the project to the client applications. Before you attempt to deploy an OracleAS TopLink application, complete the following:

1. Build the project elements, including beans, classes, and datasources.
2. Define the application mappings in the OracleAS TopLink Mapping Workbench.
3. Build the application deployment files. Use the OracleAS TopLink Mapping Workbench and the OracleAS TopLink Sessions Editor to create the files.
4. Package and deploy the application.
5. Add code to the client application to enable it to access the OracleAS TopLink application.

XML Versus Java Source Deployment

You can deploy the application mappings that you define in the OracleAS TopLink Mapping Workbench with your application as an XML file or as a compiled Java class. The OracleAS TopLink Mapping Workbench supports exporting for both of these formats.

The more traditional approach to deployment is to export Java source files from the OracleAS TopLink Mapping Workbench. It requires you to recompile the resulting Java files.

XML deployment files offer better flexibility both before and after deployment, and are easier to troubleshoot if a problem occurs. Because of this, in most cases, you should deploy your project using XML files rather than Java source files.

Creating OracleAS TopLink Deployment Files

The OracleAS TopLink Mapping Workbench provides the ability to create deployment files from a OracleAS TopLink Mapping Workbench project. After you build a project, you have two options to create the deployment files:

- Create XML deployment files that require no compiling. This approach gives you a very flexible configuration that enables you to make changes safely and easily. XML deployment files do not require third-party applications or compilers to deploy successfully.
- Create Java source files, which you compile and deploy outside of the OracleAS TopLink Mapping Workbench.

XML deployment is the preferred method of deployment, because XML files are easier to deploy and troubleshoot than compiled Java files.

This section discusses:

- XML Deployment Files
- Using Java Source Deployment Files
- Configuring Additional Files for CMP Deployment

XML Deployment Files

To deploy an OracleAS TopLink application, create a *project file,* in addition to one or more supporting files, as follows:

- If you deploy a non-EJB application, you require a session configuration file, known as the `sessions.xml` file.
- If you deploy EJBs to a J2EE container, you require the following entity bean deployment descriptors:
  - An `ejb-jar.xml` file that specifies standard EJB deployment properties
  - A J2EE container file that contains the properties specific to the J2EE container you use to deploy the application
  - A `toplink-ejb-jar.xml` that contains properties specific to OracleAS TopLink

Related beans share the same `ejb-jar.xml` file, J2EE container-specific file, and `toplink-ejb-jar.xml` file.

For more information, see "Container-Managed Persistence Applications" on page 9-18.

Project.xml File

The `project.xml` file is the core of your application. It contains the mappings and descriptors you define in the OracleAS TopLink Mapping Workbench, and also includes any named queries or finders associated with your project.

Because you must synchronize the `project.xml` file with the classes and database associated with your application, we recommend you not modify this file manually. The OracleAS TopLink Mapping Workbench ensures proper synchronization, and is the best way to make changes to the project. Simply modify the project in the OracleAS TopLink Mapping Workbench and redeploy the file `project.xml` file.

To redeploy a `project.xml` file, shut down and restart your OracleAS TopLink application.

Note: Because the `sessions.xml` file includes the name of the project file, you can save the project file with a name other than `project.xml`; however, for clarity, this discussion assumes that the file has not been renamed.

In addition to generating the deployment XML from the OracleAS TopLink Mapping Workbench, you can use either of the following methods and use the `DeploymentXMLGenerator` API:

Note: Before you use either method, ensure your the class path includes he `<ORACLE_HOME>\toplink\config` directory.

- From an application, instantiate the `DeploymentXMLGenerator` and your java source. Call the following method:

```
generate (<MW_Project.mwp>, <output file.xml>)
```

- From a command line, use:

```
java -classpath toplink.jar;toplinkmw.jar;xmlparserv2.jar;ejb.jar;.
oracle.toplink.workbench.external.api.DeploymentXMLGenerator <MW_
Project.mwp> <output file.xml>
```

Sessions.xml File

The `sessions.xml` file provides a simple and flexible way to configure, modify, and troubleshoot the application database sessions. Because of these attributes, the `sessions.xml` file is the preferred way to configure an OracleAS TopLink session.

The OracleAS TopLink Sessions Editor is a graphical tool to build and edit the `sessions.xml` file, but you can also use a text editor.

For more information about the OracleAS TopLink Sessions Editor, see "Understanding the OracleAS TopLink Sessions Editor" in the *Oracle Application Server TopLink Mapping Workbench User's Guide*.

For more information, see "Configuring Sessions with the sessions.xml File" on page 4-8.

Configuring the toplink-ejb-jar.xml File with the IBM WebSphere Server 4.0

The `toplink-ejb-jar.xml` file specifies all OracleAS TopLink-related information for an EJB entity bean deployment to a J2EE container. It includes several elements you use to configure the application.

The OracleAS TopLink deployment descriptor is included in the EJB JAR in the same META-INF directory as the `ejb-jar.xml`.

session The `session` element contains settings for the entire project. The `toplink-ejb-jar.xml` file must include a session section, which includes the following XML elements:

- `name`: A session name (unique among all deployed JARs) that is used as a key for the deployed OracleAS TopLink project (or the JAR that contains the project).
- `project-xml`: Specifies the name of the XML file that contains the OracleAS TopLink project metadata. Specify the fully qualified file name, including the `.xml` extension.
- The project deployment XML file can be stored either in the deployable JAR file at the root directory or on the file system.

> **Note:** If you wish, use a `project-class` element rather than a `project-xml` tag. With the `project-class` element, specify the fully-qualified name of the OracleAS TopLink project class. Include this class in the deployable JAR file. You can generate the project class either with the OracleAS TopLink Mapping Workbench or write it manually.

- `session-type`: The session type must always be set to `server-session`.
- `platform-class`: The platform class controls the format of the SQL generated and other database specific behavior.
- `uses-external-connection-pool` and `uses-external-transaction-controller`: For OracleAS TopLink to participate in WebSphere JTS transactions set both of these to TRUE.
- `external-transaction-controller-class`: This is the OracleAS TopLink server-specific JTS controller class required when using external transaction control. For WebSphere 4.0, use `oracle.toplink.jts.was.JTSExternalTransactionController_4_0`.
- `enable-logging`: When set to TRUE, OracleAS TopLink prints logging information for several of its operations. This is very useful for debugging.
- `logging-options`: Options for different levels of OracleAS TopLink logging.

For more information about the `toplink-was-ejb-jar_904.dtd`, see `<ORACLE_HOME>\toplink\config\dtds`.

Configuring the toplink-ejb-jar.xml File with the BEA WebLogic Server

The `toplink-ejb-jar.xml` file specifies all OracleAS TopLink-related information for an EJB entity bean deployment to a J2EE container. It includes several elements you use to configure the application.

The OracleAS TopLink deployment descriptor is included in the EJB JAR in the same META-INF directory as the `ejb-jar.xml`.

session The `session` element contains settings for the entire project. The `toplink-ejb-jar.xml` file must include a session section, which may include the following XML elements:

- `name`: Specifies the name of the session. Assign a unique session name to all projects deployed in a given server. This tag is mandatory.
- `project-class`: Specifies the name of the class that contains the OracleAS TopLink project metadata. Specify the fully qualified Java class name, but do not include the `.class` or `.java` extension.

  Use this tag (and not the `project-xml` tag) if you deploy your projects using exported and compiled Java code.
- `project-xml`: Specifies the name of the XML file that contains the OracleAS TopLink project metadata. Specify the fully qualified file name, including the `.xml` extension.

  Use this tag (and not the `project-class` tag) if you deploy your project using an exported XML file.
- `login`: Specifies the login parameters for the session. This element includes the sub elements listed in Table 9–1.

Table 9–1 login Elements

Element	Description
connection-pool	Identifies a JDBC pool for the current OracleAS TopLink project. The name of the pool must correspond to a JDBC connection pool specified in the WebLogic administration console. Specify a `connection-pool` or a `datasource` and `non-jts-datasource` to deploy entity beans.
datasource	Identifies JTA datasource for the current project. Use `datasource` in conjunction with `non-jts-datasource`. This provides an alternative to using a `connection-pool`. Use `datasource` to map to a JTA datasource, and `non-jts-datasource` to map to a non-JTS datasource. For more information about datasources, see "J2EE Integration" on page 7-44 and the J2EE container documentation.
non-jts-datasource	Identifies the read only datasource for the current project. Use `non-jts-datasource` in conjunction with datasource. This provides an alternative to using a `connection-pool`. For more information about datasources, see "J2EE Integration" on page 7-44 and the J2EE container documentation.
should-bind-all-parameters (optional)	Indicates whether all queries use parameter binding. Valid values are TRUE or FALSE. Default is FALSE.
uses-byte-array-binding (optional)	Indicates whether byte arrays are bound. Valid values are TRUE or FALSE. Default is FALSE.
uses-string-binding (optional)	Indicates whether strings are bound. Valid values are TRUE or FALSE. Default is FALSE.

- `cache-synchronization` (optional): This element indicates that changes made to one OracleAS TopLink cache in a cluster are automatically propagated to all other server caches. You can also include the optional sub elements listed in Table 9–2.

Table 9–2 Optional cache-synchronization Elements

Element	Description
is-asynchronous	Specifies whether synchronization should **NOT** wait until all sessions have been synchronized before returning. Valid values are TRUE or FALSE. Default is TRUE.
should-remove-connection-on-error	Specifies whether a synchronization connection is removed from the session if a communication error occurs. Valid values are TRUE or FALSE. Default is TRUE.

- `use-remote-relationships` (optional): OracleAS TopLink enables you to define relationships between beans in terms of their remote interfaces. This is especially useful when you port EJB 1.1 applications to EJB 2.0. When you enable this option, OracleAS TopLink defines all relationships in the JAR using remote interfaces. Valid values are TRUE or FALSE. Default is FALSE.

> **Note:** If you enable this option, your application is no longer strictly EJB 2.0 compliant, and your container may require some custom configuration. For example, when you deploy, run the `weblogic.ejbc` tool with the `-nocompliance` flag set.

- `customization-class` (optional): specifies the fully qualified name of a `DeploymentCustomization` class.

Using Java Source Deployment Files

Although XML deployment is the preferred deployment method, you can also deploy your OracleAS TopLink project as Java source files. To deploy a project as Java source files, create your project, and export the Java source files from the OracleAS TopLink Mapping Workbench. After you generate the files, compile them with an integrated development environment (IDE). This more traditional deployment method results in OracleAS TopLink applications with the following characteristics:

- They generally load more quickly than an XML-deployed project the first time they are loaded. They do not offer performance benefits after load time.
- Modifying session characteristics is a multi-step process that involves modifying the project in the OracleAS TopLink Mapping Workbench, recompiling the source files in an IDE, and redeploying the project.

In addition to generating the Java Source from the OracleAS TopLink Mapping Workbench, you can use either of the following methods and use the `JavaSourceGenerator` API:

> **Note:** Before you use either method, ensure your the class path includes he `<ORACLE_HOME>\toplink\config` directory.

- From an application, instantiate the `JavaSourceGenerator` and your java source. Call the method:

```
generate (<MW_Project.mwp>, <output file.xml>)
```

- From a command line, use:

```
java -classpath toplink.jar;toplinkmw.jar;xmlparserv2.jar;ejb.jar;.
oracle.toplink.workbench.external.api.JavaSourceGenerator  <MW_Project.mwp>
<output file.xml>
```

XML Files for Java Deployment

As with an XML deployment, a Java source deployment requires the `sessions.xml` file (for non-EJB applications) or EJB deployment descriptor files (for EJB projects). Build these files the same way you do for an XML deployment, and deploy it with your project.

For more information, see "Sessions.xml File" on page 9-5, and "Configuring the toplink-ejb-jar.xml File with the BEA WebLogic Server" on page 9-5.

Configuring Additional Files for CMP Deployment

If you deploy your application to a J2EE container that implements Container-managed Persistence (CMP), you may have to configure additional files to support the deployment. This section discusses:

- Configuring the ejb-jar.xml File
- Configuring the [J2EE-Container]-ejb-jar.xml

Configuring the ejb-jar.xml File

There is one `ejb-jar.xml` file for every JAR, although you can specify multiple beans in a single `ejb-jar.xml` file. The EJB specification you use determines the contents of this file.

Most IDEs provide facilities to create the `ejb-jar.xml` file. For more information about generating this file, see your IDE documentation.

If you build an EJB 2.0 application, the OracleAS TopLink Mapping Workbench can build the `ejb-jar.xml` file for you. Because the OracleAS TopLink Mapping Workbench can both read and write the `ejb-jar.xml`, you can either drive changes in the `ejb-jar.xml` file using the OracleAS TopLink Mapping Workbench:

- When you change the file manually outside of the OracleAS TopLink Mapping Workbench, re import the `ejb-jar.xml` file into the OracleAS TopLink Mapping Workbench project to refresh the project.
- When you change the OracleAS TopLink Mapping Workbench project, OracleAS TopLink Mapping Workbench updates the `ejb-jar.xml` file automatically when you save the project.

For more information about managing the `ejb-jar.xml` file in the OracleAS TopLink Mapping Workbench, see the *Oracle Application Server TopLink Mapping Workbench User's Guide.*

Configuring the *[J2EE-Container]*-ejb-jar.xml

The contents of the *`[J2EE-Container]`*`-ejb-jar.xml` file depends on the container to which you deploy your beans. To create this file, use the tools that accompany your container.

In most cases, the *`[J2EE-Container]`*`-ejb-jar.xml` file integrates with OracleAS TopLink without revision. However, when you deploy to a WebLogic Server container, modify the `weblogic-ejb-jar.xml`. The topics in this section explore the required modifications.

Configuring the *[J2EE-Container]*-ejb-jar.xml File for BEA WebLogic To deploy to a BEA WebLogic Server, modify the `webLogic-ejb-jar.xml` file. Within that file, each bean must have a `persistence-descriptor` entry with subentries, as follows:

- Configure the `persistence-descriptor` entry with subentries that indicate OracleAS TopLink is available and should be used:
  - If you deploy to WebLogic 6.1 (Service Pack 4), include a `persistence-type` element and a `persistence-use` element. Both elements require a `type-identifier` and a `type-version` tag. Table 9–3 lists the options for the `type-identifier` tag, and Table 9–4 lists the options for the `type-version` tag.

  - If you deploy to WebLogic 7.0 or 8.1, include a `persistence-use` element with a `type-identifier` and a `type-version` tag. Table 9–3 lists the options for the `type-identifier` tag, and Table 9–4 lists the options for the `type-version` tag.
- If you use WebLogic 6.1, add the element `type-storage` to the `persistence-type` element, and set it to `META-INF\toplink-ejb-jar.xml`.
- If you use WebLogic 7.0 or 8.1, add the element `type-storage` to the `persistence-use` element, and set it to `META-INF\toplink-ejb-jar.xml`.
- Set the `enable-call-by-reference` element to `TRUE` to enable *Call by Reference*:

```
<weblogic-enterprise-bean>
    <ejb-name>AccountBean</ejb-name>
    ...
        <enable-call-by-reference>True</enable-call-by-reference>
    ...
</weblogic-enterprise-bean>
```

Table 9–3 WebLogic type-identifier Settings

EJB Version	XML Elements
1.1	`<type-identifier>TopLink_CMP_1_1</type-identifier>`
2.0	`<type-identifier>TopLink_CMP_2_0</type-identifier>`

Table 9–4 WebLogic type-version Settings

WebLogic Version	XML Elements
6.1	`<type-version>4.0</type-version>`
7.0	`<type-version>4.5</type-version>`
8.1	`<type-version>9.0.4</type-version>`

Note: Although deprecated, the `type-version` setting of version 3.5 also functions correctly with WebLogic 6.1 (Service Pack 4) under EJB 1.1.

Unsupported weblogic-ejb-jar.xml File Tags The `weblogic-ejb-jar.xml` file includes several tags that OracleAS TopLink either does not support or does not require:

- `concurrency-strategy`: This tag specifies how WebLogic manages concurrent users for a given bean. Because OracleAS TopLink manages concurrent access internally, it does not require this element.

  For more information about OracleAS TopLink concurrency strategy, see "Locking Policy" on page 5-20.

- *db-is-shared*: Because OracleAS TopLink does not make any assumptions about the exclusivity of database access, OracleAS TopLink does not require this tag. OracleAS TopLink addresses multi-user access issues through various locking and refreshing policies.
- *delay-updates-until-end-of-tx*: OracleAS TopLink always delays updates until the end of a transaction, and does not require this tag.
- *finders-load-bean*: OracleAS TopLink always loads the bean upon execution of the finder, and does not require this tag.
- *pool*: OracleAS TopLink does not use a pooling strategy for entity beans. This avoids object-identity problems that can occur due to pooling.
- *lifecycle*: This element manages beans that follow a pooling strategy. Because OracleAS TopLink does not use a pooling strategy, OracleAS TopLink ignores this tag.
- *is-modified-method-name*: OracleAS TopLink does not require a bean developer-defined method to detect changes in object state.
- *isolation-level*: Because isolation level settings for the cache or database transactions are specified in the OracleAS TopLink project, OracleAS TopLink ignores this tag.
- *cache*: Because you define OracleAS TopLink cache properties in the OracleAS TopLink Mapping Workbench, this tag is necessary.

Packaging an OracleAS TopLink Application

The OracleAS TopLink approach to deployment involves packaging application files into a single file, such as a JAR file, or an EAR file. Each of the deployment strategies discussed in this section use this approach. The nature of the application also influences the approach you take to deploying the project. This section illustrates deployment strategies for:

- Java Applications
- Java Server Pages and Servlets Applications

- Session Bean Applications
- Container-Managed Persistence Applications
- Bean-Managed Persistence Applications

Java Applications

The OracleAS TopLink application does not use a J2EE container for deployment. Instead, it relies on OracleAS TopLink mechanisms to provide functionality and persistence. The key elements of this type of application are the lack of a J2EE container and the fact that you deploy the application by placing the application JAR on the class path.

Packaging the Java Application

You deploy Java applications simply by placing them on the class path. To follow the standard OracleAS TopLink approach of encapsulating applications in an archive, deploy the application in a JAR file, as follows:

1. Place the `sessions.xml` and `project.xml` files in the root of the JAR.
2. Include all mapped classes and any required helper classes in the JAR.
3. Place the completed JAR on the class path.

Deploying the Application to a Client

Build the JAR and place it on the class path. Include the following Java code in your client application to access the OracleAS TopLink application from a client:

```
Session mysession = SessionManager.getManager().getSession("[SESSION-NAME]");
```

Java Server Pages and Servlets Applications

Many designers build OracleAS TopLink applications that use Java server pages (JSPs) and Java servlets. This type of design generally supports Web-based applications.

Packaging Applications with JSPs and Servlets

When you build an application to deploy to the Web, package the application components in separate archives based on function. You can then assemble the separate archive files in a single deployment archive file.

The final deployment archive is an EAR file. If your client application includes application XML files, store those files in the `\meta-inf\` directory of the EAR. In addition, the EAR contains the following archive files:

A Domain JAR File The domain JAR contains the OracleAS TopLink files and domain objects required by the application, including:

- `sessions.xml`
- `project.xml` (or the compiled `project.class` file if you are not using XML files for deployment)
- The mapped classes required by the application, in a fully-resolved directory structure

When you create the JAR file, the JAR building utility automatically creates a directory structure within the JAR. Ensure that the `sessions.xml` file and the `project.xml` file (or `project.class` file) appear at the root of the JAR file. Also ensure that the class directory structure starts at the root of the JAR.

A Web Archive (WAR) File The WAR file contains the Web application files, including:

- JSPs and Servlets that provide the dynamic content for the client application
- Static HTML content for the client application
- Additional client application resources, such as images

To complete the WAR file, modify the `manifest.mf` file (located in the `\meta-inf` directory) to include a reference to the domain JAR file. The standard manifest is generally empty except for the header and two carriage returns.

Example 9–1 illustrates how to add a class path attribute.

Example 9–1 Modified manifest.mf File

```
Manifest-Version: 1.0
Created-By: 1.3.1 (Sun Microsystems Inc.)
// Add the following line
Class-Path: [Domain-Archive-Name].jar
// Two carriage returns to complete the file
[CR]
[CR]
```

Deploying the Application to a Client

After you build the WAR and JAR files, build them into an EAR file for deployment. To deploy the EAR to your JSP servlet server, copy the EAR to a well-known directory. You may also need to use server-specific deployment tools. For more information, see the server documentation.

Include the following Java code in your client application to access the OracleAS TopLink application from a client:

```
Session s = SessionManager.getManager().getSession("[SESSION-NAME]",[classloader]);
```

In most cases, `[classloader]` represents the class loader from the current thread context, specified as follows:

```
Thread.current().getContextClassLoader()
```

However, if your J2EE container does not support using this class loader, you can substitute the class loader from the current class, as follows:

```
this.getClass().getLoader()
```

Note: Oracle Application Server Containers for J2EE supports the use of the class loader from the current thread.

Session Bean Applications

Session beans generally model a process, operation, or service and as such are not persistent. You can build OracleAS TopLink applications that wrap interaction with OracleAS TopLink in session beans. Session beans execute all OracleAS TopLink-related operations on behalf of the client.

This type of design leverages JTS and externally managed transactions, but does not incur the overhead associated with CMP applications. Session bean applications also scale and deploy easily.

Packaging Applications with Session Beans

When you build an application to deploy to the Web, package the application components in separate archives based on function. You can then assemble the separate archive files in a single deployment archive file.

The final deployment archive is an EAR file. If your client application includes application XML files, store those files in the `\meta-inf\` directory of the EAR. In addition, the EAR contains the following archive files

A Domain JAR File The domain JAR contains the OracleAS TopLink files and domain objects required by the application, including:

- `sessions.xml` file
- `project.xml` file (or the compiled `project.class` file if you are not using XML files for deployment)
- mapped classes required by the application, in a fully-resolved directory structure

When you create the JAR file, the JAR building utility automatically creates a directory structure within the JAR. Ensure that the `sessions.xml` file and the `project.xml` file (or `project.class` file) appear at the root of the JAR file. Also ensure that the class directory structure starts at the root of the JAR.

An EJB JAR File The EJB JAR file specifically services the session beans in the application. It includes:

- The session bean home and remote for all session beans in the application
- Bean implementation code for all session beans in the application
- Any helper classes, such as amendment classes, required by the application
- Vendor-specific elements for the session beans
- The `ejb-jar.xml` file, stored in the `\meta-inf\` directory of the JAR

In addition, modify the `manifest.MF` file, found in the `\meta-inf\` directory, to include a reference to the domain JAR. The standard manifest is generally empty except for the header and two carriage returns.

Example 9–1 on page 9-15 illustrates how to add a class path attribute.

A WAR File The WAR file contains the Web application files, including:

- JSPs and Servlets that provide the dynamic content for the client application
- Static HTML content for the client application
- Additional client application resources, such as images

In addition, modify the `manifest.MF` file, found in the `\meta-inf\` directory, to include a reference to the domain JAR. The standard manifest is generally empty except for the header and two carriage returns.

Example 9–1 on page 9-15 illustrates how to add a class path attribute.

Deploying the Application to a Client

After you build the WAR and JAR files, build them into an EAR file for deployment. To deploy the EAR to your J2EE server, copy the EAR to a well-known directory. You may also need to use server-specific deployment tools. For more information, see the server documentation.

Include the following Java code in your client application to access the OracleAS TopLink application from a client:

```
Sessions = SessionManager.getManager().getSession("[SESSION-NAME]",[classloader]);
```

In most cases, `[classloader]` represents the class loader from the current thread context, specified as follows:

```
Thread.current().getContextClassLoader()
```

However, if your J2EE container does not support using this class loader, you can substitute the class loader from the current class, as follows:

```
this.getClass().getLoader()
```

Note: Oracle Application Server Containers for J2EE supports the use of the class loader from the current thread.

Container-Managed Persistence Applications

Many applications leverage the persistence mechanisms a J2EE container offers. OracleAS TopLink provides full support for this type of application.

The final deployment archive is an EAR file. If your client application includes application XML files, store those files in the `\meta-inf\` directory of the EAR. In addition, the EAR contains the following archive files:

An EJB JAR file The EJB JAR file specifically services the EJB entity beans in the application. It includes:

- The home and remote, and all implementation code for all mapped beans in the application
- All mapped non-EJB classes from the OracleAS TopLink Mapping Workbench project
- The home and remote, and all implementation code for any session beans included in the application
- Helper classes that contain OracleAS TopLink amendment methods, and any other classes the application requires

Store the following XML files in the `\meta-inf\` directory:

- `ejb-jar.xml` file
- *`[VENDOR-SPECIFIC]`*`-ejb-jar.xml` file
- `toplink-ejb-jar.xml` file
- `project.xml` file

> **Note:** If you do not use XML files for deployment, you do not have a `project.xml` file to include in the `\meta-inf\` directory. Instead, include the compiled `project.class` file in the appropriate directory structure in the EJB JAR.

A WAR File The WAR file contains the Web application files, including:

- JSPs and Servlets that provide the dynamic content for the client application
- Static HTML content for the client application
- Additional client application resources, such as images

General Deployment

After you build the WAR and JAR files, build them into an EAR file for deployment. To deploy the EAR to your J2EE server, copy the EAR to a well-known directory. You may also need to use server-specific deployment tools. For more information, see the server documentation.

Deploying the Application to BEA WebLogic Server

OracleAS TopLink CMP support includes integration for BEA WebLogic Server. To enable OracleAS TopLink CMP for WebLogic entity beans, use the WebLogic EJB Compiler (**ejbc**) to compile the EJB JAR, as follows:

- Run **ejbc** from the command line. Include the EJB JAR file as a command line argument. **ejbc** creates an EJB JAR that contains the original classes as well as all required generated classes and files.

When you run **ejbc**:

- It performs a partial EJB conformance check on the beans and their associated interfaces.
- It builds the internal BEA WebLogic classes that manage security and transactions, as well as the RMI stubs and skeletons that enable client access to the beans.
- OracleAS TopLink builds concrete bean subclasses and EJB finder method implementations.

For more information about running **ejbc**, see the BEA WebLogic documentation.

Troubleshooting ejbc When you start **ejbc**, it processes the data in a series of stages. If errors occur while running **ejbc**, attempt to determine which stage causes the problem. Common problems include:

- Bean classes that do not conform with the EJB specification
- Classes missing from the class path (all domain classes, required OracleAS TopLink classes, and all required BEA WebLogic classes must be on the class path)
- Java compiler (`javac`) problems, often caused by using an incorrect version of the JDK
- A failure when generating the RMI stubs and skeletons (a failure of `rmic`)

Tip: Use a command script (for example: a batch or ant script) to run **ejbc**. This enables you to pre-configure all the required variables for the command line and helps to prevent typing errors. Sample build scripts are available with the OracleAS TopLink Application Server Examples for BEA WebLogic.

For more information, see the OracleAS TopLink Examples at `<ORACLE_HOME>\toplink\doc\examples.htm`.

Deploying the Application to IBM WebSphere 4.*x* Server

OracleAS TopLink CMP support includes an integration for IBM WebSphere 4.*x* Server. Use the following procedure to deploy your application to WebSphere:

1. Use the OracleAS TopLink Deploy Tool for WebSphere to compile the EJB JAR file.

   For more information, see "Deploy Tool for WebSphere Server" on page A-17.

2. Start the WebSphere Administration Server.
3. Start the Administrator's Console and deploy the compiled JAR.

   For more information about deploying the JAR, see the IBM WebSphere documentation.

Note: When you deploy an application that contains an entity bean, set up a datasource and associate it with the bean. For more information about how to create and associate datasources, see the IBM WebSphere documentation.

It is not necessary to deploy the EJB JAR in WSAD, because deployment is carried out using the Deploy Tool (see "Deploy Tool for WebSphere Server" on page A-17).

Starting the Entity Bean You can start the bean in either the WebSphere Application Server or in WSAD.

To start the bean in IBM WebSphere Application Server:

1. Select the application that contains the entity beans.
2. Right click and choose **Start**.

A message dialog appears if the bean starts successfully. If an error occurs, consult Appendix C, "Error Codes and Messages" for troubleshooting information.

To start the bean in WSAD:

1. In WSAD, right click the EJB project and choose **Run on Server**.
2. To view the status of the process, open the **Console** tab of the Server view.

Bean-Managed Persistence Applications

OracleAS TopLink enables developers to leverage bean-managed persistence in their OracleAS TopLink applications. The OracleAS TopLink base class for the BMP entity beans implements the methods required for the EJB specification.

For more information about OracleAS TopLink BMP support, see "Overview of Bean-Managed Persistence" on page 3-57.

The final deployment archive is an EAR file. If your client application includes application XML files, store those files in the `\meta-inf\` directory of the EAR. In addition, the EAR contains the following archive files

An EJB JAR file The EJB JAR file specifically services the EJB entity beans in the application. It includes:

- The home and remote, and all implementation code for all mapped beans in the application
- All mapped non-EJB classes from the OracleAS TopLink Mapping Workbench project
- The home and remote, and all implementation code for any session beans included in the application
- Helper classes that contain OracleAS TopLink amendment methods, and any other classes the application requires

Store the following XML files as follows:

- the `ejb-jar.xml` file in the `\meta-inf\` directory
- the `sessions.xml` and the `project.xml` files in the root directory

> **Note:** If you do not use XML files for deployment, you do not have a `project.xml` file to include in the `\meta-inf\` directory. Instead, include the compiled `project.class` file in the appropriate directory structure in the EJB JAR.

A WAR File The WAR file contains the Web application files, including:

- JSPs and Servlets that provide the dynamic content for the client application
- Static HTML content for the client application
- Additional client application resources, such as images

Deploying the Application

After you build the WAR and JAR files, build them into an EAR file for deployment. To deploy the EAR to your J2EE server, copy the EAR to a well-known directory. You may also need to use server-specific deployment tools. For more information, see the server documentation.

Hot Deployment of EJBs

Many J2EE containers support *hot deployment*, a feature that enables you to deploy EJBs on a running server. Hot deployment allows you to:

- Deploy newly-developed EJBs to a running production system
- Remove (undeploy) deployed EJBs from a running server
- Modify (redeploy) the behavior of deployed EJBs by updating the bean class definition

When you take advantage of hot deployment, consider the following:

- You must deploy all related beans (all beans that share a common OracleAS TopLink project) within the same EJB JAR file. Because OracleAS TopLink views deployment on a project level, deploy all the project beans (rather than just a portion of them) to maintain consistency across the project.
- When you redeploy a bean, you automatically reset its OracleAS TopLink project. This flushes all object caches and rolls back any active object transactions associated with the project.

The client receives deployment exceptions when attempting to access undeployed or re-deployed bean instances. The client application must catch and handle the exceptions.

For more information about hot deployment, see the J2EE container documentation.

10

Tuning for Performance

Oracle Application Server TopLink applications are generally quite complex, and offer many opportunities for optimization. When you take an iterative approach to tuning, and you design your applications for peak efficiency, the result is an OracleAS TopLink application that is fast, smooth, and robust.

This chapter illustrates different methods to improve application performance. It discusses:

- Introduction to Tuning Concepts
- Profiling Performance
- General Tuning Tips
- Basic Performance Optimization
- OracleAS TopLink Reading Optimization Features
- OracleAS TopLink Writing Optimization Features
- Schema Optimization

Introduction to Tuning Concepts

The most important concept associated with tuning your OracleAS TopLink application is the idea of an iterative approach. The most effective way to tune your application is to:

- Use a profiling tool, such as the OracleAS TopLink Performance Profiler, to measure the application's performance.
- Modify application components.
- Measure performance again.

To identify the changes that improve your application performance, modify only one or two components at a time. You should also tune your application in a non-production environment before you deploy the application.

OracleAS TopLink as Part of a Larger Application

An OracleAS TopLink application is part of a larger application infrastructure that can include Web servers, external cache managers, external transactions controllers, and so on. To tune the OracleAS TopLink application most effectively, consider how the application interacts with the larger infrastructure, and include those considerations in performance testing.

An Effective Tuning Approach

To optimize performance, first check to see if a standard OracleAS TopLink feature addresses the problem you are trying to solve. The OracleAS TopLink documentation discusses the most common optimizations in the context of features they support. For example, "Query Object Performance Options" on page 6-70 offers information on how to improve query performance.

After you implement the basic optimizations, consider the more complex optimizations provided in this chapter, which include:

- General Tuning Tips
- Basic Performance Optimization
- OracleAS TopLink Reading Optimization Features
- OracleAS TopLink Writing Optimization Features
- Schema Optimization

Profiling Performance

The most important challenge to performance tuning is knowing what to optimize. To improve your application's performance, identify the areas of your application that do not operate at peak efficiency. The OracleAS TopLink Performance Profiler helps you identify performance problems.

The OracleAS TopLink Performance Profiler logs a summary of the performance statistics for every query you execute. The Profiler also logs a summary of all queries executed in a given session.

The Profiler logs the following information:

- Query class
- Domain class
- Total time, total execution time of the query (in milliseconds)
- Local time, the amount of time spent on the user's workstation (in milliseconds)
- Number of objects, the total number of objects affected
- Number of objects handled per second
- Logging, the amount of time spent printing logging messages (in milliseconds)
- SQL prepare, the amount of time spent preparing the SQL (in milliseconds)
- SQL execute, the amount of time spent executing the SQL (in milliseconds)
- Row fetch, the amount of time spent fetching rows from the database (in milliseconds)
- Cache, the amount of time spent searching or updating the object cache (in milliseconds)
- Object build, the amount of time spent building the domain object (in milliseconds)
- Query prepare, the amount of time spent to prepare the query prior to execution (in milliseconds)
- SQL generation, the amount of time spent to generate the SQL before it is sent to the database (in milliseconds)

Using the Profiler in the Web Client

The OracleAS TopLink Web Client also includes a graphical Performance Profiler.

For more information, see "Using the Performance Profiler" on page A-13.

Using the Profiler in Java

The Performance Profiler is an instance of the `PerformanceProfiler` class, found in `oracle.toplink.tools.profiler`. To access the Profiler, call the session's `getProfiler()` method.

To enable the Profiler, invoke the `setProfiler(new PerformanceProfiler())` method on the session. To end a profiling session, invoke the `clearProfiler()` method. The Profiler supports the following public API:

- `logProfile()`: enables the profiler
- `dontLogProfile()`: disables the profile
- `logProfileSummaryByQuery()`: organizes the profiler log as query summaries. This is the default profiler behavior.
- `logProfileSummaryByClass()`: organizes the profiler log as class summaries. This is an alternative to the default behavior implemented by `logProfileSummaryByQuery()`.

Example 10–1 Executing a Read Query with the Profiler

```
session.setProfiler(new PerformanceProfiler());
Vector employees = session.readAllObjects(Employee.class);
```

Example 10–2 Implementing the Performance Profiler in the sessions.xml File

```
<session>
    ...
    <profiler-class>oracle.toplink.tools.profiler.PerformanceProfiler</profiler-class>
    ...
</session>
```

Example 10–3 Performance Profiler Output

```
Begin Profile of{
ReadAllQuery(oracle.toplink.demos.employee.domain.Employee)Profile(ReadAllQuery,
# of obj=12, time=1399,sql execute=217, prepare=495, row fetch=390,
time/obj=116,obj/sec=8) */
```

```
} End Profile
```

The second line of the profile contains the following information about a query:

- `ReadAllQuery(oracle.toplink.demos.employee.domain.Employee)`: specific query profiled, and its arguments.
- `Profile(ReadAllQuery`: start of the profile and the type of query.
- `# of obj=12`: number of objects involved in the query.
- `time=1399`: total execution time of the query (in milliseconds).
- `sql execute=217`: total time spent preparing the SQL.
- `prepare=495`: total time spent preparing the SQL.
- `row fetch=390`: total time spent fetching rows from the database.
- `time/obj=116`: number of milliseconds spent on each object.
- `obj/sec=8) */`: number of objects handled per second.

Browsing the Profiler Results

To view profiler results, use the graphical Profile Browser. From your application code, launch the browser, located in the `oracle.toplink.tools.sessionconsole` package.

Example 10–4 Launching the Profile Browser

```
ProfileBrowser.browseProfiler(session.getProfiler());
```

General Tuning Tips

To substantially improve your application efficiency and throughput, Table 10–1 lists several tuning areas and offers tips to obtain the best performance from your OracleAS TopLink application.

Table 10–1 Tips for Building Efficient OracleAS TopLink Applications

Area	Recommendations	Related Information
General	Do not override OracleAS TopLink default behavior unless your application absolutely requires it. Because OracleAS TopLink default behavior is set for optimum results with the most common applications, the default is usually the most efficient choice for any given option. This is especially important for query or cache behavior.	
Mapping	Use indirection whenever possible, especially in cases where a class is normally used without its related objects.	See "Indirection" on page 3-6
Descriptors	Do not use `checkCacheThenDatabase` on descriptors unless required by the application. Query default behavior offers better performance.	See "Cache Usage" on page 6-61 See "Advanced Finder Options" on page 6-96
	Use `conformResults` on queries only when required. This avoids unnecessary resource overhead.	See "Validating a Unit of Work" on page 7-41 See "Cache Usage" on page 6-61 See "Advanced Finder Options" on page 6-96
Queries	If possible, use named queries in your application. Named queries help you avoid duplication, are easy to maintain and reuse, and easily add complex query behavior to the application.	See "Predefined Queries" on page 6-48
	Use parameterized SQL to improve write performance. Parameterized SQL improves performance by reusing the same prepared statement for multiple executions. This reduces overhead.	See "Binding and Parameterized SQL" on page 5-17 See "Parameterized SQL" on page 10-19
Sessions	Do not pool client sessions. Pooling sessions offers no performance gains.	See "Client Session" on page 4-6
	With JTA transactions, use `getActiveSession()` to access the active session for the current external transaction.	See "J2EE Integration" on page 7-44

Table 10–1 Tips for Building Efficient OracleAS TopLink Applications (Cont.)

