Oracle® TopLink

Developer's Guide

10*g* (10.1.3.1.0)

B28218-01

September 2006

ORACLE®

Oracle TopLink Developer's Guide, 10*g* (10.1.3.1.0)

B28218-01

Copyright © 1997, 2006, Oracle. All rights reserved.

Primary Author: Peter Purich

Contributing Author: Rick Sapir. Liza Rekadze

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software--Restricted Rights (June 1987). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

Contents

Preface xlix

Part I Building a TopLink Application

1 Understanding TopLink

What is TopLink? 1-1
Solving the Object-Persistence Impedance Mismatch 1-2
TopLink Key Features 1-4
TopLink Application Architectures 1-4

2 Understanding TopLink Application Development

Developing Your Application With TopLink 2-1
Typical Development Stages 2-2
Oracle Development Support 2-3
Designing Your Application With TopLink 2-3
Understanding TopLink Usage 2-3
Relational Database Usage 2-4
Object-Relational Database Usage 2-4
Oracle XML Database (XDB) Usage 2-4
Enterprise Information System (EIS) Usage 2-4
XML Usage 2-4
Understanding Target Platforms 2-5
Selecting an Architecture With TopLink 2-5
Tiers 2-5
Three Tier 2-6
Two Tier 2-6
Service Layer 2-7
EJB Session Beans 2-7
EJB Entity Beans 2-7
EJB 3.0 JPA Entities 2-8
Plain Old Java Objects (POJO) 2-8
Data Access 2-9
Data Type 2-9
Multiple Data Sources 2-9
Isolating Data Access 2-9

Historical Data Access ... 2-9
Caching ... 2-9
Cache Type ... 2-10
Refreshing ... 2-10
Cache Coordination ... 2-10
Locking ... 2-11
Optimistic Locking ... 2-11
Pessimistic Locking ... 2-11
Building and Using the Persistence Layer ... 2-11
Implementation Options ... 2-12
Persistent Class Requirements ... 2-12
Persistence Layer Components ... 2-12
Mapping Metadata ... 2-13
Session Metadata ... 2-13
Cache ... 2-13
Queries and Expressions ... 2-13
Transactions ... 2-14
Using the Persistence Layer ... 2-14
Deploying the Application ... 2-15
Understanding Deployments ... 2-15
TopLink in a J2EE Application ... 2-15
Optimizing and Customizing the Application ... 2-15
Troubleshooting the Application ... 2-16
Understanding Object Persistence ... 2-16
Application Object Model ... 2-16
Data Storage Schema ... 2-17
Primary Keys and Object Identity ... 2-17
Mappings ... 2-17
Foreign Keys and Object Relationships ... 2-17
Inheritance ... 2-18
Concurrency ... 2-18
Caching ... 2-18
Nonintrusive Persistence ... 2-19
Indirection ... 2-19
Understanding TopLink Metadata ... 2-19
Advantages of the TopLink Metadata Architecture ... 2-20
Creating Project Metadata ... 2-20
Descriptors and Mappings ... 2-21
Data Source Login Information ... 2-21
Creating Session Metadata ... 2-21
Deploying Metadata ... 2-22
Understanding the Three-Tier Architecture ... 2-22
Example Implementations ... 2-22
Advantages and Disadvantages ... 2-22
Variation Using Remote Sessions ... 2-23
Technical Challenges ... 2-23
Understanding the Two-Tier Architecture ... 2-24

Example Implementations 2-24
Advantages and Disadvantages 2-24
Technical Challenges 2-24
Understanding the EJB Session Bean Facade Architecture 2-25
Example Implementation 2-25
Advantages and Disadvantages 2-26
Understanding Session Beans 2-26
Technical Challenges 2-26
Unit of Work Merge 2-27
Understanding the EJB Entity Beans With CMP Architecture 2-27
Example Implementation 2-28
Advantages and Disadvantages 2-28
Technical Challenges 2-29
External JDBC Pools 2-29
JTA/JTS Integration 2-29
Cache Coordination 2-29
Maintaining Bidirectional Relationships 2-29
Managing Dependent Objects 2-30
Managing Collections of EJBObject Objects 2-31
Understanding the EJB Entity Beans With BMP Architecture 2-31
Example Implementations 2-32
Advantages and Disadvantages 2-32
Technical Challenges 2-33
External JDBC Pools 2-33
JTA/JTS Integration 2-33
Cache Coordination 2-33
Understanding the EJB 3.0 JPA Entity Architecture 2-33
Example Implementations 2-34
Advantages and Disadvantages 2-35
Understanding the Web Services Architecture 2-35
Example Implementations 2-35
Advantages and Disadvantages 2-36
Technical Challenges 2-36

Part II Using TopLink Development Tools

3 Understanding TopLink Development Tools

Development Environment 3-2
TopLink Run-Time Environment 3-2

4 Using TopLink Workbench

Understanding TopLink Workbench 4-1
Configuring the TopLink Workbench Environment 4-2
Working With TopLink Workbench 4-3
Using the Menus 4-5
Menu Bar Menus 4-5

Context Menus 4-5
Using the Toolbars 4-6
Standard Toolbar 4-6
Context Toolbar 4-7
Using the Navigator 4-9
Using the Editor 4-10
Using the Problems Window 4-11
Using the Online Help 4-12
Working With TopLink Workbench Preferences 4-12
General Preferences 4-13
Help Preferences 4-14
Mappings Preferences 4-15
Class Preferences 4-16
EJB Preferences 4-16
Database Preferences 4-17
Sessions Configuration Preferences 4-18
New Names Preferences 4-19
Platform Preferences 4-20
Working With Databases 4-21
Working With Database Tables in the Navigator Window 4-21
Logging In and Out of a Database 4-21
Creating New Tables 4-22
Importing Tables From a Database 4-22
Removing Tables 4-24
Renaming Tables 4-24
Refreshing Tables From the Database 4-24
Working With Database Tables in the Editor Window 4-25
Working With Column Properties 4-25
Setting a Primary Key for Database Tables 4-26
Creating Table References 4-26
Creating Field Associations 4-28
Generating Data From Database Tables 4-29
Generating SQL Creation Scripts 4-29
Generating Classes and Descriptors From Database Tables 4-30
Generating EJB Entity Beans and Descriptors From Database Tables 4-31
Generating Tables on the Database 4-32
Working With XML Schemas 4-33
Working With XML Schemas in the Navigator 4-33
Working With XML Schema Structure 4-34
Importing an XML Schema 4-34
Configuring XML Schema Reference 4-36
Using TopLink Workbench 4-36
Using Java 4-37
Configuring XML Schema Namespace 4-37
Using TopLink Workbench 4-38
Using Java 4-39
Working With Classes 4-40

Creating Classes 4-40
Using TopLink Workbench 4-40
Configuring Classes 4-41
Configuring Class Information 4-41
Configuring Class Modifiers 4-42
Configuring Class Interfaces 4-43
Adding Attributes 4-44
Configuring Attribute Modifiers 4-44
Configuring Attribute Type Information 4-45
Configuring Attribute Accessing Methods 4-46
Adding Methods 4-47
Configuring Method Modifiers 4-48
Configuring Method Type Information 4-48
Configuring Method Parameters 4-49
Importing and Updating Classes 4-50
Using TopLink Workbench 4-50
Managing Nondescriptor Classes 4-51
Renaming Packages 4-52
Using TopLink Workbench 4-52
Integrating TopLink Workbench With Apache Ant 4-53
Configuring Ant to Use TopLink Workbench Tasks 4-53
Library Dependencies 4-53
Declaring TopLink Workbench Tasks 4-54
Understanding TopLink Workbench Ant Task API 4-54
Creating TopLink Workbench Ant Tasks 4-55
mappings.validate 4-56
Parameters 4-57
Parameters Specified as Nested Elements 4-57
Examples 4-57
session.validate 4-57
Parameters 4-58
Parameters Specified as Nested Elements 4-58
Examples 4-58
mappings.export 4-58
Parameters 4-58
Parameters Specified as Nested Elements 4-59
Examples 4-59
classpath 4-59
Parameters 4-60
Parameters Specified as Nested Elements 4-60
Examples 4-60
ignoreerror 4-60
Parameters 4-60
Parameters Specified as Nested Elements 4-60
Examples 4-61
ignoreerrorset 4-61
Parameters 4-61

Parameters Specified as Nested Elements 4-61
Examples 4-61
loginspec 4-62
Parameters 4-62
Parameters Specified as Nested Elements 4-62
Examples 4-62

5 Using the Schema Manager

Understanding the Schema Manager 5-1
Schema Manager Java and Database Type Conversion 5-3
Sequencing 5-3
Creating a Table Creator 5-4
Using TopLink Workbench During Development 5-4
Using the Default Table Generator at Run Time 5-4
Using Java 5-4
Creating a TableCreator Class 5-5
Creating a TableDefinition Class 5-5
Adding Fields to a TableDefinition 5-5
Defining Sybase and Microsoft SQL Server Native Sequencing 5-6
Creating Tables With a Table Creator 5-6
Automatic Database Table Creation 5-6

6 Using an Integrated Development Environment

Configuring TopLink for Oracle JDeveloper 6-1
Using TopLink Mappings 6-1
Using TopLink Sessions 6-3
Configuring TopLink Workbench With Source Control Management Software 6-3
Using a Source Control Management System 6-3
Merging Files 6-4
Merging Project Files 6-4
Merging Table, Descriptor, and Class Files 6-5
Sharing Project Objects 6-6
Managing the ejb-jar.xml File 6-6
Working With Locked Files 6-6

Part III Deploying a TopLink Application

7 Integrating TopLink With an Application Server

Application Server Support 7-1
Application Server Integration Concepts 7-2
Software Requirements 7-2
XML Parser Platform Configuration 7-2
Configuring XML Parser Platform 7-3
Creating an XML Parser Platform 7-3
XML Parser Limitations 7-3
Security Permissions 7-4

Persistence Manager Migration ... 7-4
Clustering ... 7-4
Oracle Containers for J2EE (OC4J) ... 7-5
CMP Integration ... 7-5
Migrating OC4J Orion Persistence to OC4J TopLink Persistence ... 7-5
Overview ... 7-6
Using the TopLink Migration Tool From TopLink Workbench ... 7-9
Using the TopLink Migration Tool From the Command Line ... 7-10
Post-Migration Changes ... 7-12
Troubleshooting Your Migration ... 7-13
JTA Integration ... 7-14
BEA WebLogic Server ... 7-14
Classpath ... 7-15
CMP Integration ... 7-15
Migrating BEA WebLogic Persistence to OC4J TopLink Persistence ... 7-16
Overview ... 7-16
Using the TopLink Migration Tool From TopLink Workbench ... 7-18
Using the TopLink Migration Tool From the Command Line ... 7-18
JTA Integration ... 7-20
Security Manager ... 7-20
IBM WebSphere Application Server ... 7-21
Classpath ... 7-21
Configuring Classpath for IBM WebSphere Application Server 4.0 ... 7-21
Configuring Classpath for IBM WebSphere Application Server 5.0 and Later ... 7-21
CMP Integration ... 7-22
JTA Integration ... 7-22
Clustering on IBM WebSphere Application Server ... 7-22
Understanding Security Permissions ... 7-22
Permissions Required by TopLink Features ... 7-23
System Properties ... 7-23
Loading project.xml or sessions.xml Files ... 7-23
Cache Coordination ... 7-23
Accessing a Data Source by Port ... 7-24
Logging With java.util.logging ... 7-24
J2EE Application Deployment ... 7-24
Permissions Required When doPrivileged is Disabled ... 7-24
Disabling doPrivileged Operation ... 7-25
Configuring Miscellaneous EJB Options ... 7-25
Setter Parameter Type Checking ... 7-25
Unknown Primary Key Class Support ... 7-25
Single-Object Finder Return Type Checking ... 7-26

8 Creating TopLink Files for Deployment

Understanding TopLink Deployment File Creation ... 8-1
project.xml File ... 8-2
XSD File Format ... 8-2
Non-CMP Applications and Project Metadata ... 8-2

CMP Applications and Project Metadata ... 8-3
Creating project.xml With TopLink Workbench ... 8-3
Creating project.xml Programatically ... 8-3
sessions.xml File ... 8-4
XSD File Format ... 8-4
Non-CMP Applications and Session Metadata ... 8-4
CMP Applications and Session Metadata ... 8-4
ejb-jar.xml File ... 8-5
<J2EE-Container>-ejb-jar.xml File ... 8-5
OC4J and the orion-ejb-jar.xml File ... 8-6
BEA WebLogic Server and the weblogic-ejb-jar.xml File ... 8-6
toplink-ejb-jar.xml File ... 8-6
OC4J and the toplink-ejb-jar.xml File ... 8-7
BEA WebLogic Server and the toplink-ejb-jar.xml File ... 8-7
IBM WebSphere Application Server and the toplink-ejb-jar.xml File ... 8-8
Java Applications ... 8-8
JavaServer Pages and Servlet Applications ... 8-8
Session Bean Applications ... 8-9
CMP Applications ... 8-9
BMP Applications ... 8-9
Configuring the orion-ejb-jar.xml File for OC4J ... 8-9
Configuring persistence-manager Entries ... 8-10
Configuring pm-properties ... 8-10
Configuring cache-synchronization Properties ... 8-11
Configuring default-mapping Properties ... 8-12
Configuring the weblogic-ejb-jar.xml File for BEA WebLogic Server ... 8-13
Configuring persistence-descriptor Entries ... 8-14
Unsupported weblogic-ejb-jar.xml File Tags ... 8-15

9 Packaging a TopLink Application

Java Applications ... 9-1
JavaServer Pages and Servlet Applications ... 9-2
TopLink Domain JAR ... 9-2
Session Bean Applications ... 9-3
TopLink Domain JAR ... 9-3
EJB JAR ... 9-4
CMP Applications ... 9-4
EJB JAR ... 9-5
BMP Applications ... 9-5
TopLink Domain JAR ... 9-6
EJB JAR ... 9-7
Packaging With TopLink Metadata File Resource Paths ... 9-7

10 Deploying a TopLink Application

Java Applications ... 10-1
JavaServer Pages and Servlets ... 10-1
Session Bean Applications ... 10-1

CMP Applications 10-2
Deploying a CMP Application to OC4J 10-2
Deploying a CMP Application to BEA WebLogic Server 10-2
Troubleshooting ejbc 10-3
Deploying a CMP Application to IBM WebSphere Application Server 4.0 10-3
Starting the Entity Bean 10-3
BMP Applications 10-4
Hot Deployment of EJB 10-4
Hot Deployment in a CMP Application 10-4
Hot Deployment in a non-CMP Application 10-4
Using the WebSphere Deploy Tool 10-5
Using the Deploy Tool on its Own 10-5
Using the Deploy Tool With WebSphere Studio Application Developer 10-5
Troubleshooting 10-6

Part IV Optimizing and Customizing a TopLink Application

11 Optimization

Understanding Optimization 11-1
Sources of Application Performance Problems 11-2
Measuring TopLink Performance With the TopLink Profiler 11-2
Configuring the TopLink Performance Profiler 11-3
Accessing the TopLink Profiler Results 11-3
Measuring TopLink Performance With the Oracle Dynamic Monitoring System (DMS) 11-4
Configuring the Oracle DMS Profiler 11-6
OC4J Applications 11-7
Non-OC4J Applications 11-7
Accessing Oracle DMS Profiler Data Using JMX 11-7
Accessing Oracle DMS Profiler Data Using the DMS Spy Servlet 11-7
General Performance Optimization 11-8
Schema Optimization 11-8
Schema Case 1: Aggregation of Two Tables into One 11-9
Schema Case 2: Splitting One Table Into Many 11-9
Schema Case 3: Collapsed Hierarchy 11-11
Schema Case 4: Choosing One out of Many 11-12
Mapping and Descriptor Optimization 11-13
Session Optimization 11-13
Cache Optimization 11-13
Data Access Optimization 11-14
JDBC Driver Properties Optimization 11-14
Data Format Optimization 11-15
Batch Writing 11-15
Parameterized SQL (Binding) and Prepared Statement Caching 11-15
Query Optimization 11-16
Parameterized SQL and Prepared Statement Caching 11-17
Named Queries 11-17

Batch and Join Reading 11-17
Partial Object Queries and Fetch Groups 11-17
JDBC Fetch Size 11-18
Cursored Streams and Scrollable Cursors 11-19
Read Optimization Examples 11-19
Reading Case 1: Displaying Names in a List 11-20
Reading Case 2: Batch Reading Objects 11-23
Reading Case 3: Using Complex Custom SQL Queries 11-24
Reading Case 4: Using View Objects 11-25
Reading Case 5: Inheritance Views 11-26
Write Optimization Examples 11-26
Writing Case: Batch Writes 11-27
Unit of Work Optimization 11-30
Application Server and Database Optimization 11-30

12 Customization

Overview 12-1
Creating Custom Data Types 12-1
Using Public Source 12-2

Part V Troubleshooting a TopLink Application

13 TopLink Exception Reference

Descriptor Exceptions (1 – 201) 13-2
Concurrency Exceptions (2001 – 2009) 13-25
Conversion Exceptions (3001– 3008) 13-26
Database Exceptions (4002 – 4018) 13-27
Optimistic Lock Exceptions (5001 – 5009) 13-29
Query Exceptions (6001 – 6129) 13-31
Validation Exceptions (7001 – 7200) 13-43
EJB QL Exceptions (8001 – 8010) 13-63
Session Loader Exceptions (9000 - 9010) 13-64
Communication Exceptions (12000 - 12003) 13-65
EIS Exceptions (17007 – 17025), 90000, 91000 13-66
JMS Processing Exceptions (18001 - 18004) 13-69
Default Mapping Exceptions (20001 - 20008) 13-69
Discovery Exceptions (22001 - 22004) 13-70
Remote Command Manager Exceptions (22101 - 22111) 13-71
Transaction Exceptions (23001 - 23015) 13-73
XML Conversion Exceptions (25501) 13-74
Migration Utility Exceptions (26001 - 26020) 13-74
EJB JAR XML Exceptions (72000 – 72023) 13-77
Entity Manager Setup Exceptions (28001 – 28007) 13-80
XML Marshal Exceptions (25001 – 25020) 13-81
XML Platform Exceptions (27001 – 27006, 27101 – 27103, 27201 – 27202) 13-83

14 TopLink Workbench Error Reference

Miscellaneous Errors (1 – 89, 106 – 133) 14-1
Project Errors (100 – 102) 14-3
Descriptor Errors (200 – 399) 14-3
Mapping Errors (400 – 483) 14-13
Table Errors (500 – 610) 14-17
XML Schema Errors (700 – 706) 14-23
Session Errors (800 – 812) 14-23
Common Classpath Problems 14-25
Data Source Problems 14-25
Database Connection Problems 14-25

15 Troubleshooting Application Deployment

Generating Deployment JAR Files 15-1
Common J2SE Deployment Exceptions 15-1
Classpath Exceptions 15-2
Communication Exceptions 15-2
Descriptor Validation Exceptions 15-2
Common BEA WebLogic Server Deployment Exceptions 15-3
Common BEA WebLogic Server 6.1 Exceptions 15-5
Development Exceptions 15-6
Deployment and Run-Time Exceptions 15-6
Common BEA WebLogic 7.0 Exceptions 15-8
Development Exceptions 15-8
Deployment Exceptions 15-9
Common BEA WebLogic 8.1 Exceptions 15-10
Development Exceptions 15-10
Deployment Exceptions 15-11
Common IBM WebSphere Application Server Exceptions 15-12
Problems at Run Time 15-14
Common TopLink for IBM WebSphere Deploy Tool Exceptions 15-15

Part VI Mapping and Configuration Overview

16 Understanding TopLink Mapping and Configuration Concepts

Mapping and Configuration Concepts 16-1
Projects 16-1
Descriptors 16-2
Mappings 16-2

Part VII Projects

17 Understanding Projects

TopLink Project Types 17-1
Project Concepts 17-2

Project Architecture 17-2
Relational and Nonrelational Projects 17-2
Persistent and Nonpersistent Projects 17-2
Projects and Login 17-3
Non-CMP Session Role: Session Login 17-3
CMP Deployment Role: Deployment Login 17-3
Development Role: Development Login 17-4
Projects and Platforms 17-4
Projects and Sequencing 17-4
Configuring how to Obtain Sequence Values 17-5
Configuring Where to Write Sequence Values 17-5
XML Namespaces 17-5
Relational Projects 17-6
Building Relational Projects for a Relational Database 17-6
Building Relational Projects for an Object-Relational Database 17-6
EIS Projects 17-7
Building EIS Projects With XML Records 17-9
Building EIS Projects With Indexed or Mapped Records 17-9
XML Projects 17-9
TopLink Support for Java Architecture for XML Binding (JAXB) 17-10
Understanding JAXB-Specific Generated Files 17-10
Understanding TopLink-Specific Generated Files 17-11
Using TopLink JAXB Compiler Generated Files at Run Time 17-13
JAXB Validation 17-14
Understanding the Project API 17-14
Project Inheritance Hierarchy 17-14
Understanding Sequencing in Relational Projects 17-14
Sequencing Configuration Options 17-15
Sequencing Types 17-16
Table Sequencing 17-16
Unary Table Sequencing 17-17
Query Sequencing 17-18
Default Sequencing 17-18
Native Sequencing With an Oracle Database Platform 17-18
Native Sequencing With a Non-Oracle Database Platform 17-19
Sequencing and Preallocation Size 17-20
Sequencing With Entity Beans WIth Container-Managed Persistence 17-21
Understanding XML Namespaces 17-22
TopLink Workbench Namespace Resolution 17-22
Element and Attribute Form Options 17-23
Element Form Default Qualified and Attribute Form Default Unqualified 17-23
Element and Attribute Form Default Unqualified 17-24
Element and Attribute Form Default Qualified 17-25
TopLink Runtime Namespace Resolution 17-26

18 Creating a Project

Project Creation Overview 18-1

Using TopLink Workbench 18-2
Creating New TopLink Workbench Projects 18-2
Using Java 18-3
Creating a Project for an Existing Object and Data Model 18-5
Using TopLink Workbench 18-5
Creating a Project From an Existing Object Model 18-5
Using TopLink Workbench 18-5
Creating a Project From an Existing Data Model 18-5
Using TopLink Workbench 18-6
Creating an XML Project From an XML Schema 18-6
Using TopLink Workbench 18-6
Using the Command Line 18-8
Creating a Project by Migrating an EAR to OC4J 18-9
Creating a Project From an OC4J EJB CMP EAR at Deployment Time 18-10
Working With Projects 18-10
Opening Existing Projects 18-10
Saving Projects 18-11
Saving Projects With a New Name or Location 18-12
Generating the Project Status Report 18-12
Exporting Project Information 18-13
Exporting Deployment XML Information 18-14
Exporting Model Java Source 18-14
Exporting Project Java Source 18-14
Exporting Table Creator Files 18-15
Working With the ejb-jar.xml File 18-15
Writing to the ejb-jar.xml File 18-16
Reading From the ejb-jar.xml File 18-16

19 Configuring a Project

Configuring Common Project Options 19-1
Configuring Project Save Location 19-2
Using TopLink Workbench 19-2
Configuring Project Classpath 19-3
Using TopLink Workbench 19-3
Configuring Mapped Field Access at the Project Level 19-4
Using TopLink Workbench 19-4
Configuring Persistence Type 19-5
Using TopLink Workbench 19-6
Configuring Default Descriptor Advanced Properties 19-7
Using TopLink Workbench 19-8
Configuring Existence Checking at the Project Level 19-8
Using TopLink Workbench 19-9
Configuring Project Deployment XML Options 19-10
Using TopLink Workbench 19-10
Configuring Model Java Source Code Options 19-11
Using TopLink Workbench 19-11
Configuring Deprecated Direct Mappings 19-12

Using TopLink Workbench 19-13
Configuring Cache Type and Size at the Project Level 19-13
Using TopLink Workbench 19-14
Configuring Cache Isolation at the Project Level 19-16
Using TopLink Workbench 19-16
Configuring Cache Coordination Change Propagation at the Project Level 19-17
Using TopLink Workbench 19-18
Configuring Cache Expiration at the Project Level 19-19
Using TopLink Workbench 19-20
Configuring Project Comments 19-21
Using TopLink Workbench 19-21

20 Configuring a Relational Project

Relational Project Configuration Overview 20-1
Configuring Relational Database Platform at the Project Level 20-2
Using TopLink Workbench 20-2
Configuring Sequencing at the Project Level 20-3
Using TopLink Workbench 20-3
Using Java 20-4
Configuring Login Information 20-5
Using TopLink Workbench 20-5
Configuring Development and Deployment Logins 20-6
Using TopLink Workbench 20-6
Logging in to the Database 20-7
Configuring Named Query Parameterized SQL and Statement Caching at the Project Level 20-7
Using TopLink Workbench 20-8
Configuring Table Generation Options 20-9
Using TopLink Workbench 20-9
Configuring Table Creator Java Source Options 20-10
Using TopLink Workbench 20-10
Configuring Project Java Source Code Options 20-11
Using TopLink Workbench 20-11

21 Configuring an EIS Project

EIS Project Configuration Overview 21-1
Configuring EIS Data Source Platform at the Project Level 21-2
Using TopLink Workbench 21-2
Configuring EIS Connection Specification Options at the Project Level 21-2
Using TopLink Workbench 21-3

22 Configuring an XML Project

XML Project Configuration Overview 22-1

Part VIII Descriptors

23 Understanding Descriptors

Descriptor Types 23-1
Descriptor Concepts 23-2
Descriptor Architecture 23-2
Descriptors and Inheritance 23-3
Descriptors and EJB 23-3
Nondeferred Changes 23-3
Creating a New Entity Bean and ejbCreate / ejbPostCreate Methods 23-4
Inheritance 23-4
Fetch Groups 23-5
Amendment and After-Load Methods 23-5
Descriptors and Aggregation 23-5
Aggregate and Composite Descriptors in Relational Projects 23-5
Root and Composite Descriptors in EIS Projects 23-8
Composite Descriptors in XML Projects 23-8
Descriptor Event Manager 23-8
Descriptor Query Manager 23-8
Descriptors and Sequencing 23-9
Descriptors and Locking 23-9
Default Root Element 23-9
Relational Descriptors 23-11
Object-Relational Descriptors 23-11
EIS Descriptors 23-12
XML Descriptors 23-12
Understanding Descriptors and Inheritance 23-12
Specifying a Class Indicator 23-14
Using Class Indicator Fields 23-14
Using Class Extraction Methods 23-15
Inheritance and Primary Keys (Relational and EIS Only) 23-16
Single and Multi-Table Inheritance (Relational Only) 23-16
Single Table Inheritance 23-16
Multitable Inheritance 23-16
Aggregate and Composite Descriptors and Inheritance 23-17
Inheritance and EJB 23-17
Understanding Descriptors and Locking 23-18
Optimistic Version Locking Policies 23-18
Optimistic Version Locking Policies and Cascading 23-19
Optimistic Locking and Rollbacks 23-20
Optimistic Field Locking Policies 23-20
Pessimistic Locking Policy 23-22
Locking in a Three-Tier Application 23-22
Optimistic Locking in a Three-Tier Application 23-22
Pessimistic Locking in a Three-Tier Application 23-23
Understanding the Descriptor API 23-23
Descriptor Inheritance Hierarchy 23-23

24 Creating a Descriptor

Descriptor Creation Overview 24-1
Creating a Relational Descriptor 24-1
Using TopLink Workbench 24-2
Relational Class Descriptors 24-2
Relational Aggregate Descriptors 24-2
Relational Interface Descriptors 24-2
Using Java 24-2
Creating an Object-Relational Descriptor 24-3
Using Java 24-3
Creating an EIS Descriptor 24-4
Using TopLink Workbench 24-4
EIS Root Descriptors 24-5
EIS Composite Descriptors 24-5
Using Java 24-5
Creating an XML Descriptor 24-5
Using TopLink Workbench 24-5
Using Java 24-5
Validating Descriptors 24-6
Generating Java Code for Descriptors 24-6

25 Configuring a Descriptor

Configuring Common Descriptor Options 25-1
Configuring Primary Keys 25-3
Using TopLink Workbench 25-3
Using Java 25-4
Relational Projects 25-4
EIS Projects 25-5
Configuring Read-Only Descriptors 25-5
Using Read-Only Entity Beans 25-5
Using TopLink Workbench 25-5
Using Java 25-6
Configuring Unit of Work Conforming at the Descriptor Level 25-6
Using TopLink Workbench 25-7
Using Java 25-7
Configuring Descriptor Alias 25-7
Using TopLink Workbench 25-8
Using Java 25-9
Configuring Descriptor Comments 25-9
Using TopLink Workbench 25-9
Configuring Named Queries at the Descriptor Level 25-10
Using TopLink Workbench 25-10
Adding Named Queries 25-12
Configuring Named Query Type and Parameters 25-13
Configuring Named Query Selection Criteria 25-14
Configuring Read All Query Order 25-15
Configuring Named Query Optimization 25-16