Area	Recommendations	Related Information
	Use the OracleAS TopLink client session instead of remote session. client session is appropriate for most multi-user J2EE application server environments.	See "Client Session" on page 4-6 See "J2EE Integration" on page 7-44
Unit of Work	When you read objects, use the Unit of Work only when the objects returned from a query will be modified.	See "Transactions" on page 7-1
Cache	Tune the OracleAS TopLink cache for each class to help eliminate the need for distributed cache synchronization. Always tune these settings before implementing cache synchronization.	See "Setting Class Information" in the *Oracle Application Server TopLink Mapping Workbench User's Guide*
	Use Weak Cache for particularly volatile objects.	See "Working with Identity Maps" in the *Oracle Application Server TopLink Mapping Workbench User's Guide*
Cache Synchronization	Do not use distributed cache synchronization unless it is required by your application. Distributed cache synchronization offers performance benefits only in clustered environments in which several servers in the cluster regularly request and update the same objects.	See "Distributed Cache Synchronization" on page 8-6
	Use Java Message Service (JMS) for cache synchronization rather than Remote Method Invocation (RMI). JMS is more robust, easier to configure, and runs asynchronously. If you require synchronous cache synchronization, use RMI.	See "Distributed Cache Synchronization" on page 8-6
Code	Use the OracleAS TopLink Mapping Workbench rather than hand-coding. The OracleAS TopLink Mapping Workbench is easy to use, and implements many OracleAS TopLink features for you automatically.	

Table 10–1 Tips for Building Efficient OracleAS TopLink Applications (Cont.)

Area	Recommendations	Related Information
	Use instance or static variables to cache the results of resource intensive computations. If you use RMI or CORBA, avoid fine grain remote message sends.	

Basic Performance Optimization

Performance considerations are present at every step of the development cycle. Although this implies an awareness of performance issues in your design and implementation, it does not mean that you should expect to achieve the best possible performance in your first pass.

For example, if an optimization complicates the design, leave it until the final development phase. You should still plan for these optimizations from your first iteration, to make them easier to integrate later.

OracleAS TopLink provides a diverse set of features to optimize performance. You enable or disable most features in the descriptors or database session, making any resulting performance gains global.

OracleAS TopLink Reading Optimization Features

You can optimize certain read and write operations in an OracleAS TopLink application. To optimize reading, you can tune:

- The amount of data read from the database
- The way OracleAS TopLink queries data on the database

OracleAS TopLink provides the read optimization features listed in Table 10–2.

Table 10–2 Read Optimization Features

Feature	Function	Performance Technique
Unit of Work	Tracks object changes within the Unit of Work.	To minimize the amount of tracking required register only those objects that will change.
Object indirection	Uses valueholders as a stand-in for domain objects.	`Valueholders` can provide a major performance benefit, because they minimize database reads.

Table 10–2 Read Optimization Features (Cont.)

Feature	Function	Performance Technique
Soft cache weak identity map	Offers client-side caching for objects read from database, and drops objects from the cache when memory becomes low.	Reduces database calls and improves memory performance.
Weak identity map	Offers client-side caching for objects.	Reduces database access and maintains a cache of all referenced objects.
Full identity map	Offers client side caching for objects.	Avoids database calls for objects that have already been read. Limit the cache size. A large cache can impact system performance.
Cache identity map	Offers a fixed size client side cache.	Leverages a moderate caching strategy, and controls the impact on memory.
No identity map	Disables cache lookup.	Useful if you prefer database access over cached objects.
Batch reading and joining	Reduces database access by batching many queries into a single query that reads more data.	Dramatically reduces the number of database accesses required to perform a READ query.
Partial object reading	Allows reading of a subset of a result set of the object's attributes.	Reduces the amount of data read from the database at any one time. Reducing connection time for each read improves performance.
Report query	Similar to partial object reading, but returns only the data instead of the objects.	Supports complex reporting functions such as aggregation and group-by functions. Also enables you to compute complex results on the database, instead of reading the objects into the application and computing the results locally.

Reading Case 1: Displaying Names in a List

An application may ask the user to choose an element from a list. Because the list displays only a subset of the information contained in the objects, it is not necessary to query for all information for objects from the database.

Partial object reading and report query are two OracleAS TopLink features that optimize these types of operations. They enable you to query only the information required to display the list. The user can then select an object from the list.

Partial Object Reading

Partial object reading is a query designed to extract only the required information from a selected record in a database, rather than all the information the record contains. Because partial object reading does not fully populate objects, you can neither cache nor edit partially-read objects. Also note that the primary key is required to re-query the object (so it can be edited, for example). OracleAS TopLink does not automatically include the primary key information in a partially populated object. If you want to edit the object, specify the primary key as a required partial attribute.

In Example 10–5, the query builds complete employee objects, even though the list displays only employee last names. With no optimization, the query reads employee data.

Example 10–5 No Optimization

```
/* Read all the employees from the database, ask the user to choose one and
return it. This must read in all the information for all the employees.*/
List list;

// Fetch data from database and add to list box.
Vector employees = (Vector) session.readAllObjects(Employee.class);
list.addAll(employees);

// Display list box.
....

// Get selected employee from list.
Employee selectedEmployee = (Employee) list.getSelectedItem();

return selectedEmployee;
```

Example 10–6 demonstrates the use of partial object reading. It reads only the last name and primary key for the employees. This reduces the amount of data read from the database.

Example 10–6 Optimization Through Partial Object Reading

```
/* Read all the employees from the database, ask the user to choose one and
```

```
return it. This uses partial object reading to read just the last name of the
employees. Note that OracleAS TopLink does not automatically include the primary
key of the object. If this is needed to select the object for a query, it must
be specified as a partial attribute so that it can be included. In this way, the
object can easily be read for editing. */
List list;
// Fetch data from database and add to list box.
ReadAllQuery query = new ReadAllQuery(Employee.class);
query.addPartialAttribute("lastName");
/* OracleAS TopLink does not automatically include the primary key of the
object. If this is needed to select the object for a query, it must be specified
as a partial attribute so that it can be included.*/
query.addPartialAttribute("id");
// The next line avoids a query exception
query.dontMaintainCache();
Vector employees = (Vector) session.executeQuery(query);
list.addAll(employees);

// Display list box.
....
// Get selected employee from list.
Employee selectedEmployee =
(Employee)session.readObject(list.getSelectedItem());
return selectedEmployee;
```

ReportQuery

Report query enables you to retrieve data from a set of objects and their related objects. Report query supports database reporting functions and features.

For more information, see "ReportQuery" on page 6-73.

Example 10–7 demonstrates the use of report query to read only the last name of the employees. This reduces the amount of data read from the database compared to the code in Example 10–5, and avoids instantiating employee instances.

Example 10–7 Optimization Through Report Query

```
/* Read all the employees from the database, ask the user to choose one and
return it. This uses the report query to read just the last name of the
employees. It then uses the primary key stored in the report query result to
read the real object.*/
List list;
// Fetch data from database and add to list box.
ExpressionBuilder builder = new ExpressionBuilder();
```

```
ReportQuery query = new ReportQuery (Employee.class, builder);
query.addAttribute("lastName");
query.retrievePrimaryKeys();
Vector reportRows = (Vector) session.executeQuery(query);
list.addAll(reportRows);

// Display list box.
....

// Get selected employee from list.
ReportQueryResult result = (ReportQueryResult) list.getSelectedItem();
Employee selectedEmployee = (Employee)
   result.readobject(Employee.Class,session);
```

Although the differences between the unoptimized example (Example 10–5) and the report query optimization in Example 10–7 appear to be minor, report queries offer a substantial performance improvement.

Reading Case 2: Batch Reading Objects

The way your application reads data from the database affects performance. For example, reading a collection of rows from the database is significantly faster than reading each row individually.

A common performance challenge is to read a collection of objects that have a one-to-one reference to another object. This normally requires one read operation to read in the source rows, and one call for each target row in the one-to-one relationship.

To reduce the number of reads required, use join and batch reading. Example 10–8 illustrates the unoptimized code required to retrieve a collection of objects with a one-to-one reference to another object. Example 10–9 and Example 10–10 illustrate the use of joins and batch reading to improve efficiency.

Example 10–8 No Optimization

```
/*Read all the employees, and collect their address' cities. This takes N + 1
queries if not optimized. */

// Read all the employees from the database. This requires 1 SQL call.
Vector employees = session.readAllObjects(Employee.class,new
   ExpressionBuilder().get("lastName").equal("Smith"));

//SQL: Select * from Employee where l_name = 'Smith'
```

```
// Iterate over employees and get their addresses.
// This requires N SQL calls.
Enumeration enum = employees.elements();
Vector cities = new Vector();
while(enum.hasMoreElements()) Employee employee = (Employee) enum.nextElement();
   cities.addElement(employee.getAddress().getCity());

//SQL: Select * from Address where address_id = 123, etc }
```

Example 10–9 Optimization Through Joining

```
/* Read all the employees, and collect their address' cities. Although the code
is almost identical because joining optimization is used it only takes 1 query.
*/

// Read all the employees from the database, using joining. This requires 1 SQL
call.
ReadAllQuery query = new ReadAllQuery();
query.setReferenceClass(Employee.class);
query.setSelectionCriteria(new
   ExpressionBuilder().get("lastName").equal("Smith"));
query.addJoinedAttribute("address");
Vector employees = session.executeQuery(query);

// SQL: Select E.*, A.* from Employee E, Address A where E.l_name = 'Smith' and
E.address_id = A.address_id Iterate over employees and get their addresses. The
previous SQL already read all the addresses so no SQL is required.
Enumeration enum = employees.elements();
Vector cities = new Vector();
while (enum.hasMoreElements()) {
Employee employee = (Employee) enum.nextElement();
    cities.addElement(employee.getAddress().getCity());
```

Example 10–10 Optimization Through Batch Reading

```
/* Read all the employees, and collect their address' cities. Although the code
is almost identical because batch reading optimization is used it only takes 2
queries. */

// Read all the employees from the database, using batch reading. This requires
1 SQL call, note that only the employees are read.
ReadAllQuery query = new ReadAllQuery();
query.setReferenceClass(Employee.class);
```

```
query.setSelectionCriteria(new
   ExpressionBuilder().get("lastName").equal("Smith"));
query.addBatchReadAttribute("address");
Vector employees = (Vector)session.executeQuery(query);

// SQL: Select * from Employee where l_name = 'Smith'

// Iterate over employees and get their addresses.
// The first address accessed will cause all the addresses to be read in a
single SQL call.
Enumeration enum = employees.elements();
Vector cities = new Vector();
while (enum.hasMoreElements()) {
    Employee employee = (Employee) enum.nextElement();
    cities.addElement(employee.getAddress().getCity());
    // SQL: Select distinct A.* from Employee E, Address A
      where E.l_name = 'Smith' and E.address_id = A.address_i
}
```

Because the two-phase approach to the query (Example 10–9 and Example 10–10) accesses the database only twice, it is significantly faster than the approach illustrated in Example 10–8.

Joins offer a significant performance increase under most circumstances. Batch reading offers further performance advantage in that it allows for delayed loading through valueholders, and has much better performance where the target objects are shared.

For example, if employees in Example 10–8, Example 10–9, and Example 10–10 live at the same address, batch reading reads much less data than joining, because batch reading uses a SQL `DISTINCT` call to filter duplicate data. Batch reading is also available for one-to-many relationships, but joining is available only for one-to-one relationships.

Reading Case 3: Using Complex Custom SQL Queries

OracleAS TopLink provides a high-level query mechanism. However, if your application requires a complex query, a direct SQL call may be the best solution.

For more information about executing SQL calls, see "Custom SQL" on page 6-27.

Reading Case 4: Using View Objects

Some application operations require information from several objects rather than from just one. This can be difficult to implement, and resource intensive. Example 10–11 illustrates unoptimized code that reads information from several objects.

Example 10–11 No Optimization

```
/* Gather the information to report on an employee and return the summary of the
information. In this situation a hashtable is used to hold the report
information. Notice that this reads a lot of objects from the database, but uses
very little of the information contained in the objects. This may take 5 queries
and read in a large number of objects.*/


public Hashtable reportOnEmployee(String employeeName)
    {
    Vector projects, associations;
    Hashtable report = new Hashtable();
    // Retrieve employee from database.
    Employee employee = session.readObject(Employee.class, new
       ExpressionBuilder.get("lastName").equal(employeeName));
    // Get all the projects affiliated with the employee.
    projects = session.readAllObjects(Project.class, "SELECT P.* FROM PROJECT P,
      EMPLOYEE E WHERE P.MEMBER_ID = E.EMP_ID AND E.L_NAME = " + employeeName);
    // Get all the associations affiliated with the employee.
    associations = session.readAllObjects(Association.class, "SELECT A.*
      FROM ASSOC A, EMPLOYEE E WHERE A.MEMBER_ID = E.EMP_ID AND E.L_NAME =
      " + employeeName);
}
    report.put("firstName", employee.getFirstName());
    report.put("lastName", employee.getLastName());
    report.put("manager", employee.getManager());
    report.put("city", employee.getAddress().getCity());
    report.put("projects", projects);
    report.put("associations", associations);
    return report;
}
```

To improve application performance in these situations, define a new read-only object to encapsulate this information, and map it to a view on the database. To set the object to be read-only, use the `addDefaultReadOnlyClass()` API in the `oracle.toplink.sessions.Project` class.

Example 10–12 Optimization Through View Object

```
CREATE VIEW NAMED EMPLOYEE_VIEW AS (SELECT F_NAME = E.F_NAME, L_NAME = E.L_
NAME,EMP_ID = E.EMP_ID, MANAGER_NAME = E.NAME, CITY = A.CITY, NAME = E.NAME
FROM EMPLOYEE E, EMPLOYEE M, ADDRESS A
WHERE E.MANAGER_ID = M.EMP_ID
AND E.ADDRESS_ID = A.ADDRESS_ID)
```

Define a descriptor for the `EmployeeReport` class:

- Define the descriptor normally, but specify `tableName` as `EMPLOYEE_VIEW`.
- Map only the attributes required for the report. In the case of `numberOfProjects` and associations, use a transformation mapping to retrieve the required data.

You can now query the report from the database like any other OracleAS TopLink-enabled object.

Example 10–13 View the Report from Example 10–12

```
/* Return the report for the employee.*/
public EmployeeReport reportOnEmployee(String employeeName)
{
    EmployeeReport report;
    report = (EmployeeReport) session.readObject(EmployeeReport.class,
      new ExpressionBuilder.get("lastName").equal(employeeName));
    return report;}
```

OracleAS TopLink Writing Optimization Features

Table 10–3 lists OracleAS TopLink's write optimization features.

Table 10–3 Write Optimization Features

Feature	Effect on Performance
Unit of Work	Improves performance by updating only the changed fields and objects. Minimizes the amount of tracking required (which can be expensive) by registering only those objects that will change. **Note**: The Unit of Work supports marking classes as read-only. This avoids tracking of objects that do not change.

Table 10–3 Write Optimization Features (Cont.)

Feature	Effect on Performance
Parameterized SQL	Improves performance for frequently executed SQL statements.
Batch writing	Allows you to group all insert, update, and delete commands from a transaction into a single database call. This dramatically reduces the number of calls to the database.
Sequence number preallocation	Dramatically improves insert performance.
Does exist alternatives	The *does exist* call on write object can be avoided in certain situations by checking the cache for *does exist*, or assuming the existence of the object.

Writing Case 1: Batch Writes

The most common write performance problem occurs when a batch job inserts a large volume of data into the database. For example, consider a batch job that loads a large amount of data from one database, and then migrates the data into another. The objects involved:

- Are simple individual objects with no relationships
- Use generated sequence numbers as their primary key
- Have an address that also uses a sequence number

The batch job loads 10,000 employees from the first database and inserts them into the target database. With no optimization, the batch job reads all the records from the source database, acquires a Unit of Work from the target database, registers all objects, and commits the Unit of Work.

Example 10–14 No Optimization

```
/* Read all the employees, acquire a Unit of Work and register them. */

// Read all the employees from the database. This requires 1 SQL call, but will
be very memory intensive as 10,000 objects will be read.
Vector employees = sourceSession.readAllObjects(Employee.class);

//SQL: Select * from Employee

// Acquire a Unit of Work and register the employees.
UnitOfWork uow = targetSession.acquireUnitOfWork();
uow.registerAllObjects(employees);
```

```
uow.commit();

//SQL: Begin transaction
//SQL: Update Sequence set count = count + 1 where name = 'EMP'
//SQL: Select count from Sequence
//SQL: ... repeat this 10,000 times + 10,000 times for the addresses ...
//SQL: Commit transaction
//SQL: Begin transaction
//SQL: Insert into Address (...) values (...)
//SQL: ... repeat this 10,000 times
//SQL: Insert into Employee (...) values (...)
//SQL: ... repeat this 10,000 times
//SQL: Commit transaction}
```

This batch job performs poorly, because it requires 60,000 SQL executions. It also reads huge amounts of data into memory, which can raise memory performance issues. OracleAS TopLink offers several optimization features to improve the performance of this batch job.

To improve this operation:

1. Leverage OracleAS TopLink batch reads and cursor support.
2. Implement sequence number preallocation.
3. Use batch writing to write to the database.

   If your database does not support batch writing, use parameterized SQL to implement the write query.
4. Implement multiprocessing.

Cursors and Batch Writes

To optimize the query in Example 10–14, use a cursored stream to read the employees from the source database. You can also employ a cache identity map rather than a full identity map in both the source and target databases.

To address the potential for memory problems, use the `releasePrevious()` method after each read to stream the cursor in groups of 100. Register each batch of 100 employees in a new Unit of Work and commit them.

Although this does not reduce the amount of executed SQL, it does address potential out-of-memory issues. When your system runs out of memory, the result is performance degradation that increases over time, and excessive disk activity caused by memory swapping on disk.

Sequence Number Preallocation

SQL select calls are more resource-intensive than SQL modify calls, so you can realize large performance gains by reducing the number of select calls you issue. The code in Example 10–14 uses the select calls to acquire sequence numbers. You can substantially improve performance if you use sequence number preallocation.

In OracleAS TopLink, you can configure the sequence preallocation size on the login object (the default size is 50). Example 10–14 uses a preallocation size of 1 to demonstrate this point. If you stream the data in batches of 100 as suggested in "Cursors and Batch Writes", set the sequence preallocation size to 100. Because employees and addresses in the example both use sequence numbering, you further improve performance by letting them share the same sequence. If you set the preallocation size to 200, this reduces the number of SQL execution from 60,000 to 20,200.

Batch Writing

Batch writing enables you to combine a group of SQL statements into a single statement and send it to the database as a single database execution. This feature reduces the communication time between the application and the server, and substantially improves performance.

You can enable batch writing on the login object with the `useBatchWriting()` method. If you add batch writing to Example 10–14, you execute each batch of 100 employees as a single SQL execution. This reduces the number of SQL execution from 20,200 to 300.

Parameterized SQL

OracleAS TopLink supports parameterized SQL and prepared statement caching. Using parameterized SQL improves write performance, because it avoids the prepare cost of a SQL execution.

You cannot use batch writing and parameterized SQL together, because batch writing does not use individual statements. Because the performance benefits of batch writing are much greater than those of parameterized SQL, use batch writing if it is supported by your database.

Parameterized SQL avoids the prepare component of SQL execution, but does not reduce the number of executions. Because of this, it normally offers only moderate performance gains. However, if your database does not support batch writing, parameterized SQL improves performance. If you add parameterized SQL in Example 10–14, you must still execute 20,200 SQL executions, but parameterized SQL reduces the number of SQL PREPAREs to 4.

Multiprocessing

You can use multiple processes or multiple machines to split the batch job into several smaller jobs. In this example, splitting the batch job across threads enables you to synchronize reads from the cursored stream, and use parallel Units of Work on a single machine.

This leads to a performance increase, even if the machine has only a single processor, because it takes advantage of the wait times inherent in SQL execution. While one thread waits for a response from the server, another thread uses the waiting cycles to process its own database operation.

Example 10–15 illustrates the optimized code for this example. Note that it does not illustrate multiprocessing.

Example 10–15 Fully Optimized

```
/* Read each batch of employees, acquire a Unit of Work and register them. */
targetSession.getLogin().useBatchWriting();
targetSession.getLogin().setSequencePreallocationSize(200);

// Read all the employees from the database, into a stream. This requires 1 SQL
call, but none of the rows will be fetched.
ReadAllQuery query = new ReadAllQuery();
query.setReferenceClass(Employee.class);
query.useCursoredStream();
CursoredStream stream;
stream = (CursoredStream) sourceSession.executeQuery(query);
//SQL: Select * from Employee. Process each batch
while (! stream.atEnd()) {
    Vector employees = stream.read(100);
    // Acquire a Unit of Work to register the employees
    UnitOfWork uow = targetSession.acquireUnitOfWork();
    uow.registerAllObjects(employees);
    uow.commit();
}
//SQL: Begin transaction
//SQL: Update Sequence set count = count + 200 where name = 'SEQ'
//SQL: Select count from Sequence where name = 'SEQ'
//SQL: Commit transaction
//SQL: Begin transaction
//BEGIN BATCH SQL: Insert into Address (...) values (...)
//... repeat this 100 times
//Insert into Employee (...) values (...)
//... repeat this 100 times
//END BATCH SQL:
```

```
//SQL: Commit transactionJava optimization
```

Schema Optimization

Optimization is an important consideration when you design your database schema and object model. Most performance issues occur when the object model or database schema is too complex, which can make the database slow and difficult to query. This is most likely to happen if you derive your database schema directly from a complex object model.

To optimize performance, we recommend you design the object model and database schema together however, ensure there is no direct one-to-one correlation between the two.

Schema Case 1: Aggregation of Two Tables into One

A common schema optimization technique is to aggregate two tables into a single table. This improves read and write performance by requiring only one database operation instead of two.

Table 10–4 and Table 10–5 illustrate the table aggregation technique.

Table 10–4 Original Schema

Elements	Details
Title	ACME Member Location Tracking System
Classes	Member, Address
Tables	MEMBER, ADDRESS
Relationships	Source, Instance Variable, Mapping, Target, Member, address, one-to-one, Address

The nature of this application dictates that developers always look up employees and addresses together. As a result, querying a member based on address information requires a database join, and reading a member and its address requires two read statements. Writing a member requires two write statements. This adds unnecessary complexity to the system, and results in poor performance.

A better solution is to combine the MEMBER and ADDRESS tables into a single table, and change the one-to-one relationship to an aggregate relationship. This enables you to read all information with a single operation, and doubles the speed of updates and inserts, because they must modify only a single row in one table.

Table 10–5 Optimized Schema

Elements	Details
Classes	Member, Address
Tables	MEMBER
Relationships	Source, Instance Variable, Mapping, Target, Member, address, aggregate, Address

Schema Case 2: Splitting One Table into Many

To improve overall performance of the system, split large tables into two or more smaller tables. This significantly reduces the amount of data traffic required to query the database.

For example, the system illustrated in Table 10–6 assigns employees to projects within an organization. The most common operation reads a set of employees and projects, assigns employees to projects, and update the employees. The employee's address or job classification is also occasionally used to determine the project on which the employee is placed.

Table 10–6 Original Schema

Elements	Details	Instance Variable	Mapping	Target
Title	ACME Employee Workflow System			
Classes	Employee, Address, PhoneNumber, EmailAddress, JobClassification, Project			
Tables	EMPLOYEE, PROJECT, PROJ_EMP			
Relationships	Employee	address	aggregate	Address
	Employee	phoneNumber	aggregate	EmailAddress
	Employee	emailAddress	aggregate	EmailAddress

Table 10–6 Original Schema (Cont.)

Elements	Details	Instance Variable	Mapping	Target
	Employee	job	aggregate	JobClassification
	Employee	projects	many-to-many	Project

When you read a large volume of employees from the database, you must also read their aggregate parts. Because of this, the system suffers from general read performance issues. To resolve this, break the EMPLOYEE table into the EMPLOYEE, ADDRESS, PHONE, EMAIL, and JOB tables, as illustrated in Table 10–7.

Because you normally read only the employee information, splitting the table reduces the amount of data transferred from the database to the client. This improves your read performance by reducing the amount of data traffic by 25%.

Table 10–7 Optimized Schema

Elements	Details	Instance Variable	Mapping	Target
Title	ACME Employee Workflow System			
Classes	Employee, Address, PhoneNumber, EmailAddress, JobClassification, Project			
Tables	EMPLOYEE, ADDRESS, PHONE, EMAIL, JOB, PROJECT, PROJ_EMP			
Relationships	Employee	address	one-to-one	Address
	Employee	phoneNumber	one-to-one	EmailAddress
	Employee	emailAddress	one-to-one	EmailAddress
	Employee	job	one-to-one	JobClassification
	Employee	projects	many-to-many	Project

Schema Case 3: Collapsed Hierarchy

A common mistake when you transform an object oriented design into a relational model, is to build a large hierarchy of tables on the database. This makes querying difficult, because queries against this type of design can require a large number of joins. It is usually a good idea to collapse some of the levels in your inheritance hierarchy into a single table.

Table 10–8 represents a system that assigns clients to a company's sales representatives. The managers also track the sales representatives that report to them.

Table 10–8 Original Schema

Elements	**Details**
Title	ACME Sales Force System
Classes	Tables
Person	PERSON
Employee	PERSON, EMPLOYEE
SalesRep	PERSON, EMPLOYEE, REP
Staff	PERSON, EMPLOYEE, STAFF
Client	PERSON, CLIENT
Contact	PERSON, CONTACT

The system suffers from complexity issues that hinder system development and performance. Nearly all queries against the database require large, resource intensive joins. If you collapse the three-level table hierarchy into a single table, as illustrated in Table 10–9, you substantially reduce system complexity. You eliminate joins from the system, and simplify queries.

Table 10–9 Optimized Schema

Elements	**Details**
Classes	Tables
Person	none
Employee	EMPLOYEE
SalesRep	EMPLOYEE

Table 10–9 Optimized Schema (Cont.)

Elements	Details
Staff	EMPLOYEE
Client	CLIENT
Contact	CLIENT

Schema Case 4: Choosing One Out of Many

In a one-to-many relationship, a single source object has a collection of other objects. In some cases, the source object frequently requires one particular object in the collection, but requires the other objects only infrequently. You can reduce the size of the returned result set in this type of case by adding an instance variable for the frequently required object. This enables you to access the object without instantiating the other objects in the collection.

Table 10–10 represents a system by which an international shipping company tracks the location of packages in transit. When a package moves from one location to another, the system creates a new a location entry for the package in the database. The most common query against any given package is for its current location.

Table 10–10 Original Schema

Elements	Details	Instance Variable	Mapping	Target
Title	ACME Shipping Package Location Tracking System			
Classes	Package, Location			
Tables	PACKAGE, LOCATION			
Relationships	Package	locations	one-to-many	Location

A package in this system can accumulate several location values in its LOCATION collection as it travels to its destination. Reading all locations from the database is resource intensive, especially when the only location of interest is the current location.

To resolve this type of problem, add a specific instance variable that represents the current location. You then add a one-to-one mapping for the instance variable, and use the instance variable to query for the current location. As illustrated in

Table 10–11, because you can now query for the current location without reading all locations associated with the package, this dramatically improves the performance of the system.

Table 10–11 Optimized Schema

Elements	Details	Instance Variable	Mapping	Target
Classes	Package, Location			
Tables	PACKAGE, LOCATION			
Relationships	Package	locations	one-to-many	Location
	Package	currentLocation	one-to-one	Location

A

Application Development Tools

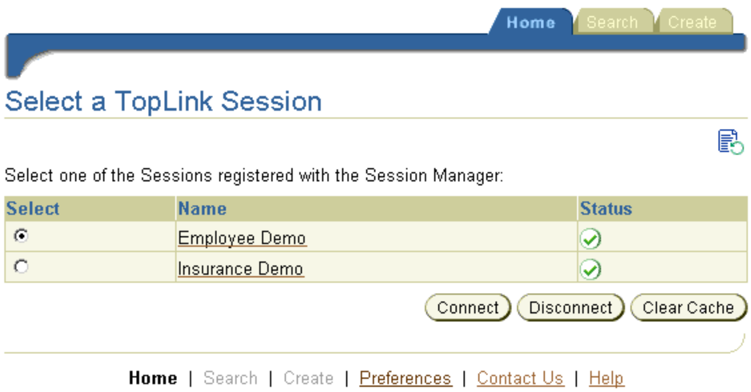
Oracle Application Server TopLink includes several tools that help you build and deploy an OracleAS TopLink application. This chapter introduces these tools and includes discussions on:

- OracleAS TopLink — Web Client
- Configuring OracleAS TopLink for Oracle JDeveloper
- Deploy Tool for WebSphere Server
- Schema Manager
- Session Management Services
- Stored Procedure Generator

OracleAS TopLink — Web Client

The OracleAS TopLink Web Client provides a Web-based interface that allows you to work with any OracleAS TopLink server session (see "Session Management Services" on page A-22) that is deployed on Oracle Application Server Containers for J2EE, IBM WebSphere 5.0, and BEA WebLogic 6.1, 7.0 or 8.1 application servers.

Figure A–1   Web Client Home



The Web Client leverages Java objects, database and OracleAS TopLink metadata to automatically create a browser based user interface to display and allow the manipulation of persistent objects obtained through server sessions. In addition, the Web Client offers utilities to profile the performance of your server session, as well as interactively execute SQL on the database connected to your server session.

The Web Client can access the following types of server sessions:

- Server sessions that any OracleAS TopLink application on the same application server have loaded into the session manager
- Server sessions that are created by the Web Client by supplying a `sessions.xml` file

Before you access server sessions, all the XML files and classes used by the server session must be accessible to the Web Client. This includes:

- the `sessions.xml` file
- the `project.xml` file (or the class files specified in the `sessions.xml` file)
- All the persistent classes mapped in the project

- All the required drivers

Configuring the Web Client

Before you build the Web Client, edit the following properties in the `<ORACLE_HOME>\toplink\config\toplinkwc\build.properties` file:

Property	Description
deployment.dir	Directory into which the EAR file is copied after you build the Web Client. Normally, this is your application server deployment directory.
domain.jar.path	The full path to your `.jar` file, when you deploy your own domain classes with the Web Client (see "Steps to Bundle a Single OracleAS TopLink Session with the Web Client:" on page A-4). To deploy the Web Client without any domain classes, leave this property blank.
use.weblogic	When you deploy to WebLogic, set to **true**.
defaultwebapp.dir	The location of the `DefaultWebApp` directory on the WebLogic server to which you are deploying. When running on WebLogic, the Web Client needs to extract resources here so that they are available to the web application.

If your OracleAS TopLink project uses a *datasource*, add the datasource information to the `<ORACLE_HOME>\config\toplinkwc\web.xml` file, as follows:

```
<resource-ref>
    <description>DataSource</description>
    <res-ref-name>jdbc/DataSourceName</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>SERVLET</res-auth>
</resource-ref>
```

In addition to the standard OracleAS TopLink `.jar` files, add the following to your application server class path:

```
<ORACLE_HOME>\jlib\uix2.jar
<ORACLE_HOME>\jlib\share.jar
```

Building the Web Client EAR File

Use the Web Client in either of the following ways to build and deploy the Web Client `.ear` file:

- To connect OracleAS TopLink server sessions already loaded into the session manager
- To bundle a single OracleAS TopLink session with the OracleAS TopLink Web Client

Steps to Connect OracleAS TopLink Server Sessions Already Loaded into the Session Manager:

1. In the `<ORACLE_HOME>\toplink\config\toplinkwc\build.properties` file, leave the **domain.jar.path** setting blank.
2. Run the `assembleWebClient` script located in the `<ORACLE_HOME>\toplink\bin` directory.

The system assembles and deploys `toplinkwc.ear` file, as specified in the `build.properties` file.

For more information, see "Configuring the Web Client" on page A-3.

Steps to Bundle a Single OracleAS TopLink Session with the Web Client:

1. Package your domain classes, OracleAS TopLink `project.xml` file, the `sessions.xml` file and any other necessary artifacts into a `.jar` file (called the *domain jar*).
2. Specify the path to your `domain.jar` file in the Web Client `build.properties` file (as specified in "Configuring the Web Client" on page A-3).
3. Run the `assembleWebClient` script located in the `<ORACLE_HOME>\toplink\bin` directory.

The system assembles and deploys `toplinkwc.ear` as specified in the `build.properties` file.

For more information, see "Configuring the Web Client" on page A-3.

Configuring the Application Server

Before using the OracleAS TopLink Web Client, configure your application server.

To use the OracleAS TopLink Web Client with Oracle Application Server Containers for J2EE:

1. Copy the `toplinkwc.ear` file to the `<ORACLE_HOME>\toplink\examples\oc4j\904\server\applications` directory.
2. Add the following line to the `server.xml` file located in the `<ORACLE_HOME>\toplink\examples\oc4j\904\server\config` directory:

```
<application name="toplinkwc" path="../applications/toplinkwc.ear"
auto-start="true" />
```

3. Add the following line to the `http-web-site.xml` file located in the `<ORACLE_HOME>\toplink\examples\oc4j\904\server\config` directory:

```
<web-app application="toplinkwc" name="toplinkwc" root="/toplinkwc" />
```

4. To start the server, run the `startServer` script located in the `<ORACLE_HOME>\toplink\examples\oc4j\904\server` directory.

   This step deploys all the OracleAS TopLink Examples, including the OracleAS TopLink Web Client.

5. To start the OracleAS TopLink Web Client, load `http://localhost:8888/toplinkwc` into a Web browser.

To use the Web Client with IBM WebSphere:

1. Copy the `toplinkwc.ear` file into the `<WEBSPHERE_INSTALL_DIR>\installableApps` directory.
2. Use the WebSphere Administration Console to install the `.ear` file and start the Web module.

   For more information about the WebSphere Administration Console, see the IBM WebSphere documentation.

3. To start the OracleAS TopLink Web Client, load `http://localhost:9080/toplinkwc` into a Web browser.

To use the Web Client with BEA WebLogic:

In the following steps, the `wls`*XX* refers to your version of BEA WebLogic. Use **61** for BEA WebLogic version 6.1, **70** for BEA WebLogic version 7.0. or **81** for BEA WebLogic version 8.1.

1. Copy the `toplinkwc.ear` file into the `<ORACLE_HOME>\toplink\examples\weblogic\wlsXX\server\config\TopLink_Domain\applications` directory.
2. To start the server, run the `startWebLogic` script located in the `<ORACLE_HOME>\toplink\examples\weblogic\wlsXX\server\config\TopLink_Domain\` directory.

   This step deploys all the OracleAS TopLink Examples, including the OracleAS TopLink Web Client.
3. To start the OracleAS TopLink Web Client, load `http://localhost:7001/toplinkwc` into a Web browser.

Connecting to OracleAS TopLink Sessions

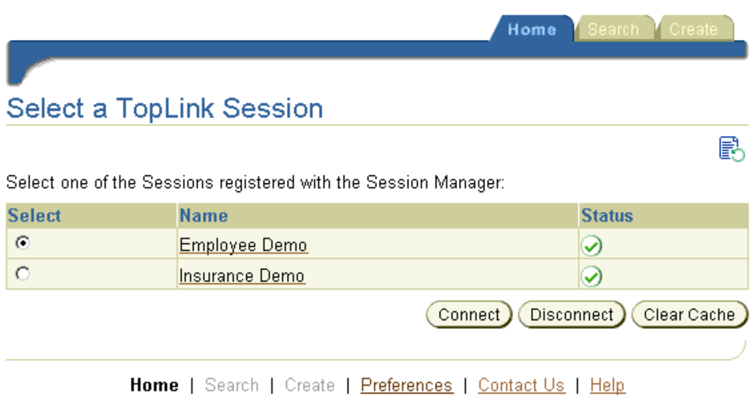
Use the Web Client Home tab to display and access the available OracleAS TopLink sessions.

To Connect to an OracleAS TopLink Session:

1. Click the **Home** tab. The Web Client displays the available (registered) OracleAS TopLink sessions and their status.

   Click **Refresh** to refresh the session list.

Figure A–2 Web Client Home



2. Choose one of the following options:

   - To connect to a session, select the session and click the **Connect** button. The session **Status** changes to .

     To select a session, click the appropriate radio button under the Select column.

   - To disconnect from a session, select the session and click the **Disconnect** button. The session **Status** changes to .

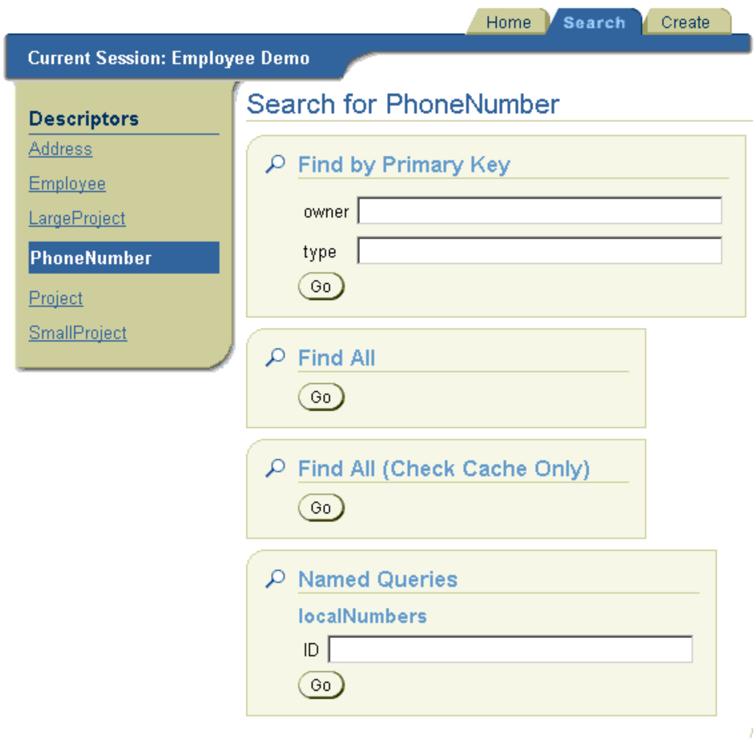
   - To clear a session cache, select the session and click the **Clear Cache button**.

   - To work with a specific session, click the session name. If the session is not already connected, the Web Client connects to the session.

Searching for Objects

Use the **Search** tab to display objects within a specific descriptor.