Configuring Named Query Attributes 25-17
Configuring Named Query Group/Order Options 25-19
Creating an EIS Interaction for a Named Query 25-20
Configuring Named Query Options 25-22
Configuring Named Query Advanced Options 25-24
Using Java 25-25
Configuring Query Timeout at the Descriptor Level 25-26
Using TopLink Workbench 25-26
Using Java 25-27
Configuring Cache Refreshing 25-27
Using TopLink Workbench 25-28
Using Java 25-29
Configuring Query Keys 25-30
Using TopLink Workbench 25-31
Using Java 25-31
Configuring Interface Query Keys 25-33
Using TopLink Workbench 25-34
Using Java 25-34
Configuring Cache Type and Size at the Descriptor Level 25-35
Using TopLink Workbench 25-36
Using Java 25-37
Configuring Cache Isolation at the Descriptor Level 25-37
Using TopLink Workbench 25-38
Using Java 25-38
Configuring Unit of Work Cache Isolation at the Descriptor Level 25-38
Using Java 25-39
Configuring Cache Coordination Change Propagation at the Descriptor Level 25-40
Using TopLink Workbench 25-40
Using Java 25-41
Configuring Cache Expiration at the Descriptor Level 25-42
Using TopLink Workbench 25-42
Using Java 25-43
Configuring Cache Existence Checking at the Descriptor Level 25-43
Using TopLink Workbench 25-44
Using Java 25-45
Configuring a Descriptor With EJB Information 25-45
Using TopLink Workbench 25-46
Using Java 25-48
Configuring CMP Information 25-48
Configuring BMP Information 25-49
Configuring Reading Subclasses on Queries 25-49
Using TopLink Workbench 25-49
Using Java 25-50
Configuring Inheritance for a Child (Branch or Leaf) Class Descriptor 25-51
Using TopLink Workbench 25-51
Using Java 25-52
Configuring Inheritance for a Parent (Root) Descriptor 25-52

Using TopLink Workbench 25-53
Using Java 25-54
Configuring Inheritance Expressions for a Parent (Root) Class Descriptor 25-55
Using Java 25-57
Configuring Inherited Attribute Mapping in a Subclass 25-58
Using TopLink Workbench 25-58
Using Java 25-59
Configuring a Domain Object Method as an Event Handler 25-59
Using TopLink Workbench 25-60
Using Java 25-62
Configuring a Descriptor Event Listener as an Event Handler 25-62
Using Java 25-64
Configuring Locking Policy 25-64
Using TopLink Workbench 25-65
Using Java 25-67
Configuring an Optimistic Locking Policy 25-67
Configuring Optimistic Locking Policy Cascading 25-67
Configuring a Pessimistic Locking Policy 25-67
Configuring Returning Policy 25-67
Using TopLink Workbench 25-68
Using Java 25-69
Configuring Instantiation Policy 25-70
Using TopLink Workbench 25-70
Using Java 25-71
Configuring Copy Policy 25-71
Using TopLink Workbench 25-72
Using Java 25-72
Configuring Change Policy 25-73
Using Java 25-73
Configuring Deferred Change Detection Policy 25-73
Configuring Object Change Tracking Policy 25-73
Configuring Attribute Change Tracking Policy 25-75
Configuring a History Policy 25-76
Using Java 25-77
Configuring Write Responsibility 25-77
Configuring Wrapper Policy 25-78
Using Java 25-79
Configuring Fetch Groups 25-79
Using Java 25-80
Configuring Amendment Methods 25-81
Using TopLink Workbench 25-81

26 Configuring a Relational Descriptor

Relational Descriptor Configuration Overview 26-1
Configuring Associated Tables 26-2
Using TopLink Workbench 26-2
Using Java 26-3

Configuring Sequencing at the Descriptor Level ... 26-3
Using TopLink Workbench ... 26-4
Using Java ... 26-5
Configuring a Sequence by Name ... 26-5
Configuring the Same Sequence for Multiple Descriptors ... 26-5
Configuring the Platform Default Sequence ... 26-6
Configuring Custom SQL Queries for Basic Persistence Operations ... 26-6
Using TopLink Workbench ... 26-7
Using Java ... 26-8
Configuring Interface Alias ... 26-10
Using TopLink Workbench ... 26-11
Using Java ... 26-11
Configuring a Relational Descriptor as a Class or Aggregate Type ... 26-11
Using TopLink Workbench ... 26-12
Using Java ... 26-12
Configuring Multitable Information ... 26-13
Using TopLink Workbench ... 26-13
Using Java ... 26-14

27 Configuring an Object-Relational Descriptor

Object-Relational Descriptor Configuration Overview ... 27-1
Configuring Field Ordering ... 27-2
Using Java ... 27-2

28 Configuring an EIS Descriptor

EIS Descriptor Configuration Overview ... 28-1
Configuring Schema Context for an EIS Descriptor ... 28-2
Using TopLink Workbench ... 28-2
Choosing a Schema Context ... 28-3
Using Java ... 28-3
Configuring Default Root Element ... 28-3
Using TopLink Workbench ... 28-4
Choosing a Root Element ... 28-4
Using Java ... 28-5
Configuring Record Format ... 28-5
Using Java ... 28-6
Configuring Custom EIS Interactions for Basic Persistence Operations ... 28-6
Using TopLink Workbench ... 28-6
Using Java ... 28-8
Configuring an EIS Descriptor as a Root or Composite Type ... 28-8
Using TopLink Workbench ... 28-9
Using Java ... 28-9

29 Configuring an XML Descriptor

XML Descriptor Configuration Overview ... 29-1
Configuring Schema Context for an XML Descriptor ... 29-1

Using TopLink Workbench ... 29-2
Choosing a Schema Context ... 29-2
Using Java ... 29-3
Configuring for Complex Type of anyType ... 29-3
Using TopLink Workbench ... 29-4
Configuring Default Root Element ... 29-5
Using TopLink Workbench ... 29-5
Choosing a Root Element ... 29-5
Configuring Document Preservation ... 29-6
Using TopLink Workbench ... 29-6
Using Java ... 29-6

Part IX Mappings

30 Understanding Mappings

Mapping Types ... 30-1
Mapping Concepts ... 30-2
Mapping Architecture ... 30-2
Example Mapping ... 30-3
Automatic Mappings ... 30-4
Automapping With TopLink Workbench at Development Time ... 30-4
Default Mapping in CMP Projects Using OC4J at Run Time ... 30-4
JAXB Project Generation at Development Time ... 30-5
Indirection ... 30-5
Value Holder Indirection ... 30-7
Transparent Indirect Container Indirection ... 30-8
Proxy Indirection ... 30-8
Indirection and EJB ... 30-9
Indirection, Serialization, and Detachment ... 30-9
Method Accessors and Attribute Accessors ... 30-10
Mapping Converters and Transformers ... 30-10
Serialized Object Converter ... 30-10
Type Conversion Converter ... 30-11
Object Type Converter ... 30-12
Simple Type Translator ... 30-12
Transformation Mappings ... 30-14
Mappings and XPath ... 30-15
XPath by Position ... 30-15
XPath by Path and Name ... 30-15
XPath by Name ... 30-16
Self XPath ... 30-16
Mappings and xsd:list and xsd:union Types ... 30-17
Mapping an xsd:union Type ... 30-17
Mapping an xsd:list Type ... 30-18
Mapping a List of Unions ... 30-18
Mapping a Union of Lists ... 30-19
Mapping a Union of Unions ... 30-19

Mappings and the jaxb:class Customization ... 30-20
all, choice, or sequence Structure ... 30-20
group Structure ... 30-21
sequence or choice Structure Containing a group ... 30-21
group Structure Containing a sequence or choice ... 30-22
group Structure Containing a group ... 30-23
Limitations of jaxb:class Customization Support ... 30-23
Mappings and JAXB Typesafe Enumerations ... 30-24
Understanding the Mapping API ... 30-25
Relational Mappings ... 30-25
Object-Relational Mappings ... 30-26
XML Mappings ... 30-27
EIS Mappings ... 30-27

31 Creating a Mapping

Mapping Creation Overview ... 31-1
Creating Mappings Manually During Development ... 31-1
Using TopLink Workbench ... 31-1
Creating Mappings Automatically During Development ... 31-2
Using TopLink Workbench ... 31-2
Creating Mappings Automatically During Deployment ... 31-2
Creating Mappings to Oracle LOB Database Objects ... 31-2
Using the Oracle JDBC OCI Driver or Server Driver ... 31-3
Using the Oracle JDBC Thin Driver ... 31-5
Removing Mappings ... 31-6
Using TopLink Workbench ... 31-6

32 Configuring a Mapping

Configuring Common Mapping Options ... 32-1
Configuring Read-Only Mappings ... 32-2
Using TopLink Workbench ... 32-3
Using Java ... 32-3
Configuring Indirection ... 32-3
Using TopLink Workbench ... 32-4
Using Java ... 32-5
Configuring ValueHolder Indirection ... 32-6
Configuring ValueHolder Indirection With Method Accessing ... 32-7
Configuring ValueHolder Indirection With EJB 3.0 on OC4J ... 32-8
Configuring IndirectContainer Indirection ... 32-8
Configuring Proxy Indirection ... 32-9
Configuring XPath ... 32-10
Using TopLink Workbench ... 32-11
Choosing the XPath ... 32-12
Configuring a Default Null Value at the Mapping Level ... 32-12
Using TopLink Workbench ... 32-13
Using Java ... 32-13

Configuring Method Accessing 32-14
Using TopLink Workbench 32-14
Using Java 32-15
Configuring Private or Independent Relationships 32-16
Using TopLink Workbench 32-17
Using Java 32-17
Configuring Mapping Comments 32-18
Using TopLink Workbench 32-18
Configuring a Serialized Object Converter 32-18
Using TopLink Workbench 32-19
Using Java 32-19
Configuring a Type Conversion Converter 32-20
Using TopLink Workbench 32-20
Using Java 32-21
Configuring an Object Type Converter 32-22
Using TopLink Workbench 32-22
Using Java 32-23
Configuring a Simple Type Translator 32-23
Using TopLink Workbench 32-24
Using Java 32-24
Configuring a JAXB Typesafe Enumeration Converter 32-25
Using Java 32-26
Configuring Container Policy 32-26
Using TopLink Workbench 32-27
Using Java 32-28
Configuring Attribute Transformer 32-29
Using TopLink Workbench 32-30
Using Java 32-31
Configuring Field Transformer Associations 32-31
Using TopLink Workbench 32-32
Specifying Field-to-Transformer Associations 32-32
Using Java 32-33
Configuring Mutable Mappings 32-33
Using TopLink Workbench 32-34
Using Java 32-34
Configuring Bidirectional Relationship 32-34
Using TopLink Workbench 32-35
Configuring the Use of a Single Node 32-36
Using TopLink Workbench 32-36
Using Java 32-37

Part X Relational Mappings

33 Understanding Relational Mappings

Relational Mapping Types 33-1
Relational Mapping Concepts 33-2
Directionality 33-2

Converters and Transformers ... 33-3
Using a Direct Mapping ... 33-3
Using a Converter Mapping ... 33-3
Using a Transformation Mapping ... 33-3
Relational Mappings and EJB ... 33-3
Direct-to-Field Mapping ... 33-4
Direct-to-XMLType Mapping ... 33-4
One-to-One Mapping ... 33-5
One-to-One Mappings and EJB ... 33-6
Variable One-to-One Mapping ... 33-6
One-to-Many Mapping ... 33-7
One-to-Many Mappings and EJB ... 33-8
Many-to-Many Mapping ... 33-8
Many-to-Many Mappings and EJB ... 33-9
Aggregate Collection Mapping ... 33-10
Aggregate Collection Mappings and Inheritance ... 33-10
Aggregate Collection Mappings and EJB ... 33-11
Implementing Aggregate Collection Mappings ... 33-11
Direct Collection Mapping ... 33-11
Direct Map Mapping ... 33-12
Aggregate Object Mapping ... 33-12
Aggregate Object Mappings with a Single Source Object ... 33-13
Aggregate Object Mappings With Multiple Source Objects ... 33-14
Implementing an Aggregate Object Relationship Mapping ... 33-14
Transformation Mapping ... 33-15

34 Configuring a Relational Mapping

Configuring Common Relational Mapping Options ... 34-1
Configuring a Database Field ... 34-2
Using TopLink Workbench ... 34-4
Configuring Reference Descriptor ... 34-5
Using TopLink Workbench ... 34-5
Configuring Batch Reading ... 34-6
Using TopLink Workbench ... 34-7
Using Java ... 34-7
Configuring Query Key Order ... 34-8
Using TopLink Workbench ... 34-8
Configuring Table and Field References (Foreign and Target Foreign Keys) ... 34-8
Using TopLink Workbench ... 34-9

35 Configuring a Relational Direct-to-Field Mapping

Relational Direct-to-Field Mapping Configuration Overview ... 35-1

36 Configuring a Relational Direct-to-XMLType Mapping

Relational Direct-to-XMLType Mapping Overview ... 36-1
Configuring Read Whole Document ... 36-1

Using TopLink Workbench 36-1

37 Configuring a Relational One-to-One Mapping

Relational One-to-One Mapping Configuration Overview 37-1
Configuring Joining at the Mapping Level 37-1
Using TopLink Workbench 37-2

38 Configuring a Relational Variable One-to-One Mapping

Relational Variable One-to-One Mapping Configuration Overview 38-1
Configuring Class Indicator 38-1
Using TopLink Workbench 38-2
Configuring Unique Primary Key 38-3
Understanding Unique Primary Key 38-3
Using TopLink Workbench 38-3
Using Java 38-4
Configuring Query Key Association 38-4
Using TopLink Workbench 38-4

39 Configuring a Relational One-to-Many Mapping

Relational One-to-Many Mapping Configuration Overview 39-1

40 Configuring a Relational Many-to-Many Mapping

Relational Many-to-Many Mapping Configuration Overview 40-1
Configuring a Relation Table 40-1
Using TopLink Workbench 40-2

41 Configuring a Relational Aggregate Collection Mapping

Relational Aggregate Collection Mapping Configuration Overview 41-1

42 Configuring a Relational Direct Collection Mapping

Relational Direct Collection Mapping Configuration Overview 42-1
Configuring Target Table 42-1
Using TopLink Workbench 42-2
Configuring Direct Value Field 42-2
Using TopLink Workbench 42-2

43 Configuring a Relational Aggregate Object Mapping

Relational Aggregate Object Mapping Configuration Overview 43-1
Configuring Aggregate Fields 43-1
Using TopLink Workbench 43-2
Configuring Allowing Null Values 43-2
Using TopLink Workbench 43-2

44 Configuring a Relational Direct Map Mapping

Relational Direct Map Mapping Configuration Overview 44-1
Configuring Direct Value Field 44-1
Using TopLink Workbench 44-1
Configuring Direct Key Field 44-2
Using TopLink Workbench 44-2
Configuring Key Converters 44-3
Using TopLink Workbench 44-3
Configuring Value Converters 44-4
Using TopLink Workbench 44-4

45 Configuring a Relational Transformation Mapping

Relational Transformation Mapping Configuration Overview 45-1

Part XI Object-Relational Mappings

46 Understanding Object-Relational Mappings

Object-Relational Mapping Types 46-1
Object-Relational Structure Mapping 46-2
Object-Relational Reference Mapping 46-2
Object-Relational Array Mapping 46-2
Object-Relational Object Array Mapping 46-2
Object-Relational Nested Table Mapping 46-3

47 Configuring an Object-Relational Mapping

Configuring Common Object-Relational Mapping Options 47-1
Configuring Reference Class 47-2
Using Java 47-2
Configuring Attribute Name 47-2
Using Java 47-3
Configuring Field Name 47-3
Using Java 47-3
Configuring Structure Name 47-4
Using Java 47-4

48 Configuring an Object-Relational Structure Mapping

Object-Relational Structure Mapping Configuration Overview 48-1

49 Configuring an Object-Relational Reference Mapping

Object-Relational Reference Mapping Configuration Overview 49-1

50 Configuring an Object-Relational Array Mapping

Object-Relational Array Mapping Configuration Overview 50-1

51 Configuring an Object-Relational Object Array Mapping

Object-Relational Object Array Mapping Configuration Overview 51-1

52 Configuring an Object-Relational Nested Table Mapping

Object-Relational Nested Table Mapping Configuration Overview 52-1

Part XII EIS Mappings

53 Understanding EIS Mappings

EIS Mapping Types 53-1
EIS Mapping Concepts 53-2
EIS Record Type 53-2
Indexed Records 53-2
Mapped Records 53-3
XML Records 53-3
XPath Support 53-3
xsd:list and xsd:union Support 53-3
jaxb:class Support 53-3
Typesafe Enumeration Support 53-3
Composite and Reference EIS Mappings 53-4
Composite EIS Mappings 53-4
Reference EIS Mappings 53-4
EIS Mapping Architecture 53-5
EIS Direct Mapping 53-5
EIS Composite Direct Collection Mapping 53-6
EIS Composite Object Mapping 53-7
EIS Composite Collection Mapping 53-7
EIS One-to-One Mapping 53-8
EIS One-to-One Mappings With Key on Source 53-9
EIS One-to-One Mappings With Key on Target 53-10
EIS One-to-Many Mapping 53-12
EIS One-to-Many Mappings With Key on Source 53-13
EIS One-to-Many Mappings With Key on Target 53-15
EIS Transformation Mapping 53-17

54 Configuring an EIS Mapping

Configuring Common EIS Mapping Options 54-1
Configuring Reference Descriptors 54-2
Using TopLink Workbench 54-2
Configuring Selection Interaction 54-3
Using TopLink Workbench 54-4

55 Configuring an EIS Direct Mapping

EIS Direct Mapping Configuration Overview 55-1

56 Configuring an EIS Composite Direct Collection Mapping

EIS Composite Direct Collection Mapping Configuration Overview 56-1

57 Configuring an EIS Composite Object Mapping

EIS Composite Object Mapping Configuration Overview 57-1

58 Configuring an EIS Composite Collection Mapping

EIS Composite Collection Mapping Configuration Overview 58-1

59 Configuring an EIS One-to-One Mapping

EIS One-to-One Mapping Configuration Overview 59-1
Configuring Foreign Key Pairs 59-1
Using TopLink Workbench 59-2

60 Configuring an EIS One-to-Many Mapping

EIS One-to-Many Mapping Configuration Overview 60-1
Configuring Foreign Key Pairs 60-1
Using TopLink Workbench 60-2
Configuring Delete All Interactions 60-3
Using TopLink Workbench 60-4

61 Configuring an EIS Transformation Mapping

EIS Transformation Mapping Configuration Overview 61-1

Part XIII XML Mappings

62 Understanding XML Mappings

XML Mapping Types 62-1
XML Mapping Concepts 62-2
Mapping to Simple and Complex Types 62-2
Mapping Order 62-3
XPath Support 62-3
xsd:list and xsd:union Support 62-3
xs:any and xs:anyType Support 62-4
jaxb:class Support 62-4
Typesafe Enumeration Support 62-4
Mapping Extensions 62-4
XML Direct Mapping 62-5
Mapping to a Text Node 62-5
Mapping to a Simple Text Node 62-5
Mapping to a Text Node in a Simple Sequence 62-6
Mapping to a Text Node in a Subelement 62-6
Mapping to a Text Node by Position 62-7
Mapping to an Attribute 62-8

Mapping to a Specified Schema Type 62-9
Mapping to a List Field With an XML Direct Mapping 62-10
Mapping to a Union Field With an XML Direct Mapping 62-10
Mapping to a Union of Lists With an XML Direct Mapping 62-12
Mapping to a Union of Unions With an XML Direct Mapping 62-12
Mapping With a Simple Type Translator 62-13
XML Composite Direct Collection Mapping 62-14
Mapping to Multiple Text Nodes 62-15
Mapping to a Simple Sequence 62-15
Mapping to a Sequence in a Subelement 62-15
Mapping to Multiple Attributes 62-16
Mapping to a Single Text Node With an XML Composite Direct Collection Mapping 62-17
Mapping to a Single Attribute With an XML Composite Direct Collection Mapping 62-18
Mapping to a List of Unions With an XML Composite Direct Collection Mapping 62-18
Mapping to a Union of Lists With an XML Composite Direct Collection Mapping 62-19
Specifying the Content Type of a Collection With an XML Composite Direct Collection Mapping 62-20
XML Composite Object Mapping 62-21
Mapping Into the Parent Record 62-21
Mapping to an Element 62-22
Mapping to Different Elements by Element Name 62-23
Mapping to Different Elements by Element Position 62-24
XML Composite Collection Mapping 62-25
XML Any Object Mapping 62-27
XML Any Collection Mapping 62-29
XML Transformation Mapping 62-31

63 Configuring an XML Mapping

Configuring Common XML Mapping Options 63-1
Configuring Reference Descriptor 63-2
Using TopLink Workbench 63-2
Configuring Maps to Wildcard 63-3
Using TopLink Workbench 63-3

64 Configuring an XML Direct Mapping

XML Direct Mapping Configuration Overview 64-1

65 Configuring an XML Composite Direct Collection Mapping

XML Composite Direct Collection Mapping Configuration Overview 65-1

66 Configuring an XML Composite Object Mapping

XML Composite Object Mapping Configuration Overview 66-1

67 Configuring an XML Composite Collection Mapping

XML Composite Collection Mapping Configuration Overview 67-1

68 Configuring an XML Any Object Mapping

XML Any Object Mapping Configuration Overview 68-1

69 Configuring an XML Any Collection Mapping

XML Any Collection Mapping Configuration Overview 69-1

70 Configuring an XML Transformation Mapping

XML Transformation Mapping Configuration Overview 70-1

Part XIV Using TopLink Overview

71 Understanding the Persistence Layer

Overview of the Persistence Layer 71-1
Sessions 71-1
Data Access 71-1
Cache 71-2
Queries and Expressions 71-2
Transactions 71-3

Part XV TopLink Sessions

72 Understanding TopLink Sessions

Session Types 72-1
Session Concepts 72-2
- Session Architecture 72-2
  - Object Cache 72-3
  - Connection Pools 72-3
  - Query Mechanism 72-4
  - Java Object Builder 72-4
- Session Configuration and the sessions.xml File 72-4
- Session Customization 72-4
- Acquiring a Session at Run Time With the Session Manager 72-5
- Managing Session Events With the Session Event Manager 72-5
  - Session Event Manager Events 72-6
  - Session Event Listeners 72-7
- Logging 72-7
  - Log Types 72-8
  - Log Output 72-10
  - Log Level 72-10
  - Logging SQL 72-10
  - Logging Chained Exceptions 72-11
  - Viewing TopLink Log Messages From the Application Server Control Console 72-11
- Profiler 72-11
  - TopLink Profiler 72-12
  - Oracle Dynamic Monitoring System (DMS) 72-12

Integrity Checker ... 72-12
Exception Handlers ... 72-12
Registering Descriptors ... 72-13
Sessions and CMP ... 72-13
Sessions and Sequencing ... 72-13
Server and Client Sessions ... 72-14
Three-Tier Architecture Overview ... 72-14
Advantages of the TopLink Three-Tier Architecture ... 72-15
Shared Resources ... 72-15
Providing Read Access ... 72-16
Providing Write Access ... 72-17
Security and User Privileges ... 72-18
Concurrency ... 72-18
Connection Allocation ... 72-18
Unit of Work Sessions ... 72-19
Isolated Client Sessions ... 72-19
Isolated Client Sessions and Oracle Virtual Private Database (VPD) ... 72-21
VPD With Oracle Database Proxy Authentication ... 72-22
VPD Without Oracle Database Proxy Authentication ... 72-22
Isolated Client Session Life Cycle ... 72-22
Isolated Client Session Limitations ... 72-24
Historical Sessions ... 72-25
Historical Session Limitations ... 72-25
Session Broker and Client Sessions ... 72-26
Session Broker Architecture ... 72-26
Committing a Transaction with a Session Broker ... 72-27
Committing a Session with a JTA Driver: Two-Phase Commits ... 72-27
Committing a Session Without a JTA Driver: Two-Stage Commits ... 72-27
Session Broker Session Limitations ... 72-28
Many-to-Many Join Tables and Direct Collection Tables ... 72-28
Session Broker Alternatives ... 72-28
Database Linking ... 72-28
Multiple Sessions ... 72-28
Database Sessions ... 72-29
Remote Sessions ... 72-30
Architectural Overview ... 72-30
Application Layer ... 72-31
Transport Layer ... 72-31
Server Layer ... 72-31
Remote Session Concepts ... 72-31
Securing Remote Session Access ... 72-32
Queries ... 72-32
Refreshing ... 72-32
Indirection ... 72-32
Cursored Streams ... 72-32
Unit of Work ... 72-32
Sessions and the Cache ... 72-33

Server and Database Session Cache 72-33
Isolated Session Cache 72-33
Historical Session Cache 72-33
Understanding the Session API 72-33

73 Creating Sessions

Session Creation Overview 73-1
Creating a Sessions Configuration 73-1
Using TopLink Workbench 73-2
Configuring a Sessions Configuration 73-2
Using TopLink Workbench 73-2
Creating a Server Session 73-4
Using TopLink Workbench 73-4
Using Java 73-5
Creating Session Broker and Client Sessions 73-6
Using TopLink Workbench 73-6
Using Java 73-8
Creating Database Sessions 73-8
Using TopLink Workbench 73-8
Using Java 73-10
Creating Remote Sessions 73-10
Using Java 73-10
Server 73-11
Client 73-11

74 Configuring a Session

Configuring Common Session Options 74-1
Configuring a Primary Mapping Project 74-2
Using TopLink Workbench 74-3
Using Java 74-3
Configuring a Session Login 74-4
Configuring Logging 74-4
Using TopLink Workbench 74-5
Using Java 74-7
Using Session Logging API 74-7
Configuring a Session to use java.util.logging Package 74-8
Configuring Logging in a CMP Application 74-9
Configuring Multiple Mapping Projects 74-9
Using TopLink Workbench 74-9
Using Java 74-10
Configuring a Performance Profiler 74-10
Using TopLink Workbench 74-11
Using Java 74-12
Configuring an Exception Handler 74-12
Using TopLink Workbench 74-12
Using Java 74-12

Configuring Customizer Class 74-13
Using TopLink Workbench 74-14
Configuring the Server Platform 74-14
Using TopLink Workbench 74-15
Using Java 74-17
Configuring Session Event Listeners 74-17
Using TopLink Workbench 74-17
Using Java 74-18
Configuring the Integrity Checker 74-18
Using Java 74-19
Configuring Connection Policy 74-19
Using TopLink Workbench 74-20
Using Java 74-21
Configuring Named Queries at the Session Level 74-21
Using Java 74-22

75 Acquiring and Using Sessions at Run Time

Session Acquisition Overview 75-1
Understanding the Session Manager 75-2
Multiple Sessions 75-2
Acquiring the Session Manager 75-2
Acquiring a Session From the Session Manager 75-3
Loading a Session From sessions.xml Using Defaults 75-3
Loading a Session From sessions.xml With an Alternative Class Loader 75-4
Loading a Session From an Alternative Session Configuration File 75-4
Loading a Session Without Logging In 75-5
Reloading and Refreshing Session Configuration 75-5
Refreshing a Session When the Class Loader Changes 75-6
Acquiring a Client Session 75-6
Acquiring an Isolated Client Session 75-7
Acquiring a Client Session That Uses Exclusive Connections 75-7
Acquiring a Client Session That Uses Connection Properties 75-8
Acquiring a Client Session That Uses a Named Connection Pool 75-8
Acquiring a Client Session That Does Not Use Lazy Connection Allocation 75-9
Acquiring a Historical Session 75-9
Logging In to a Session 75-10
Using Session API 75-10
Logging Out of a Session 75-10
Storing Sessions in the Session Manager Instance 75-10
Destroying Sessions in the Session Manager Instance 75-11

76 Configuring Server Sessions

Server Session Configuration Overview 76-1
Configuring Internal Connection Pools 76-1
Configuring External Connection Pools 76-2

77 Configuring Exclusive Isolated Client Sessions for Virtual Private Database

Exclusive Isolated Client Session Configuration Overview ... 77-1
PostAcquireExclusiveConnection Event Handler ... 77-1
Using Java ... 77-2
PreReleaseExclusiveConnection Event Handler ... 77-2
Using Java ... 77-2
NoRowsModifiedSessionEvent Event Handler ... 77-3
Using Java ... 77-3
ValidationException Handler ... 77-3

78 Configuring Historical Sessions

Historical Session Configuration Overview ... 78-1
Configuring Historical Sessions Using an Oracle Platform ... 78-1
Configuring Historical Sessions Using a TopLink HistoryPolicy ... 78-1

79 Configuring Session Broker and Client Sessions

Session Broker and Client Session Configuration Overview ... 79-1
Removing, Renaming, or Adding Sessions ... 79-1
Using TopLink Workbench ... 79-2

80 Configuring Database Sessions

Database Session Configuration Overview ... 80-1
Configuring External Connection Pools ... 80-1

Part XVI Data Access

81 Understanding Data Access

Data Access Concepts ... 81-1
Externally Managed Transactional Data Sources ... 81-1
Data Source Login Types ... 81-2
DatabaseLogin ... 81-2
EISLogin ... 81-3
Data Source Platform Types ... 81-3
Database Platforms ... 81-3
EIS Platforms ... 81-4
Authentication ... 81-5
Simple JDBC Authentication ... 81-5
Oracle Database Proxy Authentication ... 81-5
Auditing ... 81-6
Connections ... 81-6
Connection Pools ... 81-7
Internal Connection Pools ... 81-7
External Connection Pools ... 81-8
Default (Write) and Read Connection Pools ... 81-8
Sequence Connection Pools ... 81-8

Application-Specific Connection Pools ... 81-9
Understanding Data Access API ... 81-9
Login Inheritance Hierarchy ... 81-10
Platform Inheritance Hierarchy ... 81-10

82 Configuring a Data Source Login

Configuring Common Data Source Login Options ... 82-1
Configuring User Name and Password ... 82-1
Using TopLink Workbench ... 82-2
Configuring Password Encryption ... 82-2
Using Java ... 82-2
Configuring External Connection Pooling ... 82-2
Using TopLink Workbench ... 82-3
Configuring Properties ... 82-4
Using TopLink Workbench ... 82-4
Using Java ... 82-5
Configuring a Default Null Value at the Login Level ... 82-5
Using Java ... 82-6

83 Configuring a Database Login

Database Login Configuration Overview ... 83-1
Configuring a Relational Database Platform at the Session Level ... 83-1
Using TopLink Workbench ... 83-1
Configuring Database Login Connection Options ... 83-2
Using TopLink Workbench ... 83-2
Configuring Sequencing at the Session Level ... 83-4
Using TopLink Workbench ... 83-5
Using Java ... 83-5
Using the Platform Default Sequence ... 83-6
Configuring Multiple Sequences ... 83-6
Configuring Query Sequencing ... 83-7
Configuring a Table Qualifier ... 83-8
Using TopLink Workbench ... 83-8
Configuring JDBC Options ... 83-9
Using TopLink Workbench ... 83-9
Using Java ... 83-11
Configuring Advanced Options ... 83-11
Using TopLink Workbench ... 83-11
Configuring Oracle Database Proxy Authentication ... 83-12
Using Java ... 83-13

84 Configuring an EIS Login

EIS Login Configuration Overview ... 84-1
Configuring an EIS Data Source Platform at the Session Level ... 84-1
Using TopLink Workbench ... 84-1
Configuring EIS Connection Specification Options at the Session Level ... 84-2

Using TopLink Workbench 84-2

85 Creating an Internal Connection Pool

Internal Connection Pool Creation Overview 85-1
Using TopLink Workbench 85-1

86 Configuring an Internal Connection Pool

Internal Connection Pool Configuration Overview 86-1
Configuring Connection Pool Sizes 86-1
Using TopLink Workbench 86-2
Configuring Properties 86-2
Using TopLink Workbench 86-2
Using Java 86-3
Configuring a Nontransactional Read Login 86-3
Using TopLink Workbench 86-3
Configuring Connection Pool Connection Options 86-4
Using TopLink Workbench 86-4
Configuring Exclusive Read Connections 86-6
Using TopLink Workbench 86-6

Part XVII Cache

87 Understanding the Cache

Cache Architecture 87-1
Session Cache 87-2
Unit of Work Cache 87-2
Cache Concepts 87-2
Cache Type and Object Identity 87-3
Full Identity Map 87-3
Weak Identity Map 87-3
Soft and Hard Cache Weak Identity Maps 87-4
No Identity Map 87-4
Guidelines for Configuring the Cache and Identity Maps 87-4
Understanding the Internals of Soft and Hard Cache Weak Identity Map 87-5
Querying and the Cache 87-6
Handling Stale Data 87-6
Configure a Locking Policy 87-6
Configure the Cache on a Per-Class Basis 87-7
Force a Cache Refresh When Required on a Per-Query Basis 87-7
Configure Cache Invalidation 87-7
Configure Cache Coordination 87-7
Explicit Query Refreshes 87-7
Refresh Policy 87-8
EJB Finders and Refresh Policy 87-8
Cache Invalidation 87-8
Cache Coordination 87-9

Cache Isolation 87-9
Cache Locking and Transaction Isolation 87-9
Cache Optimization 87-10
Understanding Cache Coordination 87-10
When to use Cache Coordination 87-11
Coordinated Cache Architecture 87-11
Session 87-11
Descriptor 87-12
Unit of Work 87-12
Coordinated Cache Types 87-12
JMS Coordinated Cache 87-12
RMI Coordinated Cache 87-12
CORBA Coordinated Cache 87-13
Custom Coordinated Cache 87-13
Understanding the Cache API 87-13
Object Identity API 87-14
Cache Refresh API 87-14
Cache Invalidation API 87-14
Cache Coordination API 87-15

88 Configuring a Coordinated Cache

Configuring Common Coordinated Cache Options 88-1
Configuring the Synchronous Change Propagation Mode 88-2
Using TopLink Workbench 88-2
Configuring a Service Channel 88-3
Using TopLink Workbench 88-3
Configuring a Multicast Group Address 88-4
Using TopLink Workbench 88-5
Configuring a Multicast Port 88-5
Using TopLink Workbench 88-6
Configuring a Naming Service Type 88-7
Configuring JNDI Naming Service Information 88-7
Using TopLink Workbench 88-8
Configuring RMI Registry Naming Service Information 88-9
Using TopLink Workbench 88-10
Configuring an Announcement Delay 88-11
Using TopLink Workbench 88-12
Configuring Connection Handling 88-13
Using TopLink Workbench 88-13
Configuring Context Properties 88-14
Using TopLink Workbench 88-14
Configuring a Packet Time-to-Live 88-15
Using TopLink Workbench 88-16

89 Configuring a JMS Coordinated Cache

JMS Coordinated Cache Configuration Overview 89-1
Configuring a Topic Name 89-1

Using TopLink Workbench 89-1
Configuring a Topic Connection Factory Name 89-2
Using TopLink Workbench 89-2
Configuring a Topic Host URL 89-3
Using TopLink Workbench 89-3

90 Configuring an RMI Coordinated Cache

RMI Coordinated Cache Configuration Overview 90-1

91 Configuring a CORBA Coordinated Cache

CORBA Coordinated Cache Configuration Overview 91-1

92 Configuring a Custom Coordinated Cache

Custom Coordinated Cache Configuration Overview 92-1
Configuring Transport Class 92-1
Using TopLink Workbench 92-1

Part XVIII Queries

93 Understanding TopLink Queries

Query Types 93-1
Query Concepts 93-2
Call 93-3
DatabaseQuery 93-3
Data-Level and Object-Level Queries 93-3
Summary Queries 93-3
Descriptor Query Manager 93-3
TopLink Expressions 93-3
Query Keys 93-4
Query Languages 93-4
SQL Queries 93-4
EJB QL Queries 93-5
XML Queries 93-5
EIS Interactions 93-5
Query-by-Example 93-6
Building Queries 93-6
Executing Queries 93-6
Handling Query Results 93-8
Collection Query Results 93-8
Report Query Results 93-8
Stream and Cursor Query Results 93-8
Session Queries 93-9
Read-Object Session Queries 93-9
Create, Update, and Delete Object Session Queries 93-10
Database Queries 93-10

Object-Level Read Query 93-11
ReadObjectQuery 93-11
ReadAllQuery 93-11
Partial Object Queries 93-11
Join Reading and Object-Level Read Queries 93-12
Fetch Groups and Object-Level Read Queries 93-13
Data-Level Read Query 93-13
DataReadQuery 93-13
DirectReadQuery 93-13
ValueReadQuery 93-14
Object-Level Modify Query 93-14
WriteObjectQuery 93-14
UpdateObjectQuery 93-14
InsertObjectQuery 93-14
DeleteObjectQuery 93-14
UpdateAllQuery 93-14
Object-Level Modify Queries and Privately Owned Parts 93-15
Data-Level Modify Query 93-15
Report Query 93-15
Named Queries 93-16
Call Queries 93-17
SQL Calls 93-17
SQLCall 93-17
StoredProcedureCall 93-17
StoredFunctionCall 93-18
Oracle Extensions 93-18
EJB QL Calls 93-19
Enterprise Information System (EIS) Interactions 93-19
IndexedInteraction 93-19
MappedInteraction 93-19
XMLInteraction 93-20
XQueryInteraction 93-20
QueryStringInteraction 93-20
Redirect Queries 93-20
Historical Queries 93-21
Using an ObjectLevelReadQuery With an AsOfClause 93-21
Using an ObjectLevelReadQuery With Expression Operator asOf 93-22
Using an ObjectLevelReadQuery in a Historical Session 93-22
Interface and Inheritance Queries 93-22
Descriptor Query Manager Queries 93-23
Configuring Named Queries 93-23
Configuring Default Query Implementations 93-23
Configuring Additional Join Expressions 93-24
EJB Finders 93-24
Predefined Finders 93-24
Predefined CMP Finders 93-25
Predefined BMP Finders 93-25

Default Finders 93-26
Call Finders 93-26
DatabaseQuery Finders 93-26
Named Query Finders 93-27
Primary Key Finders 93-27
Expression Finders 93-27
EJB QL Finders 93-27
SQL Finders 93-28
Redirect Finders 93-28
The ejbSelect Method 93-28
Queries and the Cache 93-29
Configuring the Cache 93-29
Using In-Memory Queries 93-30
Configuring Cache Usage for In-Memory Queries 93-30
Expression Options for In-Memory Queries 93-31
Handling Exceptions Resulting From In-Memory Queries 93-33
Primary Key Queries and the Cache 93-34
Disabling the Identity Map Cache Update During a Read Query 93-34
Refreshing the Cache 93-35
Object Refresh 93-35
Cascading Object Refresh 93-35
Refreshing the Identity Map Cache During a Read Query 93-35
Caching Query Results in the Session Cache 93-36
Caching Query Results in the Query Cache 93-36
Internal Query Cache Restrictions 93-37
Caching and EJB Finders 93-37
Caching Options 93-37
Disabling Cache for Returned Finder Results 93-38
Refreshing Finder Results 93-38
Understanding the Query API 93-38

94 Using Basic Query API

Using Session Queries 94-1
Reading Objects With a Session Query 94-1
Reading an Object With a Session Query 94-2
Reading All Objects With a Session Query 94-2
Refreshing an Object With a Session Query 94-2
Creating, Updating, and Deleting Objects With a Session Query 94-3
Writing a Single Object to the Database With a Session Query 94-3
Writing All Objects to the Database With a Session Query 94-3
Adding New Objects to the Database With a Session Query 94-4
Modifying Existing Objects in the Database With a Session Query 94-4
Deleting Objects in the Database With a Session Query 94-4
Using DatabaseQuery Queries 94-4
Reading Objects Using a DatabaseQuery 94-4
Basic DatabaseQuery Read Operations 94-5
Reading Objects Using Partial Object Queries 94-6

Reading Objects Using Report Queries 94-6
Reading Objects Using Query-By-Example 94-6
Specifying Read Ordering 94-9
Specifying a Collection Class 94-9
Specifying the Maximum Rows Returned 94-10
Configuring Query Timeout at the Query Level 94-10
Using Batch Reading 94-10
Using Join Reading 94-11
Creating, Updating, and Deleting Objects With a DatabaseQuery 94-13
Write Query Overview 94-13
UpdateAll Queries 94-13
Noncascading Write Queries 94-14
Disabling the Identity Map Cache During a Write Query 94-15
Reading Data With a DatabaseQuery 94-15
Using a DataReadQuery 94-15
Using a DirectReadQuery 94-16
Using a ValueReadQuery 94-16
Updating Data With a DatabaseQuery 94-16
Specifying a Custom SQL String in a DatabaseQuery 94-16
Specifying a Custom EJB QL String in a DatabaseQuery 94-17
Using Parameterized SQL and Statement Caching in a DatabaseQuery 94-17
Using Named Queries 94-18
Using SQL Calls 94-19
Using an SQLCall 94-19
Specifying a SQLCall Input Parameter 94-20
Specifying a SQLCall Output Parameter 94-20
Specifying a SQLCall Input / Output Parameter 94-20
Using a StoredProcedureCall 94-21
Specifying an Input Parameter 94-21
Specifying an Output Parameter 94-22
Specifying an Input / Output Parameter 94-22
Using an Output Parameter Event 94-23
Using a StoredFunctionCall 94-23
Using EJB QL Calls 94-24
Using EIS Interactions 94-24
Handling Exceptions 94-25
Handling Collection Query Results 94-25
Handling Report Query Results 94-26

95 Understanding TopLink Expressions

Understanding the Expression Framework 95-1
Expressions Compared to SQL 95-1
Expression Components 95-2
Boolean Logic 95-3
Database Functions and Operators 95-3
Mathematical Functions 95-4
XMLType Functions 95-5

Platform and User-Defined Functions 95-5
Expressions for One-to-One and Aggregate Object Relationships 95-5
Expressions for Joining and Complex Relationships 95-6
Understanding Joins 95-6
Using TopLink Expression API For Joins 95-7
Parameterized Expressions 95-8
Expression Method getParameter 95-8
Expression Method getField 95-9
Query Keys and Expressions 95-10
Using Multiple Expressions 95-10
Subselects and Subqueries 95-10
Parallel Expressions 95-11
Data Queries and Expressions 95-12
getField 95-12
getTable 95-12
Creating an Expression 95-13
Using TopLink Workbench 95-13
Adding Arguments 95-14
Using Java 95-15
Creating and Using a User-Defined Function 95-16
Making a User-Defined Function Available to a Specific Platform 95-17
Making a User-Defined Function Available to All Platforms 95-17
Using a User-Defined Function 95-17

96 Using Advanced Query API

Using Redirect Queries 96-1
Creating a Redirect Query 96-1
Using Historical Queries 96-2
Using Queries With Fetch Groups 96-2
Configuring Default Fetch Group Behavior 96-3
Querying With a Static Fetch Group 96-3
Querying With a Dynamic Fetch Group 96-4
Querying on Interfaces 96-4
Querying on an Inheritance Hierarchy 96-4
Appending Additional Join Expressions 96-4
Using Java 96-5
Using Queries on Variable One-to-One Mappings 96-5
Using Oracle Database Features 96-5
Oracle Hints 96-6
Hierarchical Queries 96-6
startWith Parameter 96-7
connectBy Parameter 96-7
orderSibling Parameter 96-7
Using EJB Finders 96-7
Creating a Finder 96-8
ejb-jar.xml Finder Options 96-9
Using DatabaseQuery Finders 96-10

Using Named Query Finders ... 96-10
Using Primary Key Finders ... 96-11
Using EJB QL Finders ... 96-11
Using SQL Finders ... 96-12
Using Redirect Finders ... 96-12
Using the ejbSelect Method ... 96-15
Handling Cursor and Stream Query Results ... 96-15
Cursors and Java Iterators ... 96-16
Traversing Data With Scrollable Cursors ... 96-16
Java Streams ... 96-17
Cursored Stream Support ... 96-17
Optimizing Streams ... 96-18
Using Cursors and Streams With EJB Finders ... 96-18
Building the Query ... 96-19
Executing the Finder From the Client ... 96-19
Using Queries and the Cache ... 96-20
Caching Results in a ReadQuery ... 96-20
Configuring Cache Expiration at the Query Level ... 96-21

Part XIX Transactions

97 Understanding TopLink Transactions

Unit of Work Architecture ... 97-1
Unit of Work Transaction Context ... 97-2
Unit of Work Transaction Demarcation ... 97-2
JTA Controlled Transactions ... 97-3
OTS Controlled Transactions ... 97-3
CMP Controlled Transactions ... 97-3
Unit of Work Transaction Isolation ... 97-4
Unit of Work Concepts ... 97-4
Unit of Work Benefits ... 97-4
Unit of Work Life Cycle ... 97-5
Unit of Work and Change Policy ... 97-6
Deferred Change Detection Policy ... 97-7
Object-Level Change Tracking Policy ... 97-7
Attribute Change Tracking Policy ... 97-8
Change Policy Mapping Support ... 97-9
Clones and the Unit of Work ... 97-9
Nested and Parallel Units of Work ... 97-9
Nested Unit of Work ... 97-10
Parallel Unit of Work ... 97-10
Commit and Rollback Transactions ... 97-10
Commit Transactions ... 97-10
Rollback Transactions ... 97-11
Primary Keys ... 97-11
Unit of Work Optimization ... 97-11
Understanding the Unit of Work API ... 97-11

Unit of Work as Session 97-12
Reading and Querying Objects with the Unit of Work 97-12
Locking and the Unit of Work 97-12
Example Model Object and Schema 97-12

98 Using Basic Unit of Work API

Acquiring a Unit of Work 98-1
Creating an Object 98-2
Modifying an Object 98-2
Associating a New Target to an Existing Source Object 98-3
Associating Without Reference to the Cache Object 98-3
Associating With Reference to the Cache Object 98-4
Associating a New Source to an Existing Target Object 98-6
Associating an Existing Source to an Existing Target Object 98-7
Deleting Objects 98-7
Using privateOwnedRelationship 98-8
Explicitly Deleting From the Database 98-9
Understanding the Order in Which Objects Are Deleted 98-9

99 Using Advanced Unit of Work API

Registering and Unregistering Objects 99-1
Creating and Registering an Object in One Step 99-2
Using registerNewObject 99-2
Registering a New Object With registerNewObject 99-2
Associating New Objects With One Another 99-3
Using registerAllObjects 99-4
Using Registration and Existence Checking 99-5
Check Database 99-5
Assume Existence 99-5
Assume Nonexistence 99-5
Working With Aggregates 99-6
Unregistering Working Clones 99-6
Declaring Read-Only Classes 99-6
Configuring Read-Only Classes for a Single Unit of Work 99-6
Configuring Default Read-Only Classes 99-7
Read-Only Descriptors 99-7
Writing Changes Before Commit Time 99-7
Using Conforming Queries and Descriptors 99-8
Guidelines for Using Conforming 99-8
Ensure That the Query Supports Conforming 99-8
Consider how Conforming Affects Database Results 99-9
Register New Objects and Instantiate Relationships 99-10
Using Conforming Queries 99-11
Using Conforming Descriptors 99-12
Conforming Query Alternatives 99-12
Using Unit of Work Method writeChanges Instead of Conforming 99-12

Using Unit of Work Properties Instead of Conforming 99-13
Merging Changes in Working Copy Clones 99-13
Resuming a Unit of Work After Commit 99-14
Reverting a Unit of Work 99-15
Using a Nested or Parallel Unit of Work 99-15
Parallel Unit of Work 99-15
Nested Unit of Work 99-15
Using a Unit of Work With Custom SQL 99-16
Controlling the Order of Delete Operations 99-16
Using the Unit of Work setShouldPerformDeletesFirst Method 99-17
Using the Descriptor addConstraintDependencies Method 99-17
Using deleteAllObjects Without addConstraintDependencies 99-17
Using deleteAllObjects With addConstraintDependencies 99-17
Using Optimistic Read Locking With forceUpdateToVersionField 99-18
Forcing a Check of the Optimistic Read Lock 99-18
Forcing a Version Field Update 99-19
Disabling forceUpdateToVersionField 99-20
Implementing User and Date Auditing With the Unit of Work 99-21
Integrating the Unit of Work With an External Transaction Service 99-21
Acquiring a Unit of Work With an External Transaction Service 99-21
Using a Unit of Work When an External Transaction Exists 99-22
Using a Unit of Work When No External Transaction Exists 99-23
Using the Unit of Work to Handle External Transaction Timeouts and Exceptions 99-23
External Transaction Commit Timeouts 99-24
External Transaction Commit Exceptions 99-24
Integrating the Unit of Work With CMP 99-24
CMP Transaction Attribute 99-25
Local Transactions 99-25
Nondeferred Changes 99-26
Database Transaction Isolation Levels 99-26
General Factors Affecting Transaction Isolation Level 99-26
External Applications 99-27
TopLink Coordinated Cache 99-27
DatabaseLogin Method setTransactionIsolation 99-27
Reading Through the Write Connection 99-28
Managing Cache Access 99-29
CMP and External Transactions 99-30
Read Uncommitted Level 99-30
Read Committed Level 99-30
Repeatable Read Levels 99-30
Serializable Read Levels 99-30
Troubleshooting a Unit of Work 99-31
Avoiding the Use of Post-Commit Clones 99-31
Determining Whether or Not an Object Is the Cache Object 99-32
Dumping the Contents of a Unit of Work 99-32
Handling Exceptions 99-33
Exceptions at Commit Time 99-33

Exceptions During Conforming 99-34
Validating a Unit of Work 99-34
Validating the Unit of Work Before Commit Time 99-35

Index

Preface

Oracle TopLink Developer's Guide explains how to use Oracle TopLink to design, implement, deploy, and optimize an advanced persistence or object-to-XML transformation layer for a wide range of Java 2 Enterprise Edition (J2EE) and Java applications. It describes TopLink mapping metadata, sessions, data access, queries, transactions (both with and without an external transaction controller), and cache.

It describes how to use TopLink application development tools and how to integrate TopLink with a variety of J2EE containers. It also introduces the concepts with which you should be familiar to get the most out of TopLink.

Audience

Oracle TopLink Developer's Guide is intended for application developers creating applications that use TopLink to manage persistence.

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. Accessibility standards will continue to evolve over time, and Oracle 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 more information, visit the Oracle Accessibility Program Web site at

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

Accessibility of Code Examples in Documentation

Screen readers 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, some screen readers 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 does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Related Documentation

For more information, see the following documents:

- *Oracle TopLink Release Notes*
- Oracle Application Server Release Notes
- *Oracle TopLink Getting Started Guide*
- *Oracle TopLink API Reference*

Conventions

The following text conventions are used in this document:

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

Part I

Building a TopLink Application

This part describes the architecture of TopLink and how TopLink fits into your development process. It contains the following chapters:

- Chapter 1, "Understanding TopLink"

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

- Chapter 2, "Understanding TopLink Application Development"

  This chapter contains an overview of the key stages in the TopLink development process including choosing an application architecture and platform, object and relational mapping, building the persistence layer, and deploying and maintaining a TopLink application.

1

Understanding TopLink

Oracle TopLink is an advanced, object-persistence and object-transformation framework that provides development tools and run-time capabilities that reduce development and maintenance efforts, and increase enterprise application functionality.

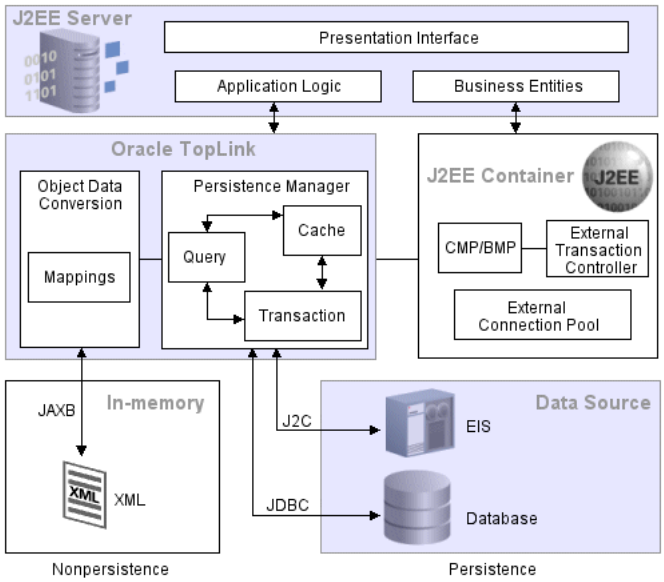
This chapter describes the following topics:

- What is TopLink?
- Solving the Object-Persistence Impedance Mismatch
- TopLink Key Features
- TopLink Application Architectures

What is TopLink?

Oracle TopLink builds high-performance applications that store persistent object-oriented data in a relational database. It successfully transforms object-oriented data into either relational data or Extensible Markup Language (XML) elements.

Figure 1–1 TopLink Runtime Architecture



Using TopLink, you can integrate persistence and object-transformation into your application, while staying focused on your primary domain problem by taking advantage of an efficient, flexible, and field-proven solution (see "Solving the Object-Persistence Impedance Mismatch" on page 1-2).

TopLink is suitable for use with a wide range of Java 2 Enterprise Edition (J2EE) and Java application architectures (see "TopLink Application Architectures" on page 1-4). Use TopLink to design, implement, deploy, and optimize an advanced, object-persistence and object-transformation layer that supports a variety of data sources and formats, including the following:

- Relational–for transactional persistence of Java objects to a relational database accessed using Java Database Connectivity (JDBC) drivers.
- Object-Relational–for transactional persistence of Java objects to special purpose structured data source representations optimized for storage in object-relational databases such as Oracle Database.
- Enterprise information system (EIS)–for transactional persistence of Java objects to a nonrelational data source accessed using a J2EE Connector architecture (J2C) adapter, and any supported EIS record type, including indexed, mapped, or XML.
- XML–for nontransactional, nonpersistent (in-memory) conversion between Java objects and XML Schema Document (XSD)-based XML documents using Java Architecture for XML Binding (JAXB).

TopLink includes support for container-managed persistence (CMP) containers from a variety of vendors–such as Oracle Containers for J2EE (OC4J), IBM WebSphere application server, and BEA WebLogic Server–and support for base classes that simplify bean-managed persistence (BMP) development.

The extensive suite of development tools that TopLink provides, including Oracle TopLink Workbench, lets you quickly capture and define object-to-data source and object-to-data representation mappings in a flexible, efficient metadata format (see "Understanding TopLink Metadata" on page 2-19).

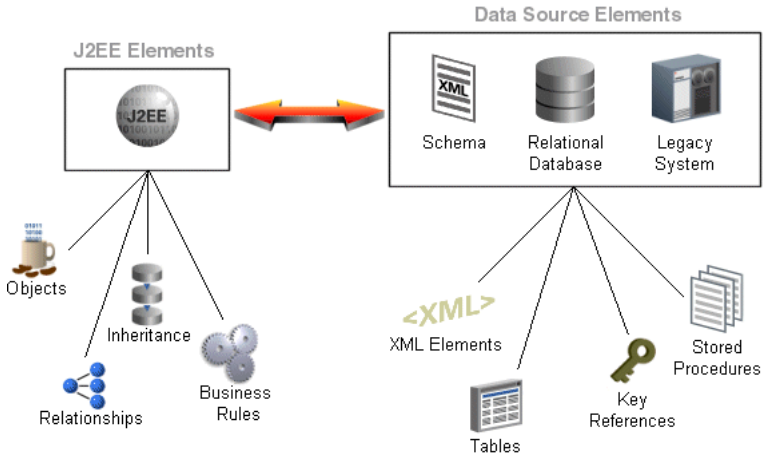
The TopLink runtime lets your application exploit this mapping metadata with a simple session facade that provides in-depth support for data access, queries, transactions (both with and without an external transaction controller), and caching.

For more information about TopLink, see "TopLink Key Features" on page 1-4.

Solving the Object-Persistence Impedance Mismatch

Java-to-data source integration is a widely underestimated problem when creating enterprise Java applications. This complex problem involves more than simply reading from and writing to a data source. The data source elements include tables, rows, columns, and primary and foreign keys. The Java and J2EE include entity classes (regular Java classes or Enterprise JavaBeans (EJB) entity beans), business rules, complex relationships, and inheritance. In a nonrelational data source, you must match your Java entities with EIS records or XML elements and schemas. These differences (as shown in Figure 1–2) are known as the object-persistence impedance mismatch.

Figure 1–2 Solving Object-Persistence Impedance Mismatch



Successful solution requires bridging these different technologies, and solving the object-persistence impedance mismatch–a challenging and resource-intensive problem. To solve this problem, you must resolve the following issues between J2EE and data source elements:

- 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 with any data source, without compromising ideal application design or data integrity. In addition, Java developers need the ability to store (that is, **persist**) and retrieve business domain objects using a relational database or a nonrelational data source as a repository.

TopLink Solution

TopLink addresses the disparity between Java objects and data sources. TopLink is a persistence framework that manages relational, object-relational, EIS, and XML mappings in a seamless manner. This allows developers to rapidly build applications that combine the best aspects of object technology and the specific data source.

TopLink lets you do the following:

- Persist Java objects to virtually *any* relational database supported by a JDBC-compliant driver.
- Persist Java objects to virtually *any* nonrelational data source supported by a J2EE Connector architecture (J2C) adapter using indexed, mapped, or XML enterprise information system (EIS) records.
- Perform in-memory conversions between Java objects and XML Schema (XSD) based XML documents using JAXB.
- Map *any* object model to *any* relational or nonrelational schema, using the Oracle TopLink Workbench graphical mapping tool.
- Use TopLink successfully, even if you are unfamiliar with SQL or JDBC, because TopLink offers a clear, object-oriented view of data sources.

TopLink Key Features

TopLink provides an extensive and thorough set of features. Java developers can use these features to rapidly build high-performance enterprise applications that are both scalable and maintainable.