Figure A–3 Web Client Search



To Search for an Object:

1. Click the **Search** tab.
2. Choose a **Descriptor** from the Descriptor list.
3. Choose one of the following search options:
   - To search for an object using its primary key, enter the primary key information in the **Find by Primary Key** area and click **Go**.
   - To find all available objects, click **Go** in the **Find All** area.

- To find all objects in the OracleAS TopLink cache, click **Go** in the **Find All (Check Cache Only)** area.
- To search for objects using a named query, enter the named query information in the **Named Queries** area and click **Go**.

Note: The **Named Queries** area appears only for objects with defined named queries.

The Web Client displays all the objects that match the search criteria.

Figure A–4 Web Client Search Results

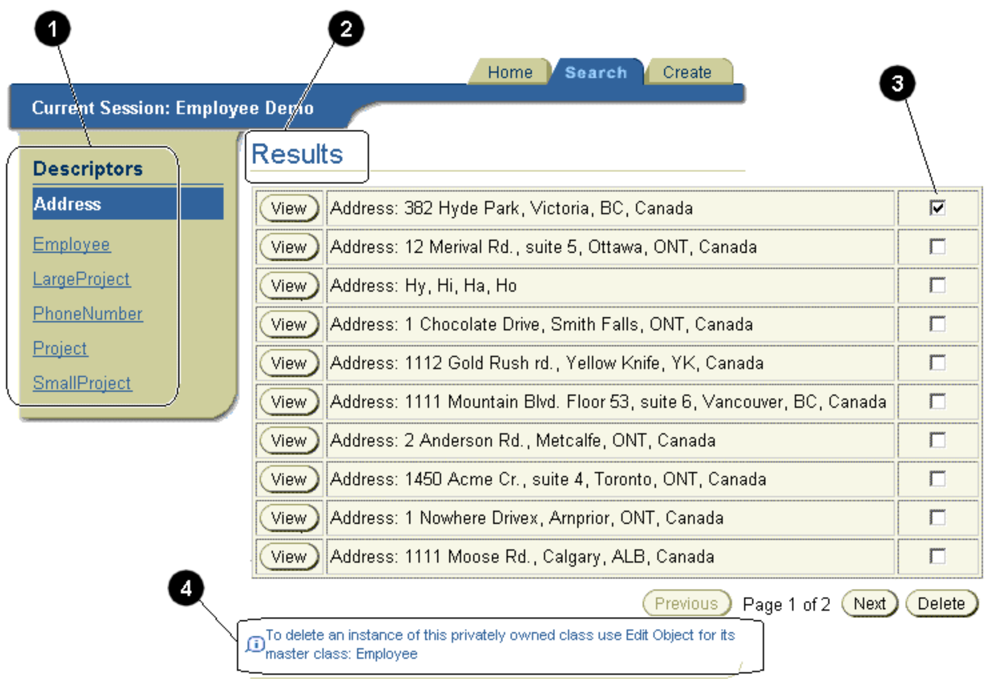


Figure A–4 identifies the following user-interface elements:

1. List of available descriptors
2. Search results

3. Select object column
4. Note for privately owned classes

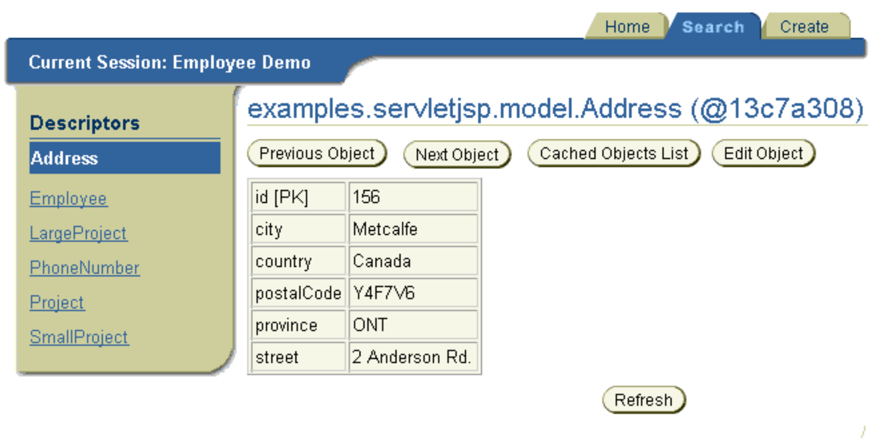
4. Choose one of the following options to delete or view an object:

   - Click **Delete** to delete an object.

   Note: You cannot delete objects for privately owned classes. Instead, edit its master class.

   - Click **View** for the object to display. The Web Client displays the object's data.

Figure A–5 Web Client View Object



5. Select further options for the viewed object:

   - Click **Previous Object** to display the previous record.
   - Click **Next Object** to display the next record.
   - Click **Cached Object List** to display all elements of the current type that exist in the OracleAS TopLink cache.
   - Click **Edit Object** to change or edit the record.

   For more information about creating and editing objects, see "Creating and Editing Objects" on page A-11.

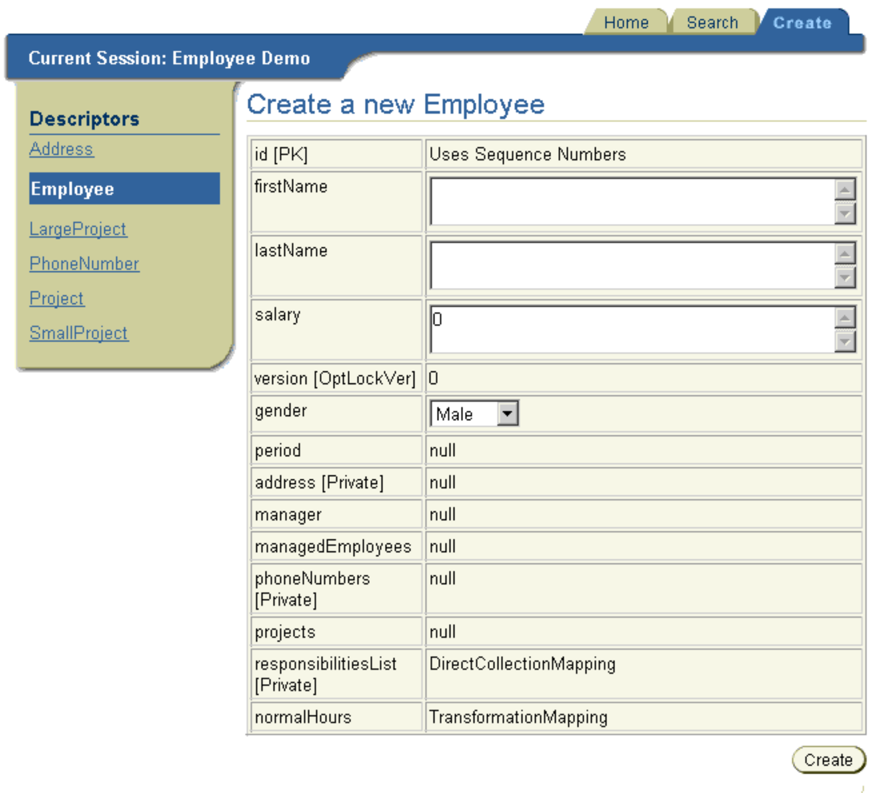
Creating and Editing Objects

Use the **Create** tab to create a new object. The information you enter on this tab is validated by the database—not the OracleAS TopLink Web Client.

To Create an Object:

1. Choose the **Descriptor** of the object to create.
2. Click the **Create** tab.

Figure A–6 Web Client Create



3. Enter the necessary information and click **Create**.

Note: You cannot create objects for privately owned classes. Instead, edit its master class.

Performing SQL Queries

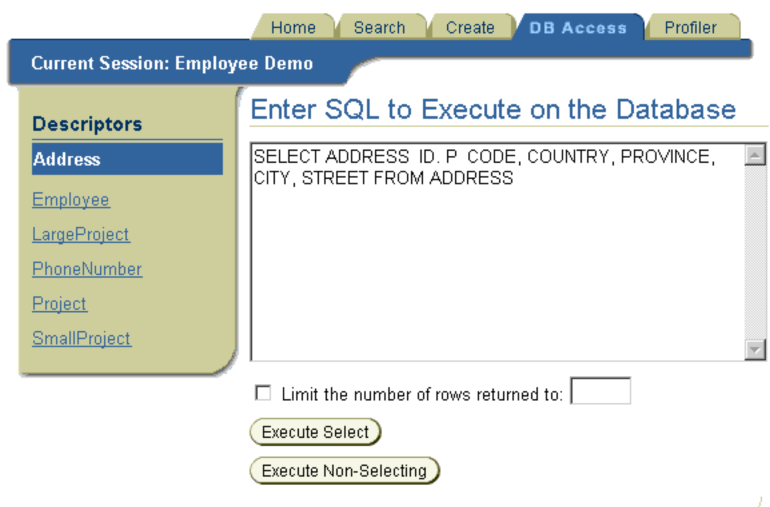
Use the **DB Access** tab to enter specific SQL queries to execute on the database.

To Perform a SQL Query:

1. Click the **DB Access** tab. If the DB Access tab is not visible, use the Web Client Preferences to enable the tab.

   For more information about setting Web Client preferences, see "Setting Web Client Preferences" on page A-14.

Figure A–7 Web Client DB Access



2. Enter the SQL query.

Note: The Web Client does not validate the SQL query.

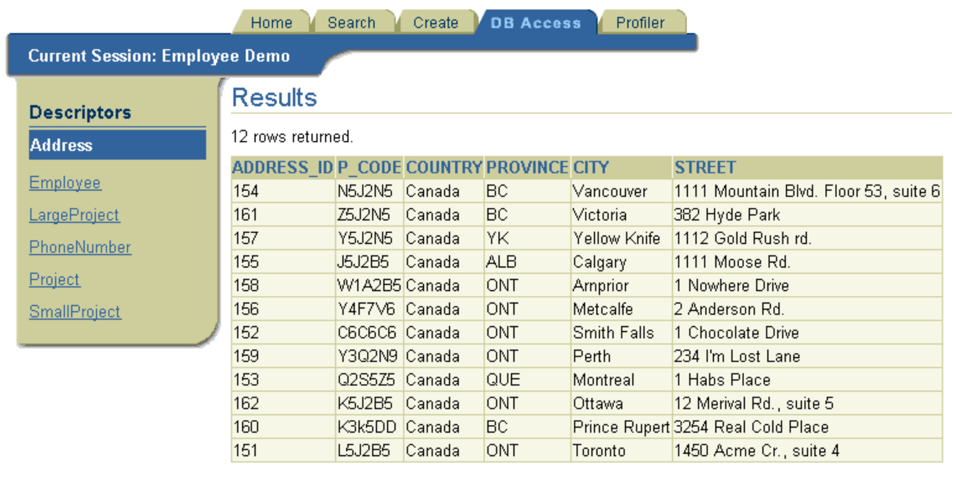
---

3. Specify whether the Web Client limits the number of rows returned from the query.

4. Choose the type of query:
   - **Execute Select**: results return the number of matches as well as the actual records.
   - **Execute Non-Selecting**: results return only the number of rows affected by the SQL statement.

   The Web Client displays the SQL results.

Figure A–8 Web Client DB Access Results



ADDRESS_ID	P_CODE	COUNTRY	PROVINCE	CITY	STREET
154	N5J2N5	Canada	BC	Vancouver	1111 Mountain Blvd. Floor 53, suite 6
161	Z5J2N5	Canada	BC	Victoria	382 Hyde Park
157	Y5J2N5	Canada	YK	Yellow Knife	1112 Gold Rush rd.
155	J5J2B5	Canada	ALB	Calgary	1111 Moose Rd.
158	W1A2B5	Canada	ONT	Arnprior	1 Nowhere Drive
156	Y4F7V6	Canada	ONT	Metcalfe	2 Anderson Rd.
152	C6C6C6	Canada	ONT	Smith Falls	1 Chocolate Drive
159	Y3Q2N9	Canada	ONT	Perth	234 I'm Lost Lane
153	Q2S5Z5	Canada	QUE	Montreal	1 Habs Place
162	K5J2B5	Canada	ONT	Ottawa	12 Merival Rd., suite 5
160	K3k5DD	Canada	BC	Prince Rupert	3254 Real Cold Place
151	L5J2B5	Canada	ONT	Toronto	1450 Acme Cr., suite 4

Using the Performance Profiler

Use the **Profiler** tab to specify the OracleAS TopLink Performance Profiler settings that appear in Figure A–9 and to display performance information.

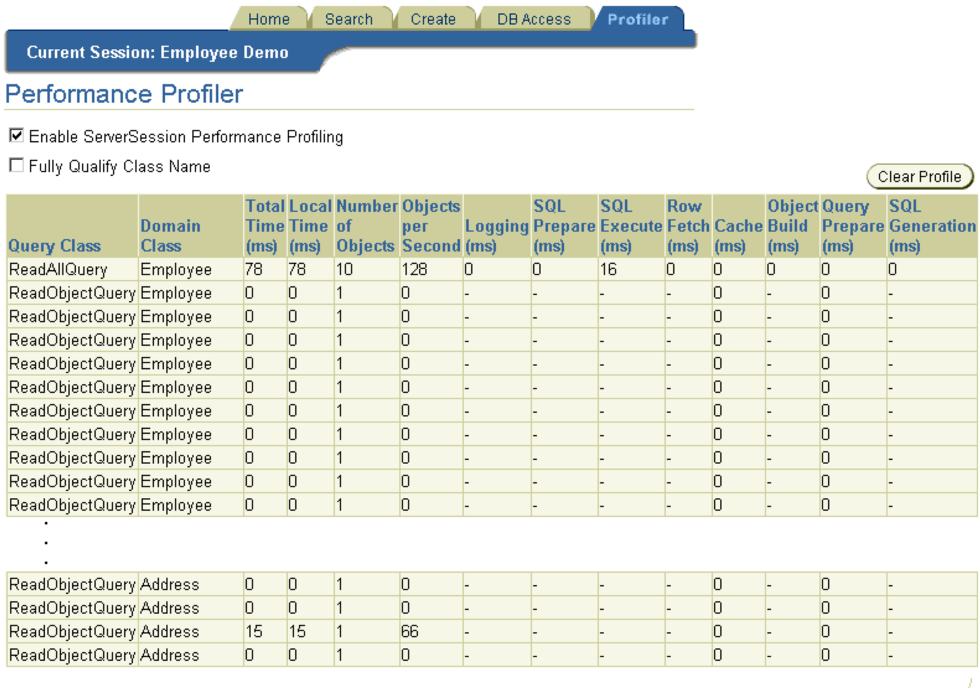
For more information about the OracleAS TopLink Performance Profiler settings, see "Profiling Performance" on page 10-3.

To Use the Performance Profiler:

1. Click the **Profiler** tab. If the Profiler tab is not visible, use the Web Client Preferences to enable the tab.

   For more information about setting Web Client preferences, see "Setting Web Client Preferences" on page A-14.

Figure A–9 Web Client Profiler



Query Class	Domain Class	Total Time (ms)	Local Time (ms)	Number of Objects	Objects per Second	Logging (ms)	SQL Prepare (ms)	SQL Execute (ms)	Row Fetch (ms)	Cache (ms)	Object Build (ms)	Query Prepare (ms)	SQL Generation (ms)
ReadAllQuery	Employee	78	78	10	128	0	0	16	0	0	0	0	0
ReadObjectQuery	Employee	0	0	1	0	-	-	-	-	0	-	0	-
ReadObjectQuery	Employee	0	0	1	0	-	-	-	-	0	-	0	-
ReadObjectQuery	Employee	0	0	1	0	-	-	-	-	0	-	0	-
ReadObjectQuery	Employee	0	0	1	0	-	-	-	-	0	-	0	-
ReadObjectQuery	Employee	0	0	1	0	-	-	-	-	0	-	0	-
ReadObjectQuery	Employee	0	0	1	0	-	-	-	-	0	-	0	-
ReadObjectQuery	Employee	0	0	1	0	-	-	-	-	0	-	0	-
ReadObjectQuery	Employee	0	0	1	0	-	-	-	-	0	-	0	-
ReadObjectQuery	Employee	0	0	1	0	-	-	-	-	0	-	0	-
ReadObjectQuery	Employee	0	0	1	0	-	-	-	-	0	-	0	-
. . .													
ReadObjectQuery	Address	0	0	1	0	-	-	-	-	0	-	0	-
ReadObjectQuery	Address	0	0	1	0	-	-	-	-	0	-	0	-
ReadObjectQuery	Address	15	15	1	66	-	-	-	-	0	-	0	-
ReadObjectQuery	Address	0	0	1	0	-	-	-	-	0	-	0	-

2. Specify the profiler settings by selecting:
   - **Enable Server Session Performance Profiling** check box
   - **Fully Qualify Class Name** check box
3. After you specify the profiler settings, the Profiler tab displays performance information for OracleAS TopLink queries that the Web Client executes.

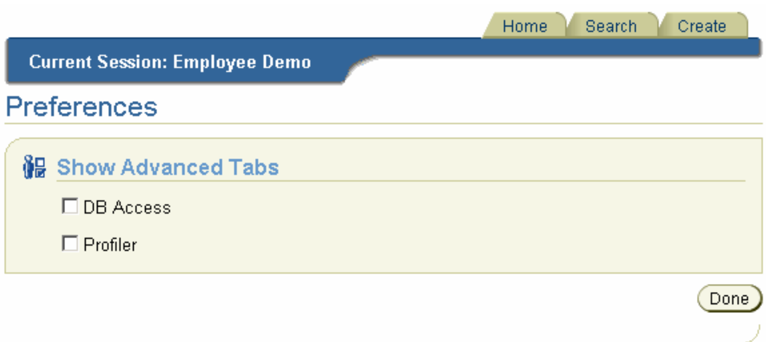
Setting Web Client Preferences

Use the Web Client Preferences to specify which advanced properties are available.

To Specify Web Client Preferences:

1. Click the **Preferences** link (at the bottom of each Web Client page). The Preferences tab appears.

Figure A–10 Web Client Preferences



2. Specify the advanced properties for this session by selecting:
   - **DB Access** check box
   - **Profiler** check box
3. Click **Done**.

Configuring OracleAS TopLink for Oracle JDeveloper

This section contains information on how to configure OracleAS TopLink for Oracle JDeveloper.

Oracle JDeveloper is a J2EE development environment with end-to-end support to develop, debug, and deploy e-business applications and Web Services.

When you use OracleAS TopLink with Oracle JDeveloper, use the following procedures to add the OracleAS TopLink JAR files to your JDeveloper projects:

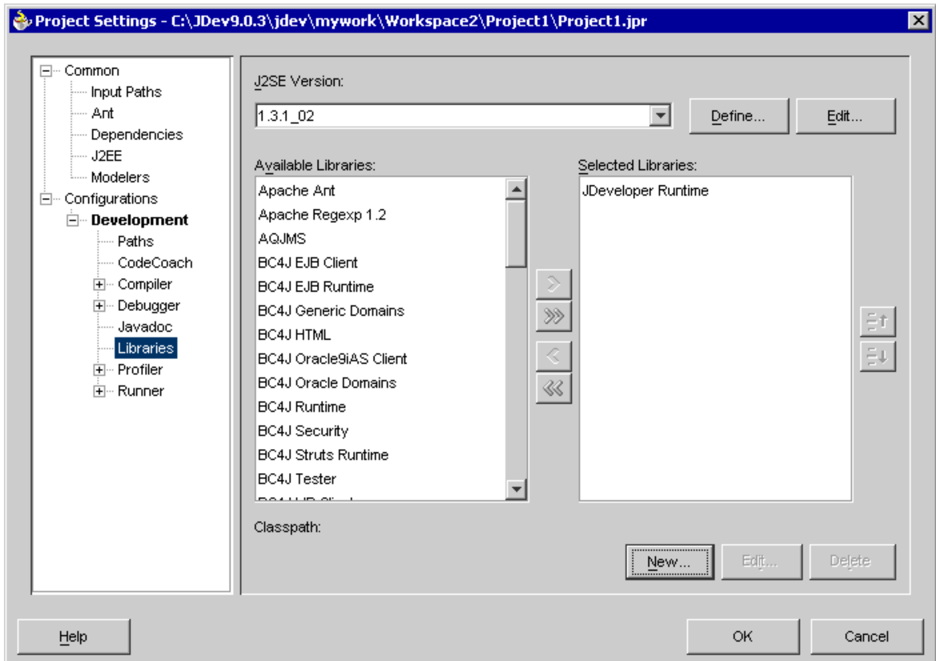
Creating an OracleAS TopLink JDeveloper Library:

1. Select a JDeveloper project in the System Navigator pane.
2. Choose **Project** > **Project Settings**.

   The Project Settings pane appears.
3. Choose **Configurations** > **Development** > **Libraries**.

   A list of predefined and user-defined libraries appears.

Figure A–11 List of Available Libraries



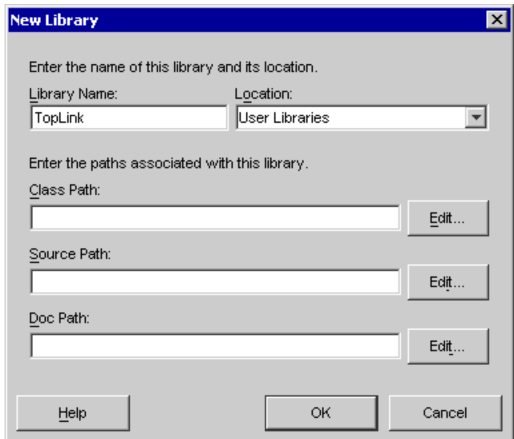
4. Click **New** to create a new library that will contain the OracleAS TopLink `.jar` files.

   The New Library dialog box appears.

5. Enter a name for the new Library—for example, OracleAS TopLink.

   Ensure that the default choice for Libraries remains as **User Libraries**.

Figure A–12 Creating a New Library



6. To edit the **Class Path** and add the OracleAS TopLink `.jar` files, click the **Edit** button.

   Add the following to the beginning of your **Class Path**:

```
<ORACLE_HOME>\toplink\jlib\toplink.jar
<ORACLE_HOME>\toplinkjlib\antlr.jar
<ORACLE_HOME>\lib\xmlparserv2.jar
```

7. Click **OK**. On the **Project Settings** pane click **OK**.

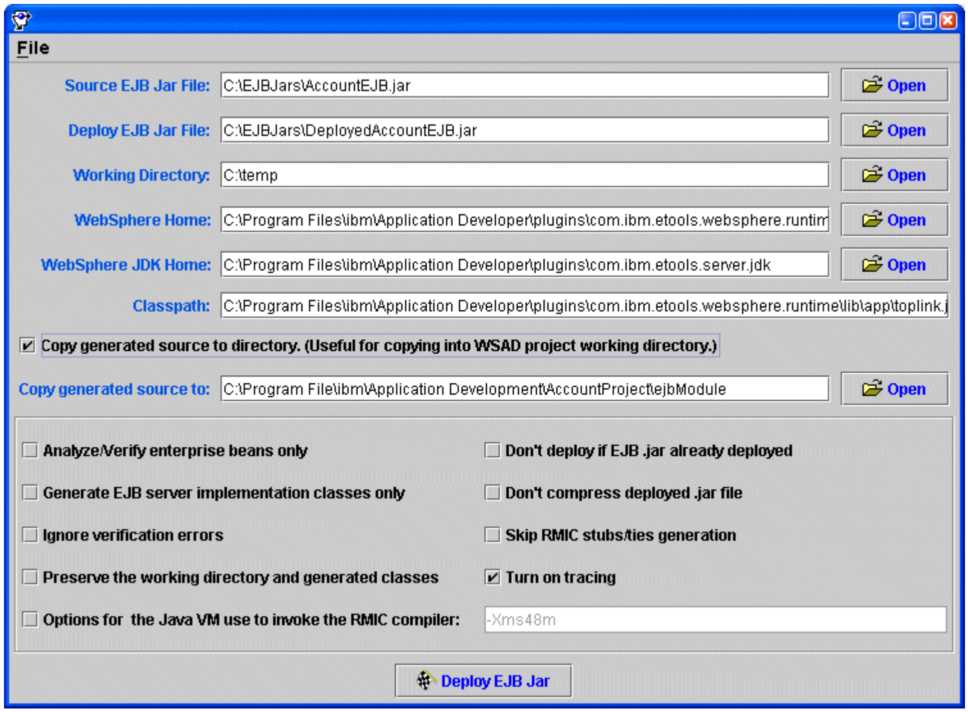
Use an existing User-Defined OracleAS TopLink Library:

After a user library is created, it can be re-referenced by any other project. Revisit the Libraries window of the Project Settings, and add the OracleAS TopLink Library to any project with which you want to use OracleAS TopLink.

Deploy Tool for WebSphere Server

OracleAS TopLink integration for IBM WebSphere Server includes a deployment tool that helps you deploy your projects to WebSphere. The Deploy Tool for WebSphere is a graphical tool that makes project deployment to WebSphere easier to configure and execute. The deploy tool also includes a command-line option that enables you to deploy your project while bypassing the graphical interface element of the tool.

Figure A–13 The Deploy Tool set up for use with WSAD



To Deploy a JAR:

1. Enable the **Copy generated source to directory** option to save a copy of the generated code in the specified directory. This is a quick and efficient way to copy the files into a WSAD project working directory.
2. Enable the **Turn on tracing** option if you want to see the details of the process.
3. Click the **Deploy EJB Jar** button.

Using the Deploy Tool with WebSphere Studio Application Developer (WSAD)

The Deploy tool is compatible with the WebSphere Studio Application Developer (WSAD).

To Deploy from the Deploy Tool to WSAD:

1. Select the EJB Project in WSAD and choose to generate **Deploy and RMIC Code**.

2. Export the EJB Project to an EJB JAR, making sure that the OracleAS TopLink project and `toplink-ejb-jar.xml` files are included in the EJB JAR.
3. Start the OracleAS TopLink - Deploy Tool. To start the server, execute the `wasDeployTool.cmd/sh` script in the `<ORACLE_HOME>\toplink\bin` directory.
4. Choose the EJB project working directory so that OracleAS TopLink overrides the WSAD deploy code with the OracleAS TopLink deploy code.
5. If the source is copied to a directory other than the WSAD EJB Project directory, manually copy the source files to the WSAD EJB Project under the **ejbModule directory** of the project.
6. Enter appropriate directories in the fields of the Deploy Tool.
7. Choose **Deploy EJB JAR** to create the deployed EJB JAR.
8. Choose **Rebuild all** from the Project menu to compile the deploy code to incorporate CMP.

Troubleshooting

The most common error you might encounter when you use the deploy tool is the `NoClassDefFoundError` exception. To resolve this error condition, add the required resources to the **Classpath**. The **Turn on tracing** option also helps to debug errors during deployment code generation.

When an obscure error appears during the generating stub phase, copy the Java command and run it at the command prompt. This gives a more detailed error message.

Schema Manager

The Schema Manager creates and modifies tables in a database from a Java application. As a a Java code *batch* facility, the Schema Manager can also create sequence numbers on an existing database and generate stored procedures.

Use the Schema Manager to re-create a production database in a nonproduction environment. Doing this enables you to build models of your existing databases, and modify and test them during development.

Using the Schema Manager to Create Tables

The Schema Manager table creation mechanism uses Java types rather than database types, it is database-independent. However, this mechanism does not account for database specific optimizations, it is best-suited for development purposes rather than production.

The OracleAS TopLink `TableDefinition` class enables you to create new database table schemas in a generic format. At runtime, OracleAS TopLink determines the database type, and uses the generic schemas to create the appropriate fields for that database.

Creating a Table Definition

The `TableDefinition` class includes all the information required to create a new table, including the names and properties of a table and all its fields.

The `TableDefinition` class has the following methods:

```
setName()
addField()
addPrimaryKeyField()
addIdentityField()
addForeignKeyConstraint()
```

All table definitions must call the `setName()` method to set the name of the table that is described by the `TableDefinition`.

Adding Fields to a Table Definition

Use the `addField()` method to add fields to the `TableDefinition`. To add the primary key field to the table, use the `addPrimaryKeyField()` method rather than the `addField()` method.

To maintain compatibility among different databases, the `type` parameter requires a Java class rather than a database field type. OracleAS TopLink translates the Java class to the appropriate database field type at runtime. For example, the `String` class translates to the CHAR type for dBase databases. However, if you are connecting to Sybase, the `String` class translates to VARCHAR.

The `addField()` method can also be called with the `fieldSize` or `fieldSubSize` parameters for column types that require size and subsize to be specified.

Some databases require a subsize, but others do not. OracleAS TopLink automatically provides the required information, as necessary.

Defining Sybase and Microsoft SQL Server Native Sequencing

The `addIdentityField()` methods have the following definitions:

```
addIdentityField(String fieldName, Class type)
addIdentityField(String fieldName, Class type, int fieldSize)
```

These methods enable you to add fields representing a generated sequence number from Sybase or Microsoft SQL Server native sequencing.

The OracleAS TopLink Two-Tier Example illustrates the table creation mechanism in the `EmployeeTableCreator.java` file located in the `<ORACLE_HOME>\toplink\examples\foundation\twotier\src\examples\sessions\twotier\` directory.

Creating Tables on the Database

OracleAS TopLink offers two methods that enable you to pass the initialized `TableDefinition` object to the `DatabaseSession` Schema Manager:

- The `createObject()` method creates a new table in the database, according to the table definition.

  ```
  SchemaManager schemaManager = new SchemaManager(session);
  schemaManager.createObject(Tables.employeeTable());
  ```

- The `replaceObject()` method destroys and re-creates the schema entity in the database.

  ```
  SchemaManager schemaManager = new SchemaManager(session);
  schemaManager.replaceObject(Tables.addressTable());
  ```

Creating the Sequence Table

If your application requires a sequence table, invoke the `createSequences()` method on the Schema Manager:

```
SchemaManager schemaManager = new SchemaManager(session);
schemaManager.createSequences();
```

The preceding code:

- Creates the sequence table as defined in the session `DatabaseLogin`
- Creates or inserts sequences for each sequence name for all registered descriptors in the session

- Creates the Oracle sequence object if you use Oracle native sequencing

Managing Java and Database Type Conversions

Table A–1 lists the field types that match a given class for each database that OracleAS TopLink supports. This list is specific to the Schema Manager and does not apply to mappings. OracleAS TopLink automatically performs conversions between any database types within mappings.

Table A–1 OracleAS TopLink Classes and Database Field Types

Class	Oracle Type	DB2 Type	dBase Type	Sybase Type	Microsoft Access Type
java.lang.Boolean	NUMBER	SMALLINT	NUMBER	BIT default 0	SHORT
java.lang.Byte	NUMBER	SMALLINT	NUMBER	SMALLINT	SHORT
java.lang.Byte[]	LONG RAW	BLOB	BINARY	IMAGE	LONGBINARY
java.lang.Integer	NUMBER	INTEGER	NUMBER	INTEGER	LONG
java.lang.Long	NUMBER	INTEGER	NUMBER	NUMERIC	DOUBLE
java.lang.Float	NUMBER	FLOAT	NUMBER	FLOAT(16)	DOUBLE
java.lang.Double	NUMBER	FLOAT	NUMBER	FLOAT(32)	DOUBLE
java.lang.Short	NUMBER	SMALLINT	NUMBER	SMALLINT	SHORT
java.lang.String	VARCHAR2	VARCHAR	CHAR	VARCHAR	TEXT
java.lang.Character	CHAR	CHAR	CHAR	CHAR	TEXT
java.lang.Character[]	LONG	CLOB	MEMO	TEXT	LONGTEXT
java.math.BigDecimal	NUMBER	DECIMAL	NUMBER	NUMERIC	DOUBLE
java.math.BigInteger	NUMBER	DECIMAL	NUMBER	NUMERIC	DOUBLE
java.sql.Date	DATE	DATE	DATE	DATETIME	DATETIME
java.sql.Time	DATE	TIME	CHAR	DATETIME	DATETIME
java.sql.Timestamp	DATE	TIMESTAMP	CHAR	DATETIME	DATETIME

Session Management Services

OracleAS TopLink provides statistical reporting and runtime configuration systems through two public APIs: `oracle.toplink.service.RuntimeServices` and `oracle.toplink.services.DevelopmentServices`.

Runtime Services

The `RuntimeServices` API enables you to monitor a running in-production system. It offers statistical functions and reporting, as well as logging functions. Typical uses for `RuntimeServices` include turning logging on or off and generating real time reports on the number and type of objects in a given cache or subcache.

For more information, see the `RuntimeServices` class in the *Oracle Application Server TopLink API Reference*.

Development Services

The `DevelopmentServices` API enables you to make changes to a running nonproduction application that can destabilize or even crash the application. For example, use the `DevelopmentServices` API to change the states of selected objects and modify and reinitialize identity maps. This feature is useful for stress and performance testing of preproduction applications and also enables you to build prototypes quickly and easily.

For more information, see the `RuntimeServices` class in the *Oracle Application Server TopLink API Reference*.

Using Session Management Services

To instantiate a session management service, you pass a session to the constructor. After instantiating the service, you can attach a graphical interface or other applications to the object to provide statistical feedback and runtime option settings.

Example A–1 Accessing Session Management Services

```
import oracle.toplink.services.RuntimeServices;
import oracle.toplink.publicinterface.Session;
...
...
RuntimeServices service = newRuntimeServices ((session) session);
java.util.List classNames = service.getClassesInSession();
```

Session Management Services and BEA WebLogic Server

OracleAS TopLink support for BEA WebLogic Server automatically deploys the session management services to the JMX server. You can retrieve the JMX Mbeans with the following object names:

```
WebLogicObjectName("TopLink_Domain:Name=Development <Session><Name> Type=Configuration");
WebLogicObjectName("TopLink_Domain:Name=Runtime <Session><Name> Type=Reporting");
```

`Session Name` represents the session type and name under which you store the required session configuration in the `toplink-ejb-jar.xml` file.

For more information about the `WebLogicObjectName` API, see

`http://e-docs.bea.com/wls/docs70/javadocs/weblogic/management/WebLogicObjectName.html`

Stored Procedure Generator

You can generate stored procedures based on the dynamic SQL that is associated with descriptors and mappings. After you generate the stored procedures, attach them to the mappings and descriptors of the domain object. At that point, access to the database is accomplished through stored procedures, rather than through SQL.

> **Note:** Implement this feature only if your database requires access by stored procedures. Doing this does not enhance performance and has the same limitations that are associated with stored procedures.

Generating Stored Procedures

You can generate stored procedures for all descriptors and most relationship mappings with the exception of many-to-many mappings, which are not supported by the stored procedure generator and stored procedures for Read operations for the Oracle platform.

Sequencing and Stored Procedures

You can generate stored procedures for sequence number updates and selects. To enable these stored procedures in OracleAS TopLink, create an amendment class that contains a method attaching the stored procedures to each descriptor.

Example A–2 Stored Procedures Generated Directly on the Database

OracleAS TopLink creates an amendment class called `com.demo.Tester` in the file `C:/temp/Tester.java`.

```
SchemaManager manager = new SchemaManager(session);
manager.outputDDLToDatabase();
manager.generateStoredProceduresAndAmendmentClass("C:/temp/","com.demo.Tester");
```

Example A–3 Generating Stored Procedures to a File

```
SchemaManager manager = new SchemaManager(session);
manager.outputDDLToFile("C:\Temp\test.sql");
manager.generateStoredProceduresAndAmendmentClass("C:/temp/","com.demo.Tester");
```

For more information about creating an amendment class, see "Customizing OracleAS TopLink Descriptors with Amendment Methods" on page 3-82.

Attaching the Stored Procedures to the Descriptors

After you create the stored procedures on the database, and after you create the amendment file, enable them on the descriptors:

- Before logging in, call a method on the generated amendment class:

  ```
  Session session = project.createDatabaseSesssion();
  com.demo.Tester.amendDescriptors(project);
  ```

For more information about creating an amendment class, see "Customizing OracleAS TopLink Descriptors with Amendment Methods" on page 3-82.

B

Configuring OracleAS TopLink for J2EE Containers

This chapter describes how to configure Oracle Application Server TopLink for use with J2EE containers and application servers. It includes sections on:

- For installation information, see "Installing and Configuring OracleAS TopLink," in the *Oracle Application Server TopLink Getting Started Guide.*

Software Requirements

To run an OracleAS TopLink application within a J2EE container, your system must meet the following software requirements:

- An application server or J2EE container such as:
  - Oracle Application Server Containers for J2EE
  - IBM WebSphere Application Server 4.0
  - BEA WebLogic Application Server 6.1 (Service Pack 4), 7.0 (Service Pack 2) or 8.1.
- A JDBC driver configured to connect with your local database system (for more information, see your database administrator)
- A Java development environment, such as:
  - Oracle JDeveloper
  - IBM WebSphere Studio Application Developer (WASD)
  - Sun Java Development Kit (JDK) 1.3.1 or higher
  - Any other Java environment that is compatible with the Sun JDK 1.3.1 or higher
- A command-line Java virtual machine (JVM) executable (such as `java.exe` or `jre.exe`)

Non-CMP Configuration

OracleAS TopLink supports several architectures that leverage a J2EE container. To enable OracleAS TopLink in these architectures configure the following:

Class Path Place the OracleAS TopLink JARs on the application server class path. Class path configuration is container-specific. For more information, see:

- Oracle Application Server Containers for J2EE Support on page B-3
- IBM WebSphere Application Server 4.0 on page B-4
- IBM WebSphere Application Server 5.0 on page B-5
- BEA WebLogic Application Server (6.1, 7.0 or 8.1) on page B-6

Datasource OracleAS TopLink applications that run in a J2EE container often use a J2EE Datasource to access JDBC connections. To leverage a J2EE datasource, configure a datasource with the server's configuration tools, and specify the name of the datasource in the `sessions.xml` file, as follows:

```
<login>
    <datasource>java:comp/env/jdbc/myJTADataSource</datasource>
    <uses-external-connection-pool>true</uses-external-connection-pool>
    ...
</login>
```

JTA integration The OracleAS TopLink Unit of Work uses the Java Transaction API (JTA) to participate in global transactions. Configure OracleAS TopLink JTA support in the `sessions.xml` file as follows:

```
    <login>
        ...
        <uses-external-transaction-controller>true</uses-external-transaction-controller>
    </login>
    <external-transaction-controller-class>oracle.toplink.jts.oracle9i.Oracle9iJTSExternal
TransactionController </external-transaction-controller-class>
```

OracleAS TopLink support for external transaction controllers requires a J2EE datasource. OracleAS TopLink provides several container-specific external transaction controllers, as well as a generic controller.