Some of the primary features of TopLink are the following:

- Nonintrusive, flexible, metadata-based architecture (see "Understanding TopLink Metadata" on page 2-19)
- Comprehensive visual TopLink Workbench
- Advanced mapping support and flexibility (relational, object-relational, EIS, and XML)
- Object caching support
- Query flexibility
- Just-in-time reading
- Caching
- Object-level transaction support and integration
- Locking
- Multiple performance tuning options
- Architectural flexibility

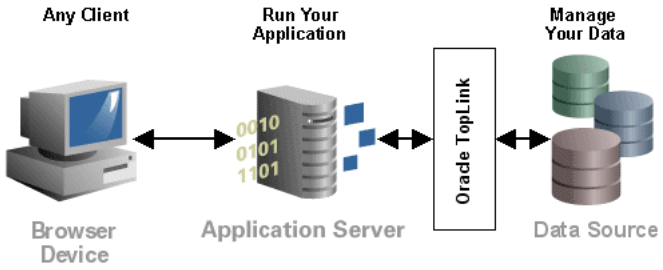
For additional information, see the TopLink page on OTN:

`http://www.oracle.com/technology/products/ias/toplink/index.html`

TopLink Application Architectures

You can use TopLink in a variety of application architectures, including three- and two-tier architectures, with or without J2EE, to access a variety of data types on both relational and nonrelational data sources.

Figure 1–3 TopLink and Your Application Architecture



For more information on strategies for incorporating TopLink into your application architecture, see "Designing Your Application With TopLink" on page 2-3.

This section introduces some of the following common enterprise architectures used by TopLink applications:

- **Three-Tier**

  The three-tier (or J2EE Web) application is one of the most common TopLink architectures. This architecture is characterized by a server-hosted environment in

which the business logic, persistent entities, and the Oracle TopLink Foundation Library all exist in a single Java Virtual Machine (JVM). See "Understanding the Three-Tier Architecture" on page 2-22 for more information.

The most common example of this architecture is a simple three-tier application in which the client browser accesses the application through servlets, JavaServer Pages (JSP) and HTML. The presentation layer communicates with 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 TopLink.

- **EJB Session Bean Facade**

  A popular variation on the three-tier application involves wrapping the business logic, including the TopLink access, in EJB session beans. This architecture provides a scalable deployment and includes integration with transaction services from the host application server. See "Understanding the EJB Session Bean Facade Architecture" on page 2-25 for more information.

  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 3.0 Entities with JPA**

  The EJB 3.0 specification includes an additional persistence specification called the Java Persistence API (JPA). You can use this API for creating, removing and querying across lightweight Java objects both within a compliant EJB 3.0 container and a standard Java SE 5 environment.

  TopLink Essentials is the open-source community edition of TopLink. It provides the JPA functionality for the EJB 3.0 reference implementation.

  > **Note:** TopLink Essentials is the default JPA implementation in OC4J.

  When using TopLink JPA, you can access the API through extensions. The extensions allow you to reach down to the TopLink API and modify the JPA implementation.

  To obtain more information, see the following:

  – `http://www.oracle.com/technology/products/ias/toplink/jpa/index.html`.

  – "Understanding the EJB 3.0 JPA Entity Architecture" on page 2-33.

- **EJB Entity Beans with CMP**

  TopLink provides CMP support for applications that require the use of EJB entity beans. This support is available on the leading application servers. TopLink CMP support provides you with an EJB 1.0, 1.1, 2.0 and 2.1 CMP solution transparent to the application code, but still offers all the TopLink run-time benefits. See "Understanding the EJB Entity Beans With CMP Architecture" on page 2-27 for more information.

  Applications can access TopLink-enabled entity beans with container-managed persistence directly from the client, or from within a session bean layer. TopLink

also offers the ability to use regular Java objects in relationships with enterprise entity beans.

Note: Even though TopLink fully supports EJB 1.0, 1.1, 2.0, 2.1 and, preliminarily EJB 3.0, this version of Oracle TopLink Developer's Guide only focuses on EJB 2.0 and 2.1. For detailed information on TopLink support for EJB 1.0 and 1.1, refer to the earlier versions of *Oracle TopLink Developer's Guide*.

- **EJB Entity Beans with BMP**

  Another option for using EJB entity beans is to use TopLink BMP in the application. This architecture enables developers to access the persistent data through the EJB application programming interface (API), but is platform independent. See "Understanding the EJB Entity Beans With BMP Architecture" on page 2-31 for complete information.

  The BMP approach is portable–that is, after you create an application, you can move it from one application server platform to another.

- **Web Services**

  A Web services architecture is similar to the three-tier or session-bean architecture. However, in a Web services architecture you encapsulate business logic (the service) in a Web service instead of (or in addition to) using session beans. In a Web services architecture, clients communicate with your application using XML.

  As in any architecture, you can use TopLink to persist objects to relational or EIS data sources. However, in a Web services architecture you can also use TopLink to map your object model to an XML schema for use with the Web service or as the Web service XML serializer.

  See "Understanding the Web Services Architecture" on page 2-35 for more information

- **Two-Tier**

  A two-tier (or client/server) application is one in which the TopLink application accesses the database directly. Although less common than the other architectures discussed here, TopLink supports this architecture for smaller or embedded data processing applications. See "Understanding the Two-Tier Architecture" on page 2-24 for more information.

2

Understanding TopLink Application Development

This chapter describes how to build a TopLink application, including suggested development processes, architectures, and technologies.

This chapter includes information on the following topics:

- Developing Your Application With TopLink
- Designing Your Application With TopLink
- Selecting an Architecture With TopLink
- Building and Using the Persistence Layer
- Deploying the Application
- Optimizing and Customizing the Application
- Troubleshooting the Application
- Understanding Object Persistence
- Understanding TopLink Metadata
- Understanding the Three-Tier Architecture
- Understanding the Two-Tier Architecture
- Understanding the EJB Session Bean Facade Architecture
- Understanding the EJB Entity Beans With CMP Architecture
- Understanding the EJB Entity Beans With BMP Architecture
- Understanding the Web Services Architecture

Developing Your Application With TopLink

To ensure the best design for your TopLink application, Oracle recommends that you follow an iterative step-by-step development process. The flexibility of TopLink lets you use *any* development tool.

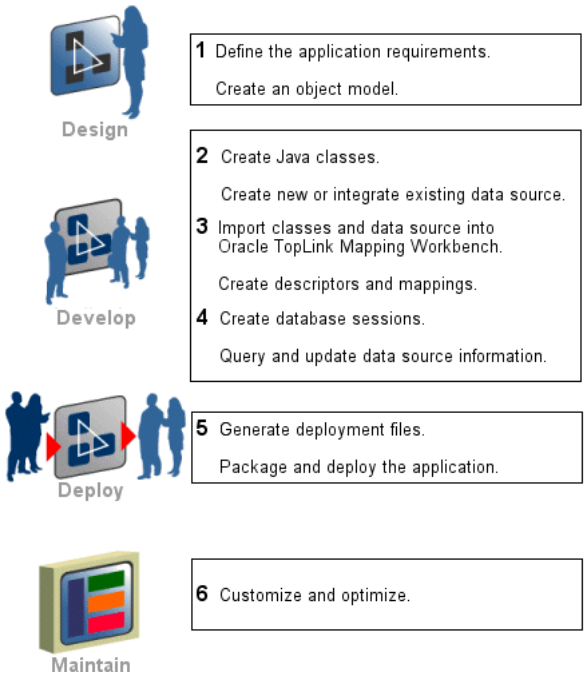
This section describes following recommended development process:

- Typical Development Stages
- Oracle Development Support

Typical Development Stages

This section describes the general development stages of a TopLink application. Figure 2–1 illustrates the TopLink development process.

Figure 2–1 TopLink Development Process



Design the Application (1)

Define your application requirements, select an architecture, and determine the target platform. See "Designing Your Application With TopLink" on page 2-3 for more information. Remember, TopLink works with *any* architecture and *any* platform.

When designing the application, you should also create an object model for the application. See "Understanding Object Persistence" on page 2-16 for more information. It is important to create the object model *before* using TopLink to map objects, because defining persistent mappings for an incorrect or rapidly changing model can be very difficult.

Develop the Application (2, 3, 4)

Create the Java classes and decide how the classes should be implemented by the data source. When working with a legacy system, decide how the classes relate to the existing data. If there is no legacy data source to integrate, decide how to store each class in the data source and create the required schema. Alternatively, you may use TopLink to create your initial tables.

Using TopLink Workbench, create descriptors and mappings for the persistent classes. Use TopLink sessions to manipulate the persistent classes, including querying and changing data. See Part II, "Using TopLink Development Tools" for more information.

Avoid building all your model's descriptors in a single iteration. Start with a small subset of your classes. Build and test their descriptors, then gradually add new descriptors and relationships. This lets you catch common problems before they proliferate through your entire design.

Write Java code to use database sessions. Sessions are used to query for database objects and write objects to the database. See Chapter 72, "Understanding TopLink Sessions" for more information.

Deploy the Application (5)

Generate, package, then deploy the necessary files to your application server. The required information will vary, depending on your environment and architecture. See Part III, "Deploying a TopLink Application" for more information.

Maintain the Application (6)

TopLink includes many options that can enhance application performance. You can customize most aspects of TopLink to suit your requirements. Use advanced TopLink features or write custom querying routines to access the database in specific ways, and to optimize performance. See Part IV, "Optimizing and Customizing a TopLink Application" for more information.

Oracle Development Support

Oracle provides additional support for you as a TopLink developer on the Oracle Technology Network (OTN), including the following:

- Metalink support
- Discussion forums
- How-to documents
- Examples

You must register online before using OTN; registration is free of charge and can be done at

`http://www.oracle.com/technology/membership`

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

`http://www.oracle.com/technology/docs`

Using your OTN user name and password, you can also access the TopLink developer's forum to post questions and get answers about using TopLink at

`http://forums.oracle.com/forums/forum.jsp?forum=48`

Designing Your Application With TopLink

When you design your application, you must choose how and where to use TopLink. You can use TopLink to perform a variety of persistence and data transformation functions (see "Understanding TopLink Usage" on page 2-3) on a variety of Java-supporting platforms (see "Understanding Target Platforms" on page 2-5). When you design your application architecture, keep these capabilities in mind (see "Selecting an Architecture With TopLink" on page 2-5).

Understanding TopLink Usage

This section describes the basic ways in which you can use TopLink, including the following usage types:

- Relational Database Usage
- Object-Relational Database Usage
- Oracle XML Database (XDB) Usage
- Enterprise Information System (EIS) Usage
- XML Usage

Relational Database Usage

You can use TopLink to persist Java objects to relational databases that support SQL data types accessed using JDBC.

For more information, see "Building Relational Projects for a Relational Database" on page 17-6.

Object-Relational Database Usage

You can use TopLink to persist Java objects to object-relational databases that support data types specialized for object storage (such as Oracle Database) accessed using JDBC.

For more information, see "Building Relational Projects for a Relational Database" on page 17-6.

Oracle XML Database (XDB) Usage

You can use TopLink to persist XML documents to an Oracle XML database using TopLink direct-to-XMLType mappings.

For more information, see "Relational Projects" on page 17-6 and "Direct-to-XMLType Mapping" on page 33-4.

Enterprise Information System (EIS) Usage

You can use TopLink to persist Java objects to an EIS data source using a J2C adapter.

In this scenario, the application invokes EIS data source-defined operations by sending EIS interactions to the J2C adapter. Operations can take (and return) EIS records. Using TopLink EIS descriptors and mappings, you can easily map Java objects to the EIS record types supported by your J2C adapter and EIS data source.

This usage is common in applications that connect to legacy data sources and is also applicable to Web services.

For more information, see "EIS Projects" on page 17-7.

XML Usage

You can use TopLink for in-memory, nonpersistent Java object-to-XML transformation with XML Schema (XSD) based XML documents and JAXB.

You can use the TopLink JAXB compiler with your XSD to generate both JAXB-specific artifacts (such as content and element interfaces, implementation classes, and object factory class) and TopLink-specific artifacts (such as sessions and project XML files and TopLink Workbench project). For more information, see "TopLink Support for Java Architecture for XML Binding (JAXB)" on page 17-10.

This usage has many applications, including messaging and Web services.

For more information, see "XML Projects" on page 17-9.

Understanding Target Platforms

TopLink supports any enterprise architecture that uses Java, including the following:

- Java application servers and J2EE containers, such as Oracle Application Server and Oracle Containers for J2EE (OC4J)
- Java-supporting databases, such as Oracle Database
- Java-compatible browsers, such as Netscape and Internet Explorer
- Server Java platforms, such as AS/400, OS/390, and UNIX

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

In addition, TopLink provides custom CMP integration for a variety of application servers.

For detailed information about supported application server versions, custom integration, and configuration requirements, see "Integrating TopLink With an Application Server" on page 7-1.

Selecting an Architecture With TopLink

This section describes some of the key aspects of application architecture that apply to TopLink and discusses the various options available for each, including the following:

- Tiers
- Service Layer
- Data Access
- Caching
- Locking

Tiers

This section describes choices you need to make when deciding on how to separate client and server functionality in your application architecture.

These choices can be summarized as follows:

- Three Tier
  - J2EE or Non-J2EE
  - Client
    - Web client
    - XML/Web service client
    - Java (fat) client
- Two Tier

Three Tier

Oracle recommends a three-tier application architecture. With a three-tier architecture, Oracle recommends using TopLink server sessions and client sessions (see "Server and Client Sessions" on page 72-14) and the TopLink unit of work (see Chapter 97, "Understanding TopLink Transactions").

For more information, see "Understanding the Three-Tier Architecture" on page 2-22.

J2EE or Non-J2EE You can use TopLink in a J2EE or non-J2EE application architecture. Oracle recommends that you use a J2EE application architecture.

With a J2EE application, you should use external connection pools (see "External Connection Pools" on page 81-8). You may consider using Java Transaction API (JTA) integration (see "JTA Controlled Transactions" on page 97-3), EJB session and/or entity beans.

With a non-J2EE application, you should use internal connection pools (see "Internal Connection Pools" on page 81-7).

Client In a three-tier application architecture, you can implement any of the following types of client:

- Web client–Oracle recommends that you implement a Web client.
- XML/Web service client–With this client type, you can use TopLink XML (see "XML Usage" on page 2-4).
- Java (fat) client–With this client type, you can choose the means of communicating with the server:
  - EJB session beans–Oracle recommends this approach. You may consider using the `UnitOfWork` method `mergeClone` to handle merging deserialized objects (see "Merging Changes in Working Copy Clones" on page 99-13). The disadvantage of this approach is that your application must handle serialization. Avoid serializing deep object graphs. You should use indirection (see "Configuring Indirection" on page 32-3). Consider using the data-transfer-object pattern.
  - XML/Web service–Use TopLink XML (see "XML Usage" on page 2-4).
  - EJB entity bean–Use TopLink CMP integration or BMP support. The disadvantage of this approach is that remote entity beans may not perform or scale well.
  - RMI–You may consider using a TopLink remote session (see "Remote Sessions" on page 72-30). The disadvantage of this approach is that a remote session is stateful and may not scale well.

See also "Service Layer" on page 2-7.

Two Tier

With a two-tier application architecture, Oracle recommends using TopLink database sessions (see "Database Sessions" on page 72-29) and the TopLink unit of work (see Chapter 97, "Understanding TopLink Transactions"). The disadvantages of this architecture are that it is not Web-enabled and does not scale well to large deployments.

For more information, see "Understanding the Two-Tier Architecture" on page 2-24.

Service Layer

This section describes choices you need to make when deciding on how to encapsulate your application's business logic (or service).

These choices can be summarized as follows:

- EJB Session Beans
  - Stateful
  - Stateless
- EJB Entity Beans
  - Container-Managed Persistence (CMP)
  - Bean-Managed Persistence (BMP)
- EJB 3.0 JPA Entities
- Plain Old Java Objects (POJO)

See also:

- "Data Access" on page 2-9
- "Caching" on page 2-9.

EJB Session Beans

Oracle recommends using EJB session beans.

With EJB session beans, you should use JTA integration (see "JTA Controlled Transactions" on page 97-3) and external connection pools (see "External Connection Pools" on page 81-8). You should acquire a unit of work using `Server` method `getActiveUnitOfWork` (see "Acquiring a Unit of Work With an External Transaction Service" on page 99-21). If your session bean and data source are not in the same JVM, you may consider using `UnitOfWork` method `mergeClone` to handle merging deserialized objects (see "Merging Changes in Working Copy Clones" on page 99-13).

For more information, see "Understanding the EJB Session Bean Facade Architecture" on page 2-25.

Stateful If you are using stateful session beans, then note that a reference to a client session cannot be passivated. In this case, you must reacquire a client session (see "Acquiring a Session at Run Time With the Session Manager" on page 72-5) on activate or per request.

Stateless If you are using stateless session beans, you must acquire new client session (see "Acquiring a Session at Run Time With the Session Manager" on page 72-5) for each request.

EJB Entity Beans

EJB entity bean architectures are slightly different from other TopLink architectures, because the EJB entity bean interfaces hide TopLink functionality completely from the client application developer.

You can use entity beans in almost any J2EE application. For TopLink, *how the application uses* the entity beans is not important; *how entity beans are mapped and implemented* is important to TopLink.

Container-Managed Persistence (CMP) Oracle recommends using entity beans with container-managed persistence. In this case, you should use the TopLink CMP integration for your application server. You must ensure that you are using a J2EE server that TopLink supports (see "Integrating TopLink With an Application Server" on page 7-1).

For more information, see "Understanding the EJB Entity Beans With CMP Architecture" on page 2-27.

Bean-Managed Persistence (BMP) If you are using entity beans with bean-managed persistence, you should use the TopLink BMP integration. The disadvantages of this architecture are that the BMP architecture is restrictive and may not provide good performance.

For more information, see "Understanding the EJB Entity Beans With BMP Architecture" on page 2-31.

EJB 3.0 JPA Entities

EJB 3.0 Java Persistence API (JPA) is a specification for persistence in Java EE and SE applications. In JPA, a persistent class is referred to as an entity. An entity is a plain old Java object (POJO) class (see "Plain Old Java Objects (POJO)" on page 2-8) that is mapped to the database and configured for usage through JPA using annotations and/or XML.

With JPA, when your application is running inside a container, all of the benefits of the container support and ease of use apply. Note that you can configure the same application to run outside the container.

You can use session beans (see "EJB Session Beans" on page 2-7) as the means for your application to interact with JPA.

If you choose to use JPA for your application development, consider the following options:

- TopLink Essentials–the open-source community edition of TopLink. TopLink Essentials is derived from TopLink providing core object-relational mapping functionality necessary for the reference implementation of JPA in EJB 3.0. TopLink Essentials is available in source and binary distributions as the entity-persistence module of GlassFish Open Source Java EE 5 Application Server.
- TopLink–a commercial implementation that provides the preview functionality of JPA.

Both options provide excellent support for this new standard, as well as additional functionality and performance benefits.

For more information, see "Understanding the EJB 3.0 JPA Entity Architecture" on page 2-33 and "TopLink Application Architectures" on page 1-4.

Plain Old Java Objects (POJO)

If you choose to build your service layer with non-EJB Java objects with a J2EE application server, you should use external connection pools (see "External Connection Pools" on page 81-8). If you use a non-J2EE Web server, you should use internal connection pools (see "Internal Connection Pools" on page 81-7). In either case, you may consider using JTA integration (see "JTA Controlled Transactions" on page 97-3).

Data Access

This section describes choices you need to make when deciding on what type of data your application architecture must support.

These choices can be summarized as follows:

- Data Type
- Multiple Data Sources
- Isolating Data Access
- Historical Data Access

See also "Locking" on page 2-11.

Data Type

You can use TopLink to manage any of the following types of data:

- relational (see "Relational Database Usage" on page 2-4)
- object-relational (see "Object-Relational Database Usage" on page 2-4)
- Oracle XDB (see "Oracle XML Database (XDB) Usage" on page 2-4)
- EIS, nonrelational, legacy data (see "Enterprise Information System (EIS) Usage" on page 2-4)
- XML and Web service data (see "XML Usage" on page 2-4)

Multiple Data Sources

If your application architecture must access more than one data source, Oracle recommends that you use a session broker (see "Session Broker and Client Sessions" on page 72-26) and JTA integration (see "JTA Controlled Transactions" on page 97-3) for two-phase commit.

Alternatively, you may use multiple sessions.

Isolating Data Access

If your application architecture requires that some data be restricted to a private cache and isolated from the TopLink shared session cache, Oracle recommends that you use an isolated session (see "Isolated Client Sessions" on page 72-19). You can also use an isolated session with the Oracle Virtual Private Database (VPD) feature (see "Isolated Client Sessions and Oracle Virtual Private Database (VPD)" on page 72-21).

Historical Data Access

If your data source maintains past or historical versions of objects, Oracle recommends that you use a TopLink historical session (see "Historical Sessions" on page 72-25) to access this historical data so that you can express read queries conditional on how your objects are changing over time.

Caching

This section describes choices you need to make when deciding on how to use the TopLink cache (see Chapter 87, "Understanding the Cache") in your application architecture.

These choices can be summarized as follows:

- Cache Type

- Refreshing
- Cache Coordination
  - Protocol
  - Synchronization

See also "Locking" on page 2-11.

Cache Type

Choose a cache type (see "Cache Type and Object Identity" on page 87-3) appropriate for the type of data your application processes. For example, consider a weak identity map for volatile data (see "Guidelines for Configuring the Cache and Identity Maps" on page 87-4).

Refreshing

Consider how your application architecture may be affected by stale data (see "Handling Stale Data" on page 87-6): for example, consider using query or descriptor refresh options (see "Refreshing" on page 2-10) or cache invalidation (see "Cache Invalidation" on page 87-8). Consider using an isolated session's cache (see "Isolated Client Sessions" on page 72-19) for volatile data.

Avoid using no identity map (see"No Identity Map" on page 87-4) for objects that are involved in relationships or that require object identity.

Cache Coordination

TopLink provides a distributed cache coordination feature that allows multiple, possibly distributed, instances of a session to broadcast object changes among each other so that each session's cache is kept up to date (see "Understanding Cache Coordination" on page 87-10). Before using cache coordination, ensure that it is appropriate for your application (see "When to use Cache Coordination" on page 87-11).

Protocol You can configure a coordinated cache to broadcast changes using any of the following communication protocols:

- Java Message Service (JMS)–Oracle recommends using a JMS coordinated cache (see "JMS Coordinated Cache" on page 87-12).
- Remote Method Invocation (RMI)–Oracle recommends that you use RMI cache coordination only if you require synchronous change propagation (see "Configuring the Synchronous Change Propagation Mode" on page 88-2). For more information, see "RMI Coordinated Cache" on page 87-12.
- Common Object Request Broker Architecture (CORBA)–Currently, TopLink provides support for the Sun ORB (see "CORBA Coordinated Cache" on page 87-13).

Synchronization You can configure synchronization strategy that a coordinated cache uses to determine what it broadcasts when an object changes. You can configure this at the project (see "Configuring Cache Coordination Change Propagation at the Project Level" on page 19-17) or descriptor ("Configuring Cache Coordination Change Propagation at the Descriptor Level" on page 25-40) level as follows:

- invalidate changed objects–Propagate an object invalidation that marks the object as invalid in all other sessions. This tells other sessions that they must update their

cache from the data source the next time this object is read. Oracle recommends using this synchronization strategy.

- synchronize changes–Propagate a change notification that contains each changed attribute.
- synchronize changes and new objects–Propagate a change notification that contains each changed attribute. For new objects, propagate an object creation (along with all the new instance's attributes).

Locking

This section describes choices you need to make when deciding on how to use TopLink locking options in your application architecture. Oracle strongly recommends always using a locking policy in a concurrent system (see "Configuring Locking Policy" on page 25-64).

These choices can be summarized as follows:

- Optimistic Locking
- Pessimistic Locking

If you are building a three-tier application, be aware of how that architecture affects the way you use locking (see "Locking in a Three-Tier Application" on page 23-22).

For more information, see "Understanding Descriptors and Locking" on page 23-18.

Optimistic Locking

Oracle recommends using TopLink optimistic locking. With optimistic locking, all users have read access to the data. When a user attempts to write a change, the application checks to ensure the data has not changed since the user read the data.

You can use version (see "Optimistic Version Locking Policies" on page 23-18) or field (see "Optimistic Field Locking Policies" on page 23-20) locking policies. Oracle recommends using version locking policies.

Pessimistic Locking

With pessimistic locking, the first user who accesses the data with the purpose of updating it locks the data until completing the update. The disadvantage of this approach is that it may lead to reduced concurrency and deadlocks.

Consider using pessimistic locking support at the query level (see "Configuring Named Query Options" on page 25-22).

If are using CMP, you may consider using bean-level pessimistic locking support (see "Configuring Named Query Options" on page 25-22).

Building and Using the Persistence Layer

Oracle TopLink requires that classes must meet certain minimum requirements before they can become persistent. TopLink also provides alternatives to most requirements. TopLink uses a nonintrusive approach by employing a metadata architecture that allows for minimal object model intrusions.

This section includes the following information:

- Implementation Options
- Persistent Class Requirements

- Persistence Layer Components
- Using the Persistence Layer

Implementation Options

Persistence layer components may be generated as metadata (see "Understanding TopLink Metadata" on page 2-19) from TopLink Workbench, or expressed as Java classes.

Oracle recommends using TopLink Workbench to create the necessary metadata (stored as XML). You can easily export and update the `project.xml` and `sessions.xml` files. This reduces development effort by eliminating the need to regenerate and recompile Java code each time you change the project. With TopLink Workbench, you write Java code only for your own application classes and any necessary amendment methods. For information about the XML structure of the `project.xml` and `sessions.xml` files, refer to the appropriate XML schemas (XSD) in the `<TOPLINK_HOME>\config\xsds` directory.

To use Java code, you must manually write code for each element of the TopLink project including: project, login, platform, descriptors, and mappings. This may be more efficient if your application is model-based and relies heavily on code generation. Depending on the type of project you are creating, TopLink Workbench can export Java code for projects, tables, and your model source (see "Exporting Project Information" on page 18-13).

Persistent Class Requirements

The following requirements apply to plain Java objects.

Note: for EJB 2.0 entity beans with container-managed persistence, the bean requirements are defined by the EJB 2.0 specification; for EJB 2.1 entity beans with container-managed persistence, the bean requirements are defined by the EJB 2.1 specification.

You can use direct access on private or protected attributes. For more information on direct and method access, see "Configuring Method Accessing" on page 32-14.

When using *nontransparent* indirection, the attributes must be of the type `ValueHolderInterface` rather than the original attribute type. The value holder does not instantiate a referenced object until it is needed.

TopLink provides *transparent* indirection for `Collection`, `List`, `Set`, and `Map` attribute types for any collection mappings. Using transparent indirection does not require the use of the `ValueHolderInterface` or any other object model requirements.

See "Indirection" on page 30-5 for more information on indirection and transparent indirection.

Persistence Layer Components

Typically, the TopLink persistence layer contains the following components:

- Mapping Metadata
- Session Metadata
- Cache

- Queries and Expressions
- Transactions

Mapping Metadata

The TopLink application metadata model is based on the TopLink project. The project includes descriptors, mappings, and various policies that customize the run-time capabilities. You associate this mapping and configuration information with a particular data source and application by referencing the project from a session.

For more information, see the following:

- "Creating Project Metadata" on page 2-20
- Chapter 17, "Understanding Projects"
- Chapter 23, "Understanding Descriptors"
- Chapter 30, "Understanding Mappings"

Session Metadata

A session is the primary interface between the client application and TopLink, and represents the connection to the underlying data source.

TopLink offers several different session types (see Chapter 72, "Understanding TopLink Sessions"), each optimized for different design requirements and architectures. The most commonly used session is the server session, a session that clients access on the server through a client session. The server session provides a shared cache and shared connection resources. You define a session with session metadata.

For CMP projects, the TopLink run-time creates and uses a session internally, but your application does not acquire or use this session directly. Depending on the application server you use, you can specify some of the parameters for this internal session.

For more information, see the following:

- "Creating Session Metadata" on page 2-21
- "Using the Persistence Layer" on page 2-14

Cache

By default, a TopLink session provides an object-level cache that guarantees object identity and enhances performance by reducing the number of times the application needs to access the data source. TopLink provides a variety of cache options, including locking, refresh, invalidation, isolation, and coordination. Using cache coordination, you can configure TopLink to synchronize changes with other instances of the deployed application. You configure most cache options at the session level. You can also configure cache options on a per-query basis or on a descriptor to apply to all queries on the reference class.

For more information, see Chapter 87, "Understanding the Cache".

Queries and Expressions

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

- TopLink expressions
- EJB Query Language (QL)

- SQL
- Stored procedures
- Query by example

With these options, you can build any type of query. Oracle recommends using 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.

When using entity beans, you can code finders completely using EJB QL (in addition to any of the other TopLink query options), enabling the application to comply with the J2EE specification.

Regardless of the architecture or persistent entity type, you are free to use any of the query options. TopLink Workbench provides the simplest way to define queries. Alternatively, you can build queries in code, using the TopLink API.

For more information, see Chapter 93, "Understanding TopLink Queries" and Chapter 95, "Understanding TopLink Expressions".

Transactions

TopLink provides the ability to write transactional code isolated from the underlying database and schema by using a **unit of work**, a specific transactional session.

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 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 coordinated cache.

If an application uses entity beans, you do not access the unit of work API directly, but you still benefit from its features: the integration between the TopLink runtime and the J2EE container automatically uses the unit of work to the application's best advantage.

For more information, see Chapter 93, "Understanding TopLink Queries".

Using the Persistence Layer

At run time, your application uses the TopLink metadata (see "Understanding TopLink Metadata" on page 2-19).

For a non-CMP project, your application loads a `session.xml` file at run time using the session manager (see Chapter 75, "Acquiring and Using Sessions at Run Time"). The `session.xml` file contains a reference to the mapping metadata `project.xml` file. Using the session, your application accesses the TopLink runtime and the `project.xml` mapping metadata.

For a CMP project, the metadata required is dependent upon the J2EE application server you deploy your application to (see "Creating TopLink Files for Deployment" on page 8-1). All application servers require an `ejb-jar.xml` and a TopLink project XML file. The session configuration is dependent on the specific J2EE application server.

Deploying the Application

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

This section discusses packaging and deployment from a TopLink perspective. However, if you deploy your application to a J2EE container, you must configure elements of your application to enable TopLink container support.

This section includes the following information:

- Understanding Deployments
- TopLink in a J2EE Application

For more information, see Part III, "Deploying a TopLink Application".

Understanding Deployments

The TopLink approach to deployment involves packaging application files into a single file, such as aJAR file, or an EAR file. This approach lets you create clean and self-contained deployments that do not require significant file management.

After creating these files, deploy the project.

TopLink in a J2EE Application

Although TopLink is an integral part of a J2EE application, in most cases the client does not interact with TopLink directly. Instead, TopLink features are invoked indirectly by way of EJB container callbacks.

As a result, the typical deployment process involves the following steps:

1. Build the project elements, including beans, classes, and data sources.
2. Define the application mappings in TopLink Workbench.
3. Build the application deployment files. Use TopLink Workbench to create the files.
4. Package and deploy the application.
5. Add code to the client application to enable it to access the TopLink application.

Optimizing and Customizing the Application

TopLink provides a diverse set of features to optimize performance including the following:

- Enhancing queries
- Tuning the cache
- Scaling to multiple server configuration

You enable or disable most features in the descriptors or session, making any resulting performance gains global.

Using TopLink EIS (see "Enterprise Information System (EIS) Usage" on page 2-4), you can integrate a TopLink application with legacy data sources using a J2C adapter. This

is the most efficient way to customize a TopLink application to accommodate unusual or nonstandard systems.

Using TopLink XML (see "XML Usage" on page 2-4), you can integrate a TopLink application with legacy data sources using a Web service.

See Part IV, "Optimizing and Customizing a TopLink Application" for details on optimizing and customizing TopLink.

Troubleshooting the Application

See Part V, "Troubleshooting a TopLink Application" for detailed information on troubleshooting all aspects of a TopLink application including development and deployment.

Understanding Object Persistence

This section includes a brief description of relational mapping and provides important information and restrictions to guide object and relational modeling. This information is useful when building TopLink applications.

This section includes information on the following:

- Application Object Model
- Data Storage Schema
- Primary Keys and Object Identity
- Mappings
- Foreign Keys and Object Relationships
- Inheritance
- Concurrency
- Caching
- Nonintrusive Persistence
- Indirection

These sections contain additional detail on these features, and explain how to implement and use them with TopLink.

Application Object Model

Object modeling refers to the design of the Java classes that represent your application objects. With TopLink, you can use your favorite integrated development environment (IDE) or Unified Modeling Language (UML) modeling tool to define and create your application object model.

Any class that registers a descriptor with a TopLink database session is called a persistent class. TopLink does not require that persistent classes provide public accessor methods for any private or protected attributes stored in the database. Refer to "Persistent Class Requirements" on page 2-12 for more information.

Data Storage Schema

Your data storage schema refers to the design that you implement to organize the persistent data in your application. This schema refers to the data itself–not the actual data source (such as a relational database or nonrelational legacy system).

During the design phase of the TopLink application development process (see "Typical Development Stages" on page 2-2), you should decide how to implement the classes in the data source. When integrating existing data source information, you must determine how the classes relate to the existing data. If no legacy information exists to integrate, decide how you will store each class, then create the necessary schema.

You can also use TopLink Workbench (see Chapter 4) or database schema manager (see Chapter 5) to create the necessary information.

Primary Keys and Object Identity

When making objects persistent, each object requires an *identity* to uniquely identify it for storage and retrieval. Object identity is typically implemented using a unique primary key. This key is used internally by TopLink to identify each object, and to create and manage references. Violating object identity can corrupt the object model.

In a Java application, object identity is preserved if each object in memory is represented by one, and only one, object instance. Multiple retrievals of the same object return references to the same object instance–not multiple copies of the same object.

TopLink supports multiple identity maps to maintain object identity (including composite primary keys). Refer to "Cache Type and Object Identity" on page 87-3 for additional information.

Mappings

TopLink uses the metadata produced by TopLink Workbench (see "Understanding TopLink Metadata" on page 2-19) to describe how objects and beans map to the data source. This approach isolates persistence information from the object model–you are free to design their ideal object model, and DBAs are free to design their ideal schema.

You use TopLink Workbench to create and manage the mapping information. At run time, TopLink uses the metadata to seamlessly and dynamically interact with the data source, as required by the application.

TopLink provides an extensive mapping hierarchy that supports the wide variety of data types and references that an object model might contain. For more information, see Chapter 30, "Understanding Mappings".

Foreign Keys and Object Relationships

A **foreign key** is a combination of columns that reference a unique key, usually the primary key, in another table. Foreign keys can be any number of fields (similar to primary key), all of which are treated as a unit. A foreign key and the primary parent key it references must have the same number and type of fields.

Foreign keys represents relationships from a column or columns in one table to a column or columns in another table. For example, if every `Employee` has an attribute `address` that contains an instance of `Address` (which has its own descriptor and table), the one-to-one mapping for the `address` attribute would specify foreign key information to find an address for a particular `Employee`.

Refer to "Configuring Table and Field References (Foreign and Target Foreign Keys)" on page 34-8 for more information.

Inheritance

Object-oriented systems allow classes to be defined in terms of other classes. For example: motorcycles, sedans, and vans are all *kinds of vehicles*. Each of the vehicle types is a *subclass* of the `Vehicle` class. Similarly, the `Vehicle` class is the *superclass* of each specific vehicle type. Each subclass inherits attributes and methods from its superclass (in addition to having its own attributes and methods).

Inheritance provides several application benefits, including the following:

- Using subclasses to provide specialized behaviors from the basis of common elements provided by the superclass. By using inheritance, you can reuse the code in the superclass many times.
- Implementing *abstract* superclasses that define generic behaviors. This abstract superclass may define and partially implement behavior, while allowing you to complete the details with specialized subclasses.

Refer to "Configuring Inheritance for a Child (Branch or Leaf) Class Descriptor" on page 25-51 and "Configuring Inherited Attribute Mapping in a Subclass" on page 25-58 for detailed information on using inheritance with TopLink.

Concurrency

To have concurrent clients logged in at the same time, the server must spawn a dedicated thread of execution for each client. J2EE application servers do this automatically. Dedicated threads enable each client to work without having to wait for the completion of other clients. TopLink ensures that these threads do not interfere with each other when they make changes to the identity map or perform database transactions.

Using the TopLink `UnitOfWork` class, your client can make transactional changes in an isolated and thread safe manner. The unit of work manages clones for the objects you modify to isolate each client's work from other concurrent clients and threads. The unit of work is essentially an object-level transaction mechanism that maintains all of the ACID (Atomicity, Consistency, Isolation, Durability) transaction principles as a database transaction. For more information on the unit of work, see Chapter 97, "Understanding TopLink Transactions".

TopLink supports configurable optimistic and pessimistic locking strategies to let you customize the type of locking that the TopLink concurrency manager uses. For more information, see "Understanding Descriptors and Locking" on page 23-18.

Caching

TopLink caching improves application performance by automatically storing data returned as objects from the database for future use. This caching provides several advantages:

- Reusing Java objects that have been previously read from the database minimizes database access
- Minimizing SQL calls to the database when objects already exist in the cache
- Minimizing network access to the database
- Setting caching policies a class-by-class and bean-by-bean basis

- Basing caching options and behavior on Java garbage collection

TopLink supports several caching polices to provide extensive flexibility. You can fine-tune the cache for maximum performance, based on individual application performance. Refer to Part XVII, "Cache" for complete information.

Nonintrusive Persistence

The TopLink nonintrusive approach of achieving persistence through a metadata architecture (see "Understanding TopLink Metadata" on page 2-19) means that there are almost no object model intrusions.

To persist Java objects, TopLink does not require any of the following:

- Persistent superclass or implementation of persistent interfaces
- Store, delete, or load methods required in the object model
- Special persistence methods
- Generating source code into or wrapping the object model

When using entity beans with container-managed persistence, TopLink does not require any additional intrusion to the object model, other than the CMP specification requirements.

See "Building and Using the Persistence Layer" on page 2-11 for additional information on this nonintrusive approach.

Indirection

An indirection object takes the place of an application object so the application object is not read from the database until it is needed. Using indirection allows TopLink to create *stand-ins* for related objects. This results in significant performance improvements, especially when the application requires the contents of only the retrieved object rather than all related objects.

Without indirection, each time the application retrieves a persistent object, it also retrieves *all* the objects referenced by that object. This may result in lower performance for some applications.

Note: Oracle strongly recommends that you use indirection in all situations.

TopLink provides several indirection models, such as proxy indirection, transparent indirection, and value holder indirection. TopLink also provides indirection support for EJB (see "Indirection and EJB" on page 30-9).

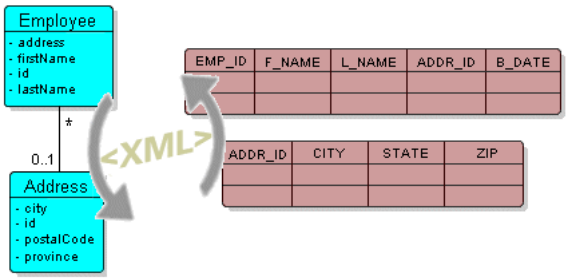
See "Indirection" on page 30-5 for more information.

Understanding TopLink Metadata

The TopLink metadata is the bridge between the development of an application and its deployed run-time environment. Capture the metadata using TopLink Workbench (see "Creating Project Metadata" on page 2-20 and "Creating Session Metadata" on page 2-21), and pass the metadata to the run-time environment using deployment XML files (such as `project.xml` and `sessions.xml`). You could also manually code these files using Java and the TopLink API, but this approach is more labor-intensive.

The metadata, encapsulated in deployment XML files, lets you pass configuration information into the run-time environment. The run-time environment uses the information in conjunction with the persistent entities (Java objects or EJB entity beans) and the code written with the TopLink API, to complete the application.

Figure 2–2 TopLink Metadata



This section describes the following:

- Advantages of the TopLink Metadata Architecture
- Creating Project Metadata
- Creating Session Metadata
- Deploying Metadata

Advantages of the TopLink Metadata Architecture

The TopLink metadata architecture provides many important benefits, including the following:

- Stores mapping information in XML descriptors–not in the domain model objects
- By using the metadata, TopLink does not intrude in the object model or the database schema
- Allows you to design the object model as needed, without forcing any specific design
- Allows DBAs to design the database as needed, without forcing any specific design
- Does not rely on code-generation (which can cause serious design, implementation, and maintenance issues)
- Is unintrusive: adapts to the object model and database schema, rather than requiring you to design their object model or database schema to suit TopLink

Creating Project Metadata

A TopLink project contains the mapping metadata that the TopLink runtime uses to map objects to a data source. The project is the primary object used by the TopLink runtime.

This section describes the principal contents of project metadata, including the following:

- Descriptors and Mappings
- Data Source Login Information

For more information about creating `project.xml` metadata, see "project.xml File" on page 8-2.

Descriptors and Mappings

TopLink maps persistent entities to the database in the application, using the descriptors and mappings you build with TopLink Workbench. TopLink Workbench supports several approaches to project development, including the following:

- 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

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

Although TopLink Workbench does offer the ability to generate persistent entities or the relational model components for an application, these utilities are intended only to assist in rapid initial development strategies–not complete round-trip application development.

For more information, see Chapter 23, "Understanding Descriptors" and Chapter 30, "Understanding Mappings".

Amending Descriptors An amendment method lets you implement a TopLink feature that is not currently supported by TopLink Workbench. Simply write a Java method to amend the descriptor after it is loaded, and specify the method in TopLink Workbench for inclusion in the project metadata. See "Configuring Amendment Methods" on page 25-81 for detailed information on implementing an amendment method for a TopLink descriptor.

Data Source Login Information

For non-CMP projects, you configure a session login in the session metadata that specifies the information required to access the data source (see "Creating Session Metadata" on page 2-21).

For CMP projects, the project contains a deployment login that specifies the information required to access the data source.

For more information, see "Projects and Login" on page 17-3.

Creating Session Metadata

A TopLink session contains a reference to a particular `project.xml` file, plus the information required to access the data source. The session is the primary object used by your application to access the features of the TopLink runtime.

The agent responsible for creating and accessing session metadata differs depending on whether or not you are creating a CMP project. In a non-CMP project, your application acquires and accesses a session directly (see "Non-CMP Applications and Session Metadata" on page 8-4). In a CMP project, your application indirectly accesses a session acquired internally by the TopLink runtime (see "CMP Applications and Session Metadata" on page 8-4).

Deploying Metadata

The `project.xml` and `sessions.xml` file are packaged for deployment differently according to the type of application you are deploying.

For more information, see the following:

- "Creating TopLink Files for Deployment" on page 8-1
- "Packaging a TopLink Application" on page 9-1

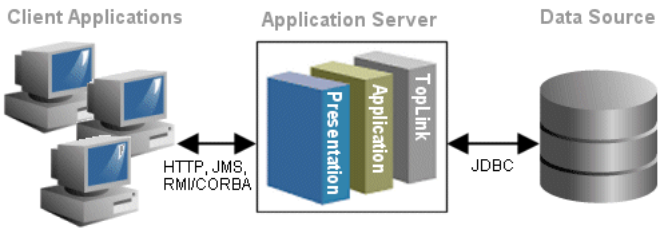
Understanding the Three-Tier Architecture

The three-tier Web application architecture generally includes the connection of a server-side Java application to the database through a JDBC connection (see Figure 2–3). In this pattern, 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 Common Object Request Broker Architecture (CORBA).

The three-tier application is a common architecture in which 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 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 2–3 Three Tier Architecture



Although not all three-tier applications are Web-based, this architecture is ideally suited to distributed Web applications. In addition, although it is also common to use EJB in a Web application, this TopLink architecture does not.

Example Implementations

Examples of three-tier architecture implementation include the following:

- A Model-View-Controller Model 2 architectural design pattern that runs in a J2EE container with servlets and JSP that uses TopLink to access data without EJB.
- A Swing or Abstract Window Toolkit (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 it is less standard than EJB.

Variation Using Remote Sessions

TopLink includes a session type called remote session. The session offers the full session API and contains a cache of its own, but exists on the client system rather than on the TopLink server. Communications can be configured to use RMI or RMI-Internet Inter-Object Request Broker Protocol (IIOP).

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

Although this is an excellent option for you if you wish to simplify the 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.

For more information, see "Remote Sessions" on page 72-30.

Technical Challenges

The three-tier application with a stateless client presents several technical challenges, including the following:

- Transaction management in a stateless environment

  A common design practice is to delimit client requests within a single unit of work (transactional session). 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, then 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 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. In this case, transport the optimistic lock values to the client in the HTTP contents of an edit page. You 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 "Configuring Locking Policy" on page 25-64.

- External JDBC pools

  By default, TopLink manages its own connection pools. You can also configure TopLink to use connection pooling offered by the host application server. This feature is useful for shared connection pools and is required for JTA/JTS integration (see "Configuring External Connection Pooling" on page 82-2).

- JTA/JTS Integration

  JTA and JTS are standard Java components that enable sessions to participate in distributed transactions. You must configure TopLink to use JTA/JTS to use session beans in the architecture (see "Integrating the Unit of Work With an External Transaction Service" on page 99-21).

- Cache coordination

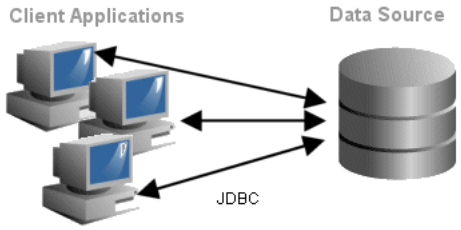
  If you choose to use multiple servers to scale your application, you may require TopLink cache coordination (see Chapter 87, "Understanding the Cache").

Understanding the Two-Tier Architecture

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

For more information, see Chapter 72, "Understanding TopLink Sessions".

Figure 2–4 Two-Tier Architecture



Although the two-tier architecture is the simplest 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.

Two-tier applications are often implemented as user interfaces that directly access the database (see Figure 2–4). They can also be non-interface processing engines. In either case, the two-tier model is not as common as the three-tier model.

The following are key elements of an efficient two-tier (client-server) architecture with 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 TopLink database session that builds the two-tier architecture provides all the TopLink features in a single session type, thereby making the two-tier architecture simple to build and use.

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

Technical Challenges

The current trend toward multitiered Web applications makes the two-tier architecture less common in production systems, but no less viable. 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 increases.

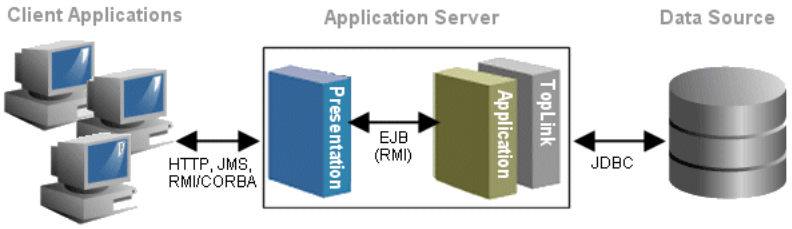
To minimize this problem, TopLink offers support for several data locking strategies. These include pessimistic locking and several variations of optimistic locking. For more information, see "Configuring Locking Policy" on page 25-64.

Understanding the EJB Session Bean Facade Architecture

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

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

Figure 2–5 Three-Tier Architecture Using Session Beans and Java Objects



A common extension to the three-tier architecture is to combine session beans and persistent Java objects managed by TopLink. The resulting application includes session beans and Java objects on a TopLink three-tier architecture (see Figure 2–5).

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 a TopLink session, the bean obtains a client session from the shared server session. This architecture has the following key features:

- Session beans delimit transactions.

  Configure 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 the objects 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 Model 2 architectural design pattern that runs in a J2EE container with servlets and JSP and uses the session bean enabled by TopLink to access data without EJB.

Advantages and Disadvantages

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

- Less overhead than an EJB entity bean application

  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 lets you 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 challenges with inheritance, indirection, and relationships.

For more information about managing inheritance, indirection and relationships, see Part IX, "Mappings".

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 Java objects enabled by TopLink. Session beans execute all operations related to TopLink on behalf of the client.

The EJB specifications describe 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

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

- Stateless session beans and the TopLink session

  Stateless beans store no information between method calls from the client. As a result, reestablish the connection of the bean to the session for each client method call. Each method call through TopLink obtains a client session, makes the appropriate calls, and releases the reference to the client session.

- Stateful session beans and the 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 a stateful session bean enabled by TopLink 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 a TopLink session instance does not survive passivation. To maintain the session between method calls, release the session during the passivation process and re-obtain it when you reactivate the bean.

- External JDBC pools

  By default, TopLink manages its own connection pools. For the session bean architecture, you must configure TopLink to use connection pooling offered by the host application server. This feature is useful for shared connection pools and is required for JTA/JTS integration (see "Configuring External Connection Pooling" on page 82-2).

- JTA/JTS integration

  JTA and JTS are standard Java components that enable sessions to participate in distributed transactions. You must configure TopLink to use JTA/JTS to use session beans in the architecture (see "Integrating the Unit of Work With an External Transaction Service" on page 99-21).

- Cache coordination

  If you choose to use multiple servers to scale your application, you may require TopLink cache coordination (see "Understanding Cache Coordination" on page 87-10).

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 99-13.

Understanding the EJB Entity Beans With CMP Architecture

CMP is the part of the J2EE component model that provides an object persistence service that an EJB container uses to persist entity beans. CMP provides distributed, transactional, secure access to persistent data, with a guaranteed portable interface.

This architecture is an extension of the three-tier architecture, in which the implementation of persistence methods is handled by the container at runtime. As a bean provider, you only need to specify in a deployment descriptor those persistent fields and relationships for which the container must handle data access and, optionally, an abstract representation of the database schema.

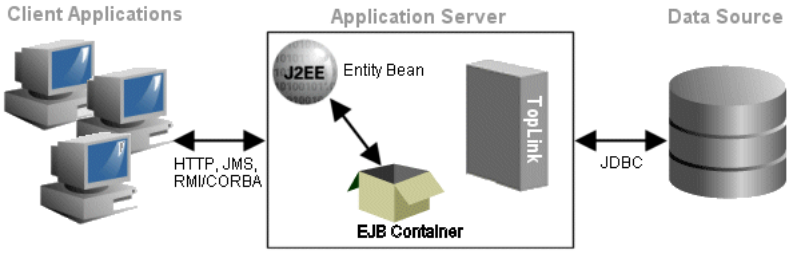
TopLink CMP is an extension of the TopLink persistence framework that provides custom integration to EJB containers of various application servers (see "Application Server Support" on page 7-1). For more information about choosing an application server, see "Understanding Target Platforms" on page 2-5. TopLink integrates with the EJB container in this architecture to augment (or, in the case of OC4J, become) the container's persistence manager.

Note: When using OC4J and Java 1.5, TopLink supports a subset of the persistence features anticipated in the final EJB 3.0 specification. For more information on EJB 3.0 support, see *Oracle Containers for J2EE Enterprise JavaBeans Developer's Guide*.

EJB 3.0 feature support is subject to change and dependent upon the contents of the final specification.

TopLink CMP integration is nonintrusive (see Figure 2–6). Through a combination of run-time integration and code generation, the container uses TopLink internally and the bean user interacts with entity beans with container-managed persistence according to their standard API. This lets you combine the standard interfaces and power of CMP and a container with TopLink flexibility, performance, and productivity.

Figure 2–6 Three-Tier CMP Architecture



For more information, see the following:

- "Integrating TopLink With an Application Server" on page 7-1
- "Creating TopLink Files for Deployment" on page 8-1
- "Packaging a TopLink Application" on page 9-1
- "Deploying a TopLink Application" on page 10-1
- "Configuring Persistence Type" on page 19-5

Example Implementation

An example of the entity beans with container-managed persistence implementation is a Model-View-Controller Model 2 architectural design pattern that runs in a J2EE container, with servlets and JSP that access either session beans or entity beans with container-managed persistence enhanced by TopLink.

Advantages and Disadvantages

A three-tier architecture using entity beans with container-managed persistence offers the following advantages:

- It allows for entity beans with container-managed persistence supplied with sophisticated TopLink features such as caching and mapping support, storing bean data across more than one table, composite primary keys, and data conversion.
- It presents a standard method to access data, which lets you create standardized, reusable business objects.

- It is well-suited to create coarse-grained objects, which TopLink relates to dependent, lightweight, regular Java objects (TopLink can also manage container-managed relationships to lightweight dependent Java objects).
- TopLink provides for lazy initialization of referenced objects and beans (see "Indirection" on page 30-5).
- TopLink provides functionality for transactional copies of beans, allowing concurrent access by several clients, rather than relying on individual serialization.
- TopLink provides advanced query capabilities, as well as dynamic querying, including the ability to define queries at the bean-level rather than the data source level and to use a rich set of querying and finder options.
- TopLink maintains bean and object identity.

The disadvantage of this architecture is that pure entity bean with container-managed persistence 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 TopLink integration with the application server or J2EE container.

Other issues include the following:

- External JDBC Pools
- JTA/JTS Integration
- Cache Coordination
- Maintaining Bidirectional Relationships
- Managing Dependent Objects
- Managing Collections of EJBObject Objects

External JDBC Pools

By default, TopLink manages its own connection pools. You can also configure TopLink to use connection pooling offered by the host application server. This feature is useful for shared connection pools and is required for JTA/JTS integration (see "Configuring External Connection Pooling" on page 82-2).

JTA/JTS Integration

JTA and JTS are standard Java components that enable sessions to participate in distributed transactions. You must configure TopLink to use JTA/JTS to use session beans in the architecture (see "Integrating the Unit of Work With an External Transaction Service" on page 99-21).

Cache Coordination

If you choose to use multiple servers to scale your application, you may require TopLink cache coordination (see "Understanding Cache Coordination" on page 87-10).

Maintaining Bidirectional Relationships

When one-to-one or many-to-many relationship is bidirectional, you must maintain the back pointers as the relationships change.

TopLink automatically maintains the relationship between two entity beans.

To set the back pointer manually, do one of the following:

- Code the entity bean to maintain the back pointer when the relationship is established or modified (recommended).
- Code the client to explicitly set the back pointer.

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

In a one-to-many relationship, a source bean might have several dependent target objects. For example, an `EmployeeBean` might own several dependent `PhoneNumber` instances. When you add a new dependent object (a `PhoneNumber`, in this example) to an employee, you must set the `PhoneNumber` instance's back pointer to its owner (the employee). 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 as Example 2–1 shows.

Example 2–1 Setting the Back-Pointer in the Entity Bean

```
// 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);

// Maintain the relationship in the Employee's addPhoneNumber method
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);
}
```

For more information, see the following:

- Configuring Bidirectional Relationship on page 32-34
- "Directionality" on page 33-2

Managing Dependent Objects

Unlike EJB, TopLink dependent persistent objects can be sent back and forth between a client and a 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 setter methods when adding dependent objects to relationship collections as Example 2–2 shows. This enables TopLink to handle merging of objects in the cache.

Example 2–2 Managing Dependent Objects

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

Managing Collections of EJBObject Objects

Collections generally use the `equals` method to compare objects. This is not a problem in the case of an object that contains a collection of `EJBObject` objects, because the EJB container collection handles equality appropriately.

Understanding the EJB Entity Beans With BMP Architecture

BMP is the part of the J2EE component model that lets you, the bean provider, implement the entity bean's persistence directly in the entity bean class or in one or more helper classes that you provide.

This architecture is an extension of the three-tier architecture, in which the persistent data is bean managed within an entity bean using code that you implement. The client code accesses the data through the entity bean interface.

TopLink BMP is an extension of the TopLink persistence framework that provides base class `BMPEntityBase` as a starting point for your BMP development. This class provides an implementation for all methods (except `ejbPassivate`) required by the EJB specifications prior to 3.0. Subclass `BMPEntityBase` to create a TopLink-enabled entity bean with bean-managed persistence.

To use the `BMPEntityBase` class, perform the following:

1. Create a TopLink session (see Chapter 72, "Understanding TopLink Sessions") for your application.
2. Add a `BMPWrapperPolicy` to each descriptor that represents an entity bean with bean-managed persistence.

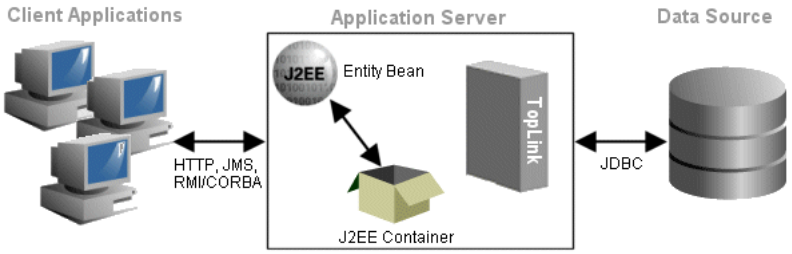
   The `BMPWrapperPolicy` provides TopLink with the information to create remote objects for entity beans and to extract the data out of a remote object.
3. Create the home and remote interfaces.
4. Create deployment descriptors (see "Integrating TopLink With an Application Server" on page 7-1 and "Creating TopLink Files for Deployment" on page 8-1).
5. Package your application (see "Packaging a TopLink Application" on page 9-1).
6. Deploy the beans (see "Deploying a TopLink Application" on page 10-1).

To make full use of TopLink session and unit of work features, TopLink provides a hook into its functionality through the `BMPDataStore` class. Use this class to translate EJB-required functionality into simple calls.

The `BMPDataStore` class provides implementations of `LOAD` and `STORE`, multiple finders, and `REMOVE` functionality. The `BMPDataStore` class requires a TopLink session. 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` method.

TopLink BMP support (see Figure 2–7) lets you combine the standard interfaces of entity beans with bean-managed persistence with TopLink flexibility, performance, and productivity.

Figure 2–7 Three-Tier BMP Architecture



TopLink supports BMP. To use BMP support, the home interface must inherit from the `oracle.toplink.ejb.EJB20Home`. To make calls to the `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`.

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.

For more information, see the following:

- "Integrating TopLink With an Application Server" on page 7-1
- "Creating TopLink Files for Deployment" on page 8-1
- "Packaging a TopLink Application" on page 9-1
- "Deploying a TopLink Application" on page 10-1
- "Configuring Persistence Type" on page 19-5
- "Configuring the Server Platform" on page 74-14

Example Implementations

An example of the entity beans with bean-managed persistence implementation is a Model-View-Controller Model 2 architectural design pattern that runs in a J2EE container, with servlets and JSP that access session beans and entity beans with bean-managed persistence enhanced by TopLink.