For more information, see "J2EE Integration" on page 7-44.

Oracle Application Server Containers for J2EE Support

To configure OracleAS TopLink support for Oracle Application Server Containers for J2EE, include the following OracleAS TopLink JARS on the Oracle Application Server Containers for J2EE class path:

```
<ORACLE_HOME>\toplink\jlib\toplink.jar
<ORACLE_HOME>\toplink\jlib\antlr.jar
```

For example, add the following code to the Oracle Application Server Containers for J2EE `application.xml` file:

```
<library path="/OraHome1/toplink/jlib/toplink.jar" />
<library path="/OraHome1/toplink/jlib/antlr.jar" />
```

Substitute your *<ORACLE_HOME>* directory for `OraHome1`.

IBM WebSphere Application Server 4.0

OracleAS TopLink provides support for IBM WebSphere Application Server 4.0. To configure this support for IBM WebSphere Application Server, copy the following OracleAS TopLink JARs to the application server class path directory:

```
<ORACLE_HOME>\toplink\jlib\toplink.jar
<ORACLE_HOME>\toplink\jlib\antlr.jar
<ORACLE_HOME>\lib\xmlparserv2.jar
```

Table B–1 lists the default application class path directories for IBM container components.

Table B–1 Class Path Directories for IBM Container Components

Container	Default Application Class Path
WebSphere Application Server 4.0 (for Windows)	`\WebSphere\AppServer\lib\app`
WebSphere Studio Application Developer 4.0 (for Windows)	`\Program Files\ibm\Application Developer\plugins\com.ibm.etools.websphere.runtime\lib\app`

Configuring IBM WebSphere Module Visibility Setting

Because of the way the WebSphere defines its class loader isolation mode, OracleAS TopLink supports only the `APPLICATION` and `MODULE` modes for module visibility.

Module Mode A J2EE application (EAR file) can have multiple EJB modules (EJB JAR files). OracleAS TopLink CMP loads one `toplink-ejb-jar.xml` per EJB module (EJB JAR). If you do not set the loader isolation mode to `MODULE`, an EJB module can load the incorrect `toplink-ejb-jar.xml` from another EJB module.

Application Mode OracleAS TopLink supports class loader isolation `APPLICATION` mode. However, each application must have only one EJB JAR file.

For more information about the module visibility in IBM WebSphere, see the IBM WebSphere documentation.

Table B–2 lists the application and module modes OracleAS TopLink supports.

Table B–2 OracleAS TopLink Support of Server-Installable Applications on Server Versus Module Visibility Mode

Installable Applications on Server	Application	Module	Compatibility	Server
Multiple applications in which each application can have multiple OracleAS TopLink EJB modules	No	Yes	No	No
Multiple applications in which each application has single OracleAS TopLink EJB module	Yes	Yes	No	No
Single application has multiple OracleAS TopLink EJB modules	No	Yes	No	No
Single application has Single OracleAS TopLink EJB module	Yes	Yes	Yes	Yes

IBM WebSphere Application Server 5.0

To enable OracleAS TopLink support for IBM WebSphere Application Server 5.0, configure the following four elements for WebSphere at the Enterprise Application level:

Creating a Shared Library Create a shared library that contains the following Toplink JARS and associate the shared library with the application:

```
<ORACLE_HOME>\toplink\jlib\toplink.jar
<ORACLE_HOME>\toplink\jlib\antlr.jar
<ORACLE_HOME>\lib\xmlparserv2.jar
```

Classloader Mode Set the application Classloader Mode to `PARENT_LAST`. Note that you can also configure the Classloader at the Server level.

For more information, see the IBM WebSphere 5.0 documentation.

Application Classloader Policy Set the Application Classloader Policy on the application server to `Multiple`.

For more information, see the IBM WebSphere 5.0 documentation.

A WAS 5.0 JTA Integration Class For applications that require JTA integration, specify the external transaction controller in the OracleAS TopLink `sessions.xml` file. To enable the WebSphere 5.0 external transaction controller, add the following line to the `sessions.xml` file:

```
<external-transaction-controller-class>oracle.toplink.jts.was.
JTSExternalTransactionController_5_0</external-transaction-controller-class>
```

For more information, see "External Transaction Controllers" on page 7-45.

BEA WebLogic Application Server (6.1, 7.0 or 8.1)

OracleAS TopLink provides support for BEA WebLogic Application Server 6.1, 7.0 and 8.1. This support requires manual configuration, and includes a sample domain and the ability to use a security manager with BEA WebLogic.

To configure OracleAS TopLink support for BEA WebLogic Server, add the following JAR files to the application server class path:

```
<ORACLE_HOME>\toplink\jlib\toplink.jar
<ORACLE_HOME>\lib\xmlparserv2.jar
```

> **Note:** When you add the `toplink.jar` and `xmlparserv2.jar` files in the application server class path, ensure they are placed before the `weblogic.jar` file.

Using a Security Manager with BEA WebLogic Server

If you use a security manager, specify a security policy file in the `weblogic.policy` file (normally located in the BEA WebLogic install directory), as follows:

```
-Djava.security.manager
-Djava.security.policy==c:\weblogic\weblogic.policy
```

The BEA WebLogic installation procedure includes a sample security policy file. You need to edit the `weblogic.policy` file to grant permission for OracleAS TopLink to use reflection.

Example B–1 A Subset of a "Grant" Section from a BEA WebLogic.policy File

This example illustrates only the permissions that OracleAS TopLink requires, but most `weblogic.policy` files contain more permissions than are shown in this example.

```
grant {
// "enableSubstitution" required to run the WebLogic console
permission java.io.SerializablePermission "enableSubstitution";
// "modifyThreadGroup" required to run the WebLogic Server
```

```
permission java.lang.RuntimePermission "modifyThreadGroup";
//grant permission for OracleAS TopLink to use reflection
    permission java.lang.reflect.ReflectPermission "suppressAccessChecks";
};
```

OracleAS TopLink CMP Configuration

OracleAS TopLink provides a CMP integration for:

- IBM WebSphere Application Server 4.0
- BEA WebLogic Application Server (6.1, 7.0 and 8.1)

IBM WebSphere Application Server 4.0

To enable the OracleAS TopLink CMP integration for IBM WebSphere 4.0, configure OracleAS TopLink J2EE support for WebSphere as described in "IBM WebSphere Application Server 4.0" on page B-4.

BEA WebLogic Application Server (6.1, 7.0 and 8.1)

To enable the OracleAS TopLink CMP integration for BEA WebLogic Server 6.1, 7.0 and 8.1, use the following procedures. These procedures assume you have already installed OracleAS TopLink.

To configure OracleAS TopLink support for WebLogic:

1. Locate the persistence directory, located above the installation drive and root directory of your BEA WebLogic Server executable, as follows:

Version	Persistence Directory (above <*WebLogic_INSTALL_DIR*>)
6.1 (Service Pack 4)	`\wlserver6.1\lib\persistence`
7.0 (Service Pack 2)	`\weblogic700\server\lib\persistence`
8.1	`\weblogic81\server\lib\persistence`

   Do one of the following:

   - Use a text editor to open the `persistence.install` file in the BEA WebLogic Server persistence directory, and add a new line that references `TopLink_CMP_Descriptor.xml`.

- Replace the WebLogic `persistence.install` file with the OracleAS TopLink `persistence.install` file found in the `<ORACLE_HOME>\toplink\config` directory.

2. Add the following JAR files to the application server class path:

```
<ORACLE_HOME>\toplink\jlib\toplink.jar
<ORACLE_HOME>\lib\xmlparserv2.jar
```

> **Note:** When you add the `toplink.jar` and `xmlparserv2.jar` files in the application server class path, ensure they are placed before the `weblogic.jar` file.

3. Start the container, and then start the OracleAS TopLink application. Where supported, use a startup script to start the server. If you write your own startup script, ensure that the class path includes the files listed in Step 2.

OracleAS TopLink in a BEA WebLogic Cluster

BEA WebLogic includes a clustering service that you can leverage with your OracleAS TopLink application. To leverage a cluster, make the OracleAS TopLink runtime JAR available to all servers to which you deploy OracleAS TopLink CMP beans. This section discusses the following cluster-related issues:

- Collocation
- Static Partitioning
- Pinning
- Pinning with Session Beans
- Cache Synchronization and the Cluster

Collocation

Although the BEA WebLogic cluster enables you to build an application across several servers, related components in the application must still be localized to a server. When you store several beans on the same server, the beans are said to be *collocated*.

Collocation in a BEA WebLogic cluster imposes the following restrictions:

- When you deploy beans on a single server, you can invoke those beans only on that server. This localizes the beans on a given server, and provides a statically-defined means of collocation.
- You must cluster bean `Home` interfaces, but not instances. When you instantiate a bean, you pin it to the server on which it was instantiated.

  For more information, see "Pinning" on page B-9.
- JTA user transactions must execute completely on a single server, and cannot span servers.

You must collocate all related beans and objects on a single server to support relationships between beans. To simplify the application, also retrieve source and target objects on the same server.

Static Partitioning

Static partitioning refers to strategically deploying all related beans on a single server. A server can contain several groups of related beans, but collocation dictates that a group of beans cannot span several servers.

Static partitioning eliminates cache inconsistency issues, because the application loads beans only on the server on which the beans are deployed. BEA provides limited failover, and bean activity determines how the application load balances across servers.

Pinning

When you create or instantiate a bean, the bean instance is associated with, or *pinned* to, the server on which it is instantiated. To localize transactions to a particular server, BEA WebLogic Server pins all instantiated beans in a given transaction to the server on which you run the transaction. If beans are pinned to other servers, you cannot localize the transaction.

You can pin beans to a given server dynamically, either through user transactions or through session beans.

Pinning with User Transactions To maintain bean localization, access beans through a transaction. If you deploy beans on multiple servers, you can initiate the transaction on any server that holds a bean in the transaction. BEA WebLogic attempts to pin all accessed beans to that server for the duration of the transaction.

Example B–2 illustrates the use of a user transaction to collocate related beans.

Example B–2 Using a Transaction to Collocate Beans

```
UserTransaction transaction = lookupUserTransaction()
// Enclose all construction of relationships in the same transaction
transaction.begin();
/* Look up the home interface and the bean even if they have already been looked up
previously */
Employee emp = lookupEmployeeHome().findByPrimaryKey(new EmployeePK(EMP_ID));
Address address = new Address(EMP_ID, “99 Bank”, “Ottawa”, “Ontario”, “Canada”,“K2P 4A1”);
emp.setAddress(address);
Project project = lookupProjectHome().findByPrimaryKey(new ProjectPK(PROJ_ID));
emp.addProject(project);
transaction.commit();
```

Pinning with Session Beans If you access entity beans through a session bean, the application instantiates the entity beans on the same server as the session bean. By moving the application logic from the client to a session bean, you enable all bean code to run on the same JVM. The client invokes a method in the session bean, and the session bean executes all required logic on the server on which it resides.

You can use session beans to manage scalability and failover.

Cache Synchronization and the Cluster

Cache synchronization propagates changes from one OracleAS TopLink cache to all other server caches. This eliminates the need for manual refresh, and provides a consistent view of cached data across the cluster.

Cache synchronization is a project-level option. If you implement cache synchronization, OracleAS TopLink propagates changes to all objects in the project.

Configuring Cache Synchronization

To configure cache synchronization in the `toplink-ejb-jar.xml` deployment descriptor, implement the following elements and sub-elements:

- `cache-synchronization`: Include this tag to enable cache synchronization. To configure synchronization, use the `is-asynchronous` and `should-remove-connection-on-error` tags.
- `is-asynchronous` (optional): Sets the synchronization mode. Set to *True* to enable asynchronous propagation, or *False* to force synchronous updates. The default value is *True*.

  For more information about synchronous and asynchronous updates, see "Synchronous and Asynchronous Propagation" on page 8-12.

- `should-remove-connection-on-error` (optional): Enables error handling at the connection. Set to *True* to enable this behavior, or *False* to disable it. The default value is *True*.

  For more information about this error handling feature, see "Error Handling" on page 8-13.

Example B–3 Specifying Cache Synchronization in the toplink-ejb-jar.xml File

```
<toplink-ejb-jar>
   <session>
      <name>ejb20_AccountDemo</name>
      <project-class>oracle.toplink.demos.ejb20.cmp.account.AccountProject</project-class>
      <login>
         <connection-pool>ejbPool</connection-pool>
      </login>
      <cache-synchronization>
         <is-asynchronous>True</is-asynchronous>
         <should-remove-connection-on-error>True</should-remove-connection-on-error>
      </cache-synchronization>
   </session>
</toplink-ejb-jar>
```

For more information about the `toplink-ejb-jar.xml` file, see "Configuring the toplink-ejb-jar.xml File with the BEA WebLogic Server" on page 9-6.

C

Error Codes and Messages

This chapter describes the Oracle Application Server TopLink exception classes and general troubleshooting issues for entity bean configuration and deployment. It includes sections on:

- OracleAS TopLink Exceptions
- Exception Error Codes and Descriptions
- Entity Deployment

OracleAS TopLink Exceptions

All OracleAS TopLink exceptions are descendants of `RuntimeException`. The `TopLinkException` class is the superclass of all runtime and development type exceptions.

Runtime Exceptions

Runtime exceptions indicate error conditions at runtime, though not necessarily fatal errors. Instead, they indicate that runtime conditions are invalid, such as the loss of database connection. All these exceptions should be handled in a try-catch block.

The following exceptions can be thrown at runtime:

- `DatabaseException`
- `OptimisticLockException`
- `CommunicationException`

Development Exceptions

Development exceptions indicate that a certain fragment of code is invalid. All development exceptions do not depend on runtime conditions and must, therefore, be solved before deploying the application. For example, the `DescriptorException` is thrown the first time you initialize an application that contains an erroneous descriptor or mapping property. Development exceptions are useful as a debugging tool to find inconsistencies in the descriptor. Because development exceptions are not normal behavior, they must not be handled in a try-catch block.

The following exceptions are not dependent on runtime conditions. If one of these exceptions is thrown, then the application code being tested is invalid and must be changed. Avoid handling these types of exceptions:

- `DescriptorException`
- `BuilderException`
- `ConcurrencyException`
- `ConversionException`
- `QueryException`
- `ValidationException`

Format of Exceptions

All exceptions return the name of the exception and a message that describes what caused the exception. The message that appears reflects the type of exception.

OracleAS TopLink exceptions include the following information:

- The name of the OracleAS TopLink exception
- A description of the most probable cause of the error
- A native error code

Exception Error Code Numbers

OracleAS TopLink does not necessarily use the full range of exception error code numbers available. Table C–1 indicates the potential range:

Table C–1 Range of OracleAS TopLink Exception Error Codes

Exceptions	Error Code Range
Descriptor Exception	1 - 1000
Builder Exception	1001 - 2000
Concurrency Exception	2001 - 3000
Conversion Exception	3001 - 4000
Database Exception	4001 - 5000
Optimistic Lock Exception	5001 - 6000
Query Exception	6001 - 7000
Validation Exception	7001 - 8000
EJB QL Exception	8001 - 8999
Session Loader Exception	9000 - 10000
EJB Exception Factory	10001 - 11000
Cache Synch Communication Exception	11001 - 12000
Communication Exception	12001 - 13000
XML Data Store Exception	13001 - 14000
Deployment Exception	14001 - 15000
Synchronization Exception	15001 - 16000

Table C–1 Range of OracleAS TopLink Exception Error Codes (Cont.)

Exceptions	Error Code Range
JDO Exception	16001 - 17000
SDK Data Store Exception	17001 - 18000
JMS Processing Exception	18001 - 19000
SDK Descriptor Exception	19001 - 20000
SDK Query Exception	20001 - 21000
Discovery Exception	22000 - 22100
Remote Command Manager Exception	22101 - 22200
XML Conversion Exception	25001 - 26000
EJB JAR XML Exception	72001 - 73000

Exception Error Codes and Descriptions

This section lists the OracleAS TopLink exception error codes, information about the likely **Cause** of the problem and a possible corrective **Action**.

Each error code corresponds to an exception class and includes the following information:

- The exception number in the format, `EXCEPTION [TOPLINK-`*`XXXX`*`]`
- A description of the problem, taken from the thrown exception

Descriptor Exception

A Descriptor exception is a development exception raised when insufficient information has been provided to the descriptor. The message that is returned includes the name of the descriptor or mapping that caused the exception. If a mapping within the descriptor caused the error, then the name and parameters of the mapping are part of the returned message.

Internal exception, mapping and descriptor appear only if OracleAS TopLink has enough information about the source of the problem to provide this information.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
```

```
INTERNAL EXCEPTION: Message
MAPPING: Database mapping
DESCRIPTOR: Descriptor
```

Example C–1 Descriptor Exception