Advantages and Disadvantages

Using BMP with a 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.
- TopLink makes BMP easier to implement.
- It enables you 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.

The main disadvantages of BMP include the following:

- You must create the persistence mechanisms in the bean code.
- It is not as transparent or efficient as CMP.
- TopLink-only Java object applications offer the same degree of independence from the application server.

Technical Challenges

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

Other issues include the following:

- External JDBC Pools
- JTA/JTS Integration
- Cache Coordination

External JDBC Pools

By default, TopLink manages its own connection pools. You can also configure TopLink to use connection pooling offered by the host application server. This feature is useful for shared connection pools and is required for JTA/JTS integration (see "Configuring External Connection Pooling" on page 82-2).

JTA/JTS Integration

JTA and JTS are standard Java components that enable sessions to participate in distributed transactions. You must configure TopLink to use JTA/JTS to use session beans in the architecture (see "Integrating the Unit of Work With an External Transaction Service" on page 99-21).

Cache Coordination

If you choose to use multiple servers to scale your application, you may require TopLink cache coordination (see "Understanding Cache Coordination" on page 87-10).

Understanding the EJB 3.0 JPA Entity Architecture

A part of the EJB 3.0 specification, the Java Persistence API (JPA) is a lightweight, POJO-based framework for Java persistence. JPA focuses on object-relational mapping and contains a full object-relational mapping specification supporting the use of Java language metadata annotations and/or XML descriptors to define the mapping between Java objects and a relational database. Object-relational mapping with the JPA is completely metadata-driven. JPA supports a SQL-like query language for both static and dynamic queries. It also supports the use of pluggable persistence providers.

JPA includes the following concepts:

- Entity–any application-defined object with the following characteristics can be an entity:
  - it can be made persistent;

  - it has a persistent identity (a key that uniquely identifies an entity instance and distinguishes it from other instances of the same entity type. An entity has a persistent identity when there is a representation of it in a data store);
  - it is partially transactional in a sense that a persistence view of an entity is transactional (an entity is created, updated and deleted within a transaction, and a transaction is required for the changes to be committed in the database). However, in-memory entities can be changed without the changes being persisted.
  - it is not a primitive, a primitive wrapper, or built-in object. An entity is a fine-graned object that has a set of aggregated state that is typically stored in a single place (such as a row in a table), and have relationships to other entities.
- Entity metadata–describes every entity. Metadata could be expressed as annotations (specifically defined types that may be attached to or place in front of Java programming elements) or XML (descriptors).
- Entity manager–enables API calls to perform operations on an entity. Until an entity manager is used to create, read, or write an entity, the entity is just a regular nonpersistent Java object. When an entity manager obtains a reference to an entity, that entity becomes managed by the entity manager. The set of managed entity instances within an entity manager at any given time is called its persistence context–only one Java instance with the same persistent identity may exist in a persistence context at any time.

  You can configure an entity manager to be able to persist or manage certain types of objects, read or write to a particular database, and be implemented by a specific persistence provider. The persistence provider supplies the backing implementation engine for JPA, including the `EntityManager` interface implementation, the `Query` implementation, and the SQL generation.

  Entity managers are provided by an `EntityManagerFactory`. The configuration for an entity manager is bound to the `EntityManagerFactory`, but it is defined separately as a persistence unit. You name persistence units to allow differentiation between `EntityManagerFactory` objects. This way your application obtains control over which configuration to use for operations on a specific entity. The configuration that describes the persistence unit is defined in a `persistence.xml` file.

  The following description expresses relationships between JPA concepts:

  - `Persistence` creates one or more `EntityManagerFactory` objects;
  - each `EntityManagerFactory` is configured by one persistence unit;
  - `EntityManagerFactory` creates one or more `EntityManager` objects;
  - one or more `EntityManager` manages one `PersistenceContext`.

For more information, see the following:

- `http://www.oracle.com/technology/products/ias/toplink/jpa/index.html`
- *Oracle Containers for J2EE Enterprise JavaBeans Developer's Guide*

Example Implementations

An example of the entity beans with bean-managed persistence implementation is a Model-View-Controller Model 2 architectural design pattern that runs in a Java EE

container, with servlets and JSP that access session beans and EJB 3.0-compliant entities using the TopLink-based JPA persistence provider.

Advantages and Disadvantages

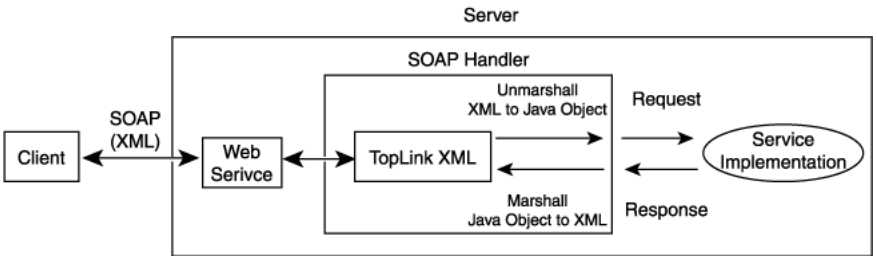
The use of TopLink JPA entities offers the following advantages:

- POJO persistence–in JPA, persistent objects are POJOs.
- Object-relational mapping is completely metadata-driven.
- The persistence API exists as a separate layer form the persistent objects and does not intrude upon them.
- Using the query framework you can query across entities and their relationships without having to use concrete foreign keys or database columns. Also, you can define queries statically in metadata or create them dynamically by passing query criteria on construction. Queries can return entities as results.
- Entities are mobile–objects are able to move from one JVM to another and back, and at the same time be usable by the application.
- You can configure persistence features through the use of Java SE 5 annotations, or XML, or a combination of both. You may also rely on defaults.
- If your application is running inside a container, the container provides support and ease of use; you can configure the same application to run outside a container.

Understanding the Web Services Architecture

A Web services architecture is similar to the three-tier (see "Understanding the Three-Tier Architecture" on page 2-22) or session bean (see "Understanding the EJB Session Bean Facade Architecture" on page 2-25) architecture, however, in a Web services architecture, you encapsulate business logic (the service) in a Web service instead of (or in addition to) using session beans. In a Web services architecture, clients communicate with your application using SOAP messages (XML over HTTP).

Figure 2–8 Web Services Architecture



As in any architecture, you can use TopLink to persist objects to relational or EIS data sources. However, in a Web services architecture, you can also use TopLink to map your object model to an XML schema for use with the Web service or as the Web service XML serializer.

Example Implementations

An example of a Web services architecture implementation is the use of a Web service to expose parts of an existing application to a remote client (typically another application) by way of SOAP messages. In this application, you can use TopLink XML

to unmarshall XML messages to Java objects to facilitate requests and marshall Java object responses back into XML for transmission to the client.

Advantages and Disadvantages

Using TopLink in Web services architecture has many advantages, including, but not limited to, the following:

- you can map XML messages to an existing Java object model.
- you can achieve a high level of complexity of mapping support
- compliance with the JAXB standards
- providing a scalable, high-performing solution

One debatable disadvantage is this solution's complexity over a simple RMI session bean service.

Technical Challenges

As with any technology, there are technical challenges associated with the use of TopLink in Web services architecture. These technical challenges are mostly related to special-case scenarios, such as when you need to implement a custom serializer because you have both the Java objects and the schema.

For more information, see the following:

- *Oracle TopLink as a Custom Serializer in a JAX-RPC 1.1 Web service* at `http://www.oracle.com/technology/products/ias/toplink/technical/tips/jaxRpc11/index.htm`
- Part XIII, "XML Mappings"

Part II

Using TopLink Development Tools

This part describes the development tools and tool support TopLink provides. It contains the following chapters.

- Chapter 3, "Understanding TopLink Development Tools"

  This chapter describes the development tools and tool support TopLink provides.

- Chapter 4, "Using TopLink Workbench"

  This chapter describes how to use TopLink Workbench including working with databases, generating data from database tables, and creating and editing a `sessions.xml` file.

- Chapter 6, "Using an Integrated Development Environment"

  This chapter explains how to integrate TopLink with an IDE. Detailed instructions are given for the Oracle JDeveloper IDE.

- Chapter 5, "Using the Schema Manager"

  This chapter explains how to use the TopLink schema manager to create databases, tables, stored procedures, and to populate database tables.

3

Understanding TopLink Development Tools

The TopLink runtime provides Java or J2EE applications with access to persistent entities stored in a data source. In addition to run-time capabilities, the TopLink Foundation Library includes the TopLink Application Programming Interface (API). This API enables applications to access TopLink run-time features.

TopLink includes additional development tools that simplify application development. These tools capture mapping and run-time configuration information in metadata files that TopLink passes to the application at run time.

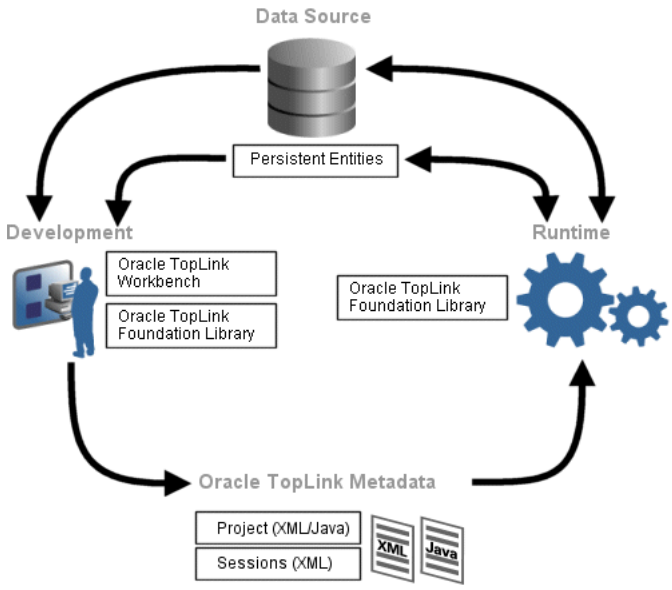
TopLink application development includes the following:

- Development Environment
- TopLink Run-Time Environment

TopLink metadata is the link between the two (see "Understanding TopLink Metadata" on page 2-19).

Figure 3–1 illustrates how these elements interact with the data source.

Figure 3–1 TopLink Components in Development Lifecycle

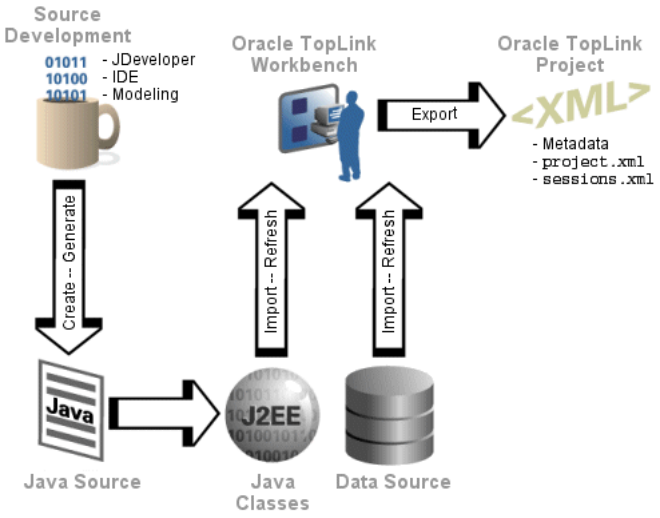


Development Environment

To create a TopLink application, use TopLink Workbench to map objects to data sources using relational and nonrelational models. Capture the resulting mappings and additional run-time configurations in the TopLink project file (`project.xml`) and build a session configuration file (`sessions.xml`). These files together represent your entire TopLink project, as shown in Figure 3–2.

During development, you can use the TopLink API to define query and transaction logic. When you use entity beans, there is generally little or no direct use of the TopLink API and there is no session or `sessions.xml` file.

Figure 3–2 TopLink Workbench in Development Environment



TopLink Workbench can import compiled entity classes (Java objects or EJB entity beans), as well as relational or nonrelational schemas through a JDBC driver (configured by the developer). Because TopLink imports the object and relational models for mapping, developers can develop the two models relatively independently from the mapping phase of a project development.

TopLink Run-Time Environment

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

In addition to J2EE environments, TopLink fully supports non-J2EE environments as well. See "Selecting an Architecture With TopLink" on page 2-5 for more information.

4

Using TopLink Workbench

This chapter provides information about understanding, using, and customizing TopLink Workbench.

This chapter includes the following sections:

- Understanding TopLink Workbench
- Configuring the TopLink Workbench Environment
- Working With TopLink Workbench
- Working With TopLink Workbench Preferences
- Working With Databases
- Working With XML Schemas
- Working With Classes
- Integrating TopLink Workbench With Apache Ant

For information on using TopLink Workbench to configure sessions XML, refer to Part XV, "TopLink Sessions".

Understanding TopLink Workbench

TopLink Workbench is a separate component from the TopLink runtime–it lets you graphically configure descriptors and map your project. TopLink Workbench can verify the descriptor options, access the data source (either a database or an XML schema), and create the database schema. Using TopLink Workbench, you can define TopLink descriptors and configurations *without using code*.

TopLink Workbench can be used during the *development* phase of the development process (see "Developing Your Application With TopLink" on page 2-1). Typically, this phase includes the following:

1. Defining an object model (a set of Java classes) to describe and solve your problem.
2. Creating a TopLink Workbench project, importing your Java classes and data sources, and using descriptors to describe how the Java classes map to your data source model.
3. Creating a TopLink session and registering your descriptors. In your application, use the session to retrieve and store objects from and to the data source.

TopLink Workbench creates a `<projectName>.mwp` file to store all TopLink project information, including object model, descriptor, and session information.

The <*projectName*>.mwp file is used only by TopLink Workbench. Typically, the only time you need to modify the <*projectName*>.mwp file is to merge changes during application development by a team of developers ("Merging Files" on page 6-4).

Using TopLink Workbench, you export this information into a `project.xml` file that your TopLink enabled application reads at run time.

For more information on using TopLink Workbench as the development environment, see Figure 3–2 on page 3-2.

Configuring the TopLink Workbench Environment

TopLink Workbench reads its environment variables from the `setenv` script in the `<TOPLINK_HOME>\bin` directory.

Before you launch TopLink Workbench, you must configure its environment as follows:

1. Use a text editor to open the `<TOPLINK_HOME>\bin\setenv` script.
   - For Windows, open the `setenv.cmd` file.
   - For UNIX, open the `setenv.sh` file.
2. Ensure that the `JAVA_HOME` environment variable is set:
   - For Windows: `set JAVA_HOME=C:/j2sdk1.4.2_04`
   - For UNIX: `JAVA_HOME=/usr/local/packages/java; export JAVA_HOME`
3. Update the `DRIVER_CLASSPATH` environment variable to add the location of the following (if necessary):

   Note: Do not include any Java classes for your persistent business objects in the `DRIVER_CLASSPATH` variable. Instead, add these persistent business objects in your TopLink Workbench project classpath (see "Configuring Project Classpath" on page 19-3).

   - JDBC drivers–if you are using relational projects (see "Relational Projects" on page 17-6).
   - J2EE Connector Architecture (J2C) adapters–if you are using EIS projects (see "EIS Projects" on page 17-7).
   - J2C `connector.jar` file–if you are using EIS projects (see "EIS Projects" on page 17-7).

     The `connector.jar` file contains `javax.resource.cci` and `javax.resource.spi` interfaces that TopLink EIS uses. By default, TopLink Workbench updates its classpath to include the Java 1.5 `connector.jar` file from <*TOPLINK_HOME*>`/j2ee/home/lib`. If this version of the `connector.jar` file is incompatible with your environment, edit the `workbench.cmd` or `workbench.sh` file in <*TOPLINK_HOME*>`/bin` to change the path to this file.

     At runtime, this `connector.jar` file (or its equivalent) must be on your application or application server classpath.

- Oracle Database `ORACLE_HOME/rdbms/jlib/xdb.jar` file - if you are using direct-to-`XMLType` mappings with an Oracle9i or higher database (see "Direct-to-XMLType Mapping" on page 33-4).
- Custom `Collection` class that you use to override the default `Collection` class that TopLink uses with a mapping container policy (see "Configuring Container Policy" on page 32-26).

Example 4–1 shows how to set the `DRIVER_CLASSPATH` variable for Windows and Example 4–2 for UNIX.

Example 4–1 Setting DRIVER_CLASSPATH on Windows

```
set DRIVER_
CLASSPATH=C:\OraHome2\jdbc\lib\ojdbc14.jar;C:\Attunity\Connect\Java\lib\attunityResourceAdapt
er.jar;C:\OraHome2\rdbms\jlib\xdb.jar
```

Note: If the path to your driver(s) contains spaces, you must enclose the path in double-quotes in the `setenv.cmd` file. For example:

```
set DRIVER_CLASSPATH="C:\Program Files\some directory\driver.jar\"
```

Example 4–2 Setting DRIVER_CLASSPATH on UNIX

```
DRIVER_
CLASSPATH=/OraHome2/jdbc/lib/ojdbc14.jar;/attunity/connect/java/lib/attunityResourceAdapter.j
ar;/OraHome2/rdbms/jlib/xdb.jar; export JDBC_CLASSPATH
```

4. Save and close the `setenv` script.

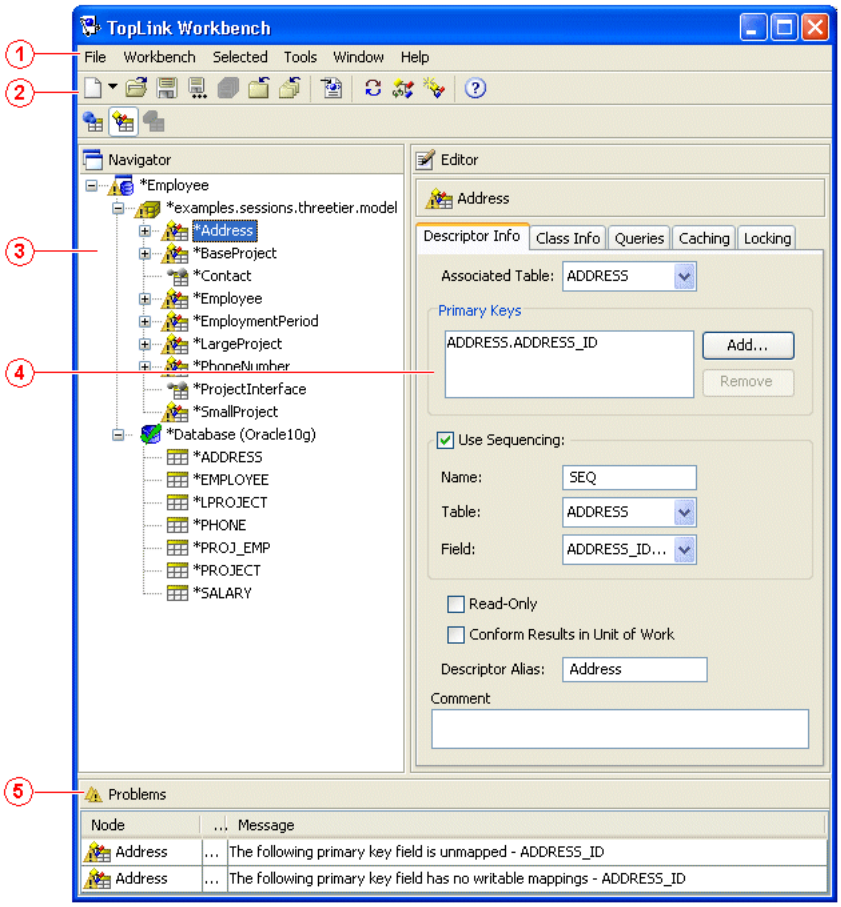
To use TopLink Workbench in a language different than your default, add the `-Duser.language` and `-Duser.country` options to the `JVM_ARGS` variable in the `workbench.cmd` or `.sh` file. For example, the following arguments will start TopLink Workbench in US English, regardless of default language of your operating system:

```
JVM_ARGS="-Duser.language=en -Duser.country=en_US"
```

Working With TopLink Workbench

Figure 4–1 shows the primary parts of TopLink Workbench window.

Figure 4–1 TopLink Workbench Window



The numbered callouts in Figure 4–1 identify the following user interface components:

1. Menu bar

   The menu bar contains menus for each TopLink Workbench function. Some objects also contain context-sensitive menus. See "Using the Menus" on page 4-5 for more information.

2. Toolbars

   The toolbars contain shortcuts to specific functions. See "Using the Toolbars" on page 4-6 for more information.

3. Navigator window section

   The Navigator window section shows the project navigation tree for all open projects (see "Using the Navigator" on page 4-9). Click the plus (+) or minus (–) sign next to an object (or double-click the object) to expand or collapse the tree. When you select an object in the **Navigator** window section, its properties appear in the **Editor** window.

4. Editor window section

   The **Editor** window section contains specific property sheets and option tabs for the currently selected object. See "Using the Editor" on page 4-10 for more information.

5. Problems window section

The **Problems** window section shows messages and errors for the currently selected object in the Navigator window section (see "Using the Problems Window" on page 4-11). Chapter 14, "TopLink Workbench Error Reference" contains detailed information on each error message.

Using the Menus

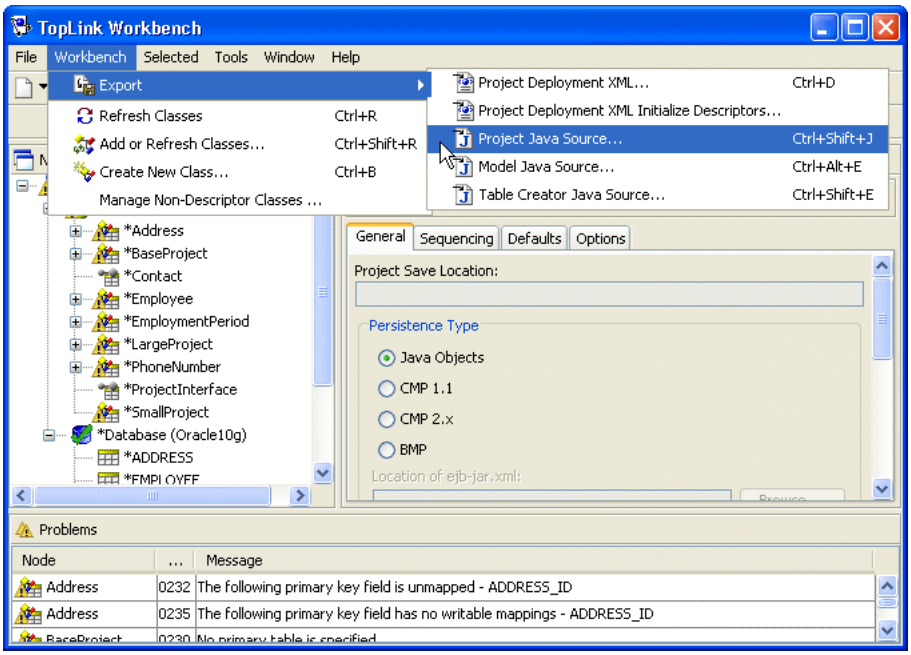
TopLink Workbench contains two types of menus:

- Menu Bar Menus
- Context Menus

Menu Bar Menus

The menu bar, located at the top of the TopLink Workbench window, provides menus for each TopLink Workbench function. Some menus (such as **Selected**) are context-sensitive; the available options may vary, depending on the currently selected object.

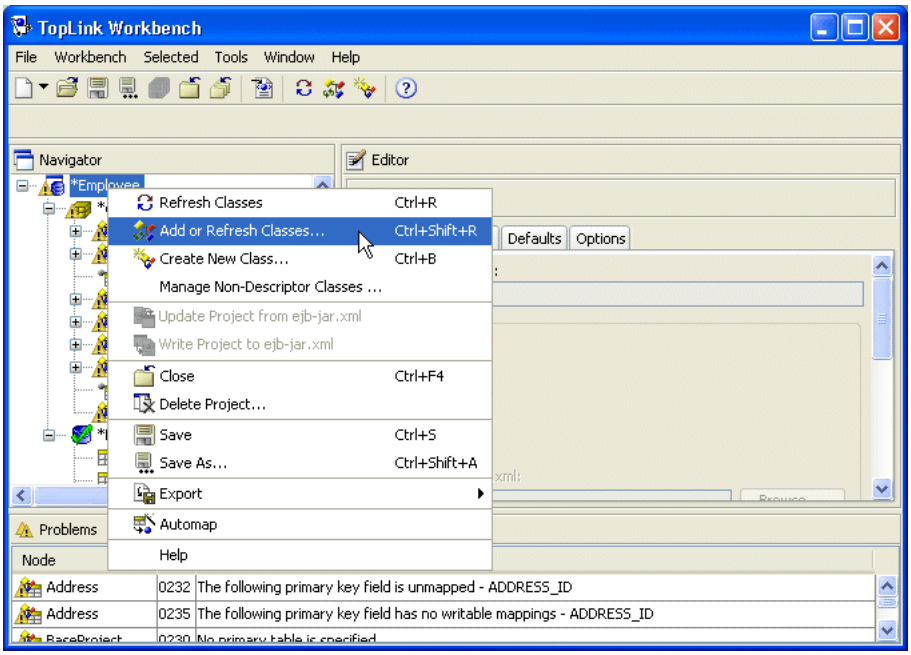
Figure 4–2 Sample Menu Bar Menu



Context Menus

When you right-click objects in the **Navigator** window, a context menu appears with functions specific to the selected object.

Figure 4–3 Sample Context Menu



Using the Toolbars

TopLink Workbench contains the following toolbars at the top of the window:

- Standard Toolbar
- Context Toolbar

Toolbars provide tool tips: each toolbar button provides a brief description when you position the mouse pointer over it.

Standard Toolbar

The standard toolbar furnishes quick access to the standard menu options (**File**, **Edit**, **Selected**, and so on).

Table 4–1 Standard Toolbar Buttons

Button	Description	Available for ...
	New	All
	Open	All
	Save	All
	Save as	All
	Save all	All
	Close	All
	Close all	All
	Help topics	All

Table 4–1 (Cont.) Standard Toolbar Buttons

Button	Description	Available for ...
	Export deployment XML	Projects
	Refresh classes	Projects
	Add or refresh classes	Projects
	Create new class	Projects

Context Toolbar

The context toolbar provides quick access to functions for the currently selected object in the **Navigator** (see "Using the Navigator" on page 4-9). The available buttons will vary, depending on which item you have selected.

You can also right-click the item and choose the appropriate option from the context menu.

Table 4–2 Context Toolbar Buttons

Button	Description	Available for ...
	Login to database	Databases
	Logout of database	Databases
	Add new table	Databases
	Add or update tables from database	Databases
	Refresh from database	Database tables
	Remove table	Database tables
	Rename	Database tables
	Add database platform	Database platform repositories
	Rename repository	Database platform repositories
	Delete platform	Database platform repositories
	Clone platform	Database platform repositories
	Add database type	Database platform repositories
	Import schema	Schemas
	Relational aggregate descriptor	Descriptors
	Relational class descriptor	Descriptors
	Relational EJB descriptor	Descriptors
	EIS composite descriptor	Descriptors
	EIS root descriptor	Descriptors

Table 4–2 (Cont.) Context Toolbar Buttons

Button	Description	Available for ...
	EIS EJB descriptor	Descriptors
	XML descriptor	Descriptors
	Direct-to-field mapping	Attributes in relational descriptors
	Object type mapping[1]	Attributes in relational descriptors
	Type conversion mapping[1]	Attributes in relational descriptors
	Serialized mapping[1]	Attributes in relational descriptors
	Direct-to-XMLType mapping	Attributes in relational descriptors
	Direct collection mapping	Attributes in relational descriptors
	Direct map mapping	Attributes in relational descriptors
	Aggregate mapping	Attributes in relational descriptors
	One-to-one mapping	Attributes in relational descriptors
	Variable one-to-one mapping	Attributes in relational descriptors
	One-to-many mapping	Attributes in relational descriptors
	Many-to-many mapping	Attributes in relational descriptors
	Direct mapping	Attributes in EIS descriptors
	Direct collection mapping	Attributes in EIS descriptors
	Composite object mapping	Attributes in EIS descriptors
	Composite collection mapping	Attributes in EIS descriptors
	One-to-one mapping	Attributes in EIS descriptors
	One-to-many mapping	Attributes in EIS descriptors
	Direct-to-XML mapping	Attributes in XML descriptors
	Direct collection mapping	Attributes in XML descriptors
	Composite object mapping	Attributes in XML descriptors
	Composite collection mapping	Attributes in XML descriptors
	Any object mapping	Attributes in XML descriptors
	Any collection mapping	Attributes in XML descriptors
	Transformation mapping	Attributes in all descriptors
	Unmap	Attributes in all descriptors

Table 4–2 (Cont.) Context Toolbar Buttons

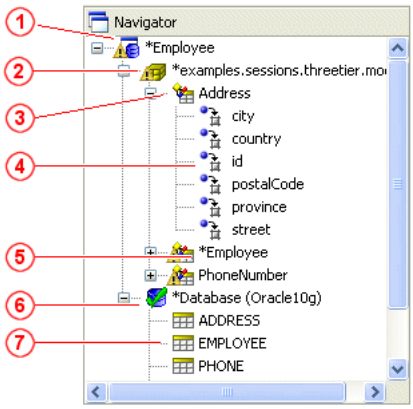
Button	Description	Available for ...
	Session	Sessions configurations
	Session Broker	Sessions configurations
	Named connection pool	Server sessions
	Sequence connection pool	Server sessions
	Write connection pool	Server sessions
	Rename	Database sessions, session brokers
	Delete session	Database sessions, session brokers

[1] Deprecated. For more information, see "Using a Converter Mapping" on page 33-3

Using the Navigator

TopLink displays the items included in each project (descriptors, mappings, data source, and so on) in the **Navigator** on the left side of the TopLink Workbench window, as Figure 4–4 shows.

Figure 4–4 Sample Navigator



The numbered callouts on Figure 4–4 identify the following user interface components:

1. Project (relational project)
2. Package
3. TopLink Descriptor (relational descriptor)
4. Attribute/mapping (direct to field mapping)
5. Unsaved/changed item
6. Database
7. Database table

Click the plus (+) or minus (–) sign next to the item, or double-click the item name to expand or collapse the item.

TopLink Workbench identifies items that have been changed but not yet saved by adding an asterisk (*) in front of the item name.

When you select an item in the **Navigator**, its properties appear in the **Editor** (see "Using the Editor" on page 4-10).

To perform specific functions for an item, select the item in the **Navigator** and do one of the following:

- Right-click on the object and select the function from the context menu (see "Context Menus" on page 4-5).
- Choose a function from the **Selected** menu (see "Menu Bar Menus" on page 4-5).

For information on using the Navigator with a database in relational projects, see "Working With Database Tables in the Navigator Window" on page 4-21.

For information on using the Navigator with an XML schema in EIS projects (using XML records) and XML projects, see "Working With XML Schemas in the Navigator" on page 4-33.

Active and Inactive Descriptors

Inactive descriptors appear dimmed in the **Navigator**. Inactive descriptors are not registered with the session when the project is loaded into Java. This feature lets you define and test subsets of descriptors. To activate or deactivate a descriptor, right-click the descriptor and select **Activate/Deactivate Descriptor** from the context menu.

Figure 4–5 Sample Active and Inactive Descriptors

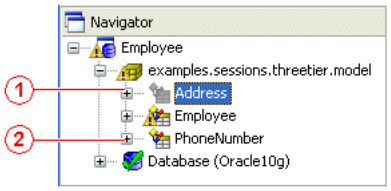


Figure 4–5 numbered callouts show the following user interface components:

1. Inactive descriptor
2. Active descriptor

Errors and Missing Information



If an element in the project (such as a descriptor or mapping) contains an error or some deficiency (sometimes called *neediness*), a warning icon appears beside the element icon in the **Navigator**, and TopLink Workbench displays a message in the Problems window (see "Using the Problems Window" on page 4-11).

Chapter 14, "TopLink Workbench Error Reference", contains more information on each TopLink Workbench error message.

Using the Editor

The **Editor**, on the right side of the TopLink Workbench window, displays the property sheet associated with the currently selected item in the **Navigator**, as Figure 4–6 shows.

Figure 4–6 Sample Editor

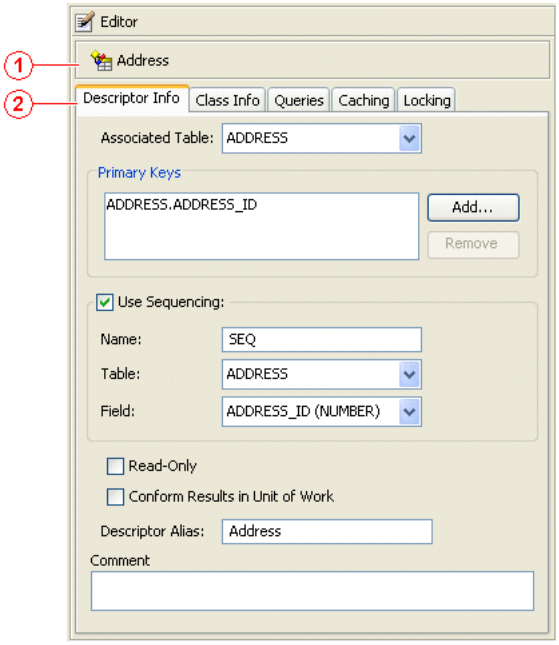


Figure 4–6 numbered callouts identify the following user interface components:

1. Selected element (from the **Navigator**)
2. Editor property tabs

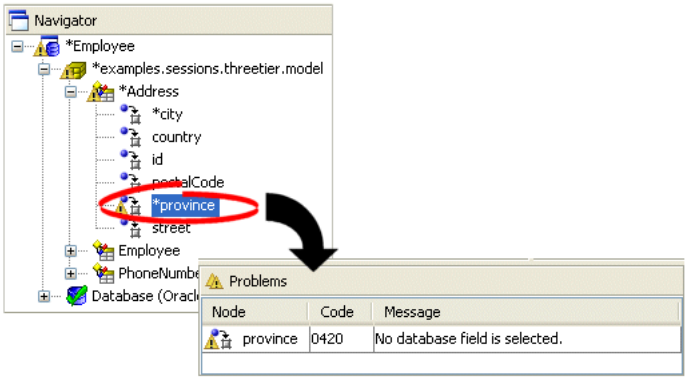
Using the Problems Window



If an element in the project (such as a descriptor or mapping) contains an error or some deficiency (sometimes called *neediness*), the TopLink Workbench displays a caution icon (represented by a yellow triangle with a black exclamation point in the middle) to the left of the deficient element in the **Navigator** (see "Using the Navigator" on page 4-9) and displays a message in the Problems window as Figure 4–7 shows.

If you select the error, then TopLink Workbench displays the complete error message in the **Problems** window. Chapter 14, "TopLink Workbench Error Reference" contains detailed information on each error message.

Figure 4–7 Sample Deficient Mapping



Double-click any error message in the **Problems** window to automatically highlight the specific node in the **Navigator**. To display or hide the **Problems** window, select **Window > Show Problems** from the menu.

You can also create a status report (see "Generating the Project Status Report" on page 18-12) that includes all errors in a selected project.

Using the Online Help

TopLink Workbench contains an extensive online Help system to assist you in developing TopLink applications. You can use the online Help system in a *hosted* or *local* environment (see "Help Preferences" on page 4-14).

To receive help on any field, tab, or element in TopLink Workbench, right-click the element and select **Help** from the context menu or press **F1**.

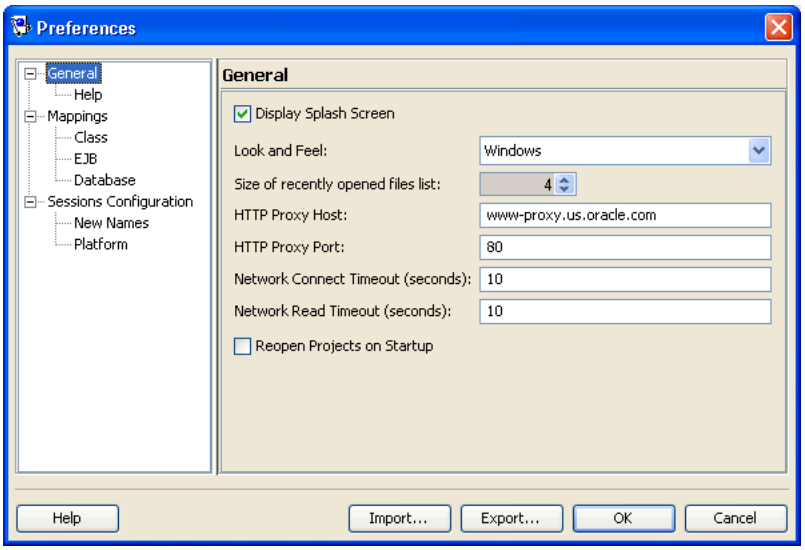


To review the complete TopLink documentation and Quick Start, click **Help**.

Working With TopLink Workbench Preferences

To customize TopLink Workbench, select **Tools > Preferences** from the menu. The **Preferences** dialog box appears.

Figure 4–8 Preferences Dialog Box



TopLink Workbench provides the following preferences:

- General Preferences
  - Help Preferences
- Mappings Preferences
  - Class Preferences
  - EJB Preferences
  - Database Preferences
- Sessions Configuration Preferences
  - New Names Preferences

– Platform Preferences

Use this dialog box to configure TopLink Workbench preferences. After changing preferences, you must restart TopLink Workbench.

To import your preferences from an existing file, click **Import** and select the file.

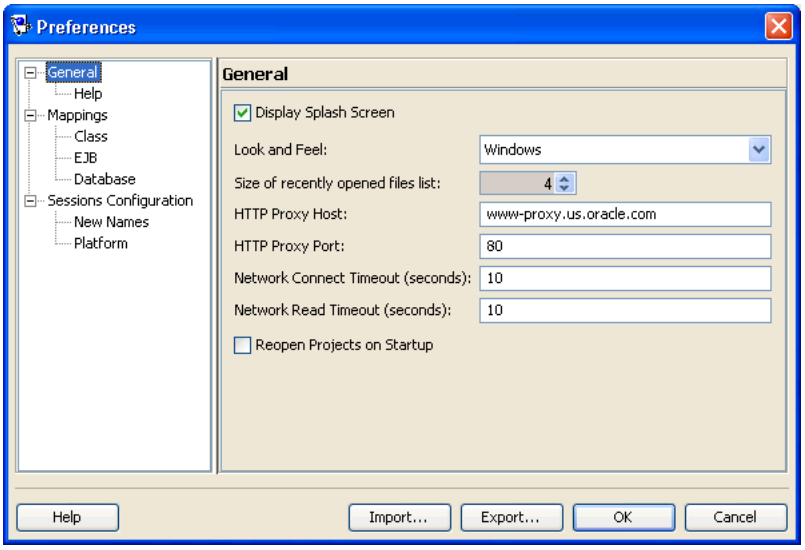
To export your preferences, click **Export** and select a directory location and filename.

General Preferences

Use the General preferences to customize the look and feel (the graphical user interface) of TopLink Workbench as well as to specify any proxy information required to access the Internet (for example, to allow TopLink to access XML schemas and on-line documentation hosted on Internet sites). Follow these steps to customize the General preferences:

1. Select **Tools > Preferences** from the menu. The **Preferences** dialog box appears.
2. Select **General** in the **Category** window.

Figure 4–9 Preferences–General Dialog Box



Use the following information to enter data in each field of the dialog box:

Field	Description
Display Splash Screen	Specify if TopLink Workbench should show the graphical splash screen when starting.
Look and Feel	Select the look and feel to use for TopLink Workbench.
Size of recently opened files list	Select the number of projects to maintain in the **File** > **Reopen** option. See "Opening Existing Projects" on page 18-10 for more information.
HTTP Proxy Host	Specify if your PC requires a proxy server to access the internet.
HTTP Proxy Port	Specify the port used by your proxy host.
Network Connect Timeout	Specify the timeout (in seconds) to establish a network or internet connection.

Field	Description
Network Read Timeout	Specify the timeout (in seconds) when accessing data from a network or internet connection.
Reopen Projects on Startup	Select to reopen the projects that were open the last time you exited the TopLink Workbench.

You must restart TopLink Workbench to apply the changes.

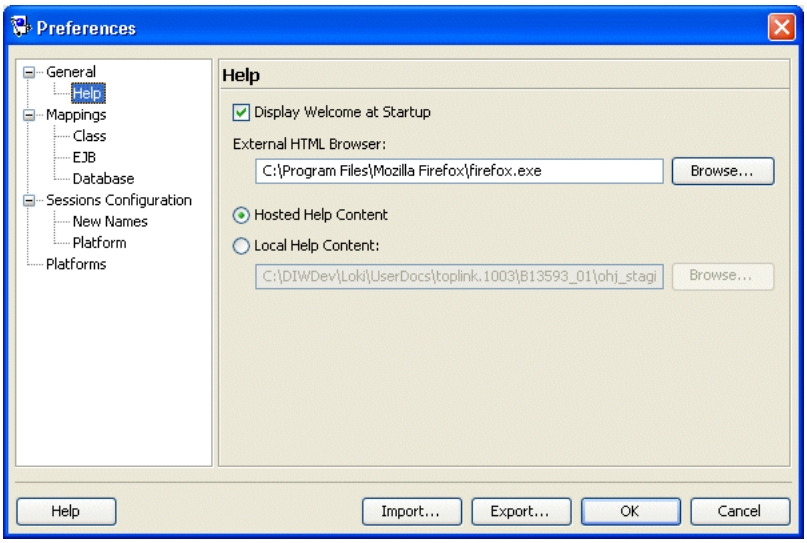
Help Preferences

Use the Help preferences to select the location of the TopLink documentation (in addition to the online Help) and other Help system preferences.

By default, TopLink installs includes only the online Help and release notes. All other documentation (such as *Oracle TopLink Getting Started Guide* and *Oracle TopLink Developer's Guide*) can be accessed from a hosted location.

1. Select **Tools > Preferences** from the menu. The **Preferences** dialog box appears.
2. Expand **General** in the **Category** window and select **Help**. The **Preferences–Help** dialog box appears.

Figure 4–10 Preferences–General–Help Dialog Box



Use the following information to enter data in each field:

Field	Description
Display Welcome at Startup	Specify if TopLink should show the **Welcome** screen each time you start TopLink Workbench.
External HTML Browser	Click **Browse** and select the location of your default Web browser. You must specify a Web browser to access the Quick Tour, Javadoc (API), and other Web-based material.

Field	Description
Hosted Help Content[1]	Access the complete TopLink documentation set located on Oracle Technology Network (OTN). Selecting this option requires your PC to have Internet access. **Note**: Depending on your Internet connection, using **Hosted Documentation** may impact performance when searching the documentation.
Local Help Content[1]	Access the complete TopLink documentation set from a local location (such as your PC or your company's intranet). Click **Browse** and select the help JAR file location.

[1] You must restart TopLink Workbench to apply changes to this option.

Note: When using **Hosted Help Content**, ensure that your proxy information, specified in the General Preferences, is correct.

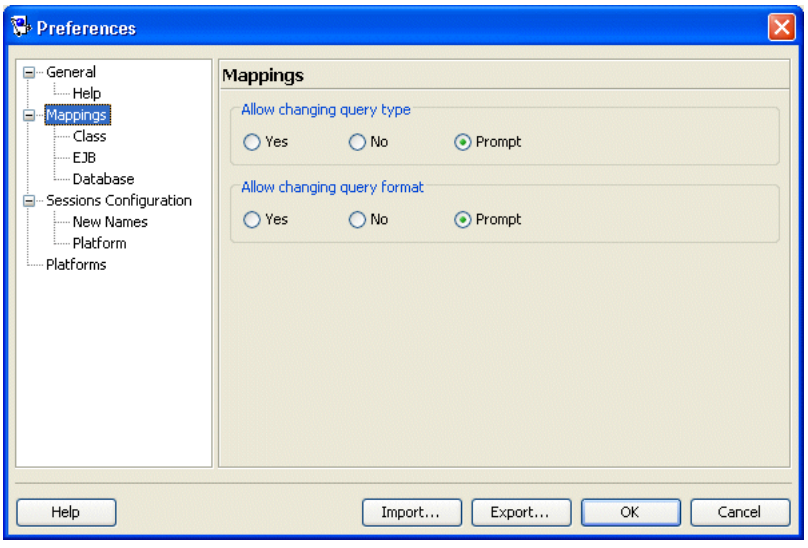
When using **Local Help Content**, download the complete documentation from OTN. See *Oracle TopLink Getting Started Guide* for more information.

Mappings Preferences

Use the Mappings preferences to specify general mapping preferences. Follow these steps to set the Mapping preferences:

1. Select **Tools > Preferences** from the menu. The **Preferences** dialog box appears.
2. Select **Mappings** in the **Category** window. The **Mappings** dialog box appears.

Figure 4–11 Preferences–Mappings Dialog Box



Use the following information to enter data in each field:

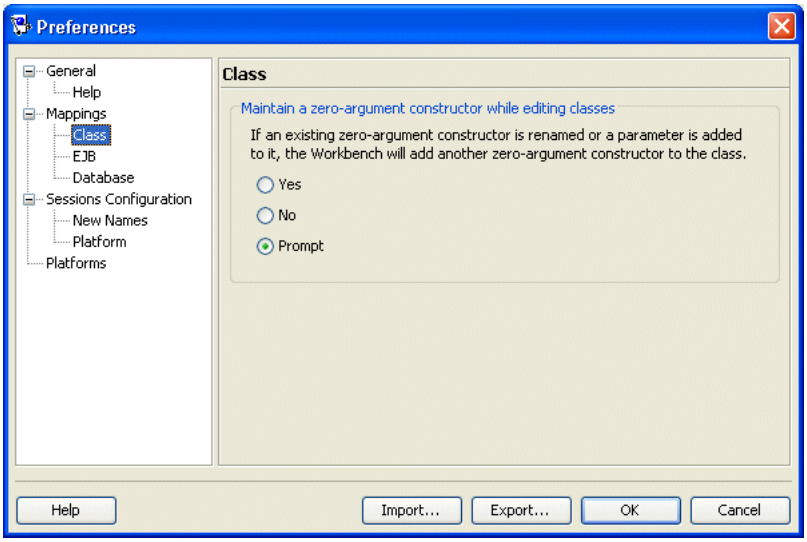
Field	Description
Allow changing query type	Configure whether or not TopLink Workbench always allows, never allows, or prompts before allowing you to change the query type associated with a descriptor.
Allow changing query format	Configure whether or not TopLink Workbench always allows, never allows, or prompts before allowing you to change the configuration of a query associated with a descriptor.

Class Preferences

Use the Class preferences to specify how TopLink Workbench maintains classes when renaming or editing a zero-argument constructor. Follow these steps to set the Class preferences:

1. Select **Tools > Preferences** from the menu. The **Preferences** dialog box appears.
2. Expand **Mappings** in the **Category** window and select **Class**.

Figure 4–12 Preferences – Mappings – Class Dialog Box



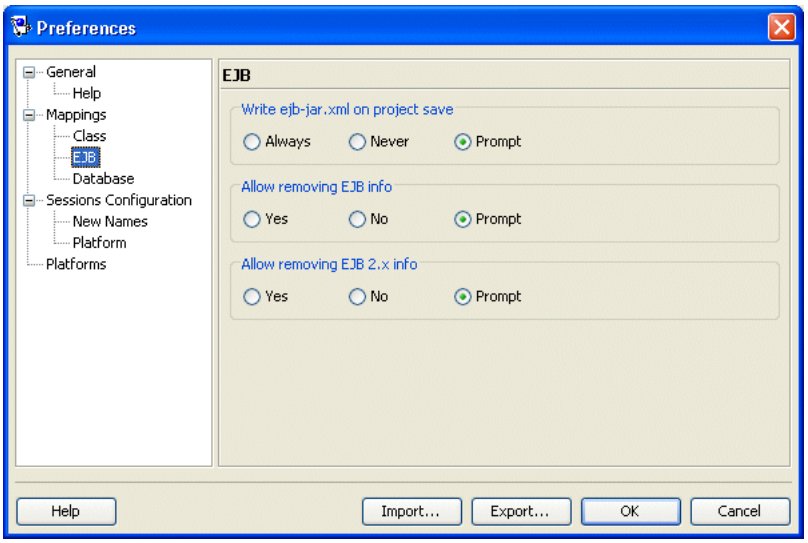
On the **Preferences**–**Mappings**–**Class** dialog box, specify how TopLink Workbench maintains classes when renaming or editing a zero-argument constructor.

EJB Preferences

Use the EJB preferences to specify how TopLink Workbench updates the `ejb-jar.xml` file when saving EJB projects. Follow these steps to set the EJB preferences:

1. Select **Tools > Preferences** from the menu. The **Preferences** dialog box appears.
2. Expand **Mappings** in the **Category** window and select **EJB**.

Figure 4–13 Preferences–Mappings–EJB Preferences Dialog Box



Use the following information to select how TopLink Workbench will update the `ejb-jar.xml` file:

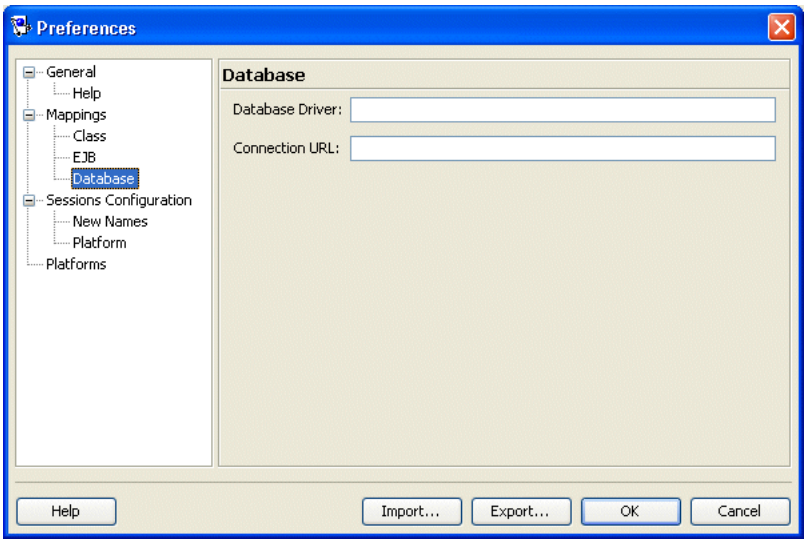
Field	Description
Write ejb-jar.xml on project save	Configure whether or not TopLink Workbench always updates, never updates, or prompts before updating the `ejb-jar.xml` file each time you save the project.
Allow removing EJB info	Configure whether or not TopLink Workbench always allows, never allows, or prompts before allowing you to remove the EJB information associated with a descriptor. See "Configuring a Descriptor With EJB Information" on page 25-45 for more information.
Allow removing EJB 2.x info	Configure whether or not TopLink Workbench always allows, never allows, or prompts before allowing you to remove the EJB 2.0 or 2.1 information associated with a descriptor. See "Configuring a Descriptor With EJB Information" on page 25-45 for more information.

Database Preferences

Use the Database preferences to specify custom database divers and connection URLs for TopLink Workbench. These drivers and URLs can then be used when defining database logins. Follow these steps to set the Database preferences:

1. Select **Tools > Preferences** from the menu. The **Preferences** dialog box appears.
2. Expand **Mappings** in the **Category** window and select **Database**.

Figure 4–14 Preferences–Mappings–Database Preferences Dialog Box



Use the following information to enter data in each field:

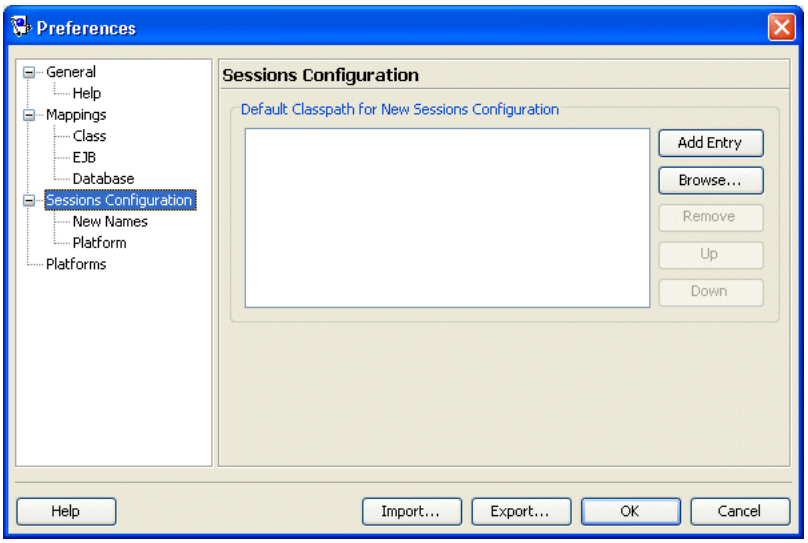
Field	Description
Database Driver	Enter the custom database driver class name.
Connection URL	Enter the custom database connection URL.

Sessions Configuration Preferences

Use the Sessions preferences to specify default classpaths to be added to each newly created TopLink sessions configuration for features that require an external Java class (for example, session event listeners). The entries added here will automatically appear on the Sessions Configuration property sheet (see "Configuring a Sessions Configuration" on page 73-2). Follow these steps to set the Sessions Configuration preferences:

1. Select **Tools > Preferences** from the menu. The **Preferences** dialog box appears.
2. Select **Sessions Configuration** in the **Category** window.

Figure 4–15 Preferences–Sessions Configuration Dialog Box



To add a JAR or ZIP file, click **Add Entry** or **Browse** and add the JAR or ZIP files that contain the default compiled Java classes for this sessions configuration.

To remove a JAR or ZIP file, select the file and click **Remove**.

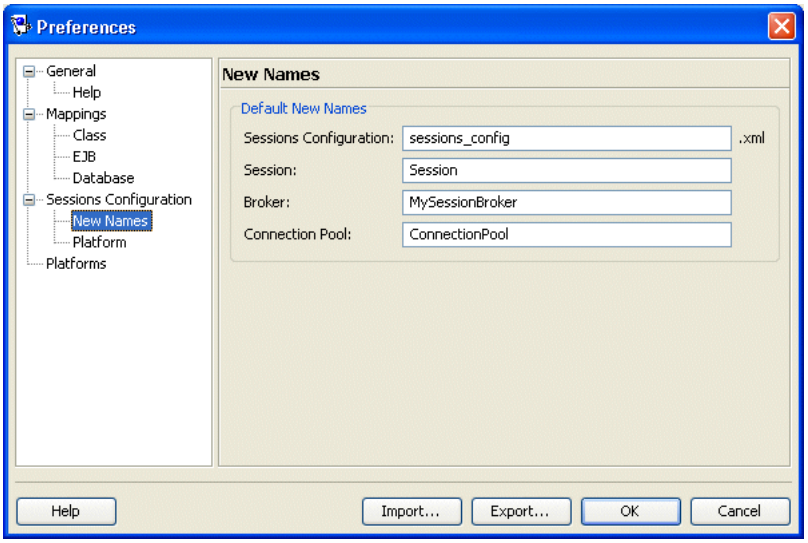
To change the order in which TopLink searches these JAR or ZIP files, select a file and click **Up** to move it up, or click **Down** to move it down in the list.

New Names Preferences

Use the New Names preferences to specify the default values and names of newly created sessions, session brokers, and connection pools. Follow these steps to set the New Names preferences:

1. Select **Tools > Preferences** from the menu. The **Preferences** dialog box appears.
2. Expand **Sessions Configuration** in the **Category** window and select **New Names**.

Figure 4–16 Preferences–Sessions Configuration–New Names Dialog Box



Use the following information to enter data in each field:

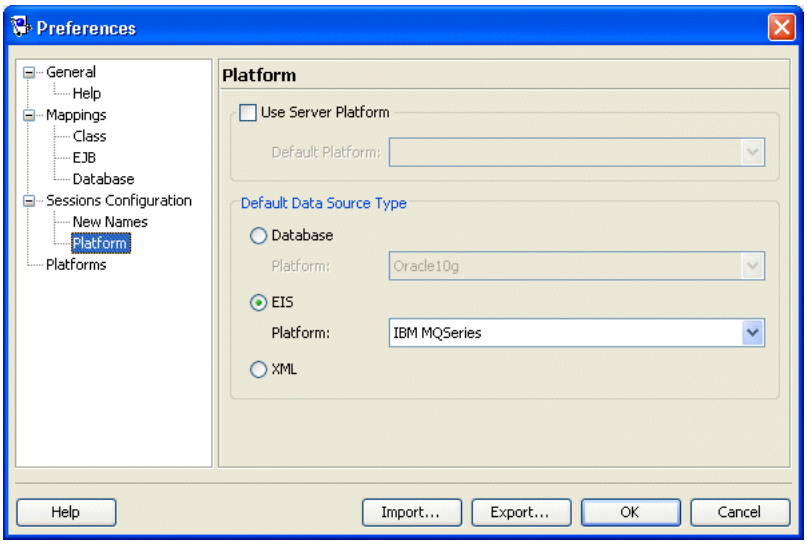
Field	Description
Sessions Configuration	Specify the default name for newly created sessions configuration files (default, `sessions.xml`). See "Creating a Sessions Configuration" on page 73-1 for more information.
Session	Specify the default name for newly created sessions (default, `Session`). See "Session Creation Overview" on page 73-1 for more information.
Broker	Specify the default name for newly created session brokers (default, `SessionBroker`). See "Creating Session Broker and Client Sessions" on page 73-6 for more information.
Connection Pool	Specify the default name for newly created connection pools (default, `ConnectionPool`). See Chapter 85, "Creating an Internal Connection Pool" for more information.

Platform Preferences

Use the Platform preferences to specify the default data source type for newly created sessions. The type selected here will automatically appear on the Create New Session dialog box. Follow these steps to set the Platform preferences:

1. Select **Tools > Preferences** from the menu. The **Preferences** dialog box appears.
2. Expand **Sessions Configuration** in the **Category** window and select **Platform**.