```
EXCEPTION [TOPLINK – 75]: oracle.toplink.exceptions.DescriptorException
EXCEPTION DESCRIPTION: The reference class is not specified.
```

Error Codes 1 – 176

Error code: 1

ATTRIBUTE_AND_MAPPING_WITH_INDIRECTION_ MISMATCH

Cause: `attributeName` is not declared as type `ValueHolderInterface`, but the mapping uses indirection. The mapping is set to use indirection, but the related attribute is not defined as type `ValueHolderInterface`. It is thrown on foreign reference mappings.

Action: If you want to use indirection on the mapping, change the attribute to type `ValueHolderInterface`. Otherwise, change the mapping associated with the attribute so that it does not use indirection.

Error code: 2

ATTRIBUTE_AND_MAPPING_WITHOUT_INDIRECTION_ MISMATCH

Cause: `attributeName` is declared as type `ValueHolderInterface`, but OracleAS TopLink is unable to use indirection. The attribute is defined to be of type `ValueHolderInterface`, but the mapping is not set to use indirection. It is thrown on foreign reference mappings.

Action: If you do not want to use indirection on the mapping, change the attribute to not be of type `ValueHolderInterface`. Otherwise, change the mapping associated with the attribute to use indirection.

Error code: 6

ATTRIBUTE_NAME_NOT_SPECIFIED

Cause: The attribute name is missing or not specified in the mapping definition.

Action: Specify the attribute name in the mapping by calling method `setAttributeName(String attribute name)`.

Error code: 7

ATTRIBUTE_TYPE_NOT_VALID

Cause: When using Java 2, the specified `attributeName` is not defined as type vector, or a type that implements Map or Collection. This occurs in one-to-many mapping, many-to-many mapping, and collection mapping when mapping is set not to use indirection, and the attribute type is not declared.

Action: Declare the attribute to be of type `java.util.Vector`.

Error code: 8

CLASS_INDICATOR_FIELD_NOT_FOUND

Cause: The class indicator field is defined, but the descriptor is set to use inheritance. When using inheritance, a class indicator field or class extraction method must be set. The class indicator field is used to create the right type of domain object.

Action: Set either a class indicator field or class extraction method.

Error code: 9

DIRECT_FIELD_NAME_NOT_SET

Cause: The direct field name from the target table is not set in the direct collection mapping.

Action: Specify the direct field name by calling method `setDirectFieldName(String fieldName)`.

Error code: 10

FIELD_NAME_NOT_SET_IN_MAPPING

Cause: The field name is not set in the mapping. It is thrown from direct to field mapping, array mapping, and structure mapping.

Action: Specify the field name by calling method `setFieldName(String fieldName)`.

Error code: 11

FOREIGN_KEYS_DEFINED_INCORRECTLY

Cause: One-to-one mapping foreign key is defined incorrectly. Multiple foreign key fields were set for one-to-one mapping by calling method `setForeignKeyFieldName(String fieldName)`.

Action: Use method `addForeignKeyFieldName(String sourceForeignKeyName`, `String targetPrimaryKeyFieldName)` to add multiple foreign key fields.

Error code: 12

IDENTITY_MAP_NOT_SPECIFIED

Cause: The descriptor must use an identity map to use the **Check cache does exist** option. The descriptor has been set to not use identity map, but the existence checking is set to be performed on identity map.

Action: Either use identity map, or set the existence checking to some other option.

Error code: 13

ILLEGAL_ACCESS_WHILE_GETTING_VALUE_THRU_ INSTANCE_ VARIABLE_ACCESSOR

Cause: OracleAS TopLink is unable to access the `attributeName` instance variable in object `objectName`. The instance variable in the domain object is not accessible. This exception is thrown when OracleAS TopLink tries to access the instance variable using Java reflection. The error is a purely Java exception, and OracleAS TopLink wraps only the reflection exception.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 14

ILLEGAL_ACCESS_WHILE_CLONING

Cause: OracleAS TopLink is unable to clone the object `domainObject` because the clone method `methodName` is not accessible. The method name specified using `useCloneCopyPolicy(String cloneMethodName)` or the `clone()` method to create the clone on the domain object, is not accessible by OracleAS TopLink using Java reflection. The error is a purely Java exception, and OracleAS TopLink wraps only the reflection exception.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 15

ILLEGAL_ACCESS_WHILE_CONSTRUCTOR_INSTANTIATION

Cause: The domain class does not define a public default constructor, which OracleAS TopLink needs to create new instances of the domain class.

Action: Define a public default constructor or use a different instantiation policy.

Error code: 16

ILLEGAL_ACCESS_WHILE_EVENT_EXECUTION

Cause: The descriptor callback method `eventMethodName` with `DescriptorEvent` as argument is not accessible. This exception is thrown when OracleAS TopLink tries to access the event method using Java reflection. The error is a purely Java exception, and OracleAS TopLink wraps only the reflection exception.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 17

ILLEGAL_ACCESS_WHILE_GETTING_VALUE_THRU_ METHOD_ ACCESSOR

Cause: Trying to invoke inaccessible `methodName` on the object `objectName`. The underlying get accessor method to access an attribute in the domain object is not accessible. This exception is thrown when OracleAS TopLink tries to access an attribute through method using Java reflection. The error is a purely Java exception, and OracleAS TopLink wraps only the reflection exception.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 18

ILLEGAL_ACCESS_WHILE_INSTANTIATING_METHOD_ BASED_PROXY

Cause: The method used by the transformation mapping using a valueholder is illegal. This exception is thrown when OracleAS TopLink tries to access the method using Java reflection. The problem occurs when the method base valueholder is instantiated.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 19

ILLEGAL_ACCESS_WHILE_INVOKING_ATTRIBUTE_METHOD

Cause: On transformation mapping, the underlying attribute method that is used to retrieve values from the database row while reading the transformation mapped attribute is not accessible.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 20

ILLEGAL_ACCESS_WHILE_INVOKING_FIELD_TO_METHOD

Cause: On transformation mapping, the method `methodName` that is used to retrieve value from the object while writing the transformation mapped attribute is not accessible. The error is a purely Java exception, and OracleAS TopLink wraps only the reflection exception.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 21

ILLEGAL_ACCESS_WHILE_INVOKING_ROW_EXTRACTION_ METHOD

Cause: OracleAS TopLink was unable to extract data `row`, because the OracleAS TopLink can not access the row specified in the `databaseRow` argument of the method. The method to extract class from row on the domain object is not accessible. The error is a purely Java exception, and OracleAS TopLink wraps only the reflection exception.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 22

ILLEGAL_ACCESS_WHILE_METHOD_INSTANTIATION

Cause: OracleAS TopLink is unable to create a new instance, because the method `methodName` that creates instances on the domain class is not accessible. The error is a purely Java exception, and OracleAS TopLink wraps only the reflection exception.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 23

ILLEGAL_ACCESS_WHILE_OBSOLETE_EVENT_EXECUTION

Cause: The descriptor callback method `eventMethodName` with `Session` as an argument is inaccessible. This exception is thrown when OracleAS TopLink tries to access the event method using Java reflection. The error is a purely Java exception, and OracleAS TopLink wraps only the reflection exception.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 24

ILLEGAL_ACCESS_WHILE_SETTING_VALUE_THRU_ INSTANCE_ VARIABLE_ACCESSOR

Cause: The `attributeName` instance variable in the object `objectName` is not accessible through Java reflection. The error is thrown by Java, and OracleAS TopLink wraps only the reflection exception.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 25

ILLEGAL_ACCESS_WHILE_SETTING_VALUE_THRU_ METHOD_ ACCESSOR

Cause: OracleAS TopLink is unable to invoke a method `setMethodName` on the object with parameter `parameter`. The attribute's set accessor method is not accessible through Java reflection. The error is thrown by Java and OracleAS TopLink wraps only the reflection exception.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 26

ILLEGAL_ARGUMENT_WHILE_GETTING_VALUE_ THRU_INSTANCE_ VARIABLE_ACCESSOR

Cause: OracleAS TopLink is unable to get a value for an instance variable `attributeName` of type `typeName` from the object. The specified object is not an instance of the class or interface declaring the underlying field. An object is accessed to get the value of an instance variable that does not exist.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 27

ILLEGAL_ARGUMENT_WHILE_GETTING_VALUE_THRU_ METHOD_ ACCESSOR

Cause: OracleAS TopLink is unable to invoke method `methodName` on the object `objectName`. The get accessor method declaration on the domain object differs from the one that is defined. The number of actual and formal parameters differ, or an unwrapping conversion has failed.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 28

ILLEGAL_ARGUMENT_WHILE_INSTANTIATING_METHOD_ BASED_ PROXY

Cause: The method that used by the method-based proxy uses in a Transformation mapping is getting illegal arguments when the valueholder is getting instantiated. This exception is thrown when OracleAS TopLink tries to access the method using Java reflection.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 29

ILLEGAL_ARGUMENT_WHILE_INVOKING_ATTRIBUTE_ METHOD

Cause: The number of actual and formal parameters differ, or an unwrapping conversion has failed. On transformation mapping, the method used to retrieve value from the database row while reading the transformation mapped attribute is getting an illegal argument.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 30

ILLEGAL_ARGUMENT_WHILE_INVOKING_FIELD_TO_ METHOD

Cause: The number of actual and formal parameters differ for method `methodName`, or an unwrapping conversion has failed. On transformation mapping, the method used to retrieve value from the object while writing the transformation mapped attribute is getting an illegal argument. The error is a purely Java exception, and OracleAS TopLink wraps only the reflection exception.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 31

ILLEGAL_ARGUMENT_WHILE_OBSOLETE_EVENT_ EXECUTION

Cause: The number of actual and formal parameters for the descriptor callback method `eventMethodName` differs, or an unwrapping conversion has failed. The callback event method is invoked with an illegal argument. This exception is thrown when OracleAS TopLink tries to invoke the event method using Java reflection. The error is a purely Java exception, and OracleAS TopLink wraps only the reflection exception.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 32

ILLEGAL_ARGUMENT_WHILE_SETTING_VALUE_THRU_ INSTANCE_ VARIABLE_ACCESSOR

Cause: An illegal value is being assigned to the attribute instance variable. OracleAS TopLink is unable to set a `value` for an instance variable `attributeName` of type `typeName` in the object. The specified object is not an instance of the class or interface that is declaring the underlying field, or an unwrapping conversion has failed.

OracleAS TopLink assigns value by using Java reflection. Java throws the error and OracleAS TopLink wraps only the reflection exception.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 33

ILLEGAL_ARGUMENT_WHILE_SETTING_VALUE_THRU _METHOD_ACCESSOR

Cause: An illegal argument is being passed to the attribute's set accessor method. OracleAS TopLink is unable to invoke method `setMethodName` on the object. The number of actual and formal parameters differs, or an unwrapping conversion has failed. Java throws the error and OracleAS TopLink wraps only the reflection exception.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 34

INSTANTIATION_WHILE_CONSTRUCTOR_INSTANTIATION

Cause: The class does not define a public default constructor, or the constructor raised an exception. This error occurs when you invoke the default constructor for the domain object to create a new instance of the object while building new domain objects if:

- The Class represents an abstract class, an interface, an array class, a primitive type, or void.
- The instantiation fails for some other reason.

Java throws the error and OracleAS TopLink wraps only the reflection exception.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 35

INVALID_DATA_MODIFICATION_EVENT

Cause: Applications should never encounter this exception. This exception usually occurs at the time of developing OracleAS TopLink, although in cases where the developer writes new mapping, it is possible to get this exception. In direct collection mapping and many-to-many mapping, the target table and relational table are populated at the end of the commit process, and if a data modification event is sent to any other mapping, then this exception is thrown.

Action: Contact Technical Support.

Error code: 36

INVALID_DATA_MODIFICATION_EVENT_CODE

Cause: An application should never encounter this exception. This exception usually occurs at the time of developing OracleAS TopLink, although in cases where developers write new mappings, it is possible to get this exception. In direct collection mapping and many-to-many mapping, the target table and relational table are populated at the end of the commit process, and if a data modification event is sent to these two mappings with wrong event code, then this exception is thrown.

Action: Contact Technical Support.

Error code: 37

INVALID_DESCRIPTOR_EVENT_CODE

Cause: An application should never encounter this exception. This exception usually occurs at the time of developing OracleAS TopLink. The exception means that the descriptor event manager does not support the event code passed in the event.

Action: Contact Technical Support.

Error code: 38

INVALID_IDENTITY_MAP

Cause: The identity map constructor failed because an invalid identity map was specified. The identity map class given in the descriptor cannot be instantiated. The exception is a Java exception thrown by a Java reflection when OracleAS TopLink instantiates the identity map class. OracleAS TopLink wraps only the Java exception.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 39

JAVA_CLASS_NOT_SPECIFIED

Cause: The descriptor does not define a Java class. The Java class is not specified in the descriptor.

Action: Specify the Java Class.

Error code: 40

DESCRIPTOR_FOR_INTERFACE_IS_MISSING

Cause: A descriptor for the referenced interface is not added to the session.

Action: Add that descriptor to the session.

Error code: 41

MAPPING_FOR_SEQUENCE_NUMBER_FIELD

Cause: A non-read-only mapping is not defined for the sequence number field. A mapping is required so that OracleAS TopLink can put and extract values for the primary key.

Action: Define a mapping.

Error code: 43

MISSING_CLASS_FOR_INDICATOR_FIELD_VALUE

Cause: OracleAS TopLink is missing the class for indicator field value `classFieldValue` of type `type`. There was no class entry found in the inheritance policy for the indicator field value that was read from the database. It is likely that the method `addClassIndicator(Class class, Object typeValue)` was not called for the field value. The class and `typeValue` is stored in the hashtable, and later the class is extracted from the hashtable by passing `typeValue` as a key. Because `Integer(1)` is not equivalent to `Float(1)`, this exception occurs when the type of `typeValue` is incorrectly specified.

Action: Verify the descriptor.

Error code: 44

MISSING_CLASS_INDICATOR_FIELD

Cause: The class indicator field is missing from the database `row` that was read from the database. This is performed in the inheritance model where after reading rows from the database, child domain objects are to be constructed depending upon the type indicator values.

Action: Verify the printed row for correct spelling.

Error code: 45

MISSING_MAPPING_FOR_FIELD

Cause: OracleAS TopLink is missing mapping for `field`; a mapping for the field is not specified.

Action: Define a mapping for the field.

Error code: 46

NO_MAPPING_FOR_PRIMARY_KEY

Cause: A mapping for the primary key is not specified. There should be one non-read-only mapping defined for the primary key field.

Action: Define a mapping for the primary key.

Error code: 47

MULTIPLE_TABLE_PRIMARY_KEY_NOT_SPECIFIED

Cause: The multiple table primary key mapping must be specified when a custom multiple table join is used. If multiple tables are specified in the descriptor and the join expression is customized, then the primary keys for all the tables must be specified. If the primary keys are not specified, then the exception occurs.

Action: Call method `addMultipleTablePrimaryKeyFieldName(String fieldNameInPrimaryTable, String fieldNameInSecondaryTable)` on the descriptor to set the primary keys.

Error code: 48

MULTIPLE_WRITE_MAPPINGS_FOR_FIELD

Cause: Multiple writable mappings for the field `fieldName` are defined in the descriptor. Exactly one must be defined writable; the others must be specified as read-only. When multiple write mappings are defined for the field, OracleAS TopLink is unable to choose the appropriate mapping for writing the value of the field in the database row. Therefore, the exception is thrown during the validation process of descriptors.

The most common cause of this problem occurs when the field has direct-to-field mapping, as well as one-to-one mapping. In this case, the one-to-one mapping must either be read-only or a target foreign key reference.

Action: Make one of those mappings read only.

Error code: 49

NO_ATTRIBUTE_TRANSFORMATION_METHOD

Cause: The attribute transformation method name in the transformation mapping is not specified. This method is invoked internally by OracleAS TopLink to retrieve value to store in the domain object.

Action: Define a method and set the method name on the mapping by calling method `setAttributeTransformation(String methodName)`.

Error code: 50

NO_FIELD_NAME_FOR_MAPPING

Cause: No field name is specified in direct-to-field mapping.

Action: Set the field by calling `setFieldName(String FieldName)`.

Error code: 51

NO_FOREIGN_KEYS_ARE_SPECIFIED

Cause: Neither the selection criteria nor the foreign keys were specified on one-to-one mapping. If the selection criterion is not specified then OracleAS TopLink tries to build one from the foreign keys specified in the mapping.

Action: Specify the fields.

Error code: 52

NO_REFERENCE_KEY_IS_SPECIFIED

Cause: No query key named: `queryKey` is found in: `descriptor`. No reference key from the target table is specified on direct collection mapping.

Action: Specify the fields by calling method `setReferenceKeyFieldName(String fieldName)`.

Error code: 53

NO_RELATION_TABLE

Cause: The relation table name is not set in this many-to-many mapping.

Action: Set the relation table name by calling method `setRelationTableName(String tableName)`.

Error code: 54

NO_SOURCE_RELATION_KEYS_SPECIFIED

Cause: There are no source relation keys specified in this many-to-many mapping.

Action: Add source relation keys to the mapping.

Error code: 55

NO_SUCH_METHOD_ON_FIND_OBSOLETE_METHOD

Cause: OracleAS TopLink cannot find the descriptor callback method `selector` on the domain class. It must take a `Session` or a `DescriptorEvent` as its argument. OracleAS TopLink tries to invoke the method using Java reflection. It is a Java exception and OracleAS TopLink is wrapping only the main exception.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 56

NO_SUCH_METHOD_ON_INITIALIZING_ ATTRIBUTE_METHOD

Cause: OracleAS TopLink cannot find the method `attributeMethodName` with parameters `databaseRow` or `databaseRow`, `session`. OracleAS TopLink wraps the Java reflection exception that is caused when the method is being created from the method name. This method is set by calling `setAttributeMethodName(String aMethodName)`.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 57

NO_SUCH_METHOD_WHILE_CONSTRUCTOR_ INSTANTIATION

Cause: The constructor is inaccessible to OracleAS TopLink. OracleAS TopLink wraps the Java reflection exception that is caused when it is creating a new instance of the domain.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 58

NO_SUCH_METHOD_WHILE_CONVERTING_TO_METHOD

Cause: The method `methodName` is not found with parameters () or `Session`. OracleAS TopLink wraps the Java reflection exception that is caused when it is creating a Method type from the method names in transformation mapping.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 59

NO_SUCH_FIELD_WHILE_INITIALIZING_ATTRIBUTES_ IN_INSTANCE_ VARIABLE_ACCESSOR

Cause: The instance variable `attributeName` is not defined in the domain class, or it is not accessible. OracleAS TopLink wraps the Java reflection exception that is caused when it is creating a Field type from the attribute name.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 60

NO_SUCH_METHOD_WHILE_INITIALIZING_ ATTRIBUTES_IN_METHOD_ ACCESSOR

Cause: The accessor method `setMethodName` or `getMethodName` is not defined for the attribute in the domain class `javaClassName`, or it is not accessible. OracleAS TopLink wraps the Java reflection exception that is caused when it is creating a Method type from the method name.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 61

NO_SUCH_METHOD_WHILE_INITIALIZING_ CLASS_EXTRACTION_ METHOD

Cause: The static class extraction method `methodName` with `databaseRow` as argument does not exist, or is not accessible. A Java reflection exception wrapped in an OracleAS TopLink exception is thrown when class extraction method is being created from the method name in the inheritance policy.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 62

NO_SUCH_METHOD_WHILE_INITIALIZING_COPY_POLICY

Cause: The clone method `methodName` with no arguments does not exist, or is not accessible. A Java reflection exception wrapped in an OracleAS TopLink exception is thrown when a method to create clones is being created from the method name in the copy policy.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 63

NO_SUCH_METHOD_WHILE_INITIALIZING_ INSTANTIATION_POLICY

Cause: The instance creation method `methodName` with no arguments does not exist, or is not accessible. A Java reflection exception wrapped in an OracleAS TopLink exception is thrown when a method to create the new instance is being created from the method name in the instantiation policy.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 64

NO_TARGET_FOREIGN_KEYS_SPECIFIED

Cause: The foreign keys in the target table are not specified in one-to-many mappings. These fields are not required if a selection criterion is given in the mapping, but otherwise they must be specified.

Action: Set target foreign keys or selection criteria.

Error code: 65

NO_TARGET_RELATION_KEYS_SPECIFIED

Cause: There are no target relation keys specified in many-to-many mappings.

Action: Call method `addTargetRelationKeyFieldName(String targetRelationKeyFieldName`, `String targetPrimaryKeyFieldName)` to set the fields.

Error code: 66

NOT_DESERIALIZABLE

Cause: The object cannot be deserialized from the byte array that is read from the database. The exception is thrown when the serialized object mapping is converting the byte array into an object.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 67

NOT_SERIALIZABLE

Cause: The object cannot be serialized into a byte array. The exception is thrown when a serialized object mapping is converting the object into a byte array.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 68

NULL_FOR_NON_NULL_AGGREGATE

Cause: The value of aggregate in the source object `object` is null. Null values are not allowed for aggregate mappings unless **allow null** is specified in the aggregate mapping.

Action: Call method `allowNull()` on the mapping.

Error code: 69

NULL_POINTER_WHILE_GETTING_VALUE_THRU_ INSTANCE_ VARIABLE_ACCESSOR

Cause: An object is accessed to get the value of an instance variable through Java reflection. This exception is thrown only on some Java virtual machines (JVM).

Action: Inspect the internal exception, and see the Java documentation.

Error code: 70

NULL_POINTER_WHILE_GETTING_VALUE_THRU_ METHOD_ACCESSOR

Cause: The get accessor method is invoked to get the value of an attribute through Java reflection. This exception is thrown only on some Java virtual machines (JVM).

Action: Inspect the internal exception, and see the Java documentation.

Error code: 71

NULL_POINTER_WHILE_SETTING_VALUE_THRU_ INSTANCE_VARIABLE_ ACCESSOR

Cause: Null Pointer Exception has been thrown while setting the value of the `attributeName` instance variable in the object to value. An object is accessed to set the value of an instance variable through Java reflection. This exception is thrown only on some Java virtual machines (JVM).

Action: Inspect the internal exception, and see the Java documentation.

Error code: 72

NULL_POINTER_WHILE_SETTING_VALUE_THRU_ METHOD_ACCESSOR

Cause: A Null Pointer Exception has been thrown while setting the value through `setMethodName` method in the object with an argument `argument`. The set accessor method is invoked to set the value of an attribute through Java reflection. This exception is thrown only on some Java virtual machines (JVM).

Action: Inspect the internal exception, and see the Java documentation.

Error code: 73

PARENT_DESCRIPTOR_NOT_SPECIFIED

Cause: OracleAS TopLink is unable to find descriptor for the parent class. The descriptor of a subclass has no parent descriptor.

Action: The method `setParentClass(Class parentClass)` on the subclass descriptor must be called.

Error code: 74

PRIMARY_KEY_FIELDS_NOT_SPECIFIED

Cause: The primary key fields are not set for this descriptor.

Action: Add primary key field names using method `setPrimaryKeyFieldName(String fieldName)` or `setPrimaryKeyFieldName(String fieldName)`.

Error code: 75

REFERENCE_CLASS_NOT_SPECIFIED

Cause: The reference class is not specified in the foreign reference mapping.

Action: Set the reference class by calling method `setReferenceClass(Class aClass)`.

Error code: 77

REFERENCE_DESCRIPTOR_IS_NOT_AGGREGATE

Cause: The referenced descriptor for `className` should be set to an aggregate descriptor. An aggregate mapping should always reference a descriptor that is aggregate.

Action: Call method `descriptorIsAggregate()` on the referenced descriptor.

Error code: 78

REFERENCE_KEY_FIELD_NOT_PROPERLY_SPECIFIED

Cause: The table for the reference field must be the reference table. If the reference field name that is specified in the direct collection mapping is qualified with the table name, then the table name should match the reference table name.

Action: Qualify the field with the proper name, or change the reference table name.

Error code: 79

REFERENCE_TABLE_NOT_SPECIFIED

Cause: The reference table name in the direct collection mapping is not specified.

Action: Use method `setReferenceTableName(String tableName)` on the mapping to set the table name.

Error code: 80

RELATION_KEY_FIELD_NOT_PROPERLY_SPECIFIED

Cause: The table for the relation key field must be the relation table. If the source and target relation fields name that is specified in the many-to-many mapping are qualified with the table name, then the table name should match the relation table name.

Action: Qualify the field with the proper name, or change the relation table name.

Error code: 81

RETURN_TYPE_IN_GET_ATTRIBUTE_ACCESSOR

Cause: The method `attributeMethodName` that is specified in the transformation mapping should have a return type set in the attribute because this method is used to extract value from the database row.

Action: Verify the method and make appropriate changes.

Error code: 82

SECURITY_ON_FIND_METHOD

Cause: The descriptor callback method `selector` with `DescriptorEvent` as an argument is not accessible. Java throws a security exception when a Method type is created from the method name using Java reflection. The method is a descriptor event callback on the domain object that takes `DescriptorEvent` as its parameter.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 83

SECURITY_ON_FIND_OBSOLETE_METHOD

Cause: The descriptor callback method `selector` with `session` as an argument is not accessible. Java throws a security exception when a Method type is created from the method name using Java reflection. The method is a descriptor event callback on the domain object, which takes class and session as its parameters.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 84

SECURITY_ON_INITIALIZING_ATTRIBUTE_METHOD

Cause: Access to the method `attributeMethodName` with parameters `databaseRow` or `databaseRow`, `Session` has been denied. Java throws a security exception when a Method type is created from the attribute method name using Java reflection. The attribute method that is specified in the transformation mapping is used to extract value from the database row and set by calling `setAttributeTransformation(String methodName)`.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 85

SECURITY_WHILE_CONVERTING_TO_METHOD

Cause: Method: `methodName` ACCESS DENIED with <> or `session` parameters. Java throws a security exception when a Method type is created from the method name using Java reflection. These are the methods that extract the field value from the domain object in the transformation mapping.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 86

SECURITY_WHILE_INITIALIZING_ATTRIBUTES_IN_ INSTANCE_ VARIABLE_ACCESSOR

Cause: Access to the instance variable `attributeName` in the class `javaClassName` is denied. Java throws a security exception when creating a Field type from the given attribute name using Java reflection.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 87

SECURITY_WHILE_INITIALIZING_ATTRIBUTES_IN_ METHOD_ACCESSOR

Cause: The methods `setMethodName` and `getMethodName` in the object `javaClassName` are inaccessible. Java throws a security exception when creating a Method type from the given attribute accessor method name using Java reflection.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 88

SECURITY_WHILE_INITIALIZING_CLASS_ EXTRACTION_METHOD

Cause: The static class extraction method `methodName` with `DatabaseRow` as an argument is not accessible. Java throws a security exception when creating a Method type from the given class extraction method name using Java reflection. The method is used to extract the class from the database row in inheritance policy.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 89

SECURITY_WHILE_INITIALIZING_COPY_POLICY

Cause: The clone method `methodName` with no arguments is inaccessible. Java throws a security exception when creating Method type from the given method name using Java reflection. This method on copy policy is used to create clones of the domain object.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 90

SECURITY_WHILE_INITIALIZING_INSTANTIATION_POLICY

Cause: The instance creation method `methodName` with no arguments is inaccessible. Java throws a security exception when creating Method type from the given method name using Java reflection. This method on instantiation policy is used to create new instances of the domain object.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 91

SEQUENCE_NUMBER_PROPERTY_NOT_SPECIFIED

Cause: Either the sequence field name or the sequence number name is missing. To use sequence generated IDs both the sequence number name and field name properties must be set.

Action: To use sequence-generated IDs, set both the sequence number name and field name properties.

Error code: 92

SIZE_MISMATCH_OF_FOREIGN_KEYS

Cause: The size of the primary keys on the target table does not match the size of the foreign keys on the source in one-to-one mapping.

Action: Verify the mapping and the reference descriptor's primary keys.

Error code: 93

TABLE_NOT_PRESENT

Cause: The table `tableName` is not present in the descriptor.

Action: Verify the qualified field names that are specified in the mappings and descriptor so that any fields that are qualified with the table name reference the correct table.

Error code: 94

TABLE_NOT_SPECIFIED

Cause: No table is specified in the descriptor. The descriptor must have a table name defined.

Action: Call method `addTableName(String tableName)` or `setTableName(String tableName)` to set the tables on the descriptor.

Error code: 96

TARGET_FOREIGN_KEYS_SIZE_MISMATCH

Cause: The size of the foreign keys on the target table does not match the size of the source keys on the source table in the one-to-many mapping.

Action: Verify the mapping.

Error code: 97

TARGET_INVOCATION_WHILE_CLONING

Cause: OracleAS TopLink has encountered a problem in cloning the object `domainObject` clone method. The `methodName` triggered an exception. Java throws this exception when the cloned object is invoked while the object is being cloned. The clone method is specified on the copy policy that is usually invoked to create clones in Unit of Work.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 98

TARGET_INVOCATION_WHILE_EVENT_EXECUTION

Cause: A descriptor callback method `eventMethodName` that includes a `DescriptorEvent` as argument. The exception occurs when the descriptor event method is invoked using Java reflection.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 99

TARGET_INVOCATION_WHILE_GETTING_VALUE_ THRU_METHOD_ ACCESSOR

Cause: The method `methodName` on the object `objectName` is throwing an exception. Java is throwing an exception while getting an attribute value from the object through a method accessor.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 100

TARGET_INVOCATION_WHILE_INSTANTIATING_ METHOD_BASED_ PROXY

Cause: A method has thrown an exception. Java throws this exception while instantiating a method based proxy and instantiating transformation mapping.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 101

TARGET_INVOCATION_WHILE_INVOKING_ ATTRIBUTE_METHOD

Cause: The underlying method throws an exception. Java is throwing an exception while invoking an attribute transformation method on transformation mapping. The method is invoked to extract value from the database row to set into the domain object.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 102

TARGET_INVOCATION_WHILE_INVOKING_FIELD_ TO_METHOD

Cause: The method `methodName` is throwing an exception. Java is throwing exception while invoking field transformation method on transformation mapping. The method is invoked to extract value from the domain object to set into the database row.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 103

TARGET_INVOCATION_WHILE_INVOKING_ROW_ EXTRACTION_ METHOD

Cause: OracleAS TopLink encountered a problem extracting the class type from row `row` while invoking a class extraction method.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 104

TARGET_INVOCATION_WHILE_METHOD_INSTANTIATION

Cause: OracleAS TopLink is unable to create a new instance. The creation method `methodName` caused an exception.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 105

TARGET_INVOCATION_WHILE_OBSOLETE_ EVENT_EXECUTION

Cause: The underlying descriptor callback method `eventMethodName` with `session` as argument throws an exception. Java is throwing an exception while invoking a descriptor event method that takes a session as its parameter.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 106

TARGET_INVOCATION_WHILE_SETTING_VALUE_THRU_ METHOD_ ACESSOR

Cause: The method `setMethodName` on the object throws an exception. Java is throwing an exception while invoking a set accessor method on the domain object to set an attribute value into the domain object.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 108

VALUE_NOT_FOUND_IN_CLASS_INDICATOR_MAPPING

Cause: The indicator value is not found in the class indicator mapping in the parent descriptor for the class.

Action: Verify the `addClassIndicator(Class childClass, Object typeValue)` on the inheritance policy.

Error code: 109

WRITE_LOCK_FIELD_IN_CHILD_DESCRIPTOR

Cause: The child descriptor has a write lock field defined. This is unnecessary, because it inherits any required locking from the parent descriptor.

Action: Check your child descriptor, and remove the field.

Error code: 110

DESCRIPTOR_IS_MISSING

Cause: The descriptor for the reference class `className` is missing from the mapping.

Action: Verify the session to see if the descriptor for the reference class was added.

Error code: 111

MULTIPLE_TABLE_PRIMARY_KEY_MUST_BE_ FULLY_QUALIFIED

Cause: Multiple table primary key field names are not fully qualified. These field names are given on the descriptor if it has more than one table.

Action: Specify the field names with the table name.

Error code: 112

ONLY_ONE_TABLE_CAN_BE_ADDED_WITH_THIS_METHOD

Cause: You have tried to enter more than one table through this method.

Action: Use `addTableName(String tableName)` to add multiple tables to the descriptor.

Error code: 113

NULL_POINTER_WHILE_CONSTRUCTOR_INSTANTIATION

Cause: The constructor is inaccessible. Java is throwing this exception while invoking a default constructor to create new instances of the domain object.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 114

NULL_POINTER_WHILE_METHOD_INSTANTIATION

Cause: The new instance `methodName` creation method is inaccessible. Java is throwing an exception while calling a method to a build new instance of the domain object. This method is given by the user to override the default behavior of creating new instances through a class constructor.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 115

NO_ATTRIBUTE_VALUE_CONVERSION_TO_FIELD_VALUE_PROVIDED

Cause: The field conversion value for the attribute value `attributeValue` was not given in the object type mapping.

Action: Verify the attribute value, and provide a corresponding field value in the mapping.

Error code: 116

NO_FIELD_VALUE_CONVERSION_TO_ATTRIBUTE_ VALUE_PROVIDED

Cause: The attribute conversion value for the `fieldValue` was not given in the object type mapping.

Action: Verify the field value, and provide a corresponding attribute value in the mapping.

Error code: 118

LOCK_MAPPING_CANNOT_BE_READONLY

Cause: The domain object `className` cannot have a read only mapping for the write lock fields when the version value is stored in the object.

Action: Verify the mappings on the write lock fields.

Error code: 119

LOCK_MAPPING_MUST_BE_READONLY

Cause: The domain object `className` does not have a read only mapping for the write lock fields when the version value is stored in the cache.

Action: Verify the mappings on write lock fields.

Error code: 120

CHILD_DOES_NOT_DEFINE_ABSTRACT_QUERY_KEY

Cause: The queryKey `queryKeyName` is defined in the parent descriptor but not in the child descriptor. The descriptor has not defined the abstract query key.

Action: Define any implementors of the interface descriptor by the abstract query key in the interface descriptor.

Error code: 122

SET_EXISTENCE_CHECKING_NOT_UNDERSTOOD

Cause: The interface descriptor `parent` does not have at least one abstract query key defined. The string given to the method `setExistenceChecking(String token)` is not understood.

Action: Verify that the string passed is one of the following:

- Check cache
- Check database
- Assume existence
- Assume non-existence

Error code: 125

VALUE_HOLDER_INSTANTIATION_MISMATCH

Cause: The mapping for the attribute `mapping.getAttributeName()` uses indirection and must be initialized to a new valueholder.

Action: Ensure that the mapping uses indirection and that the attribute is initialized to a new valueholder.

Error code: 126

NO_SUB_CLASS_MATCH

Cause: No subclass matches this class `theClass` when inheritance is in aggregate relationship mapping.

Action: Verify the subclass and the relationship mapping.

Error code: 127

RETURN_AND_MAPPING_WITH_INDIRECTION_MISMATCH

Cause: The get method return type for the attribute `mapping.getAttributeName`() is not declared as type `ValueHolderInterface`, but the mapping is using indirection.

Action: Verify that the get method returns a valueholder, or change the mapping to not use indirection.

Error code: 128

RETURN_AND_MAPPING_WITHOUT_INDIRECTION_ MISMATCH

Cause: The get method return type for the attribute `mapping.getAttributeName()` is declared as type `ValueHolderInterface`, but the mapping is not using indirection.

Action: Ensure that the mapping is using indirection, or change the return type from valueholder.

Error code: 129

PARAMETER_AND_MAPPING_WITH_INDIRECTION_ MISMATCH

Cause: The set method parameter type for the attribute `mapping.getAttributeName()` is not declared as type `ValueHolderInterface`, but the mapping is using indirection.

Action: Ensure that the set method parameter is declared as a valueholder, or change the mapping to not use indirection.

Error code: 130

PARAMETER_AND_MAPPING_WITHOUT_INDIRECTION_ MISMATCH

Cause: The set method parameter type for the attribute `mapping.getAttributeName()` is declared as type `ValueHolderInterface`, but the mapping is not using indirection.

Action: Ensure that the mapping is changed to use indirection, or that the method parameter is not declared as a valueholder.

Error code: 131

GET_METHOD_RETURN_TYPE_NOT_VALID

Cause: The get method return type for the attribute `mapping.getAttributeName()` is not declared as type vector (or a type that implements Map or Collection if using Java 2).

Action: Declare the get method return type for the attribute as type vector (or a type that implements Map or Collection if using Java 2).

Error code: 133

SET_METHOD_PARAMETER_TYPE_NOT_VALID

Cause: The set method parameter type for the attribute `mapping.getAttributeName()` is not declared as type vector (or a type that implements Map or Collection, if using Java 2).

Action: Declare the set method parameter type for the attribute as type vector (or a type that implements Map or Collection, if using Java 2).

Error code: 135

ILLEGAL_TABLE_NAME_IN_MULTIPLE_TABLE_ FOREIGN_KEY

Cause: The table in the multiple table foreign key relationship refers to an unknown table.

Action: Verify the table name.

Error code: 138

ATTRIBUTE_AND_MAPPING_WITH_TRANSPARENT_ INDIRECTION_ MISMATCH

Cause: The attribute `mapping.getAttributeName()` is not declared as a super-type of `validTypeName`, but the mapping is using transparent indirection.

Action: Verify the attribute's type and the mapping setup.

Error code: 139

RETURN_AND_MAPPING_WITH_TRANSPARENT_ INDIRECTION_ MISMATCH

Cause: The get method return type for the attribute `mapping.getAttributeName()` is not declared as a super-type of `validTypeName`, but the mapping is using transparent indirection.

Action: Verify the attribute's type and the mapping setup.

Error code: 140

PARAMETER_AND_MAPPING_WITH_TRANSPARENT_ INDIRECTION_ MISMATCH

Cause: The set method parameter type for the attribute `mapping.getAttributeName()` is not declared as a super-type of `validTypeName`, but the mapping is using transparent indirection.

Action: Verify the attribute's type and the mapping setup.

Error code: 141

FIELD_IS_NOT_PRESENT_IN_DATABASE

Cause: The field `fieldname` is not present in the table `tableName` in the database.

Action: Verify the field name for the attribute.

Error code: 142

TABLE_IS_NOT_PRESENT_IN_DATABASE

Cause: The `descriptor.getTableName()` is not present in the database.

Action: Verify the table name for the descriptor.

Error code: 143

MULTIPLE_TABLE_INSERT_ORDER_MISMATCH

Cause: The multiple table insert order vector specified `aDescriptor.getMultipleTableInsertOrder()` has more or fewer tables than are specified in the descriptor `aDescriptor.getTables()`. All the tables must be included in the insert order vector.

Action: Ensure that all table names for the descriptor are present and that there are no extra tables.

Error code: 144

INVALID_USE_OF_TRANSPARENT_INDIRECTION

Cause: Transparent indirection is being used with a mapping other than `CollectionMappings`.

Action: Verify the mapping. It must be a collection mapping.

Error code: 145

MISSING_INDIRECT_CONTAINER_CONSTRUCTOR

Cause: The indirect container class does not implement the constructor.

Action: Implement the constructor for the container.

Error code: 146

COULD_NOT_INSTANTIATE_INDIRECT_CONTAINER_CLASS

Cause: OracleAS TopLink is unable to instantiate the indirect container class using the constructor.

Action: Validate the constructor for the indirect container class.

Error code: 147

INVALID_CONTAINER_POLICY

Cause: You have used a Container Policy with an incompatible version of the JDK. This container policy must only be used in JDK 1.3.1 or higher.

Action: Validate the container policy being used.

Error code: 148

INVALID_CONTAINER_POLICY_WITH_TRANSPARENT_ INDIRECTION

Cause: The container policy is incompatible with transparent indirection.

Action: Change the container policy to be compatible with transparent indirection, or do not use transparent indirection.

Error code: 149

INVALID_USE_OF_NO_INDIRECTION

Cause: No indirection should not receive this message.

Action: Change to use no indirection.

Error code: 150

INDIRECT_CONTAINER_INSTANTIATION_MISMATCH

Cause: The mapping for the attribute `mapping.getAttributeName()` uses transparent indirection and must be initialized to an appropriate container.

Action: Initialize the mapping to an appropriate container.

Error code: 151

INVALID_MAPPING_OPERATION

Cause: An invalid mapping operation has been used.

Action: See the documentation for valid mapping operations.

Error code: 152

INVALID_INDIRECTION_POLICY_OPERATION

Cause: An invalid indirection policy operation has been used.

Action: See the documentation for valid indirection policy operations.

Error code: 153

REFERENCE_DESCRIPTOR_IS_NOT_ AGGREGATECOLLECTION

Cause: The reference descriptor for `className` is not set to an aggregate collection descriptor.

Action: Set the reference descriptor to an aggregate collection descriptor.

Error code: 154

INVALID_INDIRECTION_CONTAINER_CLASS

Cause: An invalid indirection container class has been used.

Action: Verify the container class.

Error code: 155

MISSING_FOREIGN_KEY_TRANSLATION

Cause: The mapping does not include a foreign key field linked to the primary key field.

Action: Link the foreign key to the appropriate primary key.

Error code: 156

STRUCTURE_NAME_NOT_SET_IN_MAPPING

Cause: The structure name is not set.

Action: Set the structure name appropriately.

Error code: 157

NORMAL_DESCRIPTORS_DO_NOT_SUPPORT_ NON_RELATIONAL_ EXTENSIONS

Cause: Normal descriptors do not support non-relational extensions.

Action: Contact Technical Support.

Error code: 158

PARENT_CLASS_IS_SELF

Cause: The descriptor's parent class has been set to itself.

Action: Contact Technical Support.

Error code: 159

PROXY_INDIRECTION_NOT_AVAILABLE

Cause: An attempt to use proxy indirection has been made, but JDK 1.3.1 or higher is not being used.

Action: Use JDK 1.3.1 or higher.

Error code: 160

INVALID_ATTRIBUTE_TYPE_FOR_PROXY_INDIRECTION

Cause: The attribute was not specified in the list of interfaces given to use proxy indirection.

Action: Verify the attribute.

Error code: 161
INVALID_GET_RETURN_TYPE_FOR _PROXY_INDIRECTION

Cause: The return type for the indirection policy is invalid for the indirection policy.

Action: Ensure that the parameter type of the attribute's `get` method is correct for the indirection policy.

Error code: 162
INVALID_SET_PARAMETER_TYPE_FOR_PROXY_ INDIRECTION

Cause: The parameter for the set method is incorrect for the indirection type.

Action: Ensure that the parameter type of the attribute's `set` method is correct for the indirection policy.

Error code: 163
INCORRECT_COLLECTION_POLICY

Cause: The container policy is invalid for the collection type.

Action: Ensure that the container policy is correct for the collection type.

Error code: 164
INVALID_AMENDMENT_METHOD

Cause: The amendment method that is provided is invalid, not public, or cannot be found.

Action: Ensure that the amendment method is public, static, returns void, and has a single argument: `Descriptor`.

Error code: 165
ERROR_OCCURRED_IN_AMENDMENT_METHOD

Cause: The specified amendment method threw an exception.

Action: Examine the returned exception for further information.

Error code: 166
VARIABLE_ONE_TO_ONE_MAPPING_IS_NOT_DEFINED

Cause: There is no mapping for the attribute.

Action: Validate the mapping and attribute.

Error code: 168

TARGET_INVOCATION_WHILE_CONSTRUCTOR_ INSTANTIATION

Cause: The constructor is missing.

Action: Create the required constructor.

Error code: 169

TARGET_INVOCATION_WHILE_CONSTRUCTOR_ INSTANTIATION_OF_ FACTORY

Cause: The constructor is missing.

Action: Create the required constructor.

Error code: 170

ILLEGAL_ACCESS_WHILE_CONSTRUCTOR_ INSTANTIATION_OF_ FACTORY

Cause: Permissions do not allow access to the constructor.

Action: Adjust the Java security permissions to permit access to the constructor.

Error code: 171

INSTANTIATION_WHILE_CONSTRUCTOR_ INSTANTIATION_OF_ FACTORY

Cause: An instantiation failed inside the associated constructor.

Action: Determine which objects are being instantiated, and verify that all are instantiated properly.

Error code: 172

NO_SUCH_METHOD_WHILE_CONSTRUCTOR_ INSTANTIATION_OF_ FACTORY

Cause: A message send invoked from inside the constructor is invalid because the method does not exist.

Action: Correct the message send, ensuring that the message exists.

Error code: 173

NULL_POINTER_WHILE_CONSTRUCTOR_ INSTANTIATION_OF_ FACTORY

Cause: A message was sent from inside a constructor to a null object.

Action: Avoid sending a message to an object that is null.

Error code: 174

ILLEGAL_ACCESS_WHILE_METHOD_ INSTANTIATION_OF_FACTORY

Cause: A message was sent to an object from inside a factory instantiation, and Java has determined this message to be illegal.

Action: Determine why the message sent is illegal, and replace the message with the proper legal one.

Error code: 175

TARGET_INVOCATION_WHILE_METHOD_ INSTANTIATION_OF_ FACTORY

Cause: There is an error inside the factory associated with the invocation of a target.

Action: Determine the faulty target, and replace it with the correct target or proper message send.

Error code: 176

NULL_POINTER_WHILE_METHOD_ INSTANTIATION_OF_FACTORY

Cause: A message was sent to null inside a factory instantiation.

Action: Avoid sending a message to null.

Builder Exceptions

A Builder exception is a development exception that is raised when the Builder file format for the descriptor is not in a proper state. If OracleAS TopLink is able to determine the source and line number of the descriptor file that caused the exception, the displayed message includes this information. Otherwise, the information does not appear in the error message.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
INTERNAL EXCEPTION: Message
SOURCE: The source to the descriptor file that caused the error.
LINE NUMBER: The line number that caused the exception to be raised. This is the
line number in the descriptor file.
```

Example C–2 Builder Exception

```
EXCEPTION [TOPLINK - 1038]: oracle.toplink.tools.builderreader.BuilderException
EXCEPTION DESCRIPTION: No such section token: ABC
```

Error Codes 1001 – 1042

Error code: 1001

No such method

Cause: Tokens in the builder-generated files are the subsets of all the tokens a Project Reader can understand. Each token has a related public method on OracleAS TopLink. The exception means that the method name is incorrect.

Action: If the project files are not manually edited and corrupted, then this exception is usually internal to OracleAS TopLink and must be reported to Technical Support. But if the file was manually edited or corrupted, then generate the files again.

Error code: 1002

Could not find post load method `methodName` **on class** `aClass`

Cause: The post load method that was defined in the descriptor properties is not defined on the related domain class.

Action: Define the method on the specified class.

Error code: 1003

Cannot write parameter `object` **of class** `type`

Cause: While creating a project class, the parameter tokens are read from the file and are converted to actual types before sending them to the methods. An unknown type causes this exception.

Action: If the project files are not manually edited and corrupted, then this exception is usually internal to OracleAS TopLink and must be reported to Technical Support. But if the file was manually edited or corrupted, then generate the files again.

Error code: 1004

Could not access method `method`

Cause: Java is throwing an illegal access reflection exception while invoking the method on the object. OracleAS TopLink wraps only that exception.

Action: Inspect the internal exception, and see the Java documentation. If the project files are not manually edited and corrupted, then this exception is usually internal to OracleAS TopLink and must be reported to Technical Support. But if the file was manually edited or corrupted, then generate the files again.

Error code: 1005

Invoking `applyResultMethod` **raised exception** `exception`

Cause: Java is throwing an invocation reflection exception while invoking the method on the object. OracleAS TopLink wraps only that exception.

Action: Inspect the internal exception, and see the Java documentation. If the project files are not manually edited and corrupted, then this exception is usually internal to OracleAS TopLink and must be reported to Technical Support. But if the file was manually edited or corrupted, then generate the files again.

Error code: 1006

Invalid arguments invoking: `applyResultMethod` **with** `receiver`

Cause: Java is throwing an invalid argument reflection exception while invoking the method on the object. OracleAS TopLink wraps only that exception.

Action: Inspect the internal exception, and see the Java documentation. If the project files are not manually edited and corrupted, then this exception is usually internal to OracleAS TopLink and must be reported to Technical Support. But if the file was manually edited or corrupted, then generate the files again.

Error code: 1007

Could not access `applyResultMethod` **with** `receiver`

Cause: Java is throwing a reflection exception while invoking the method on the object. OracleAS TopLink wraps only that exception.

Action: Inspect the internal exception, and see the Java documentation. If the project files are not manually edited and corrupted, then this exception is usually internal to OracleAS TopLink and must be reported to Technical Support. But if the file was manually edited or corrupted, then generate the files again.

Error code: 1008

Parameter mismatch `method`; received `size` parameters

Cause: The number of parameters for the token read from the project or descriptor file do not match the number of parameters that a related method can take.

Action: Inspect the internal exception, and see the Java documentation. If the project files are not manually edited and corrupted, then this exception is usually internal to OracleAS TopLink and must be reported to Technical Support. But if the file was manually edited or corrupted, then generate the files again.

Error code: 1009

Accessing `methodName` on `className` with `parameters`

Cause: Java is throwing a reflection exception while invoking the method on the object. OracleAS TopLink wraps only that exception.

Action: Inspect the internal exception, and see the Java documentation. If the project files are not manually edited and corrupted, then this exception is usually internal to OracleAS TopLink and must be reported to Technical Support. But if the file was manually edited or corrupted, then generate the files again.

Error code: 1010

Could not find section definition `section` when building section definitions for `target`

Cause: An invalid section name was found in the project or descriptor file.

Action: Inspect the internal exception, and see the Java documentation. If the project files are not manually edited and corrupted, then this exception is usually internal to OracleAS TopLink and must be reported to Technical Support. But if the file was manually edited or corrupted, then generate the files again.

Error code: 1011

Could not convert `object` into an accessible Java class

Cause: The parameter read from the file cannot be converted to an appropriate type.

Action: Inspect the internal exception, and see the Java documentation. If the project files are not manually edited and corrupted, then this exception is usually internal to OracleAS TopLink and must be reported to Technical

Support. But if the file was manually edited or corrupted, then generate the files again.

Error code: 1012

File not found

Cause: The project or descriptor file was not found.

Action: Ensure that the path was given correctly in a project reader and also that the path is correct in the project file.

Error code: 1013

Invalid class/method name format

Cause: The application attempted to use the URL string to read the INI file.

Action: Use another method to read the INI files.

Error code: 1015

Open failed for URL `url`

Cause: OracleAS TopLink is unable to open the URL.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 1016

Could not resolve INIFile location: `sourceString` **using search paths** `searchPaths`

Cause: The file was not found on the given search paths.

Action: Check your search paths.

Error code: 1017

Invoking `method` **on** `receiver`

Cause: Java is throwing a reflection exception while invoking the method on the object. OracleAS TopLink wraps only that exception.

Action: Inspect the internal exception, and see the Java documentation. If the project files are not manually edited and corrupted, then this exception is usually internal to OracleAS TopLink and must be reported to Technical Support. But if the file was manually edited or corrupted, then generate the files again.

Error code: 1018

Invoking `method` on `receiver`

Cause: Java is throwing a reflection exception while invoking the method on the object. OracleAS TopLink wraps only that exception.

Action: If the project files are not manually edited and corrupted, then this exception is usually an internal to OracleAS TopLink and must be reported to Technical Support. But if the file was manually edited or corrupted, then generate the files again.

Error code: 1019

Invalid character value; expecting $* format

Cause: An invalid character format was written to the file.

Action: If the project files are not manually edited and corrupted, then this exception is usually internal to OracleAS TopLink and must be reported to Technical Support. But if the file was manually edited or corrupted, then generate the files again.

Error code: 1020

Unexpected character: {

Cause: An unexpected character was found while reading vector values from the file.

Action: If the project files are not manually edited and corrupted, then this exception is usually an internal to OracleAS TopLink and must be reported to Technical Support. But if the file was manually edited or corrupted, then generate the files again.

Error code: 1021

Unexpected character: }

Cause: An unexpected character was found while reading vector values from the file.

Action: If the project files are not manually edited and corrupted, then this exception is usually internal to OracleAS TopLink and must be reported to Technical Support. But if the file was manually edited or corrupted, then generate the files again.

Error code: 1022

Expecting `object`**, found token** `nextToken`

Cause: An unexpected token was found while reading from the file.

Action: If the project files are not manually edited and corrupted, then this exception is usually internal to OracleAS TopLink and must be reported to Technical Support. But if the file was manually edited or corrupted, then generate the files again.

Error code: 1023

Unexpected word

Cause: An unexpected token was found while reading from the file.

Action: If the project files are not manually edited and corrupted, then this exception is usually internal to OracleAS TopLink and must be reported to Technical Support. But if the file was manually edited or corrupted, then generate the files again.

Error code: 1024

`setExistenceChecking token`**; not understood**

Cause: The existence checking string that was specified on the descriptor is not understood.

Action: If the project files are not manually edited and corrupted, then this exception is usually internal to OracleAS TopLink and must be reported to Technical Support. But if the file was manually edited or corrupted, then generate the files again.

Error code: 1025

Class `className` **not found**

Cause: Java is throwing a reflection exception while invoking the method on the object. OracleAS TopLink wraps only that exception.

Action: Inspect the internal exception, and see the Java documentation. If the project files are not manually edited and corrupted, then this exception is usually internal to OracleAS TopLink and must be reported to Technical Support. But if the file was manually edited or corrupted, then generate the files again.

Error code: 1026

Not enough INI elements. Found `count`

Cause: The line in an INI file has fewer tokens than expected.

Action: If the project files are not manually edited and corrupted, then this exception is usually internal to OracleAS TopLink and must be reported to Technical Support. But if the file was manually edited or corrupted, then generate the files again.

Error code: 1027

Too many INI elements. Found `count`

Cause: The line in an INI file has more tokens then needed.

Action: If the project files are not manually edited and corrupted, then this exception is usually internal to OracleAS TopLink and must be reported to Technical Support. But if the file was manually edited or corrupted, then generate the files again.

Error code: 1028

Error writing `writeString`

Cause: OracleAS TopLink is unable to write the string into the file. A common cause of this error is an incorrectly specified directory.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 1029

Illegal access exception

Cause: Java is throwing a reflection exception while invoking the method on the object. OracleAS TopLink wraps only that exception.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 1030

Invocation target exception

Cause: Java is throwing a reflection exception while invoking the method on the object. OracleAS TopLink wraps only that exception.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 1031

Attempting to instantiate `className` with default constructor

Cause: Java is throwing a reflection exception while instantiating the object. OracleAS TopLink wraps only that exception.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 1032

Attempting to instantiate `className` with default constructor

Cause: Java is throwing a reflection exception while instantiating the object. OracleAS TopLink wraps only that exception.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 1033

IO Exception in next token

Cause: Java is throwing a reflection. OracleAS TopLink wraps only that exception.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 1034

IOException on close

Cause: Java is throwing a reflection. OracleAS TopLink wraps only that exception.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 1035

Invalid INI(URL) Method: `method`. Should return a string

Cause: Do not use the URL to read INI files. This feature is untested and undocumented.

Action: Use another method to read the INI files.

Error code: 1036

Could not cast using `castString`

Cause: An error occurred during an attempt to cast using the `castString`.

Action: Validate the `castString`.

Error code: 1037

A writer or a target file name must be specified

Cause: A writer or a target file name is not specified.

Action: Specify a writer or a target file name.

Error code: 1039

IOException on open

Cause: Java is throwing a reflection exception. OracleAS TopLink wraps only that exception.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 1040

Post Load Method Not Static

Cause: The method specified is not static.

Action: Modify the method to be static.

Error code: 1041

Project Not Found

Cause: No projects were found in the specified directory.

Action: Verify the directory.

Error code: 1042

Multiple Projects With Name

Cause: More than one project with the same name was found.

Action: Verify the project name.

Concurrency Exception

A Concurrency exception is a development exception that is raised when a Java concurrency violation occurs. Only when the running thread is interrupted, causing the Java virtual machine (JVM) to throw `InterruptedException`, is an internal exception information displayed with the error message.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
INTERNAL EXCEPTION: Message
```

Example C–3   Concurrency Exception

```
EXCEPTION [TOPLINK – 2004]: oracle.toplink.exceptions.ConcurrencyException
EXCEPTION DESCRIPTION: Signal attempted before wait on concurrency manager.
This normally means that an attempt was made to commit or rollback a transaction
```

```
before being started, or rolledback twice.
```

Error Codes 2001 – 2006

Error code: 2001

WAIT_WAS_INTERRUPTED

Cause: In a multi-threaded environment, one of the waiting threads was interrupted.

Action: Such exceptions are dependent on the application.

Error code: 2002

WAIT_FAILURE_SERVER

Cause: A request for a connection from the connection pool has been forced to wait, and that wait has been interrupted.

Action: Such exceptions are dependent on the application.

Error code: 2003

WAIT_FAILURE_CLIENT

Cause: A request for a connection from the connection pool has been forced to wait, and that wait has been interrupted.

Action: Such exceptions are dependent on the application.

Error code: 2004

SIGNAL_ATTEMPTED_BEFORE_WAIT

Cause: A signal was attempted before a wait on concurrency manager. This normally means that an attempt was made to commit or rollback a transaction before it was started, or to rollback a transaction twice.

Action: Verify transactions in the application.

Error code: 2005

WAIT_FAILURE_SEQ_DATABASE_SESSION

Cause: An `InterruptedException` was thrown while `DatabaseSession` sequencing waited for a separate connection to become available.

Action: Examine concurrency issues involving object creation with your `DatabaseSession`.

Error code: 2006

SEQUENCING_MULTITHREAD_THRU_CONNECTION

Cause: Several threads attempted to concurrently obtain sequence objects from the same `DatabaseSession` or `ClientSession`.

Action: Avoid concurrent writing through the same `DatabaseSession` or `ClientSession`.

Conversion Exception

A Conversion exception is a development exception that is raised when a conversion error occurs by an incompatible type conversion. The message that is returned indicates which type cast caused the exception.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
INTERNAL EXCEPTION: Message
```

Example C–4 Conversion Exception

```
EXCEPTION [TOPLINK – 3006]: oracle.toplink.exceptions.ConversionException
EXCEPTION DESCRIPTION: object must be of even length to be converted to a
ByteArray
```

Error Codes 3001 – 3007

Error code: 3001

COULD_NOT_BE_CONVERTED

Cause: The object `object` of class `objectClass` cannot be converted to `javaClass`. The object cannot be converted to a given type.

Action: Ensure that the object being converted is of the right type.

Error code: 3003

INCORRECT_DATE_FORMAT

Cause: The date in `dateString` is in an incorrect format. The expected format is **YYYY-MM-DD**.

Action: Verify the date format.

Error code: 3004

INCORRECT_TIME_FORMAT

Cause: The time in `timeString` is in an incorrect format. The expected format is **HH:MM:SS**.

Action: Verify the time format.

Error code: 3005

INCORRECT_TIMESTAMP_FORMAT

Cause: The timestamp `timestampString` is in an incorrect format. The expected format is **YYYY-MM-DD HH:MM:SS.NNNNNNNNN**.

Action: Verify the timestamp format.

Error code: 3006

COULD_NOT_CONVERT_TO_BYTE_ARRAY

Cause: The String object must be of even length to be converted to a ByteArray. This object cannot be converted to a ByteArray

Action: Verify the object being converted.

Error code: 3007

COULD_NOT_BE_CONVERTED_TO_CLASS

Cause: The object `object` of class `objectClass` cannot be converted to `javaClass`. The class `javaClass` is not on the class path.

Action: Ensure that the class `javaClass` is on the class path.

Database Exception

A Database exception is a runtime exception that is raised when data read from the database, or the data that is to be written to the database, is incorrect. The exception may also act as a wrapper for `SQLException`. If this is the case, the message contains a reference to the error code and error message. This exception can occur on any database type operation.

This exception includes internal exception and error code information when the exception is wrapping a `SQLException`.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
```

```
INTERNAL EXCEPTION: Message
ERROR CODE: Error code
```

Example C–5 Database Exception

```
EXCEPTION [TOPLINK – 4002]: oracle.toplink.exceptions.DatabaseExceptions
EXCEPTION DESCRIPTION: java.sql.SQLException: [INTERSOLV][ODBC dBase driver]
Incompatible datatypes in expression: >
INTERNAL EXCEPTION: java.sql.SQLException: [INTERSOLV][ODBC dBase driver]
Incompatible datatypes in expression: >
ERROR CODE: 3924
```

Error Codes 4002 – 4018

Error code: 4002

SQL_EXCEPTION

Cause: A SQL exception was encountered, thrown by the underlying JDBC bridge. OracleAS TopLink wraps only that exception.

Action: Inspect the internal exception that was thrown.

Error code: 4003

CONFIGURATION_ERROR_CLASS_NOT_FOUND

Cause: The driver class name was not found.

Action: Verify the class name given in JDBCLogin.

Error code: 4005

DATABASE_ACCESSOR_NOT_CONNECTED

Cause: The session is not connected to the database while attempting to read or write on the database.

Action: An application may have to login again because the connection to the database might have been lost.

Error code: 4006

ERROR_READING_BLOB_DATA

Cause: An error occurred reading BLOB data from the database. There are two possibilities for this exception: either the BLOB data was not read properly from the result set or OracleAS TopLink cannot process the BLOB data using `ByteArrayOutputStream`.

Action: Verify whether the underlying driver supports BLOBs properly. If it does, then report this problem to Technical Support.

Error code: 4007

COULD_NOT_CONVERT_OBJECT_TYPE

Cause: Cannot convert object type on internal `error.java.sql.TYPES = type`. The object from the result set cannot be converted to the that was type returned from the metadata information.

Action: Verify whether the underlying driver supports the conversion type properly. If it does, then report this problem to Technical Support.

Error code: 4008

LOGOUT_WHILE_TRANSACTION_IN_PROGRESS

Cause: An attempt has been made to logout while the transaction is still in progress. You cannot logout while a transaction is in progress.

Action: Wait until the transaction is finished.

Error code: 4009

SEQUENCE_TABLE_INFORMATION_NOT_COMPLETE

Cause: The sequence information given to OracleAS TopLink is not sufficiently complete to get the set of sequence numbers from the database. This usually happens on native sequencing on an Oracle database.

Action: Verify the data given, especially the sequence name given in OracleAS TopLink.

Error code: 4011

ERROR_PREALLOCATING_SEQUENCE_NUMBERS

Cause: An error occurred preallocating sequence numbers on the database; the sequence table information is not complete.

Action: Ensure the sequence table was properly created on the database.

Error code: 4014

CANNOT_REGISTER_SYNCHRONIZATIONLISTENER_ FOR_ UNITOFWORK

Cause: OracleAS TopLink cannot register the synchronization listener: *underlying_exception_string*. When the OracleAS TopLink session is configured with an `ExternalTransactionController`, any Unit of Work requested by a client must operate within the context of a JTS external global transaction. When a Unit of Work is created and the external global transaction is not in existence, or if the system cannot acquire a reference to it, this error is reported.

Action: Verify that a JTS transaction is in progress before acquiring the Unit of Work.

Error code: 4015

SYNCHRONIZED_UNITOFWORK_DOES_NOT_ SUPPORT_ COMMITANDRESUME

Cause: A synchronized `UnitOfWork` does not support the `commitAndResume` operation. When the OracleAS TopLink session is configured with an `ExternalTransactionController`, any Unit of Work requested by a client must operate within the context of a JTS external global transaction (see Error code: 4014). The JTS specification does not support the concept of check pointing a transaction—that is, committing the work performed and then continuing to work within the same transaction context. JTS does not support nested transactions, either. As a result, if a client code invokes `commitAndResume()` on a synchronized Unit of Work, this error is reported.

Action: None required.

Error code: 4016

CONFIGURATION_ERROR_NEW_INSTANCE_ INSTANTIATION_ EXCEPTION

Cause: A configuration error occurred when OracleAS TopLink attempted to instantiate `Driver: javaClass`. toplink cannot instantiate the driver.

Action: Check the driver.

Error code: 4017

CONFIGURATION_ERROR_NEW_INSTANCE_ILLEGAL_ ACCESS_ EXCEPTION

Cause: A configuration error occurred when OracleAS TopLink attempted to instantiate `Driver: javaClass`. toplink cannot instantiate the driver.

Action: Check the driver.

Error code: 4018

TRANSACTION_MANAGER_NOT_SET_FOR_JTS_DRIVER

Cause: The transaction manager has not been set for the `JTSSynchronizationListener`.

Action: Set a transaction manager for the `JTSSynchronizationListener`.

Optimistic Lock Exception

An Optimistic Lock exception is a runtime exception that is raised when the row on the database that matches the desired object is missing or when the value on the database does not match the registered number. It is used in conjunction with the optimistic locking feature. This applies only on an update or delete operation.

For more information about optimistic locking, see the *Oracle Application Server TopLink Mapping Workbench User's Guide*. These exceptions should be handled in a try-catch block.

Format

```
EXCEPTION [TOPLINK – error code]: Exception Name
EXCEPTION DESCRIPTION: Message
```

Example C–6 Optimistic Lock Exception

```
EXCEPTION [TOPLINK – 5003]: oracle.toplink.exceptions.OptimisticLockException
EXCEPTION DESCRIPTION: The object, object.toString() cannot be deleted because
it has changed or been deleted since it was last read.
```

Error Codes 5001 – 5008

Error code: 5001

NO_VERSION_NUMBER_WHEN_DELETING

Cause: An attempt was made to delete the object `object`, but it has no version number in the identity map. This object either was never read or has already been deleted.

Action: Use SQL logging to determine the reason for the exception. The last delete shows the object being deleted when the exception was thrown.

Error code: 5003

OBJECT_CHANGED_SINCE_LAST_READ_WHEN_DELETING

Cause: The object state has changed in the database. The object `object` cannot be deleted because it has changed or been deleted since it was last read. This usually means that the row in the table was changed by some other application.

Action: Refresh the object, which updates it with the new data from the database.

Error code: 5004

NO_VERSION_NUMBER_WHEN_UPDATING

Cause: An attempt has been made to update the object `object` but it has no version number in the identity map. It may not have been read before being updated or has been deleted.

Action: Use SQL logging to determine the reason for the exception. The last update shows the object being updated when the exception was thrown.

Error code: 5006

OBJECT_CHANGED_SINCE_LAST_READ_WHEN_UPDATING

Cause: The object state has changed in the database. The object `object` cannot be updated because it has changed or been deleted since it was last read. This usually means that the row in the table was changed by some other application.

Action: Refresh the object, which updates it with the new data from the database.

Error code: 5007

MUST_HAVE_MAPPING_WHEN_IN_OBJECT

Cause: The object `aClass` must have a nonread-only mapping corresponding to the version lock field. The mapping, which is needed when the lock value is stored in the domain object rather than in a cache, was not defined for the locking field.

Action: Define a mapping for the field.

Error code: 5008

NEED_TO_MAP_JAVA_SQL_TIMESTAMP

Cause: A write lock value that is stored in a domain object is not an instance of `java.sql.Timestamp`.

Action: Change the value of the attribute to be an instance of `java.sql.Timestamp`.

Query Exception

A Query exception is a development exception that is raised when insufficient information has been provided to the query. If possible, the message indicates the query that caused the exception. A query is optional and is displayed if OracleAS TopLink is able to determine the query that caused this exception.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
QUERY:
```

Example C–7 Query Exception

```
EXCEPTION [TOPLINK – 6026]: oracle.toplink.exceptions.QueryException
EXCEPTION DESCRIPTION: The query is not defined. When executing a query on the
session, the parameter that takes query is null.
```

Error Codes 6001 – 6098

Error code: 6001

ADDITIONAL_SIZE_QUERY_NOT_SPECIFIED

Cause: Cursored SQL queries must provide an additional query to retrieve the size of the result set. Failure to include the additional query causes this exception.

Action: Specify a size query.

Error code: 6002

AGGREGATE_OBJECT_CANNOT_BE_DELETED

Cause: Aggregated objects cannot be written or deleted independent of their owners. No identity is maintained on such objects.

Action: Do not try to delete aggregate objects directly.

Error code: 6003

ARGUMENT_SIZE_MISMATCH_IN_QUERY_AND_ QUERY_DEFINITION

Cause: The number of arguments provided to the query for execution does not match the number of arguments provided with the query definition.

Action: Check the query and the query execution.

Error code: 6004

BACKUP_CLONE_IS_ORIGINAL_FROM_PARENT

Cause: The object `clone` of class `clone.getClass()` with identity hashcode `(System.identityHashCode()) System.identityHashCode(clone)` is not from this Unit of Work space but from the parent session. The object was never registered in this Unit of Work but read from the parent session and related to an object registered in the Unit of Work.

Action: Verify that you are correctly registering your objects. If you are still having problems, use the `UnitOfWork.validateObjectSpace()` method to help debug where the error occurred.

Error code: 6005

BACKUP_CLONE_IS_ORIGINAL_FROM_SELF

Cause: The object clone of class `clone.getClass()` with identity hashcode `(System.identityHashCode()) <System.identityHashCode(clone)>` is the original to a registered new object. Because the Unit of Work clones new objects that are registered, ensure that an object is registered before it is reference by another object. If you do not want the new object to be cloned, used the `UnitOfWork.registerNewObject(Object)` API.

Action: Verify that you are correctly registering your objects. If you are still having problems, use the `UnitOfWork.validateObjectSpace()` method to help debug where the error occurred.

Error code: 6006

BATCH_READING_NOT_SUPPORTED

Cause: This mapping does not support batch reading. The optimization of batching the read of all the target rows is not supported for the mapping.

Action: The problem is an OracleAS TopLink development problem, and the user should never encounter this error code unless the mapping is a new custom mapping. Contact Technical Support.

Error code: 6007

DESCRIPTOR_IS_MISSING

Cause: The descriptor for `reference Class` is missing. The descriptor related to the class or the object is not found in the session.

Action: Verify whether the related descriptor was added to the session, and whether the query is performed on the right object or class.

Error code: 6008

DESCRIPTOR_IS_MISSING_FOR_NAMED_QUERY

Cause: The descriptor `domain Class Name` for the query named `queryName` is missing. The descriptor where named query is defined is not added to the session.

Action: Verify whether the related descriptor was added to the session, and whether query is performed on the right class.

Error code: 6013

INCORRECT_SIZE_QUERY_FOR_CURSOR_STREAM

Cause: The size query given on the queries returning cursor streams is not correct. The execution of the size query did not return any size.

Action: If the cursor stream query was a custom query, than check the size query that was specified, or report this problem to Technical Support.

Error code: 6014

INVALID_QUERY

Cause: Objects cannot be written in a Unit of Work using modify queries. They must be registered.

Action: Objects are registered in the Unit of Work, and during commit, the Unit of Work performs the required changes to the database.

Error code: 6015

INVALID_QUERY_KEY_IN_EXPRESSION

Cause: The query key `key` does not exist. Usually this happens because of a misspelled query key.

Action: Check the query key that was specified in the expression and verify that a query key was added to the descriptor.

Error code: 6016

INVALID_QUERY_ON_SERVER_SESSION

Cause: Objects and the database cannot be changed through the server session: all changes must be performed through a client session's Unit of Work. The objects cannot be changed on the server session by modifying queries. Objects are changed in the client sessions that are acquired from this server session.

Action: Use the client session’s Unit of Work to change the object.

Error code: 6020

NO_CONCRETE_CLASS_INDICATED

Cause: No concrete class is indicated for the type in this row. The type indicator read from the database row has no entry in the type indicator hashtable or if class extraction method was used, it did not return any concrete class type. The exception is thrown when subclasses are being read.

Action: Check class extraction method if specified or check the descriptor to verify all the type indicator values were specified.

Error code: 6021

NO_CURSOR_SUPPORT

Cause: No cursor support is provided for abstract class multiple table descriptors using expressions.

Action: Consider using custom SQL or multiple queries.

Error code: 6023

OBJECT_TO_INSERT_IS_EMPTY

Cause: There are no fields to be inserted into the table. The fields to insert into the table, `table` are empty.

Action: Define at least one mapping for this table.

Error code: 6024

OBJECT_TO_MODIFY_NOT_SPECIFIED

Cause: An object to modify is required for a modify query.

Action: Verify that the query contains an object before executing.

Error code: 6026

QUERY_NOT_DEFINED

Cause: The query is not defined. When executing a query on the session, the parameter that takes query is null.

Action: Verify whether the query is passed properly.

Error code: 6027

QUERY_SENT_TO_INACTIVE_UNIT_OF_WORK

Cause: The Unit of Work has been released and is now inactive.

Action: The Unit of Work, once released, cannot be reused unless `commitAndResume` is called.

Error code: 6028

READ_BEYOND_QUERY

Cause: An attempt has been made to read from the cursor streams beyond its limits (beyond the end of the stream).

Action: Ensure that the stream is checked for an end of stream condition before attempting to retrieve more objects.

Error code: 6029

REFERENCE_CLASS_MISSING

Cause: The reference class in the query is not specified. A reference class must be provided.

Action: Check the query.

Error code: 6030

REFRESH_NOT_POSSIBLE_WITHOUT_CACHE

Cause: Refresh is not possible if caching is not set. The read queries that skip the cache to read objects cannot be used to refresh the objects. Refreshing is not possible without identity.

Action: Check the query.

Error code: 6031

SIZE_ONLY_SUPPORTED_ON_EXPRESSION_QUERIES

Cause: OracleAS TopLink did not find a size query. Size is supported only on expression queries unless a size query is given.

Action: The cursor streams on a custom query should also define a size query.

Error code: 6032

SQL_STATEMENT_NOT_SET_PROPERLY

Cause: The SQL statement has not been properly set. The user should never encounter this error code unless queries have been customized.

Action: Contact Technical Support.

Error code: 6034

INVALID_QUERY_ITEM

Cause: OracleAS TopLink is unable to validate a query item expression.

Action: Validate the expression being used.

Error code: 6041

SELECTION_OBJECT_CANNOT_BE_NULL

Cause: The selection object that was passed to a read object or refresh was null.

Action: Check `setSelectionObject()` on the read query.

Error code: 6042

UNNAMED_QUERY_ON_SESSION_BROKER

Cause: Data read and data modify queries are being executed without the session name. Only object-level queries can be directly executed by the session broker, unless the query is named.

Action: Specify the session name.

Error code: 6043

REPORT_RESULT_WITHOUT_PKS

Cause: `ReportQuery` without primary keys cannot read the objects. The report query result that was returned is without primary key values. An object from the result can be created only if primary keys were also read.

Action: See the documentation about `retrievePrimaryKeys()` on report query.

Error code: 6044

NULL_PRIMARY_KEY_IN_BUILDING_OBJECT

Cause: The primary key that was read from the row `databaseRow` during the execution of the query was detected to be null; primary keys must not contain null.

Action: Check the query and the table on the database.

Error code: 6045

NO_DESCRIPTOR_FOR_SUBCLASS

Cause: The subclass has no descriptor defined for it.

Action: Ensure the descriptor was added to the session, or check class extraction method.

Error code: 6046

CANNOT_DELETE_READ_ONLY_OBJECT

Cause: The class you are attempting to delete is a read-only class.

Action: Contact Technical Support.

Error code: 6047

INVALID_OPERATOR

Cause: The operator `data` used in the expression is not valid.

Action: Check `ExpressionOperator` class to see a list of all the operators that are supported.

Error code: 6048

ILLEGAL_USE_OF_GETFIELD

Cause: This is an illegal use of `getField` `data` in the expression. This is an OracleAS TopLink development exception that users should not encounter.

Action: Report this problem to Technical Support.

Error code: 6049

ILLEGAL_USE_OF_GETTABLE

Cause: This is an illegal use of `getTable` `data` in the expression. This is an OracleAS TopLink development exception that users should not encounter.

Action: Report this problem to Technical Support.

Error code: 6050

REPORT_QUERY_RESULT_SIZE_MISMATCH

Cause: The number of attributes requested does not match the attributes returned from the database in report query. This can happen as a result of a custom query on the report query.

Action: Check the custom query to ensure it is specified, or report the problem to Technical Support.

Error code: 6051

CANNOT_CACHE_PARTIAL_OBJECT

Cause: Partial Objects are never put in the cache. Partial object queries are not allowed to maintain the cache or to be edited. Set `dontMaintainCache()`.

Action: Call the `dontMaintainCache()` method before executing the query.

Error code: 6052

OUTER_JOIN_ONLY_VALID_FOR_ONE_TO_ONE

Cause: An outer join (`getAllowingNull`) is valid only for one-to-one mappings and cannot be used for the mapping.

Action: Do not attempt to use `getAllowingNull` for mappings other than one-to-one.

Error code: 6054

CANNOT_ADD_TO_CONTAINER

Cause: OracleAS TopLink is unable to add `anObject` to a `containerClass` using `policy`. This is OracleAS TopLink development exception, and the user should never encounter this problem unless a custom container policy has been written.

Action: Contact Technical Support.

Error code: 6055

METHOD_INVOCATION_FAILED

Cause: The method invocation of `aMethod` on the object `anObject` threw a Java reflection exception while accessing the method.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 6056

CANNOT_CREATE_CLONE

Cause: Cannot create a clone of `anObject` using `policy.` This is an OracleAS TopLink development exception, and the user should never encounter this problem unless a custom container policy has been written.

Action: Report this problem to Technical Support.

Error code: 6057

METHOD_NOT_VALID

Cause: The method `methodName` is not valid to call on object `aReceiver`. This is an OracleAS TopLink development exception, and the user should never encounter this problem unless a custom container policy has been written.

Action: Contact Technical Support.

Error code: 6058

METHOD_DOES_NOT_EXIST_IN_CONTAINER_CLASS

Cause: The method named `methodName` was not found in class `aClass`. Thrown when looking for clone method on the container class. The clone is needed to create clones of the container in Unit of Work.

Action: Define clone method on the container class.

Error code: 6059

COULD_NOT_INSTANTIATE_CONTAINER_CLASS

Cause: The class `aClass` cannot be used as the container for the results of a query since it cannot be instantiated. The exception is a Java exception thrown when a new interface container policy is being created using Java reflection. OracleAS TopLink wraps only the Java exception.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 6060

MAP_KEY_NOT_COMPARABLE

Cause: Cannot use the object `anObject` of type `objectClass` as a key into `aContainer` which is of type `containerClass`. The key cannot be compared with the keys currently in the map. Throws a Java reflection exception while accessing the method. OracleAS TopLink wraps only the Java exception.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 6061

CANNOT_ACCESS_METHOD_ON_OBJECT

Cause: Cannot reflectively access the method `aMethod` for object: `anObject` of type `anObjectClass`. Throws a Java reflection exception while accessing method. OracleAS TopLink wraps only the Java exception.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 6062

CALLED_METHOD_THREW_EXCEPTION

Cause: The method `aMethod` was called reflectively on `objectClass` and threw an exception. Throws a Java reflection exception while accessing method. OracleAS TopLink wraps only the Java exception.

Action: Inspect the internal exception, and see the Java documentation.

Error code: 6063

INVALID_OPERATION

Cause: This is an invalid operation `operation` on the cursor. The operation is not supported.

Action: Check the class documentation and look for the corresponding method to use.

Error code: 6064

CANNOT_REMOVE_FROM_CONTAINER

Cause: Cannot remove `anObject` of type `anObjectClass` from `aContainerClass` using `policy`. This is an OracleAS TopLink development exception and, the user should never encounter this problem unless a custom container policy has been written.

Action: Contact Technical Support.

Error code: 6065

CANNOT_ADD_ELEMENT

Cause: Cannot add element to the collection container policy (cannot add `anObject` of type `anObjectClass` to a `aContainerClass`).

Action: Inspect the internal exception, and see the Java documentation.

Error code: 6066

BACKUP_CLONE_DELETED

Cause: Deleted objects cannot have reference after being deleted. The object clone of class `clone.getClass()` with identity hashcode `(System.identityHashCode()) System.identityHashCode(clone)` has been deleted, but it still has references.

Action: Ensure that you are correctly registering your objects. If you are still having problems, use the `UnitOfWork.validateObjectSpace()` method to help identify where the error occurred.

Error code: 6068

CANNOT_COMPARE_TABLES_IN_EXPRESSION

Cause: Cannot compare table reference to `data` in expression.

Action: Check the expression.

Error code: 6069

INVALID_TABLE_FOR_FIELD_IN_EXPRESSION

Cause: Field has invalid table in this context for field `data` in expression.

Action: Check the expression.

Error code: 6070

INVALID_USE_OF_TO_MANY_QUERY_KEY_IN_EXPRESSION

Cause: This is an invalid use of a query key representing a one-to-many relationship `data` in expression.

Action: Use the `anyOf` operator instead of `get`.

Error code: 6071

INVALID_USE_OF_ANY_OF_IN_EXPRESSION

Cause: This is an invalid use of `anyOf` for a query key not representing a to-many relationship `data` in expression.

Action: Use the `get` operator instead of `anyOf`.

Error code: 6072

CANNOT_QUERY_ACROSS_VARIABLE_ONE_TO_ ONE_MAPPING

Cause: Querying across a variable one-to-one mapping is not supported.

Action: Change the expression such that the query in not performed across a variable one-to-one mapping.

Error code: 6073

ILL_FORMED_EXPRESSION

Cause: This is an ill-formed expression in query, attempting to print an object reference into a SQL statement for `queryKey`.

Action: Contact Technical Support.

Error code: 6074

CANNOT_CONFORM_EXPRESSION

Cause: This expression cannot determine if the object conforms in memory. Set the query to check the database.

Action: Change the query such that it does not attempt to conform to the results of the query.

Error code: 6075

INVALID_OPERATOR_FOR_OBJECT_EXPRESSION

Cause: Object comparisons can only use the `equal` or `notEqual` operators, other comparisons must be performed through query keys or direct attribute level comparisons.

Action: Ensure the query uses only `equal` and `notEqual` if object comparisons are being used.

Error code: 6076

UNSUPPORTED_MAPPING_FOR_OBJECT_COMPARISON

Cause: Object comparisons can only be used with one-to-one mappings; other mapping comparisons must be performed through query keys or direct attribute level comparisons.

Action: Use a query key instead of attempting to compare objects across the mapping.

Error code: 6077

OBJECT_COMPARISON_CANNOT_BE_PARAMETERIZED

Cause: Object comparisons cannot be used in parameter queries.

Action: Change the query so that it does not attempt to use object when using parameterized queries.

Error code: 6078

INCORRECT_CLASS_FOR_OBJECT_COMPARISON

Cause: The class of the argument for the object comparison is incorrect.

Action: Ensure the class for the query is correct.

Error code: 6079

CANNOT_COMPARE_TARGET_FOREIGN_KEYS_ TO_NULL

Cause: Object comparison cannot be used for target foreign key relationships.

Action: Query on source primary key.

Error code: 6080

INVALID_DATABASE_CALL

Cause: This is an invalid database call. The call must be an instance of `DatabaseCall: call`.

Action: Ensure the call being used is a `DatabaseCall`.

Error code: 6081

INVALID_DATABASE_ACCESSOR

Cause: Invalid database accessor. The accessor must be an instance of `DatabaseAccessor: accessor`.

Action: Ensure the accessor being used is a `DatabaseAccessor`.

Error code: 6082

METHOD_DOES_NOT_EXIST_ON_EXPRESSION

Cause: The method `methodName` with argument types `argTypes` cannot be invoked on Expression.

Action: Ensure the method being used is a supported method.

Error code: 6083

IN_CANNOT_BE_PARAMETERIZED

Cause: Queries using `IN` cannot be parameterized.

Action: Disable the query prepare or binding.

Error code: 6084

REDIRECTION_CLASS_OR_METHOD_NOT_SET

Cause: The redirection query was not configured properly, the class or method name was not set.

Action: Verify the configuration for the redirection class.

Error code: 6085

REDIRECTION_METHOD_NOT_DEFINED_CORRECTLY

Cause: The redirection query's method is not defined or it defines with the wrong arguments. It must be public static and have the following arguments: `DatabaseQuery`, `DatabaseRow`, or `Session` (the interface).

Action: Check the redirection query's method as above.

Error code: 6086

REDIRECTION_METHOD_ERROR

Cause: The static invoke method provided to `MethodBaseQueryRedirector` threw an Exception when invoked.

Action: Check the static invoke method for problems.

Error code: 6087

EXAMPLE_AND_REFERENCE_OBJECT_CLASS_MISMATCH

Cause: There is a class mismatch between the example object and the reference class specified for this query.

Action: Ensure that the example and reference classes are compatible.

Error code: 6088

NO_ATTRIBUTES_FOR _REPORT_QUERY

Cause: A `ReportQuery` has been built with no attributes specified.

Action: Specify the attribute for the query.

Error code: 6089

NO_EXPRESSION_BUILDER_CLASS_FOUND

Cause: The expression has not been initialized correctly. Only a single `ExpressionBuilder` should be used for a query. For a parallel expressions, the query class must be provided to the `ExpressionBuilder` constructor, and the query’s `ExpressionBuilder` must always be on the left side of the expression.

Action: Contact Technical Support.

Error code: 6090

CANNOT_SET_REPORT_QUERY_TO_CHECK_CACHE_ONLY

Cause: The checkCacheOnly method was invoked on a `ReportQuery`. You cannot invoke the `checkCacheOnly` method on a `ReportQuery` because a `ReportQuery` returns data rather than objects and the OracleAS TopLink cache is built with objects.

Action: Do not use a `ReportQuery` in this case.

Error code: 6091

TYPE_MISMATCH_BETWEEN_ATTRIBUTE_AND_ CONSTANT_ ON_ EXPRESSION

Cause: The type of the constant used for comparison in the expression does not match the type of the attribute.

Action: Contact Technical Support.

Error code: 6092

MUST_INSTANTIATE_VALUEHOLDERS

Cause: Uninstantiated valueholders have been detected.

Action: Instantiate the valueholders for the collection you want to query on.

Error code: 6093

MUST_BE_ONE_TO_ONE_OR_ONE_TO_MANY_MAPPING

Cause: The `buildSelectionCriteria` method was invoked on a mapping that was neither one-to-one nor one-to-many. Only the one-to-one and one-to-many mapping exposes this public API to build selection criteria. Using the `buildSelectionCriteria` method with other mapping types will not return correct results.

Action: Only use the `buildSelectionCriteria` method with one-to-one and one-to-many mappings.

Error code: 6094

PARAMETER_NAME_MISMATCH

Cause: An unmapped field was used in a parameterized expression.

Action: Map the field or define an alternate expression that does not rely on the unmapped field.

Error code: 6095

CLONE_METHOD_REQUIRED

Cause: A delegate class of an `IndirectContainer` implementation does not implement `Cloneable`. If you implement `IndirectContainer` you must also implement `Cloneable`. For example, see `oracle.toplink.indirection.IndirectSet`. The `clone` method must clone the delegate. For example, the `IndirectSet` implementation uses reflection to invoke the `clone` method because it is not included in the common interface shared by `IndirectSet` and its base delegate class, `HashSet`.

Action: Ensure that your `IndirectContainer` implementation or its delegate class implements `Cloneable`.

Error code: 6096

CLONE_METHOD_INACCESSIBLE

Cause: A delegate class of an `IndirectContainer` implementation implements `Cloneable` but the `IndirectContainer` implementation does not have access to the specified clone method. That is, a `java.lang.IllegalAccessException` was thrown when the delegate's clone method was invoked.

Action: Ensure that both the delegate clone method and the delegate class are public. Ensure permission is set for Java reflection in your VM security settings. See also `java.lang.reflect.Method.invoke()`.

Error code: 6097

CLONE_METHOD_THORW_EXCEPTION

Cause: A delegate class of an `IndirectContainer` implementation implements `Cloneable` and the `IndirectContainer` implementation has access to the specified clone method, but the specified clone method throws a `java.lang.reflect.InvocationTargetException` when invoked.

Action: Verify the implementation of the delegate's clone method.

Error code: 6098

UNEXPECTED_INVOCATION

Cause: A proxy object method throws an unexpected exception when invoked (that is, some exception other than `InvocationTargetException` and `ValidationException`.)

Action: Review the proxy object to see where it is throwing the exception described in the exception Message. Ensure this exception is no longer thrown.

Validation Exception

A Validation exception is a development exception that is raised when an incorrect state is detected or an API is used incorrectly.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
```

Example C–8 Validation Exception

```
EXCEPTION [TOPLINK – 7008]: oracle.toplink.exceptions.ValidationException
EXCEPTION DESCRIPTION: The Java type javaClass is not a valid database type. The
Java type of the field to be written to the database has no corresponding type
on the database.
```

Error Codes 7001 – 7104

Error code: 7001

LOGIN_BEFORE_ALLOCATING_CLIENT_SESSIONS

Cause: You attempted to allocate client sessions before logging into the server.

Action: Ensure you have called `login()` on your server session or database session. This error also appears in multithreaded environments as a result of concurrency issues. Check that all your threads are synchronized.

Error code: 7002

POOL_NAME_DOES_NOT_EXIST

Cause: The pool name used while acquiring client session from the server session does not exist.

Action: Verify the pool name given while acquiring client session and all the existing pools on the server session.

Error code: 7003

MAX_SIZE_LESS_THAN_MIN_SIZE

Cause: The maximum number of connections in a connection pool should be more than the minimum number of connections.

Action: Check `addConnectionPool(String poolName, JDBCLogin login, int minNumberOfConnections, int maxNumberOfConnections)` on the server session.

Error code: 7004

POOLS_MUST_BE_CONFIGURED_BEFORE_LOGIN

Cause: Pools must all be added before login on the server session has been done. Once logged in, you cannot add pools.

Action: Check `addConnectionPool(String poolName, JDBCLogin login, int minNumberOfConnections, int maxNumberOfConnections)` on server session. This method should be called before logging in on the server session.

Error code: 7008

JAVA_TYPE_IS_NOT_A_VALID_DATABASE_TYPE

Cause: The Java type `javaClass` is not a valid database type. The Java type of the field to be written to the database has no corresponding type on the database.

Action: Check table or stored procedure definition.

Error code: 7009

MISSING_DESCRIPTOR

Cause: The descriptor `className` is not found in the session.

Action: Ensure that the related descriptor to the class was properly registered with the session.

Error code: 7010

START_INDEX_OUT_OF_RANGE

Cause: This is an OracleAS TopLink development exception and users should never encounter this problem. It happens when a copy of a vector is created with a start and end index.

Action: Report this problem to Technical Support.

Error code: 7011

STOP_INDEX_OUT_OF_RANGE

Cause: This is an OracleAS TopLink development exception and users should never encounter this problem. It happens when a copy of a vector is created with a start and end index.

Action: Report this problem to Technical Support.

Error code: 7012

FATAL_ERROR_OCCURRED

Cause: This is an OracleAS TopLink development exception and users should never encounter this problem. It happens when test cases are executed.

Action: Report this problem to Technical Support. This error commonly occurs if you attempt to `commit()` an invalid (or previously committed) `UnitOfWork`.

If `ValidationException.cannotCommitUOWAgain()` appears in the stack trace, verify that call `commit()` on valid `UnitOfWork` instances.

Error code: 7013

NO_PROPERTIES_FILE_FOUND

Cause: The `toplink.properties` file cannot be found on the system class path.

Action: Ensure that there is a `toplink.properties` file located on the system class path.

Error code: 7017

CHILD_DESCRIPTORS_DO_NOT_HAVE_IDENTITY_MAP

Cause: An identity map is added to the child descriptor. A child descriptor shares its parent's identity map.

Action: Check child descriptor and remove identity map from it.

Error code: 7018

FILE_ERROR

Cause: The user should never encounter this problem. It happens when test cases are executed.

Action: Contact Technical Support.

Error code: 7023

INCORRECT_LOGIN_INSTANCE_PROVIDED

Cause: The login instance provided to the `login()` method is incorrect. A JDBCLogin must be provided.

Action: Use a JDBCLogin.

Error code: 7024

INVALID_MERGE_POLICY

Cause: This is an OracleAS TopLink development exception and users should never encounter it.

Action: Contact Technical Support.

Error code: 7025

ONLY_FIELDS_ARE_VALID_KEYS_FOR_ DATABASE_ROWS

Cause: The key on the database row is not either of type String or of type `DatabaseField`.

Action: Contact Technical Support.

Error code: 7027

SEQUENCE_SETUP_INCORRECTLY

Cause: The sequence `sequenceName` is setup incorrectly, increment does not match pre-allocation size.

Action: Contact Technical Support.

Error code: 7030

CANNOT_SET_READ_POOL_SIZE_AFTER_LOGIN

Cause: OracleAS TopLink is unable to set read pool size after the server session has already been logged in.

Action: The size should be set before login.

Error code: 7031

CANNOT_ADD_DESCRIPTORS_TO_SESSION_BROKER

Cause: OracleAS TopLink cannot add descriptors to a session broker.

Action: Descriptors are added to the sessions contained in the session broker.

Error code: 7032

NO_SESSION_REGISTERED_FOR_CLASS

Cause: The descriptor related to the domain class `domainClass` was not found in any of the sessions registered in the session broker.

Action: Check the sessions.

Error code: 7033

NO_SESSION_REGISTERED_FOR_NAME

Cause: The session with the given name `sessionName` is not registered in the session broker.

Action: Check the session broker.

Error code: 7038

LOG_IO_ERROR

Cause: Error while logging message to session's log.

Action: Check the internal exception.

Error code: 7039

CANNOT_REMOVE_FROM_READ_ONLY_CLASSES_ IN_NESTED_UNIT_ OF_WORK

Cause: OracleAS TopLink is unable to remove from the set of read-only classes in a nested Unit of Work. A nested Unit of Work's set of read-only classes must be equal to or a superset of its parent's set of read-only classes.

Action: Contact Technical Support.

Error code: 7040

CANNOT_MODIFY_READ_ONLY_CLASSES_SET_ AFTER_USING_UNIT_ OF_WORK

Cause: OracleAS TopLink is unable to change the set of read-only classes in a Unit of Work after that Unit of Work has been used. Changes to the read-only set must be made when acquiring the Unit of Work or immediately after.

Action: Contact Technical Support.

Error code: 7042

PLATFORM_CLASS_NOT_FOUND

Cause: The platform class `className` was not found and a reflection exception was thrown.

Action: Check the internal exception.

Error code: 7043

NO_TABLES_TO_CREATE

Cause: A `project` does not have any tables to create on the database.

Action: Validate the project and tables you are attempting to create.

Error code: 7044

LLEGAL_CONTAINER_CLASS

Cause: The container class specified `className` cannot be used as the container because it does not implement the `Collection` or `Map` interfaces.

Action: Implement either the `Collection` or `Map` interfaces in the container class.

Error code: 7047

ONTAINER_POLICY_DOES_NOT_USE_KEYS

Cause: Invalid Map class was specified for the container policy. The container specified (of class `aPolicyContainerClass`) does not require keys. You tried to use `methodName`.

Action: Use map class that implements the `Map` interface.

Error code: 7048

METHOD_NOT_DECLARED_IN_ITEM_CLASS

Cause: The key method on the map container policy is not defined. The instance method <*methodName*> does not exist in the reference class <*className*> and therefore cannot be used to create a key in a Map. A map container policy represents how to handle an indexed collection of objects. Usually the key is the primary key of the objects stored, so the policy needs to know the name of the primary key get method, to extract it from each object using reflection. For instance a user might call policy.setKeyMethodName("getId");

Action: Check the second parameter of your DatabaseQuery.useMapClass() call.

Error code: 7051

MISSING_MAPPING

Cause: Missing attribute `attributeName` for descriptor `descriptor` called from `source`. This is an OracleAS TopLink development exception and a user should never encounter it.

Action: Contact Technical Support.

Error code: 7052

ILLEGAL_USE_OF_MAP_IN_DIRECTCOLLECTION

Cause: The method `useMapClass` was called on a `DirectCollectionMapping`. It is illegal to call `useMapClass()` on a `DirectCollectionMapping`. OracleAS TopLink cannot instantiate Java attributes mapped using a `DirectCollectionMapping` with a Map. The `useMapClass()` API is supported for `OneToManyMappings` and `ManyToManyMappings`. The Java 2 `Collection` interface is supported using the `useCollectionClass()` method.

Action: Use the `useCollectionClass()` API. Do not call `useMapClass()` on `DirectCollectionMappings`.

Error code: 7053

CANNOT_RELEASE_NON_CLIENTSESSION

Cause: OracleAS TopLink is unable to release a session that is not a client session. Only client sessions can be released.

Action: Modify the code to ensure the client session is not released.

Error code: 7054

CANNOT_ACQUIRE_CLIENTSESSION_FROM_SESSION

Cause: OracleAS TopLink is unable to acquire a session that is not a client session. Client sessions can only be acquired from server sessions.

Action: Modify the code to ensure acquire is attempted only from server sessions.

Error code: 7055

OPTIMISTIC_LOCKING_NOT_SUPPORTED

Cause: Optimistic Locking is not supported with stored procedure generation.

Action: Do not use `OptimisticLocking` with stored procedure generation.

Error code: 7056

WRONG_OBJECT_REGISTERED

Cause: The wrong object was registered into the Unit of Work. It should be the object from the parent cache.

Action: Ensure that the object is from the parent cache.

Error code: 7058

INVALID_CONNECTOR

Cause: The connector selected is invalid and must be of type `DefaultConnector`.

Action: Ensure that the connector is of type `DefaultConnector`.

Error code: 7059

INVALID_DATA_SOURCE_NAME

Cause: Invalid datasource name: `name`.

Action: Verify that the datasource name.

Error code: 7060

CANNOT_ACQUIRE_DATA_SOURCE

Cause: OracleAS TopLink is unable to acquire datasource: `name` or an error has occurred in setting up the datasource.

Action: Verify the datasource name. Check the nested SQL exception to determine the cause of the error. Typical problems include:

- The connection pool was not configured in your `config.xml`.
- The driver is not on the class path.
- The user or password is incorrect.
- The database server URL or driver name is not properly specified.

Error code: 7061

JTS_EXCEPTION_RAISED

Cause: An exception occurred within the Java Transaction Service (JTS).

Action: Examine the JTS exception and see the JTS documentation.

Error code: 7062

FIELD_LEVEL_LOCKING_NOTSUPPORTED_ OUTSIDE_A_UNIT_OF_WORK

Cause: `FieldLevelLocking` is not supported outside a Unit of Work. In order to use field level locking, a Unit of Work must be used for ALL writes.

Action: Use a Unit of Work for writing.

Error code: 7063

EJB_CONTAINER_EXCEPTION_RAISED

Cause: An exception occurred within the EJB container.

Action: Examine the EJB exception and see the JTS documentation.

Error code: 7064

EJB_PRIMARY_KEY_REFLECTION_EXCEPTION

Cause: An exception occurred in the reflective EJB bean primary key extraction.

Action: Ensure that your primary key object is defined correctly.

Error code: 7065

EJB_CANNOT_LOAD_REMOTE_CLASS

Cause: The remote class for the bean cannot be loaded or found, for the bean.

Action: Ensure that the correct class loader is set correctly.

Error code: 7066

EJB_MUST_BE_IN_TRANSACTION

Cause: OracleAS TopLink is unable to create or remove beans unless a JTS transaction is present, `bean=bean`.

Action: Ensure that the JTS transaction is present.

Error code: 7068

EJB_INVALID_PROJECT_CLASS

Cause: The platform class `platformName` was not found for the `projectName` using default class loader.

Action: Validate the project and platform.

Error code: 7069

PROJECT_AMENDMENT_EXCEPTION_OCCURED

Cause: An exception occurred while looking up or invoking the project amendment method, `amendmentMethod` on the class `amendmentClass`.

Action: Validate the amendment method and class.

Error code: 7070

EJB_TOPLINK_PROPERTIES_NOT_FOUND

Cause: A `toplink.properties` resource bundle must be located on the class path in an OracleAS TopLink directory.

Action: Validate the class path and the location of the OracleAS TopLink resource bundle.

Error code: 7071

CANT_HAVE_UNBOUND_IN_OUTPUT_ARGUMENTS

Cause: You cannot use input output parameters without using binding.

Action: Use binding on the `StoredProcedureCall`.

Error code: 7072

EJB_INVALID_PLATFORM_CLASS

Cause: `SessionManager` failed to load the class identified by the value associated with properties `platform-class` or `external-transaction-controller-class` during initialization when it loads the OracleAS TopLink session common properties from the OracleAS TopLink global properties file (`sessions.xml` for non-EJB applications or `toplink-ejb-jar.xml` for EJB applications).

Action: Ensure that your OracleAS TopLink global properties file is correctly configured. Pay particular attention to the `platform-class` and `external-transaction-controller-class` properties.

Error code: 7073

ORACLE_OBJECT_TYPE_NOT_DEFINED

Cause: The Oracle object type with type name `typeName` is not defined.

Action: Ensure that the Oracle object type is defined.

Error code: 7074

ORACLE_OBJECT_TYPE_NAME_NOT_DEFINED

Cause: The Oracle object type `typeName` is not defined.

Action: Ensure that the Oracle object type is defined.

Error code: 7075

ORACLE_VARRAY_MAXIMIM_SIZE_NOT_DEFINED

Cause: The Oracle VARRAY type `typeName` maximum size is not defined.

Action: Verify the maximum size for the Oracle VARRAY.

Error code: 7076

DESCRIPTOR_MUST_NOT_BE_INITIALIZED

Cause: When generating the project class the descriptors must not be initialized.

Action: Ensure that the descriptors are not initialized before generating the project class.

Error code: 7077

EJB_INVALID_FINDER_ON_HOME

Cause: The `Home` interface `homeClassName.toString()` specified during creation of `BMPWrapperPolicy` does not contain a correct `findByPrimaryKey` method. A `findByPrimaryKey` method must exist that takes the PrimaryKey class for this bean.

Action: Ensure that a `FindByPrimaryKey` method exists and is correct.

Error code: 7078

EJB_NO_SUCH_SESSION_SPECIFIED_IN_PROPERTIES

Cause: The `sessionName` specified on the deployment descriptor does not match any session specified in the `toplink.properties` file.

Action: Contact Technical Support.

Error code: 7079

EJB_DESCRIPTOR_NOT_FOUND_IN_SESSION

Cause: The descriptor was not found in the session.

Action: Check the project being used for this session.

Error code: 7080

EJB_FINDER_EXCEPTION

Cause: A `FinderException` was thrown when attempting to load an object from the class with the primary key.

Action: Contact Technical Support.

Error code: 7081

CANNOT_REGISTER_AGGREGATE_OBJECT_IN_ UNIT_OF_ WORK

Cause: The aggregate object cannot be directly registered in the Unit of Work. It must be associated with the source (owner) object.

Action: Contact Technical Support.

Error code: 7082

MULTIPLE_PROJECTS_SPECIFIED_IN_PROPERTIES

Cause: The `toplink.properties` file specified multiple project files for the server. Only one project file can be specified.

Action: Specify either `projectClass`, `projectFile`, or `xmlProjectFile`.

Error code: 7083

NO_PROJECT_SPECIFIED_IN_PROPERTIES

Cause: The `toplink.properties` file does not include any information on the OracleAS TopLink project to use for the server. One project file must be specified.

Action: Specify either `projectClass`, `projectFile`, or `xmlProjectFile`.

Error code: 7084

INVALID_FILE_TYPE

Cause: The specified file is not a valid type for reading. `ProjectReader` must be given the deployed XML project file.

Action: Contact Technical Support.

Error code: 7085

SUB_SESSION_NOT_DEFINED_FOR_BROKER

Cause: Unable to create an instance of the external transaction controller specified in the properties file.

Action: Contact Technical Support.

Error code 7086:
EJB_INVALID_SESSION_TYPE_CLASS

Cause: The session manager cannot load the class corresponding to the session’s type class name.

Action: Ensure that the class name of the session's type is fully qualified in the `sessions.xml` file or `toplink.properties` file.

Error code 7087:
EJB_SESSION_TYPE_CLASS_NOT_FOUND

Cause: The session manager cannot load the class corresponding to the session’s type class name.

Action: Ensure that the class name of the session's type is fully qualified in the `sessions.xml` file or `toplink.properties` file.

Error code 7088:
CANNOT_CREATE_EXTERNAL_TRANSACTION_ CONTROLLER

Cause: The session manager cannot load the class corresponding to the external transaction controller's class name.

Action: Ensure that the class name of the external transaction controller is valid and fully qualified in the `sessions.xml` file or `toplink.properties` file.

Error code 7089:
SESSION_AMENDMENT_EXCEPTION_OCCURED

Cause: The session manager cannot load the class corresponding to the amendment class name or it cannot load the method on the amendment class corresponding to the amendment method name.

Action: Ensure that the class name of the amendment class is fully qualified and the amendment method exists in the amendment class in the `sessions.xml` file or `toplink.properties` file.

Error code 7091
SET_LISTENER_CLASSES_EXCEPTION

Cause: OracleAS TopLink is unable to create the listener class that implements `SessionEventListener` for the internal use of `SessionXMLProject`.

Action: Contact Technical Support.

Error code 7092

EXISTING_QUERY_TYPE_CONFLICT

Cause: OracleAS TopLink has detected a conflict between a custom query with the same name and arguments to a session.

Action: Ensure that no query is added to the session more than once or change the query name so that the query can be distinguished from others.

Error code 7093

QUERY_ARGUMENT_TYPE_NOT_FOUND

Cause: OracleAS TopLink is unable to create an instance of the query argument type.

Action: Ensure that the argument type is a fully qualified class name and the argument class is included in the class path environment.

Error code 7094

ERROR_IN_SESSIONS_XML

Cause: The `sessions.xml` or `toplink.properties` files cannot be loaded.

Action: Ensure that the path to either of the files exist on the class path environment.

Error code 7095

NO_SESSIONS_XML_FOUND

Cause: The `sessions.xml` or `toplink.properties` files cannot be loaded.

Action: Ensure that the path to either of the files exist on the class path environment.

Error code 7096

CANNOT_COMMIT_UOW_AGAIN

Cause: OracleAS TopLink cannot invoke `commit()` on an inactive Unit of Work that was committed or released.

Action: Ensure you invoke `commit()` on a new Unit of Work or invoke `commitAndResume()` so that the Unit of Work can be reused. For more information about the `commitAndResume()` method, see the *Oracle Application Server TopLink API Reference.*

Error code 7097:
OPERATION_NOT_SUPPORTED

Cause: OracleAS TopLink cannot invoke a nonsupport operation on an object.

Action: Do not use the operation indicated in the stack trace.

Error Code: 7099
PROJECT_XML_NOT_FOUND

Cause: The filename specified for the XML-based project is incorrect.

Action: Verify the name and location of the file.

Error Code: 7101
NO_TOPLINK_EJB_JAR_XML_FOUND

Cause: The `toplink-ejb-jar.xml` file was not found.

Action: Ensure that the file is on your class path.

Error Code: 7102
NULL_CACHE_KEY_FOUND_ON_REMOVAL

Cause: Encountered a null value for a cache key while attempting to remove an object from the identity map. The most likely cause of this situation is that the object has already been garbage-collected and therefore does not exist within the identity map.

Action: Ignore. The `Session.removeFromIdentityMap` method is intended to allow garbage collection, which has already been done.

Error Code: 7103
NULL_UNDERLYING_VALUEHOLDER_VALUE

Cause: A null reference was encountered while attempting to invoke a method on an object that uses proxy indirection.

Action: Please check that this object is not null before invoking its methods.

Error Code: 7104
INVALID_SEQUENCING_LOGIN

Cause: A separate connection(s) for sequencing was requested but sequencing login uses external transaction controller.

Action: Either provide a sequencing login that does not use an external transaction controller or do not use separate connection(s) for sequencing.

EJB QL Exception

An EJB QL exception is a runtime exception raised when the EJB QL string does not parse properly, or the contents are not resolvable within the context of the OracleAS TopLink session. The associated message typically includes a reference to the EJB QL string that caused the problem.

Error Codes 8001 – 8010

Error Code: 8001

`recognitionException`

Cause: The OracleAS TopLink EJB QL parser does not recognize a clause in the EJB QL string.

Action: Validate the EJB QL string.

Error Code: 8002

`generalParsingException`

Cause: OracleAS TopLink has encountered a problem while parsing the EJB QL string.

Action: Check the internal exception for details on the root cause of this exception.

Error Code: 8003

`classNotFoundException`

Cause: The class specified in the EJB QL string was not found.

Action: Ensure that the class is on the appropriate class path.

Error Code: 8004

`aliasResolutionException`

Cause: OracleAS TopLink was unable to resolve the alias used in the EJB QL string.

Action: Validate the identifiers used in the EJB QL string.

Error Code: 8005

`resolutionClassNotFoundException`

Cause: OracleAS TopLink was unable to resolve the class for an alias. This means that the class specified cannot be found.

Action: Ensure that the class is specified properly and is on the class path.

Error Code: 8006

`missingDescriptorException`

Cause: The class specified in the query has no OracleAS TopLink descriptor.

Action: Ensure that the class has been mapped and is specified correctly in the EJB QL string.

Error Code: 8009

`expressionNotSupported`

Cause: An unsupported expression was used in the EJB QL.

Action: Change the query to use only supported expressions.

Error Code: 8010

`generalParsingException`

Cause: OracleAS TopLink has encountered a problem while parsing the EJB QL string.

Action: Check the internal exception for details on the root cause of this exception.

Session Loader Exception

A Session Loader Exception is a runtime exception thrown if the Session Manager encounters a problem loading session information from a `sessions.xml` (for non-EJB applications) or `toplink-ejb-jar.xml` (for EJB applications) properties file.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
```

Example C–9 Session Loader Exception

```
EXCEPTION [TOPLINK – 9004]: oracle.toplink.exceptions.SessionLoaderException
EXCEPTION DESCRIPTION: The <project-xml> file MyProject was not found on the
classpath, nor on the filesystem.
```

Error Codes 9000 - 9009

Error Code: 9000

FINAL_EXCEPTION

Cause: The session loader caught one or more XML parsing exceptions while loading session information. The specific XML exceptions follow.

Action: Verify your session configuration XML file.

Error Code: 9001

UNKNOWN_TAG

Cause: An unknown tag was encountered in the specified XML node.

Action: Examine the specified XML node in your session configuration XML file. Ensure that you use only the tags defined for that node in the appropriate OracleAS TopLink DTD. See `<ORACLE_HOME>/toplink/config/dtds`.

Error Code: 9002

UNABLE_TO_LOAD_PROJECT_CLASS

Cause: The specified class loader could not load a class with the name given by the `project-name` property.

Action: Verify the value of the `project-name` property and if correct, ensure that a class with that name is in your classpath.

Error Code: 9003

UNABLE_TO_PROCESS_TAG

Cause: The session loader caught an exception while either parsing the value of the specified tag or calling the set-method associated with the specified tag.

Action: Verify the value shown for the specified tag.

Error Code: 9004

COULD_NOT_FIND_PROJECT_XML

Cause: The session loader could not find the file identified by the `project-xml` tag on either the classpath or the filesystem.

Action: Verify the value of the `project-xml` tag and if correct, ensure that a project XML file with that name exists in your classpath or filesystem.

Error Code: 9005

FAILED_TO_LOAD_PROJECT_XML

Cause: The session loader caught an exception while trying to load the file identified by the `project-xml` tag either because the file could not be found or because the file could not be parsed.

Action: Verify the configuration of the project XML file and ensure that a project XML file with that name specified by the `project-xml` tag exists in your classpath or filesystem.

Error Code: 9006

UNABLE_TO_PARSE_XML

Cause: The session loader caught a SAX exception while trying to parse the XML at the given line and column of the specified XML file.

Action: Verify that the XML is correctly formatted at the given line and column. Alternatively, ensure the Oracle parser is in your class path and that is appears before any other XML parser.

Error Code: 9007

NON_PARSE_EXCEPTION

Cause: The session loader caught an exception unrelated to XML parsing (for example, a premature end-of-file exception) while trying to parse the specified XML file.

Action: Verify the integrity of the XML file.

Error Code: 9008

UN_EXPECTED_VALUE_OF_TAG

Cause: The value of an XML tag does not correspond to any known OracleAS TopLink required values.

Action: Please verify the list of values for this tag.

Error Code: 9009

UNKNOWN_ATTRIBUTE_OF_TAG

Cause: Incorrect name value pair when processing transport properties for the RCM tag.

Action: Please verify that the all properties have both the name and the value filled in, in the session configuration XML.

EJB Exception Factory

An EJB Exception Factory Exception is a runtime exception thrown if a `BeanManager` specific to a given application server encounters a problem during any stage of an EJB's life cycle.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
```

Example C–10 EJB Exception Factory Exception

```
EXCEPTION [TOPLINK – 10008]: javax.ejb.CreateException
EXCEPTION DESCRIPTION: Cannot find bean.
```

Error Codes 10001 - 10048

Error Code: 10001

CREATE_EXCEPTION

Cause: The PersistenceManager for the given application server failed to create an EJB (for example, a problem was encountered during the create, such as a NullPointerException).

Action: Check the exception contained in the CreateException for additional information.

Error Code: 10002

REMOVE_EXCEPTION

Cause: The PersistenceManager for the given application server failed to remove an EJB (for example, a problem was encountered during the remove, such as a NullPointerException).

Action: Check the exception contained in the RemoveException for additional information.

Error Code: 10003

EJB_EXCEPTION

Cause: An internal, unexpected Exception was thrown.

Action: See the Exception message provided.

Error Code: 10004

FINDER_EXCEPTION1

Cause: Unexpected exception encountered while executing finder.

Action: See the Exception message provided.

Error Code: 10005

FINDER_EXCEPTION2

Cause: Unexpected exception encountered while executing finder.

Action: See the Exception message provided.

Error Code: 10007

DUPLICATE_KEY_EXCEPTION

Cause: The PersistenceManager for a given application server failed to create an EJB, because an EJB with the given primary key already exists.

Action: Verify the application logic to ensure the primary key is unique.

Error Code: 10008

OBJECT_NOT_FOUND_EXCEPTION

Cause: A scalar finder (one that returns a single object) was invoked on a home interface, and returned null.

Action: Verify the application logic to ensure the desired EJB exists.

Error Code: 10009

OBJECT_NOT_FOUND_PKEY_EXCEPTION

Cause: A find using the primary key indicated, returned null.

Action: Verify the application logic to ensure the desired EJB exists.

Error Code: 10010

CANNOT_CREATE_READ_ONLY

Cause: An attempt was made to create an entity marked as read-only using session().getProject().setDefaultReadOnlyClasses(aVector). You cannot create a read-only entity.

Action: Read-only entities should be read from the database (not created by the home interface). Adjust the application to read the required entities beforehand.

Error Code: 10011

CANNOT_REMOVE_READ_ONLY

Cause: An attempt was made to delete an entity marked as read-only using session().getProject().setDefaultReadOnlyClasses(aVector). You cannot delete a read-only entity.

Action: Determine whether the object should be read-only or not. If it should, do not try to remove it.

Error Code: 10014

ERROR_IN_NON_TX_COMMIT

Cause: The `PersistenceManager` for a given application server failed to end a local transaction (made up of a non-synchronized, non-JTA `UnitOfWork`) after a remove, create, business method, or home method invocation.

Action: See the Exception message provided.

Error Code: 10021

ERROR_ASSIGNING_SEQUENCES

Cause: The `PersistenceManager` for a given application server, whose `shouldAssignSequenceNumbers` method returns true, failed to assign a sequence number to an entity.

Action: See the Exception message provided.

Error Code: 10022

LIFECYCLE_REMOTE_EXCEPTION

Cause: A `java.rmi.RemoteException` was thrown when an entity was activated, loaded, passivated, or stored.

Action: See the Exception message provided.

Error Code: 10023

SEQUENCE_EXCEPTION

Cause: An exception was thrown while handling a post-insert `DescriptorEvent` preventing the specified entity from being assigned a primary key.

Action: See the Exception message provided.

Error Code: 10024

NO_SUCH_ENTITY_EXCEPTION

Cause: A conforming find, using the same query as a find by primary key, failed with a `javax.ejb.ObjectNotFoundException`.

Action: See the Exception message provided.

Error Code: 10025

INTERNAL_ERROR_ACCESSING_CTX

Cause: Internal error.

Action: Please contact support if required.

Error Code: 10026

INTERNAL_ERROR_FINDING_GENSUBCLASS

Cause: Internal error.

Action: Please contact support.

Error Code: 10027

INTERNAL_ERROR_INITIALIZING_CTX

Cause: Internal error.

Action: Please contact support.

Error Code: 10028

INTERNAL_ERROR_INVALID_MAPPING

Cause: The `SessionAccessor.registerOrMergeAttribute` method, called from within an EJB setter method, failed to obtain a `DatabaseMapping` for the given attribute from the `PersistenceManager`.

Action: Verify that the given attribute belongs to the EJB class and if it does, verify that a mapping exists for it.

Error Code: 10029

INTERNAL_ERROR_ACCESSING_PK

Cause: Failed to wrap an EJB for return to the application because the attempt to extract the primary key from the bean failed.

Action: See the Exception message provided.

Error Code: 10030

INTERNAL_ERROR_ACCESSING_PKFIELD

Cause: Failed to initialize primary key fields due to `java.lang.NoSuchFieldException`.

Action: See the Exception message provided.

Error Code: 10031

INTERNAL_ERROR_PREPARING_BEAN_INVOKE

Cause: One of the following failed with an exception other than `javax.ejb.ObjectNotFoundException`: a conforming find using the same query as a find by primary key; an Oracle Application Server Containers for J2EE `startCall` method invocation for a BUISNESS_METHOD operation; or a WebLogic `preInvoke` method invocation.

Action: See the Exception message provided

Error Code: 10032

FINDER_NOT_IMPLEMENTED

Cause: Associated finder has no implementation.

Action: Provide an implementation for the finder.

Error Code: 10033

FINDER_FINDBYPK_NULLPK

Cause: A find by primary key was called with a null primary key value.

Action: Ensure the primary key is not null when the finder is invoked

Error Code: 10034

REMOVE_NULLPK_EXCEPTION

Cause: A find by primary key was called with a null primary key value.

Action: Ensure the primary key is not null when the finder is invoked.

Error Code: 10036

ERROR_DURING_CODE_GEN

Cause: The `PersistenceManager` for a given application server failed to code-generate a bean subclass.

Action: See the Exception message provided.

Error Code: 10037

ERROR_EXECUTING_EJB_SELECT

Cause: An EJB select failed with an exception other than `javax.ejb.ObjectNotFoundException`.

Action: See the Exception message provided.

Error Code: 10038

ERROR_EXECUTING_EJB_HOME

Cause: The invocation of a Home interface method (excluding finders or create methods) failed.

Action: See the Exception message provided.

Error Code: 10040

NO_ACTIVE_TRANSACTION

Cause: A create or remove EJB failed because the PersistenceManager does not have a transaction.

Action: Ensure your application has a transaction available. This may be a configuration problem related to your ejb-jar.xml or an application logic problem in your client code.

Error Code: 10043

FINDER_RESULTS_ALREADY_WRAPPED

Cause: The results of a finder query could not be wrapped because they were already wrapped.

Action: If a redirect query is used, be sure to call the `setShouldUseWrapperPolicy(false)` method first.

Error Code: 10045

LOCAL_WRAPPER_MISSING

Cause: Error resolving the local interface.

Action: Please double check your local interface configuration.

Error Code: 10046

REMOTE_WRAPPER_MISSING

Cause: Error resolving the remote interface.

Action: Please double check your remote interface configuration.

Error Code: 10047

CREATE_NULLPK_EXCEPTION

Cause: The PersistenceManager for a given application server failed to create a bean because the primary key was not defined.

Action: Make sure the primary key is defined properly, either in the application logic or through the sequence number configuration.

Communication Exception

A Communication Exception is a runtime exception that wraps all RMI, CORBA, or input and output (I/O) exceptions that occur.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
```

Example C–11 Communication Exception

```
EXCEPTION [TOPLINK – 12000]: oracle.toplink.exceptions.CommunicationException
EXCEPTION DESCRIPTION: Error Sending connection service to myService.
```

Error Codes 12000 - 12004

Error Code: 12000

ERROR_SENDING_CONNECTION_SERVICE

Cause: Failed to add a connection to `CacheSynchronizationManager` or `RemoteCommandManager`.

Action: See generated exception for root cause.

Error Code: 12001

UNABLE_TO_CONNECT

Cause: `CacheSynronizationManager` failed to connect to the specified service.

Action: See generated exception for root cause.

Error Code: 12003

UNABLE_TO_PROPAGATE_CHANGES

Cause: `CacheSynronizationManager` failed to propagate changes to the specified service.

Action: See generated exception for root cause.

Error Code: 12004

ERROR_IN_INVOCATION

Cause: Error invoking a remote call.

Action: See generated exception for root cause.

XML Data Store Exception

An XML Data Store Exception is a runtime exception thrown when using OracleAS TopLink to persist objects in the form of XML files (rather than using a relational database.)

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
```

Example C–12 XML Data Store Exception

```
EXCEPTION [TOPLINK – 13000]: oracle.toplink.xml.XMLDataStoreException
EXCEPTION DESCRIPTION: File not found: C:\data\myTable\row.xml.
```

Error Codes 13000 - 13020

Error Code: 13000

FILE_NOT_FOUND

Cause: Failed to create a WriteStream for an XML file (an individual file or a file extracted from a ZIP archive) because the file could not be found in the file system. This can happen if the XML DataAccessor is trying to update an XML file and the file does not exist. This indicates an inconsistent state between the application and what is on disk.

Action: Verify that the specified file exists.

Error Code: 13001

UNABLE_TO_CLOSE_WRITE_STREAM

Cause: After writing a row to the XML data store, failed to close the `WriteStream` used due to a `java.io.IOException`. This can happen if the disk is full.

Action: See the generated exception for the root cause. Verify that there is sufficient disk space available for this operation.

Error Code: 13002

NOT_A_DIRECTORY

Cause: Creating or deleting a file source failed because the `File` being created or deleted was not a directory or a file exists with the same name as the directory indicated.

Action: Verify that OracleAS TopLink has permissions to create the necessary directories. Verify that there is sufficient disk space available for this operation.

Error Code: 13003

DIRECTORY_COULD_NOT_BE_CREATED

Cause: Checking or creating a file or document source failed because the `File.mkdirs` method failed to create the directory named by the specified abstract pathname, including any necessary but nonexistent parent.

Action: Verify that OracleAS TopLink has permissions to create the necessary directories. Verify that there is sufficient disk space available for this operation.

Error Code: 13004

DIRECTORY_NOT_FOUND

Cause: Directory does not exist and OracleAS TopLink has not been set to create directories as needed (`createsDirectoriesAsNeeded` policy is false.)

Action: Either create the appropriate directory or configure OracleAS TopLink to create directories as needed (set `createsDirectoriesAsNeeded` true.)

Error Code: 13005

FILE_ALREADY_EXISTS

Cause: OracleAS TopLink is attempting to create a file and the file already exists. OracleAS TopLink expects to be able to create a new version of the file and will not overwrite an existing file. This can happen if the XML

DataAccessor is trying to insert an XML file and the file already exists. This indicates an inconsistent state between the application and what is on disk.

Action: Change where OracleAS TopLink is writing or remove the existing file.

Error Code: 13006

UNABLE_TO_CREATE_WRITE_STREAM

Cause: Failed to create a `WriteStream` due to a `java.io.IOException`.

Action: See the generated exception for the root cause.

Error Code: 13007

INVALID_FIELD_VALUE

Cause: Failed to construct an XML element to represent an object because the object was an invalid type. For a direct collection, one or more of the elements had a type that was not null or `String`. For a nested row, one or more of the elements had a type that was not `DatabaseRow`.

Action: See the generated exception for the root cause. Verify the configuration of the object being persisted to ensure that it can be persisted in an XML data store.

Error Code: 13008

CLASS_NOT_FOUND

Cause: Failed to load the specified class due to a `java.lang.ClassNotFoundException`. This indicates a problem either with the OracleAS TopLink JAR (it is missing the class `oracle.toplink.xml.xerces.DefaultXMLTranslator`) or an improperly configured custom class loader (see the `DatabaseLogin.setXMLParserJARFileNames` method).

Action: See the generated exception for the root cause. Confirm that the OracleAS TopLink JAR contains `oracle.toplink.xml.xerces.DefaultXMLTranslator`. If you are using a custom class loader, confirm that this class is included in the list of JAR files passed into `DatabaseLogin.setXMLParserJARFileNames` method.

Error Code: 13009

SAX_PARSER_ERROR

Cause: Failed to parse the specified XML file due to an `org.xml.sax.SAXParseException`.

Action: See the generated exception for the root cause including the line and column number at which the SAXParseException was thrown.

Error Code: 13010

GENERAL_EXCEPTION

Cause: An operation failed due to something other than an `org.xml.sax.SAXParseException`.

Action: An exception was thrown either when trying to build a parser or to build a document that caused that action to fail. See the generated exception for the root cause.

Error Code: 13011

IOEXCEPTION

Cause: A `ReadStream` or `WriteStream` could not be created due to a `java.io.IOException`.

Action: See the generated exception for the root cause.

Error Code: 13012

UNABLE_TO_CLOSE_READ_STREAM

Cause: After reading a row from the XML data store, failed to close the `ReadStream` used due to a `java.io.IOException`.

Action: See the generated exception for the root cause.

Error Code: 13013

HETEROGENEOUS_CHILD_ELEMENTS

Cause: Composite elements are being stored in a DirectCollectionMapping. DirectCollectionMappings in the SDK will only work with simple elements. Simple elements contain only one child of type text in XML.

Action: Ensure elements that are mapped as direct collections only contain simple elements.

Cause: Child elements of a complex element are not the same type: the type of each child element must be the same as that of the first child element.

Action: Verify that the XML document is not corrupt. If it is valid, ensure that it meets SDK requirements as illustrated in Example C–13 and Example C–14.

Cause: Child elements of a complex element do not have the same name.

Action: Verify that the XML document is not corrupt. If it is valid, ensure that it meets SDK requirements as in Example C–13 and Example C–14.

Example C–13 XML Supported by the SDK

```
<foo>
    <bar> [string or nested elements] </bar>
    <bar> [must match the first child: either string or nested elements] </bar>
    <bar> [must match the first child: either string or nested elements] </bar>
</foo>
```

Example C–14 XML Not Supported by the SDK

```
<foo>
    <bar> ... </bar>
    <fred> [this element will cause the exception] </fred>
    <bar> ... </bar>
</foo>
```

Error Code: 13017

INSTANTIATION_EXCEPTION

Cause: Failed to instantiate the specified class due to a `java.lang.InstantiationException`. This indicates a problem either with the OracleAS TopLink JAR (it is missing the class `oracle.toplink.xml.xerces.DefaultXMLTranslator`) or an improperly configured custom class loader (see the `DatabaseLogin.setXMLParserJARFileNames` method).

Action: See the generated exception for the root cause. Ensure that the specified class is not an interface or an abstract class. Confirm that the OracleAS TopLink JAR contains oracle.toplink.xml.xerces.DefaultXMLTranslator. If you are using a custom class loader, confirm that this class is included in the list of JAR files passed into DatabaseLogin.setXMLParserJARFileNames method.

Error Code: 13018

INSTANTIATION_ILLEGAL_ACCESS_EXCEPTION

Cause: Failed to instantiate the specified class due to a `java.lang.IllegalAccessException`. This indicates a problem either with the OracleAS TopLink JAR (it is missing the class oracle.toplink.xml.xerces.DefaultXMLTranslator) or an improperly configured custom class loader (see the DatabaseLogin.setXMLParserJARFileNames method).

Action: See the generated exception for the root cause. Ensure that the specified class is public. Ensure permission is set for Java reflection in your VM security settings. Ensure that the specified class is not an interface or an abstract class. Confirm that the OracleAS TopLink JAR contains oracle.toplink.xml.xerces.DefaultXMLTranslator. If you are using a custom class loader, confirm that this class is included in the list of JAR files passed into DatabaseLogin.setXMLParserJARFileNames method.

Error Code: 13020

ELEMENT_DATA_TYPE_NAME_IS_REQUIRED

Cause: Failed to build XML for a given object because the object's data type name is null or zero length.

Action: Ensure that the element datatype name is provided

Deployment Exception

A Deployment Exception is a runtime exception thrown if problems are detected during deployment of an EJB. During deployment, project, sessions, and ejb-jar XML files (or their Java Class equivalents) are read and the necessary objects instantiated and initialized.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
```

Example C–15 Deployment Exception

```
EXCEPTION [TOPLINK – 14001]: oracle.toplink.ejb.DeploymentException
EXCEPTION DESCRIPTION: No OracleAS TopLink project was specified for this bean.
```

Error Codes 14001 - 14027

Error Code: 14001

NO_PROJECT_SPECIFIED

Cause: Neither project name nor class could be read from the deployment descriptor.

Action: Verify your project configuration in your deployment descriptor. Double check that either project-xml or project-class is specified.

Error Code: 14003

NO_SUCH_PROJECT_IDENTIFIER

Cause: No project exists with the identifier requested.

Action: Verify that the project name matches exactly the project name specified in your project XML file.

Error Code: 14004

ERROR_CREATING_CUSTOMIZATION

Cause: Could not create an instance of the DeploymentCustomization class.

Action: Verify the implementation of the class implementing the DeploymentCustomization interface. Start with the constructor, then proceed to the remainder of the implementation.

Error Code: 14005

ERROR_RUNNING_CUSTOMIZATION

Cause: An exception was thrown when either the `DeploymentCustomization.afterLoginCustomization` method or `DeploymentCustomization.beforeLoginCustomization` method was called.

Action: See the generated exception for the root cause. Verify the implementation of your `DeploymentCustomization`.

Error Code: 14011

ERROR_CONNECTING_TO_DATA_SOURCE

Cause: The data source could not be located in JNDI, or was not properly specified.

Action: Verify the datasource attribute of the login element in your sessions XML file. Ensure that the datasource is present and properly configured.

Error Code: 14016

ERROR_CREATING_PROJECT

Cause: The project XML file name or class was specified, but a general error occurred creating the project.

Action: See the generated exception for the root cause. Verify your project XML file.

Error Code: 14020
ERROR_IN_DEPLOYMENT_DESCRIPTOR

Cause: Error parsing the toplink-ejb-jar.xml.

Action: See the generated exception for the root cause. Verify your toplink-ejb-jar.xml file.

Error Code: 14023
CANNOT_FIND_GENERATED_SUBCLASS

Cause: An internal, unexpected Exception was thrown.

Action: See the Exception message provided.

Error Code: 14024
CANNOT_READ_TOPLINK_PROJECT

Cause: An internal, unexpected Exception was thrown reading the project.

Action: See the exception message provided.

Error Code: 14026
MUST_USE_TRANSPARENT_INDIRECTION

Cause: Your project contains either a one-to-many or many-to-many relationship (between EJB2.0 entity beans) which is not using transparent indirection.

Action: Verify your project is using transparent indirection for all one-to-many and many-to-many relationships involving EJB2.0 entity beans.

Error Code: 14027
MUST_USE_VALUEHOLDER

Cause: Your project contains a one-to-one relationship (between EJB2.0 entity beans) which is not using basic indirection.

Action: Verify your project is using basic indirection for all one-to-one relationships involving EJB2.0 entity beans.

Synchronization Exception

A Synchronization exception is a runtime exception that is raised when a cache synchronization update by OracleAS TopLink to a distributed session was unsuccessful. When this occurs, the message contains a reference to the error code and error message.

Error Codes 15001 - 15025

Error Code: 15001
UNABLE_TO_PROPAGATE_CHANGES

Cause: An error occurred when sending changes to remote system.

Action: See exception generated for cause.

Error Code: 15008
ERROR_DOING_LOCAL_MERGE

Cause: The local shared cache has become corrupt.

Action: Restart the session on this server or initializes.

Error Code: 15010
ERROR_LOOKING_UP_LOCAL_HOST

Cause: An IO exception occurred when attempting to discover local system IP address.

Action: See generated exception for root cause.

Error Code: 15011
ERROR_BINDING_CONTROLLER

Cause: An IO error occurred when attempting to register remote service.

Action: See generated exception for root cause and resolve.

Error Code: 15012
ERROR_LOOKING_UP_CONTROLLER

Cause: Unable to find remote server's remote service.

Action: See generated exception for root cause. Verify that remote server is running.

Error Code: 15013
LOOKING_UP_JMS_SERVICE

Cause: Unable to find the specified JMS service.

Action: Ensure that the IP address and port of the JMS service has been specified correctly in the session configuration and that the service is running.

Error Code: 15016

ERROR_GETTING_SYNC_SERVICE

Cause: An error occurred when attempting to initialize the Synchronization Service and specified in the Session configuration.

Action: See generated exception for root cause. Verify that the service has been properly specified in the session configuration.

Error Code: 15017

ERROR_NOTIFYING_CLUSTER

Cause: An error occurred when attempting to contact other OracleAS TopLink Sessions.

Action: See generated exception for root cause.

Error Code: 15018

ERROR_JOINING_MULTICAST_GROUP

Cause: An error occurred when attempting to join multicast group for OracleAS TopLink clustering handshaking phase.

Action: See generated exception for root cause.

Error Code: 15023

ERROR_RECEIVING_ANNOUNCEMENT

Cause: An IO error occurred when attempting to receive a session existence announcement from a remote OracleAS TopLink Session.

Action: See generated exception for root cause.

Error Code: 15025

FAIL_TO_RESET_CACHE_SYNCH

Cause: API on the Development Services called to reset OracleAS TopLink Cache Synchronization, including participation in the cluster, failed.

Action: See generated Exception for root cause.

JDO Exception

A JDO Exception is a runtime exception thrown when Java Data Objects are used.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
```

```
EXCEPTION DESCRIPTION: Message
```

Example C–16 JDO Exception

```
EXCEPTION [TOPLINK – 16004]: oracle.toplink.exceptions.JDOException
EXCEPTION DESCRIPTION: Cannot execute transactional read query without an active
transaction.
```

Error Codes 16001 - 16006

Error Code: 16001

OBJECT_IS_NOT_TRANSACTIONAL

Cause: Failed to delete an object because it was not registered with the currently active `UnitOfWork`.

Action: Register the specified object with the `UnitOfWork` before deleting.

Error Code: 16002

ARGUMENT_OBJECT_IS_NOT_JDO_OBJECTID

Cause: Failed to get an object by id because the `ObjectId` used was not a `JDOObjectId`.

Action: Ensure that you pass a `JDOObjectId` (not an `ObjectId`) when using the `JDOPersistenceManager`.

Error Code: 16003

OBJECT_FOR_ID_DOES_NOT_EXIST

Cause: Failed to get an object by id: no object with the specified `JDOObjectId` was found.

Action: Ensure that your application handles this exception appropriately.

Error Code: 16004

TRANSACTIONAL_READ_WITHOUT_ACTIVE_TRANSACTION

Cause: A query failed because although the `JDOPersistenceManager` is in a transactional read, the current transaction is inactive.

Action: Ensure there is an active transaction before doing a transactional read.

Error Code: 16005

TRANSACTION_IS_ALREADY_ACTIVE

Cause: Failed to begin a `JDOTransaction` because the transaction is already active.

Action: Commit or rollback the transaction before trying to begin.

Error Code: 16006

TRANSACTION_IS_NOT_ACTIVE

Cause: Failed to commit or rollback a `JDOTransaction` because the transaction is not active.

Action: Ensure that the transaction state is not altered by another application and is active before commit or rollback.

SDK Data Store Exception

An SDK Data Store Exception is a runtime exception thrown when SDK Classes are used to customize OracleAS TopLink.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
```

Example C–17 SDK Data Store Exception

```
EXCEPTION [TOPLINK – 17001]: oracle.toplink.sdk.SDKDataStoreException
EXCEPTION DESCRIPTION: The OracleAS TopLink SDK does not currently support
Cursor.
```

Error Codes 17001 - 17006

Error Code: 17001

UNSUPPORTED

Cause: A method call failed because it is not currently supported by the SDK.

Action: Avoid using the unsupported method.

Error Code: 17002

INCORRECT_LOGIN_INSTANCE_PROVIDED

Cause: An instance of `oracle.toplink.sdk.SDKAccessor` was passed the wrong type of `Login` (the SDK expects an instance of `DatabaseLogin`).

Action: Verify that your SDK-based application is being passed the expected type of `Login`.

Error Code: 17003

INVALID_CALL

Cause: When the `QueryManager` owned by an `SDKDescriptor` is initialized, an instance of `InvalidSDKCall` is set for each type of `Call` that is not configured. If you invoke an unconfigured `Call`, this INVALID_CALL error is logged rather than simply throwing a `NullPointerException` because the INVALID_CALL error contains more information.

Action: Avoid using the unconfigured `Call` or provide a `Call` implementation in your SDK-based application.

Error Code: 17004

IE_WHEN_INSTANTIATING_ACCESSOR

Cause: Failed to instantiate the specified class due to a `java.lang.InstantiationException`.

Action: Ensure that the specified class is not an interface or an abstract class.

Error Code: 17005

IAE_WHEN_INSTANTIATING_ACCESSOR

Cause: Failed to instantiate the specified class due to a `java.lang.IllegalAccessException`.

Action: Ensure that specified class is public. Ensure permission is set for Java reflection in your VM security settings.

Error Code: 17006

SDK_PLATFORM_DOES_SUPPORT_SEQUENCES

Cause: Unsupported `SDKPlatform` methods `buildSelectSequenceCall` or `buildUpdateSequenceCall` were called.

Action: Avoid using these methods or subclass `SDKPlatform` and override them with your own implementation.

JMS Processing Exception

A JMS Processing Exception is a runtime exception thrown when processing Java Messaging Service messages.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
```

Example C–18 JMS Processing Exception

```
EXCEPTION [TOPLINK – 18001]: oracle.toplink.exceptions.JMSProcessingException
EXCEPTION DESCRIPTION: Error while processing incomming JMS message.
```

Error Codes 18001 - 18002

Error Code: 18001

DEFAULT

Cause: Failed to process incoming JMS message.

Action: See generated exception for root cause.

Error Code: 18002

NO_TOPIC_SET

Cause: `JMSClusteringService` failed to start because the `Topic` created in the JMS service for the interconnection of sessions is null.

Action: Ensure that the `Topic` created in the JMS service for the interconnection of sessions is set in the `JMSClusteringService`.

SDK Descriptor Exception

An SDK Descriptor Exception is a runtime exception thrown when using SDK Classes to customize OracleAS TopLink Descriptors.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
```

Example C–19 SDK Descriptor Exception

```
EXCEPTION [TOPLINK – 19001]: oracle.toplink.sdk.SDKDescriptorException
EXCEPTION DESCRIPTION: The OracleAS TopLink SDK does not currently support query
result ordering.
```

Error Codes 19001 - 19003

Error Code: 19001
UNSUPPORTED

Cause: A method call failed because it is not currently supported by the SDK.

Action: Avoid using the unsupported method.

Error Code: 19002
CUSTOM_SELECTION_QUERY_REQUIRED

Cause: An `SDKObjectCollectionMapping` was used without a custom selection query.

Action: Set custom selection query on the `SDKObjectCollectionMapping`.

Error Code: 19003
SIZE_MISMATCH_OF_FIELD_TRANSLATIONS

Cause: Mapping field name array and data store field name array are of different lengths.

Action: The sizes of the field translation arrays must be equal.

SDK Query Exception

An SDK Query Exception is a runtime exception thrown when using SDK Classes to customize OracleAS TopLink Queries.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
```

Example C–20 SDK Query Exception

```
EXCEPTION [TOPLINK – 20002]: oracle.toplink.sdk.SDKQueryException
EXCEPTION DESCRIPTION: Invalid SDK mechanism state - only one call is allowed.
```

Error Codes 20001 - 20004

Error Code: 20001
INVALID_SDK_CALL

Cause: The passed call is not an instance of `SDKCall`.

Action: Use an instance of `SDKCall`.

Error Code: 20003
INVALID_SDK_ACCESSOR

Cause: Accessor set into `SDKQuery` is not an instance of `SDKAccessor`.

Action: Set `SDKAccessor`.

Error Code: 20004
INVALID_ACCESSOR_CLASS

Cause: The `SDKLogin.setAccessorClass` method was passed a class that does not implement the interface referred to by `ClassConstants.Accessor_Class`.

Action: Ensure that you pass a Class that implements the `Accessor` interface.

Discovery Exception

A Discovery Exception is a runtime exception thrown when DiscoveryManager is operating.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
```

Example C–21 Discovery Exception

```
EXCEPTION [TOPLINK – 22001]: oracle.toplink.exception.DiscoveryException
EXCEPTION DESCRIPTION: Could not join multicast group.
```

Error Codes 22001 - 22004

Error Code: 22001

ERROR_JOINING_MULTICAST_GROUP

Cause: `DiscoveryManager` failed to join a multicast group due to a `java.io.IOException`: either a `MulticastSocket` could not be created or the invocation of the `MulticastSocket.joingGroup` method failed.

Action: See the generated exception for root cause.

Error Code: 22002

ERROR_SENDING_ANNOUNCEMENT

Cause: `DiscoveryManager` failed to inform other services that its service has started up.

Action: Consider increasing the announcement delay: the amount of time in milliseconds that the service should wait between the time that this remote service is available and a session announcement is sent out to other discovery managers. This may be needed to give some systems more time to post their connections into the naming service. See the `DiscoveryManager.setAnnouncementDelay` method.

Error Code: 22004

ERROR_RECEIVING_ANNOUNCEMENT

Cause: `DiscoveryManager` caught a `java.io.IOException` while blocking for announcements from other `DiscoveryManagers`.

Action: See the generated exception for root cause.

Remote Command Manager Exception

A Remote Command Manager Exception is a runtime exception thrown when the Remote Command Module is used.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
```

Example C–22 Remote Command Manager Exception

```
EXCEPTION [TOPLINK – 22104]:
```

```
oracle.toplink.exceptions.RemoteCommandManagerException
EXCEPTION DESCRIPTION: Could not look up hostname.
```

Error Codes 22101 - 22105

Error Code: 22101

ERROR_OBTAINING_CONTEXT_FOR_JNDI

Cause: Failed to get a JNDI context with the specified properties due to a `javax.naming.NamingException`.

Action: See generated exception for root cause. Verify that the properties for looking up the context is correct.

Error Code: 22102

ERROR_BINDING_CONNECTION

Cause: Failed to post a connection in the local naming service.

Action: See generated exception for root cause.

Error Code: 22103

ERROR_LOOKING_UP_REMOTE_CONNECTION

Cause: Failed to look up a remote connection with the specified name and URL.

Action: See generated exception for root cause. Verify that remote connection and URL are correct.

Error Code: 22104

ERROR_GETTING_HOST_NAME

Cause: The `java.net.InetAddress.getLocalHost` method failed to look up the specified hostname.

Action: See generated exception for root cause. Verify that the host is on-line and reachable.

Error Code: 22105

ERROR_PROPAGATING_COMMAND

Cause: Failed to propagate a command to the specified connection.

Action: See generated exception for root cause. Verify that the remote host of the specified connection is on-line and reachable if the generated exception included a `CommunicationException`.

XML Conversion Exception

An XML Conversion Exception is a runtime exception thrown when conversion between OracleAS TopLink instances and XML failed. This exception is used in cache synchronization that uses XML change set.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
```

Example C–23 XML Conversion Exception

```
EXCEPTION [TOPLINK – 25001]: oracle.toplink.exceptions.XMLConversionException
EXCEPTION DESCRIPTION: Cannot create URL for file  [\\FILE_SERVER\command.xml].
```

Error Code 25001

Error Code: 25001

ERROR_CREATE_URL

Cause: Failed to create a URL for the specified file.

Action: Ensure the specified file exists and is accessible.

EJB JAR XML Exception

An EJB JAR XML Exception is a runtime exception thrown at deployment time when the ejb-jar XML file is read and required concrete EJB Classes code generated.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
```

Example C–24 EJB JAR XML Exception

```
EXCEPTION [TOPLINK – 72000]: oracle.toplink.exceptions.EJBJarXMLException
```

```
EXCEPTION DESCRIPTION: Error reading ejb-jar.xml file.
```

Error Codes 72000 - 72023

Error Code: 72000

READ_EXCEPTION

Cause: Failed to read an ejb-jar XML file due to a `java.io.IOException` or `javax.xml.parsers.ParserConfigurationException`.

Action: See generated exception for root cause.

Error Code: 72001

INVALID_DOC_TYPE

Cause: Failed to parse the specified file because it did not use the expected `doctype: -//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN`

Action: Verify that your ejb-jar XML file uses the correct doc type.

Error Code: 72023

NO_CMR_FIELD_FOR_BEAN_ABSTRACT_SETTER

Cause: Code generation of a one-to-one bean setter method body failed because: the `Descriptor` was null, the `Descriptor` has no `InheritancePolicy`, the `Descriptor InheritancePolicy` has a null parent class, or no Container Managed Relation field defined.

Action: Verify the configuration of this Deployment Descriptor in your ejb-jar XML file.

Entity Deployment

This section discusses some of the general troubleshooting issues surrounding entity bean configuration and deployment. It lists many of the common exceptions and error messages that you may run across when attempting to deploy and persist entity beans using OracleAS TopLink.

If you encounter any problems installing OracleAS TopLink, using the OracleAS TopLink Mapping Workbench, or require more information on any runtime exceptions that are generated by OracleAS TopLink, consult the appropriate documentation.

Generating Deployment JARs

If you experience trouble generating the JARs for deployment,

- Ensure that all environment entries (class path, and so on) are configured properly.
- Identify which step of the build is failing (copying, compiling, running EJB compiler, and so on.)

Running the Enterprise JavaBean (EJB) compiler utility involves several processes, such as compiling, code-generation, EJB compliance verification, compiling RMI stubs by running `rmic`, and so on. If an error occurs during the running of the EJB compiler utility, try to determine which stage may be causing the failure.

For more information about the EJB compiler, see the server documentation.

Common BEA WebLogic Deployment Exceptions

The following are some of the most common errors that are encountered when you deploy to a BEA WebLogic applications server.

For more information about specific versions, see

- "Common BEA WebLogic 6.1 Exceptions" on page C-126
- "Common BEA WebLogic 7.0 Exceptions" on page C-130
- "Common BEA WebLogic 8.1 Exceptions" on page C-132

Assertion Error

```
weblogic.utils.AssertionError: ***** ASSERTION FAILED *****[
   Could not load class
   'oracle.toplink.internal.ejb.cmp.wls.WlsCMPDeployer':
   java.lang.ClassNotFoundException:
   oracle.toplink.internal.ejb.cmp.wls.WlsCMPDeployer
ERROR: ejbc found errors
```

Cause: This error occurs if the `toplink.jar` file is not properly set on your class path.

Action: Ensure the `<ORACLE_HOME>/toplink/jlib/toplink.jar` file is specified on your system class path.

Error Deploying Application

Cause: A **DeploymentException** has occurred.

Action: Refer to the specific error code. The error code appears in the square brackets in the exception message, such as `[TopLink-8001]`). These errors may be refer to errors in the specification of the project location reading in the properties file or validation errors due to improper mappings.

Exception 8001 `<Error> <J2EE> <Error deploying application Account:`

```
Unable to deploy EJB: AccountBean from Account.jar:

LOCAL EXCEPTION STACK:

EXCEPTION [TOPLINK-8001] (TopLink (WLS CMP) - X.X.X):
oracle.toplink.ejb.DeploymentException

EXCEPTION DESCRIPTION: No OracleAS TopLink project was
specified for this bean.

   at

oracle.toplink.ejb.DeploymentException.noPro
jectSpecified(DeploymentException.java:132) at
oracle.toplink.internal.ejb.cmp.ProjectDeployment.readProject(
ProjectDeployment.java:378)
```

Cause: This error occurs if the OracleAS TopLink project file is not specified in the `toplink-ejb-jar.xml`.

Action: Ensure there is an entry in the `toplink-ejb-jar.xml` file for either the `project-xml` or `project-class`.

Exception 8016 `<Error> <J2EE> <Error deploying application Account:`

```
Unable to deploy EJB: AccountBean from Account.jar:

LOCAL EXCEPTION STACK:

EXCEPTION [TOPLINK-8016] (TopLink (WLS CMP) - X.X.X):
oracle.toplink.ejb.DeploymentException

EXCEPTION DESCRIPTION: An error occurred while setting up the
project: [java.io.FileNotFoundException: Account.xml]

INTERNAL EXCEPTION: java.io.FileNotFoundException: Account.xml

   at
```

```
oracle.toplink.ejb.DeploymentException.errorCreatingProject(Un
known Source)
```

Cause: This error can occur if the location of the OracleAS TopLink project file for the bean is not properly specified.

Action: Check the file name as it is specified in the `toplink-ejb-jar.xml` file, and the location of the project file on the file system.

Cannot Startup Connection Pool

```
<Error> <JDBC> <Cannot startup connection pool "ejbPool"
   weblogic.common.ResourceException: Cannot load driver class:
   org.hsqldb.jdbcDriver>
...
```

Cause: An error has occurred in setting up the connection pool.

Action: Check the nested SQL exception to determine the cause of the error. Typical problems include:

- The driver is not on the class path.
- The user or password is incorrect.
- The database server URL or driver name is not properly specified.

Please consult the BEA WebLogic documentation and your JDBC Driver documentation for help on the specific error raised by BEA WebLogic.

Error Message `weblogic.utils.AssertionError: ***** ASSERTION FAILED *****[ Could not create an instance of class 'null': java.lang.NullPointerException`

```
at java.lang.Class.forName0(Native Method)
at java.lang.Class.forName(Class.java:120)
at weblogic.ejb20.persistence.PersistenceType.
loadClass(PersistenceType.java:309)
```

Cause: This problem occurs if using the GA version of BEA WebLogic Server 6.0.

Action: Upgrade to at least WebLogic 6.0 (Service Pack 1).

EJBC Found Errors

```
ERROR: ejbc found errors
Error from ejbc: Error while loading persistence resource
TopLink_CMP_Descriptor.xml Make sure that the persistence type
is in your class path.
```

Cause: This error occurs if the `toplink.jar` file is not properly set on your class path.

Action: Ensure the `<ORACLE_HOME>/toplink/jlib/toplink.jar` file is specified on your system class path.

EJB Deployment Exception

```
weblogic.ejb20.EJBDeploymentException: Error Deploying CMP EJB:;
   nested exception is: weblogic.ejb20.cmp.rdbms.RDBMSException:
   An error occurred setting up the project:
EXCEPTION [TOPLINK-13000] (vX.X [TopLink for WebLogic X.X]
JDK1.2): oracle.toplink.xml.XMLDataStoreException
```

EXCEPTION DESCRIPTION: File not found...

Cause: This error occurs if the location of the OracleAS TopLink project file for the bean is not properly specified.

Action: Check the file name as it is specified in the `toplink-ejb-jar.xml` file, and the location of the OracleAS TopLink project file on the file system.

Deploying EJB Component

```
Error deploying EJB Component: ...
   weblogic.ejb20.EJBDeploymentException: Exception in EJB
   Deployment; nested exception is:
Error while deploying bean..., File ... Not Found at
weblogic.ejb20.persistence.PersistenceType.setup
Deployer(PersistenceType.java:273)
```

Cause: A typical cause of this error is that the `toplink-ejb-jar.xml` file is referring to a local DTD file using a file name or location that is incorrect.

Action: Ensure that all XML files refer to valid DTD files and locations.

Cannot Startup Connection Pool ejbPool

```
Cannot startup connection pool "ejbPool"
   weblogic.common.ResourceException:
Could not create pool connection. The DBMS driver exception
was:

...
```

Action: An error has occurred in setting up the connection pool. Check the nested SQL exception to determine the cause of the error. Typical problems include:

- The driver is not on the class path.
- The user name or password is incorrect.
- The database server URL or driver name is not properly specified.

Please consult the BEA WebLogic documentation and your JDBC driver documentation for help on the specific error raised by BEA WebLogic.

Other Errors

Occasionally, changes made to the server's configuration file (`config.xml`) do not appear to be applied when the server is restarted. If this occurs, try removing the temp directories created by BEA WebLogic. You can find them under the `wlserver6.1` directory, at the same level as the `config` directory.

Common IBM WebSphere Server Exceptions

When the IBM WebSphere Server is started, it attempts to deploy the JAR files that are specified for deployment within the application server.

Errors that occur when the server is started are usually configuration problems that involve class path issues, environment variable configuration, and database login configuration. Review the IBM WebSphere Server documentation on starting the server.

This section contains some of the exceptions and errors that can be encountered when running the IBM WebSphere Server, along with their possible causes and recommended solutions.

Class Not Found Exceptions

Cause: The class not found is not included on the WebSphere application extensions class path or in the EJB or WAR module.

Action: Ensure that all required classes are included in the correct location. For more information about class path locations, see the *IBM WebSphere InfoCenter*.

Cause: The required OracleAS TopLink JARs have not been copied into the application extensions class path.

Action: Ensure that `toplink.jar` and `antlr.jar` are copied into the `<`*`WebSphere install`*`>\lib\app` directory.

oracle.toplink.exceptions.DatabaseException

Cause: An OracleAS TopLink Exception has occurred.

Action: Refer to the specific error code. The error code appears in the square brackets in the exception message, such as `[TopLink-1016]`). Errors observed here may be errors in reading in the properties file, or validation errors due to improper mappings.

Exception [6066]

```
oracle.toplink.exceptions.QueryException: The object <Object>
of class <class> with identity hashcode <hashcode> is not from
this Unit of Work object space but the parent session's. The
object was never registered in this Unit of Work, but read
from the parent session and related to an object registered in
the Unit of Work. Ensure that you are correctly registering
your objects. If you are still having problems, you can use
the UnitOfWork.validateObjectSpace() method to help debug
where the error occurred. Please see the manual and FAQ for
more information.
```

Cause: A bean was created outside of a transaction and then a second bean was created either in or out of a transaction.

Action: Ensure that all creates are performed within the context of a transaction.

Cause: The bean was not cleared out during `ejbPassivate`.

Action: Ensure that the `ejbPassivate` clears out the bean.

Cause: A bean-to-object relationship is not privately owned.

Action: Ensure that all bean-to-object relationships are privately owned.

Exception [7064]

`oracle.toplink.exceptions.ValidationException: Exception occured in reflective EJB bean primary key extraction, please ensure your primary key object is defined correctly: key = 301, bean = <beanName>`

Cause: An incorrect primary key object is being used with a bean.

Action: Ensure that you are using the correct primary key object for a bean.

Exception [7066]

`oracle.toplink.exceptions.ValidationException: Cannot create or remove beans unless a JTS transaction is present, bean=<bean>`

Cause: An attempt was made to create or remove a been outside of a transaction.

Action: Ensure that all removing and creating of beans is performed within a transaction.

Exception [7068]

`oracle.toplink.exceptions.ValidationException: The project class <projectclass> was not found for the <toplink_session_name> using default class loader.`

Cause: The project class that is specified in the `toplink.properties` file for the session specified on the **toplink_session_name environment** variable cannot be found.

Action: Ensure that the project class given in the exception is on the WebSphere dependent class path.

Exception [7069]

`oracle.toplink.exceptions.ValidationException: An exception occured looking up or invoking the project amendment method, <amendmentMethod> on the class <amendmentClass>;`

Cause: An amendment method was called, but cannot be found.

Action: Ensure that the required amendment method exists on the class that is specified.

Exception [7070]

`oracle.toplink.exceptions.ValidationException: A toplink.properties resource bundle must be located on the class path in an OracleAS TopLink directory.`

Cause: The `toplink.properties` file cannot be found.

Action: Ensure that the location of the `toplink.properties` file is on the class path.

Exception [7079]

`EXCEPTION DESCRIPTION: The descriptor for [<bean class>] was not found in the session [<session name>]. Check the project being used for this session.`

Cause: The descriptor that is listed was not found in the session that is specified on the deployment descriptor.

Action: Ensure that the project that is specified in the `toplink-ejb-jar.xml` file is the desired project. Also check that the project includes a descriptor for the missing bean class.

Exception [7101]

`No "meta-inf/toplink-ejb-jar.xml" could be found in your class path. The CMP session could not be read in from file.`

Cause: The `toplink-ejb-jar.xml` file was not found.

Action: Ensure that the `toplink-ejb-jar.xml` file is located in the deployed `ejb-jar` file under the `meta-inf` directory.

Exception [9002]

`EXCEPTION [TOPLINK-9002] (TopLink - X.X.X): oracle.toplink.exceptions.SessionLoaderExceptionEXCEPTION DESCRIPTION: Unable to load Project class [<project class>].`

Cause: The project class that is specified for the session in the `toplink-ejb-jar.xml` file cannot be found.

Action: Ensure that the project class has been included in the deployed JAR with the entity beans.

Problems at Runtime

This section lists some of the common exceptions and errors that can occur at runtime when using the OracleAS TopLink CMP for IBM WebSphere Application Server.

Exception [6026]

`oracle.toplink.exceptions: Query is not defined`

Cause: A required named query does not exist.

Action: Implement the named query. The stacktrace of the exception contains the finder that failed.

Common OracleAS TopLink for IBM WebSphere Deploy Tool Exceptions

This following section lists common exceptions and errors that may occur when running the OracleAS TopLink for IBM WebSphere Deploy Tool.

Class Not Found Exceptions

Cause: The class that is specified was not found; it is not included on the deploy tool class path or the system class path.

Action: Ensure that all required classes are included on the correct class path. For more information about class path setup, see the *IBM WebSphere Getting Started*.

Note: The Deploy Tool calls external IBM classes to generate deployed code. Any exceptions that are thrown from these classes is described on `System.out`. Check the **Tracing** button to view the most detailed information possible.

Common BEA WebLogic 6.1 Exceptions

Following are a few of the most common errors you may encounter when deploying JAR files with OracleAS TopLink and BEA WebLogic 6.1.

Development Exceptions

Missing Persistence Type `ERROR: Error from ejbc: Persistence type 'TopLink_CMP_2_0' with version 'X.X which is referenced in bean`

'Account' is not installed. The installed persistence types are: (WebLogic_CMP_RDBMS, 6.0), (WebLogic_CMP_RDBMS, 5.1.0).

`ERROR: ejbc found errors`

Cause: There is no entry in the `persistence.install` file for OracleAS TopLink CMP. This may occur if the OracleAS TopLink installation was interrupted or a BEA WebLogic Service Pack was applied.

Action: In the `<`*`WebLogic InstallDir`*`>/wlserver6.1/lib/persistence` directory, edit the `persistence.install` file to add a new line `TopLink_CMP_Descriptor.xml`, or replace your existing `persistence.install` file with the version of the file in the `<`*`ORACLE_HOME`*`>/toplink/config` directory.

Error Loading Persistence Resource `Error while loading persistence resource TopLink_CMP_Descriptor.xml Make sure that the persistence type is in your class path.`

Cause: The `toplink.jar` file is not properly set in your class path.

Action: Ensure that the class path includes the `<`*`ORACLE_HOME`*`>/toplink/jlib/toplink.jar` file.

Wrong BEA WebLogic Version `C:\<`*`ORACLE_HOME`*`>\toplink\examples\weblogic\wls61\examples\ejb\cmp20\singlebean\Account.java:10: cannot resolve symbol`

```
symbol  : class EJBLocalObject
location: interface examples.ejb.cmp20.singlebean.Account
public interface Account extends EJBLocalObject {
```

Cause: You are trying to compile your code using BEA WebLogic 6.0.

Action: Compile using BEA WebLogic 6.1.

Deployment and Runtime Exceptions

Missing Persistence Type `Persistence type 'TopLink_CMP_2_0' with version 'X.X which is referenced in bean 'Account' is not installed. The installed persistence types are: (WebLogic_CMP_RDBMS, 6.0), (WebLogic_CMP_RDBMS, 5.1.0).`

Cause: There is no entry in the `persistence.install` file for OracleAS TopLink CMP. This may occur if the OracleAS TopLink installation was interrupted, or a BEA WebLogic Service Pack was applied.

Action: In the <*WebLogic InstallDir*>/wlserver6.1/lib/persistence directory, edit the `persistence.install` file to add a new line: `TopLink_CMP_Descriptor.xml`. You can also replace your existing `persistence.install` file with the version of the file in the <*ORACLE_HOME*>/toplink/config directory.

Error Loading Persistence Resource <*DATE and TIME*> <Error> <J2EE> <Error deploying application ejb20_cmp_order:

```
Unable to deploy EJB: C:\<ORACLE_
HOME>\toplink\examples\weblogic\wls61\server\config\TopLink_
Domain\applications\wlnotdelete\wlap64280\ejb20_cmp_order.jar
from ejb20_cmp_order.jar:

Error while loading persistence resource TopLink_CMP_
Descriptor.xml Make sure that the persistence type is in your
class path.

   at

weblogic.ejb20.persistence.InstalledPersistence.initialize(Ins
talledPersistence.java:214)

   at

weblogic.ejb20.persistence.InstalledPersistence.getInstalledTy
pe(InstalledPersistence.java:113)
```

Cause: The `toplink.jar` file is not properly set in your class path.

Action: Ensure that the class path includes the <*ORACLE_HOME*>/toplink/jlib/toplink.jar file.

Wrong Persistence Version *DATE and TIME*> <Error> <J2EE> <Error deploying application ejb20_cmp_account:

```
Unable to deploy EJB: Account from ejb20_cmp_account.jar:

java.lang.AbstractMethodError

   at

weblogic.ejb20.deployer.ClientDrivenBeanInfoImpl.deploy(Client
DrivenBeanInfoImpl.java:807)

   at

weblogic.ejb20.deployer.Deployer.deployDescriptor(Deployer.jav
a:1234)
```

```
    at
weblogic.ejb20.deployer.Deployer.deploy(Deployer.java:947)
    at
weblogic.j2ee.EJBComponent.deploy(EJBComponent.java:30)
```

Cause: You may be using a `persistence-version` meant for BEA WebLogic 7.0.

Action: Use a `persistence-version` of 4.0.

Cannot Startup Datasource `EXCEPTION [TOPLINK-7060] (TopLink (WLS CMP)-X.X):oracle.toplink.exceptions.ValidationException`

```
EXCEPTION DESCRIPTION: Cannot acquire datasource
[jdbc/ejbNonJTSDataSource].

INTERNAL EXCEPTION: javax.naming.NameNotFoundException: Unable
to resolve jdbc.ejbNonJTSDataSource Resolved: ''
Unresolved:'jdbc' ; remaining name 'ejbNonJTSDataSource'
```

Cause: An error has occurred in setting up the datasource.

Action: Check the nested SQL exception to determine the cause of the error. For more information, see "Error code: 7060". For more information on a specific error raised by WebLogic, see the BEA WebLogic documentation and your JDBC Driver documentation.

Wrong WebLogic Version `<DATE and TIME> <Error> <Management> <Error parsing XML descriptor for application TopLink_Domain:Name=ejb20_cmp_account, Type=Application`

```
weblogic.xml.process.ProcessorFactoryException: Could not
locate processor for public id = "-//Sun Microsystems,
Inc.//DTD J2EE Application 1.3//EN"
    at
weblogic.xml.process.ProcessorFactory.getProcessor(ProcessorFa
ctory.java:181)
    at
weblogic.xml.process.ProcessorFactory.getProcessor(ProcessorFa
ctory.java:164)
```

Cause: You are trying to compile your code using BEA WebLogic 6.0.

Action: Compile using BEA WebLogic 6.1.

Common BEA WebLogic 7.0 Exceptions

Following are a few of the most common errors you may encounter when deploying JAR files with OracleAS TopLink and BEA WebLogic 7.0.

Development-time Exceptions:

Missing Persistence Type `Persistence type 'TopLink_CMP_2_0' with version 'X.0 which is referenced in bean 'Account' is not installed. The installed persistence types are: (WebLogic_CMP_RDBMS, 6.0), (WebLogic_CMP_RDBMS, 5.1.0), (WebLogic_CMP_RDBMS, 7.0)`

`ERROR: ejbc found errors`

Cause: There is no entry in the `persistence.install` file for OracleAS TopLink CMP. This may occur if the OracleAS TopLink installation was interrupted, or a BEA WebLogic Service Pack was applied.

Action: In the `<`*`WebLogic InstallDir`*`>/weblogic700/lib/persistence` directory, edit the `persistence.install` file to add a new line: `TopLink_CMP_Descriptor.xml`. You can also replace your existing `persistence.install` file with the version of the file in the `<`*`ORACLE_HOME`*`>/toplink/config` directory.

Error Loading Persistence Resource `ERROR: Error from ejbc: Error while loading persistence resource TopLink_CMP_Descriptor.xml`

```
Make sure that the persistence type is in your class path.
```

`ERROR: ejbc found errors`

or

```
ERROR: at

weblogic.ejb20.persistence.InstalledPersistence.initialize(InstalledPersistence.java:214)

   at

weblogic.ejb20.persistence.InstalledPersistence.getInstalledType(InstalledPersistence.java:113)

   at
```

```
weblogic.ejb20.deployer.MBeanDeploymentInfoImpl.getPersistence
Type(MBeanDeploymentInfoImpl.java:584
```

Cause: The `toplink.jar` file is not properly set in your class path.

Action: Ensure that the class path includes the *<ORACLE_HOME>*`/toplink/jlib/toplink.jar` file.

Wrong WebLogic Version `ERROR: Error processing 'META-INF/weblogic-ejb-jar.xml': The public id, "-//BEA Systems, Inc.//DTD WebLogic 7.0.0 EJB//EN", specified in the XML document is invalid. Use one of the following valid public ids:`

`"-//BEA Systems, Inc.//DTD WebLogic 5.1.0 EJB//EN"`

`"-//BEA Systems, Inc.//DTD WebLogic 6.0.0 EJB//EN"`

`ERROR: ejbc found errors`

Cause: You are trying to compile your JAR using BEA WebLogic 6.1.

Action: Compile using BEA WebLogic 7.0.

Deployment/runtime Exceptions:

Missing Persistence Type `Error from ejbc: Persistence type 'TopLink_CMP_2_0' with version 'X.0 which is referenced in bean 'Account' is not installed. The installed persistence types are: (WebLogic_CMP_RDBMS, 6.0), (WebLogic_CMP_RDBMS, 5.1.0), (WebLogic_CMP_RDBMS, 7.0).`

```
Persistence type 'TopLink_CMP_2_0' with version 'X.0 which is
referenced in bean 'Account' is not installed. The installed
persistence types are: (WebLogic_CMP_RDBMS, 6.0), (WebLogic_
CMP_RDBMS, 5.1.0), (WebLogic_CMP_RDBMS, 7.0)
```

Cause: There is no entry in the `persistence.install` file for OracleAS TopLink CMP. This may occur if the OracleAS TopLink installation was interrupted or a BEA WebLogic Service Pack was applied.

Action: In the *<WebLogic InstallDir>*`/weblogic7.0/lib/persistence` directory, edit the `persistence.install` file to add a new line: `TopLink_CMP_Descriptor.xml`. You can also replace your existing `persistence.install` file with the version of the file in the *<ORACLE_HOME>*`/toplink/config` directory.

Error Loading Persistence Resource `java.lang.NullPointerException`

```
   at
weblogic.ejb20.deployer.EJBDeployer.deactivate(EJBDeployer.jav
a:1513)
at
weblogic.ejb20.deployer.EJBDeployer.undeploy(EJBDeployer.java:
301)
   at
weblogic.ejb20.deployer.Deployer.deploy(Deployer.java:875)
   at
weblogic.j2ee.EJBComponent.deploy(EJBComponent.java:70)
```

Cause: The `toplink.jar` file is not properly set in your class path.

Action: Ensure that the class path includes the `<ORACLE_HOME>/toplink/jlib/toplink.jar` file.

Cannot Startup Datasource `EXCEPTION [TOPLINK-7060] (TopLink (WLS CMP) - X.X.X): oracle.toplink.exceptions.ValidationException`

```
EXCEPTION DESCRIPTION: Cannot acquire datasource
[jdbc/ejbNonJTSDataSource].
INTERNAL EXCEPTION: javax.naming.NameNotFoundException: Unable
to resolve jdbc.ejbNonJTSDataSource Resolved: ''
Unresolved:'jdbc' ; remaining name 'ejbNonJTSDataSource'
```

Cause: An error has occurred in setting up the datasource.

Action: Check the nested SQL exception to determine the cause of the error. For more information, see "Error code: 7060". For more information on a specific error raised by WebLogic, see the BEA WebLogic documentation and your JDBC Driver documentation.

Common BEA WebLogic 8.1 Exceptions

Following are a few of the most common errors you may encounter when deploying JAR files with OracleAS TopLink and BEA WebLogic 8.1.

Development-time Exceptions:

Missing Persistence Type `Persistence type 'TopLink_CMP_2_0' with version 'X.0 which is referenced in bean 'Account' is not installed.  The installed persistence types are: (WebLogic_CMP_RDBMS, 7.0), (WebLogic_CMP_RDBMS, 6.0), (WebLogic_CMP_RDBMS, 5.1.0).`

`ERROR: ejbc couldn’t invoke compiler`

Cause: There is no entry in the `persistence.install` file for OracleAS TopLink CMP. This may occur if the OracleAS TopLink installation was interrupted, or a BEA WebLogic Service Pack was applied.

Action: In the `<`*`WebLogic InstallDir`*`>/weblogic81/lib/persistence` directory, edit the `persistence.install` file to add a new line: `TopLink_CMP_Descriptor.xml`. You can also replace your existing `persistence.install` file with the version of the file in the `<`*`ORACLE_HOME`*`>/toplink/config` directory.

Error Loading Persistence Resource `Error occurred while loading persistence resource TopLink_CMP_Descriptor.xml. Make sure that the persistence type is in your class path.`

`ERROR: ejbc couldn’t invoke compiler`

Cause: The `toplink.jar` file is not properly set in your class path.

Action: Ensure that the class path includes the `<`*`ORACLE_HOME`*`>/toplink/jlib/toplink.jar` file.

Wrong WebLogic Version `ERROR: ejbc found errors while processing the descriptor for std_cmp20-singlebean.jar:`

```
ERROR: ejbc found errors while processing
   'META-INF/weblogic-ejb-jar.xml': The public id, "-//BEA
   Systems, Inc.//DTD WebLogic 8.1.0 EJB//EN", specified in
   the XML document is invalid.  Use one of the following
   valid public ids:
"-//BEA Systems, Inc.//DTD WebLogic 5.1.0 EJB//EN"
"-//BEA Systems, Inc.//DTD WebLogic 6.0.0 EJB//EN"
"-//BEA Systems, Inc.//DTD WebLogic 7.0.0 EJB//EN"
ERROR: ejbc found errors
```

Cause: You are trying to compile your using BEA WebLogic 7.0.

Action: Compile using BEA WebLogic 8.1.

Deployment/runtime Exceptions:

Missing Persistence Type `Error Deployer BEA-149201 Failed to complete the deployment task with ID 0 for the application _appsdir_cmp20-singlebean_ear. weblogic.management.ApplicationException: Exception:weblogic.management.ApplicationException: prepare failed for cmp20-singlebean.jar`

`Module: cmp20-singlebean.jar Error: Exception preparing module: EJBModule(cmp20-singlebean.jar,status=NEW)`

`Persistence type 'TopLink_CMP_2_0' with version 'X.0 which is referenced in bean 'Account' is not installed. The installed persistence types are: (WebLogic_CMP_RDBMS, 7.0), (WebLogic_CMP_RDBMS, 6.0), (WebLogic_CMP_RDBMS, 5.1.0)`

Cause: There is no entry in the `persistence.install` file for OracleAS TopLink CMP. This may occur if the OracleAS TopLink installation was interrupted or a BEA WebLogic Service Pack was applied.

Action: In the `<`*`WebLogic InstallDir`*`>/weblogic81/lib/persistence` directory, edit the `persistence.install` file to add a new line: `TopLink_CMP_Descriptor.xml`. You can also replace your existing `persistence.install` file with the version of the file in the `<`*`ORACLE_HOME`*`>/toplink/config` directory.

Error Loading Persistence Resource `Error Deployer BEA-149201 Failed to complete the deployment task with ID 2 for the application _appsdir_cmp20-relationships_ear.`

`weblogic.management.ApplicationException: Exception:weblogic.management.ApplicationException: prepare failed for cmp20-relationships.jar`

`Module: cmp20-relationships.jar Error: Exception preparing module: EJBModule(cmp20-relationships.jar,status=NEW)`

`Unable to deploy EJB: .\TopLink_Demos\stage\_appsdir_cmp20-relationships_ear\cmp20-relationships.jar from cmp20-relationships.jar:`

`[EJB:011004]Error occurred while loading persistence resource TopLink_CMP_Descriptor.xml. Make sure that the persistence type is in your classpath.`

`at weblogic.ejb20.persistence.InstalledPersistence.initialize(InstalledPersistence.java:212)`

```
at weblogic.ejb20.persistence.InstalledPersistence.getIn-
stalledType(InstalledPersistence.java:114)
```

Index

C

D

E

F

G

H

K

L

M

N

O

P

Q

R

S

T

U

V

W

X