Figure 4–17 Preferences–Sessions Configuration–Platform Preferences Dialog Box



Use the following information to enter data in each field:

Field	Description
Use Server Platform	Specify the default application server platform for newly created sessions configuration files (default, `sessions.xml`). See "Creating a Sessions Configuration" on page 73-1 for more information.
Default Data Source Type	Select the default data source type (**Database**, **EIS**, or **XML**) and platform for newly created sessions. See "Configuring the Server Platform" on page 74-14 for more information.

Working With Databases

In relational projects, when you expand the database object in the **Navigator**, TopLink Workbench displays the database tables associated with the project. You can associate tables by importing them from the database, or by creating them within TopLink Workbench.

Figure 4–18 Sample Database Tables

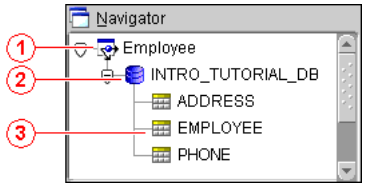


Figure 4–18 numbered callouts identify the following database icons.

1. Project
2. Database
3. Database table

Each database table property sheet contains the following tabs in the **Editor**:

- **Columns**–Add or modify the table's fields, and specify each field's properties.
- **References**–Specify references between tables.

This section includes information on the following topics:

- Working With Database Tables in the Navigator Window
- Working With Database Tables in the Editor Window
- Generating Data From Database Tables

Working With Database Tables in the Navigator Window

This section describes the following options:

- Logging In and Out of a Database
- Creating New Tables
- Importing Tables From a Database
- Removing Tables
- Renaming Tables
- Refreshing Tables From the Database

See "Working With Database Tables in the Editor Window" on page 4-25 for more information.

Logging In and Out of a Database

To log in or out of a relational database, do the following:

1. Create a database login (see "Database Login Configuration Overview" on page 83-1).

2. To log in to a relational database, right-click the database object in the **Navigator**, and choose **Log In to Database** from the context menu or choose **Selected** > **Log In to Database** from the menu.
3. To log out of a relational database, right-click the database object in the **Navigator** and choose **Log Out of Database** from the context menu or choose **Selected** > **Log Out of Database** from the menu.

Creating New Tables

To create a new database table within TopLink Workbench, use the following procedure:



1. Select the database object in the **Navigator** window and click **Add New Table**. The **New Table** dialog box appears.

   You can also right-click the database object and choose **Add New Table** from the context menu, or choose **Selected** > **Add New Table** from the menu.

Figure 4–19 New Table Dialog Box



Use the following information to enter data in each field:

Field	Description
Catalog	Use to identify specific database information for the table. Consult your database administrator for more information.
Schema	Use to identify specific database information for the table. Consult your database administrator for more information.
Table Name	Specify the name of this database table.

TopLink Workbench adds the database table to the project.

Although the database table has been added to the project, it has not been written to the actual database. See "Generating Tables on the Database" on page 4-32 for more information on creating the table in the database.

Continue with "Working With Database Tables in the Editor Window" on page 4-25 to use these tables in your project.

Importing Tables From a Database

TopLink Workbench can automatically read the schema for a relational database and import the table data into the project as long as your JDBC driver supports the following JDBC methods:

- `getTables`

- `getTableTypes`
- `getImportedKeys`
- `getCatalogs`
- `getPrimaryKeys`

The JDBC driver must be on the TopLink Workbench classpath (see "Configuring the TopLink Workbench Environment" on page 4-2).

To import tables from the database, use the following procedure:



1. Select the database object in the **Navigator,** and click **Add/Update Existing Tables from Database**. The **Import Tables from Database** dialog box appears.

   You can also right-click on the database object in the **Navigator** and choose **Add/Update Existing Tables from Database** from the context menu or choose **Selected** > **Add/Update Existing Tables from Database** from the menu.

Figure 4–20 Import Tables from Database Dialog Box

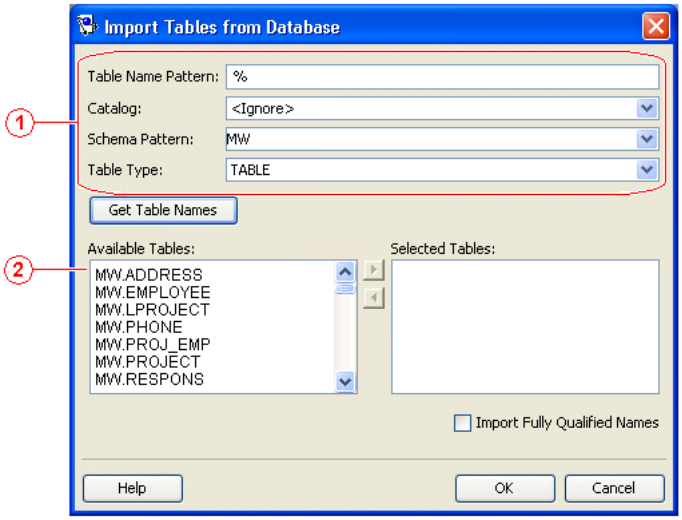


Figure 4–20 numbered callouts identify the following user interface components:

1. Filters
2. Database tables that match the filters

Use the following information to enter data in each field of the dialog box:

Field	Description
Table Name Pattern	Specify the name of database table(s) to import. Use percent character (%) as a wildcard. Tables that match the **Table Name Pattern** can be imported.
Catalog	Specify the catalog of database table(s) to import.
Schema Pattern	Specify the schema of database table(s) to import.
Table Type	Specify the type of database table(s) to import.
Available Tables	Click **Get Table Names** to make TopLink display tables that match **Table Name Pattern**, **Catalog**, **Schema Pattern**, and **Table Type** settings.

Field	Description
Selected Tables	Select the tables in the **Available Tables** area to import, and click the right-arrow button. TopLink adds the table to the **Selected Tables** field. Click **OK** to import the tables from the database into the TopLink Workbench project.
Import Fully Qualified Names	Specify whether or not the tables' names are fully qualified against the schema and catalog.

Examine each table's properties to verify that the imported tables contain the correct information. See "Working With Database Tables in the Editor Window" on page 4-25 for more information.

Removing Tables

To remove a database table from the project, use the following procedure:



1. Select a database table in the **Navigator,** and click **Remove Table** on the toolbar. TopLink Workbench prompts for confirmation.

   You can also right-click on the database object and choose **Remove** from the context menu or choose **Selected** > **Remove Table** from the menu.

2. Click **OK**. TopLink Workbench removes the table from the project.

> **Note:** Although you have removed the table from the TopLink Workbench project, the table remains in the database.

Renaming Tables

To rename a database table in the TopLink Workbench project, use the following procedure:

1. Right-click the table in the **Navigator** and choose **Rename** from the context menu. The Rename dialog box appears.

   You can also select the table and choose **Selected** > **Rename** from the menu.

2. Enter a new name and click **OK**. TopLink Workbench renames the table.

> **Note:** Although you have renamed the table in the TopLink Workbench project, the original table name remains in the database.

Refreshing Tables From the Database

To refresh (that is, reload) the database tables in the TopLink Workbench project, use this procedure:

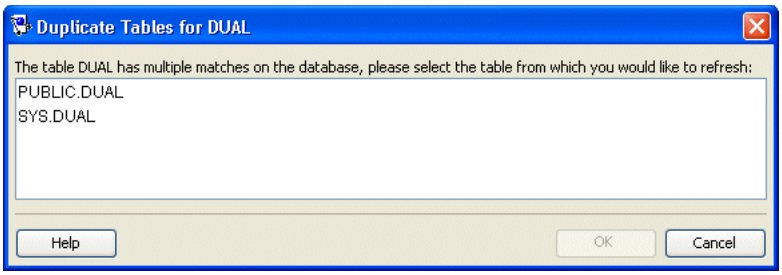


Select a database table in the **Navigator**, and click **Refresh from Database** on the toolbar.

You can also select the table and choose **Selected** > **Refresh from Database** from the menu, or click **Refresh**. TopLink Workbench reloads the database table.

When refreshing tables from the database, if there are multiple database tables with similar names, the **Duplicate Tables** dialog box appears.

Figure 4–21 Duplicate Table Dialog Box



Select the specific database table to update, and then click **OK**.

Working With Database Tables in the Editor Window

When you select a database table in the **Navigator**, its properties appear in the **Editor**. Each database table contains the following property tabs:

- **Columns**–Add or modify the table fields, and specify each field properties.
- **References**–Specify references between tables.

This section describes how to use these tabs to configure the following:

- Working With Column Properties
- Setting a Primary Key for Database Tables
- Creating Table References
- Creating Field Associations

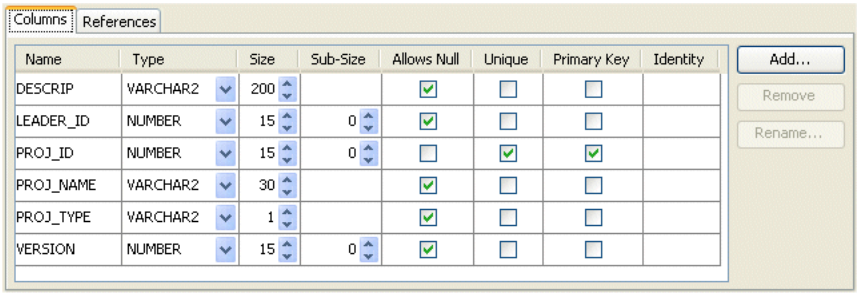
Working With Column Properties

Use the database table's **Column** tab to specify properties for the database table's fields.

To specify a table's column properties, use this procedure:

1. Select a database table in the **Navigator**. The table's property sheet displays in the **Editor**.
2. Click the Columns tab.

Figure 4–22 Fields Properties



Use the following information to fill each column on the Columns tab:

Field	Description
Name	Specify the name of the field.
Type	Use the drop-down list to select the field's type. Note: The valid values will vary, depending on the database.
Size	Specify the size of the field.
Sub-Size	Specify the sub-size of the field.
Allows Null	Specify if this field can be null.
Unique	Specify whether the value must be unique within the table.
Primary Key	Specify whether or not this field is a primary key for the table (see "Setting a Primary Key for Database Tables" on page 4-26).
Identity	Use to indicate a Sybase, SQL Server or Informix identity field.

Note: Some properties may be unavailable, depending on your database type.

To add a new field, click **Add**.

To remove a field, select the field and click **Remove**.

To rename a field, select the field and click **Rename**.

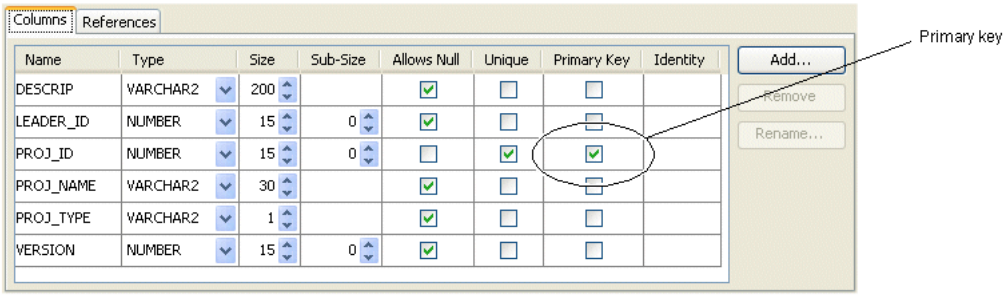
Setting a Primary Key for Database Tables

To set a primary key(s) for a database table, use this procedure:

Note: TopLink Workbench can automatically import primary key information if supported by the JDBC driver.

1. Select a database table in the **Navigator**. Its property sheet appears in the **Editor**.
2. Click the **Columns** tab.

Figure 4–23 Setting Primary Key for a Database Table



3. Select the **Primary Key** field(s) for the table.

Creating Table References

References are table properties that contain the foreign key; they may or may not correspond to an actual constraint that exists on the database. TopLink Workbench

uses these references when you define relationship mappings and multiple table associations.

When importing tables from the database, TopLink Workbench can automatically create references (if the driver supports this), or you can define references from the workbench. See "Importing Tables From a Database" on page 4-22.

To create a new table reference, use this procedure:

1. Select a database table in the **Navigator**. The table's properties display in the **Editor**.
2. Click the **References** tab.

Figure 4–24 References Tab

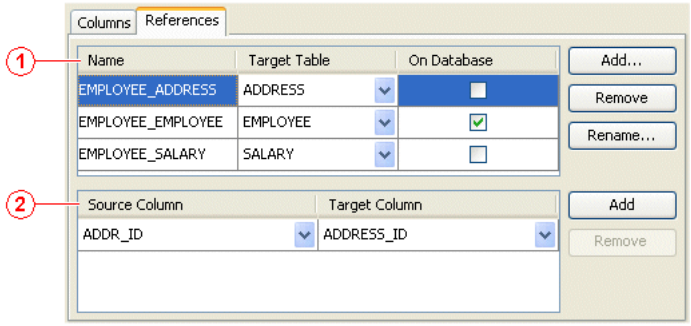
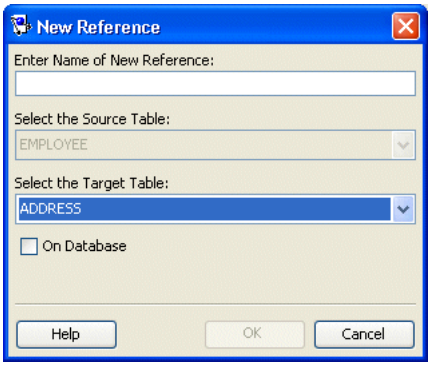


Figure 4–26 numbered callouts identify the following user interface components:

1. Table References area
2. Key Pairs area

3. In the **References** area, click **Add**. The **New Reference** dialog box appears.

Figure 4–25 New Reference Dialog Box



Use the following information to enter data in each field of the dialog box:

Field	Description
Enter Name of New Reference	Specify the name of the reference table. If you leave this field blank, TopLink Workbench automatically creates a name based on the format: SOURCETABLE_TARGETTABLE.

Field	Description
Select the Source Table	Specify the name of the source database table (the currently selected table in the Navigator).
Select the Target Table	Use the list to specify the target table for this reference.
On Database	Specify if you want to create the reference on the database when you create the table. Not all database drivers support this option.

Continue with "Creating Field Associations" on page 4-28.

Creating Field Associations

For each table reference, you can specify one or more field associations that define how fields in the source table relate to fields in the target table. See "Creating Table References" on page 4-26.

To create new field references, use this procedure:

1. Select a database table in the Navigator. The table's properties display in the **Editor**.
2. Click the **References** tab.

Figure 4–26 References Tab

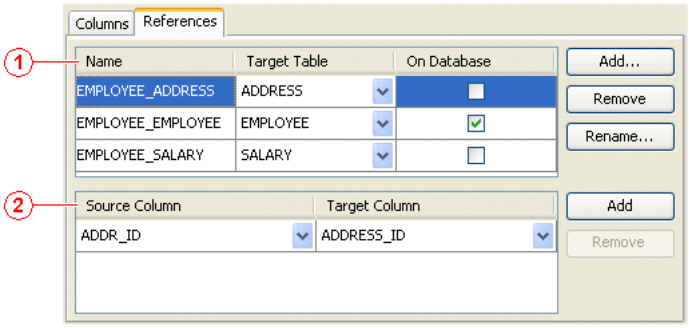


Figure 4–26 numbered callouts identify the following user interface components:

1. Table references area
2. Key pairs area

3. Select a table reference from the references area.
4. To create a new key pair, click **Add** in the key pairs area and complete each field in the key pairs area using the following information:

Field	Description
Table References Area	
Reference Name	Specify the name of this table reference
Target Table	Specify the database table that is the *target* of this reference.
On Table	Specify if the reference exists on the database.
Key Pairs Area	
Source Field	Select the database field from the source table.

Field	Description
Target Field	Select the database field from the target table.

Generating Data From Database Tables

TopLink Workbench can automatically generate a variety of information from the database tables. This section describes the following:

- Generating SQL Creation Scripts
- Generating Classes and Descriptors From Database Tables
- Generating EJB Entity Beans and Descriptors From Database Tables
- Generating Tables on the Database

Generating SQL Creation Scripts

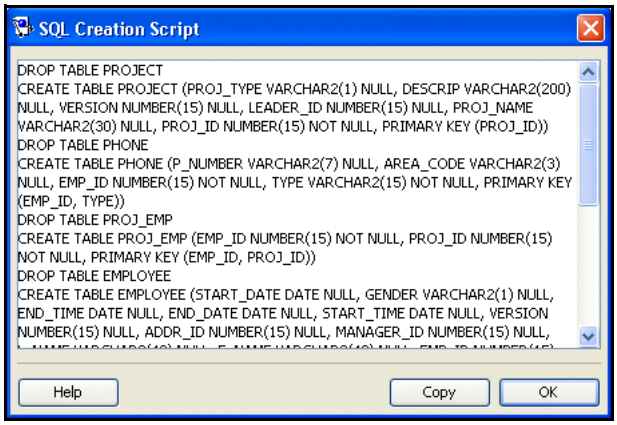
Using the TopLink Workbench, you can generate SQL scripts that you can use to create tables in a relational database.

To automatically generate SQL scripts to create the tables in a project, use this procedure:

1. Select the database table(s) in the **Navigator**.
2. Right-click the table(s) and choose **Generate Creation Script for** > **Selected Table** or **All Tables** from the context menu. The SQL Creation Script dialog box appears.

   You can also choose **Selected** > **Generate Creation Script for** > **Selected Table** or **All Tables** from the menu.

Figure 4–27 SQL Creation Script Dialog Box



Copy the script and paste it into a file. You may need to edit the file to include additional SQL information that TopLink Workbench could not generate. If the database table or column name is an SQL reserved word, you must edit the SQL script and enclose the database table or column in quotes. See "Oracle Database Reserved Words" in the *Oracle Database SQL Reference Guide* for more information.

Note: If TopLink cannot determine how a particular table feature should be implemented in SQL, it generates a descriptive message in the script.

Generating Classes and Descriptors From Database Tables

TopLink Workbench can automatically generate Java class definitions, descriptor definitions, and associated mappings from the information in database tables. You can later edit the generated information if necessary.

For each table, TopLink Workbench does the following:

- Creates a class definition and a descriptor definition.
- Adds attributes to the class for each column in the table.
- Automatically generates access methods, if specified.
- Creates direct-to-field mappings for all direct (nonforeign key) fields in the table.
- Creates relationship mappings (one-to-one and one-to-many) if there is sufficient foreign key information. You may be required to determine the exact mapping type.

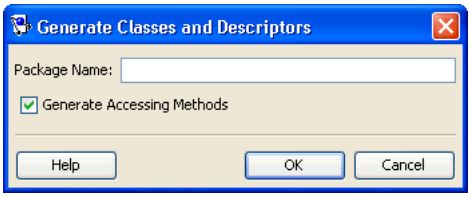
> **Note:** Class and attribute names are generated based on the table and column names. You can edit the class properties to change their names.

To generate classes and descriptors from database tables, use the following procedure:

1. Select the database table(s) in the **Navigator**.
2. Right-click the table(s) and choose **Generate Classes and Descriptors from** > **Selected Table** or **All Tables** from the context menu.

   You can also choose **Selected** > **Generate Classes and Descriptors from** > **Selected Table** or **All Tables** from the menu.
3. Click **Yes**. The Generate Classes and Descriptors dialog box appears.

Figure 4–28 Generate Classes and Descriptors Dialog Box

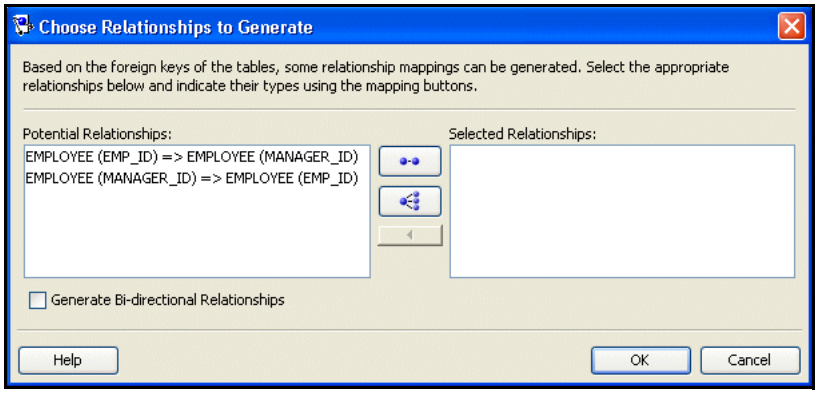


Use the following information to enter data in each field:

Field	Description
Package Name	Specify the name of package to generate. The package name must comply with Java naming standards.
Generate Accessing Methods	Specify if TopLink Workbench generates accessing methods for each class and descriptor.

If the table contains foreign key fields that may represent relationship mappings, then the Choose Relationships to Generate dialog box appears.

Figure 4–29 Choose Relationships to Generate Dialog Box





Select an entry from **Potential Relationships** and click the **1:1 Mapping** or **1:M Mapping** button, located between the Potential Relationships and Selected Relationships windows. See Chapter 33, "Understanding Relational Mappings" for more information on mappings.

You can also specify whether the relationships are bidirectional. See "Configuring Bidirectional Relationship" on page 32-34 for more information.

Click **OK** to automatically create the relationships.

The newly created descriptors appear in the **Navigator** of TopLink Workbench.

Generating EJB Entity Beans and Descriptors From Database Tables

Using TopLink Workbench, you can automatically generate EJB entity beans and descriptors for each database table, including the following:

- One EJB descriptor that implements the `<javax.ejb.EntityBean>` and entity bean classes
- Bean relation attributes (CMP or BMP)
- Java source for each class
- EJB-compliant method stubs

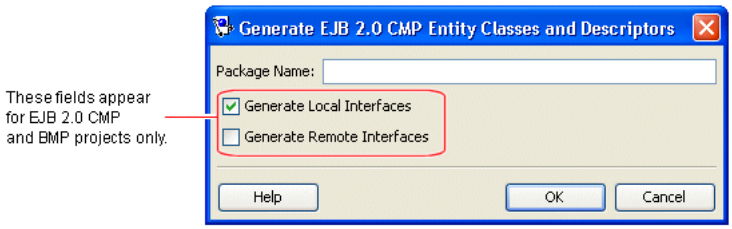
> **Note:** This option is available only for projects with container-managed or bean-managed persistence. See "Configuring Persistence Type" on page 19-5 for more information.

To automatically generate EJB entity beans and descriptors for each database table, use this procedure:

1. Select the database table(s) in the **Navigator**.
2. Right-click the table(s) and choose **Generate EJB Entities and Descriptors from** > **Selected Table** or **All Tables** from the context menu. TopLink Workbench prompts you to save your project.

   You can also choose **Selected** > **Generate EJB Entities and Descriptors from** > **Selected Table** or **All Tables** from the menu.
3. Click **Yes** to save your project before generating EJB entities. The Generate EJB Entity Classes and Descriptors dialog box appears.

Figure 4–30 Generate EJB Entity Classes and Descriptors Dialog Box



Use the following information to enter data in each field on the Generate EJB Entity Classes and Descriptors dialog box:

Field	Description
Package Name	Name of the package to contain the generated entity beans and descriptors.
Generate Local Interfaces[1]	Specify if TopLink creates local interfaces for the EJB entity beans.
Generate Remote Interfaces[1]	Specify if TopLink creates remote interfaces for the EJB entity beans.

[1] For CMP and BMP projects only. See "Configuring a Descriptor With EJB Information" on page 25-45 for more information.



If the table contains foreign key fields that may represent relationship mappings, then the Choose Relationships to Generate dialog box appears. Select a potential relationship and click the **1:1 Mapping** or **1:M Mapping** button, located between the Potential Relationships and **Selected Relationships** windows.

You can also specify if the relationships are bidirectional (see "Configuring Bidirectional Relationship" on page 32-34).

Repeat for all appropriate sets of tables.

Click **OK** to generate the relationship mappings.

The system creates the remote primary key, home, and bean classes for each bean and adds this information to the project.

Generating Tables on the Database

To create a table in the database, based on the information in TopLink Workbench, use this procedure:

> **Note:** You must log in the database before creating tables. See "Logging in to the Database" on page 20-7 for more information.

1. Select the database table(s) in the **Navigator**.
2. Right-click the table(s) and choose **Create on Database** > **Selected Table** or **All Tables** from the context menu.

   You can also create tables by selecting **Selected** > **Create on Database** > **Selected Table** or **All Tables** from the menu.

TopLink Workbench creates the tables on the database.

Alternatively, you can generate tables at run time by exporting the information in TopLink Workbench to a `TableCreator` class (see "Understanding the Schema Manager" on page 5-1).

Working With XML Schemas

For XML and EIS projects, TopLink Workbench maps each TopLink descriptor to your XML schema.

This section includes information on the following topics:

- Working With XML Schemas in the Navigator
- Working With XML Schema Structure
- Importing an XML Schema
- Configuring XML Schema Reference
- Configuring XML Schema Namespace

Working With XML Schemas in the Navigator

After you import one or more XML schemas into your project (see Importing an XML Schema on page 4-34) and you expand the schema object in the **Navigator**, TopLink Workbench displays the schemas associated with the project.

Figure 4–31 Sample XML Schemas

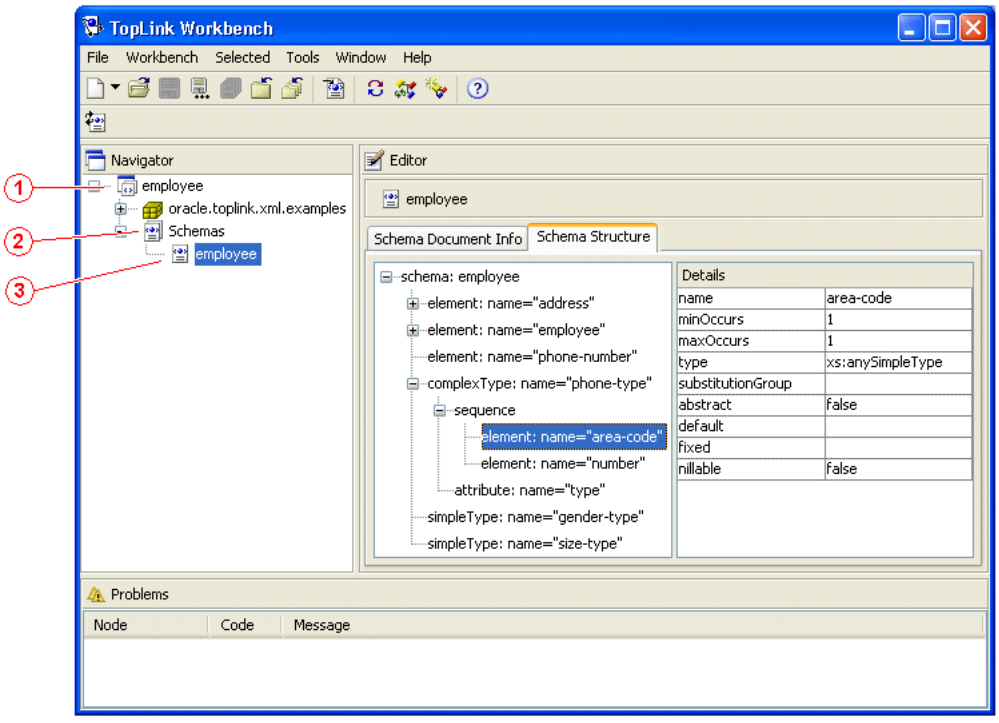


Figure 4–31 numbered callouts identify the following schema icons:

1. Project
2. Schemas object
3. Specific schema

For more information, see the following:

- "Working With XML Schema Structure" on page 4-34
- "Configuring XML Schema Reference" on page 4-36
- "Configuring XML Schema Namespace" on page 4-37

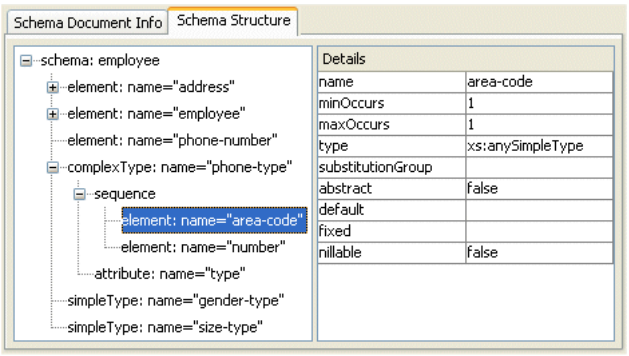
Working With XML Schema Structure

When you select a specific XML schema in the **Navigator**, you can display the structure and details of the schema using the Schema Structure tab.

To display the structure and details of a schema, use this procedure:

1. Select a schema element in the **Navigator**. Its properties appear in the **Editor**.
2. Click the **Schema Structure** tab. The Schema Structure tab appears.
3. Select an element in the schema. The element's details appear.

Figure 4–32 Schema Structure Tab



Use the following information to verify data in each field in the Schema Document Info tab:

Field	Description
Schema Structure	Displays the elements of the schema, listed in alphabetical order, in an expandable or collapsible tree structure.
Details	Displays detailed information (such as name and type) for the currently selected element in the **Schema Structure** area.

These fields are for display only and cannot be changed in TopLink Workbench.

Importing an XML Schema

The first step in configuring an EIS project (using XML records) or XML project is importing the XML schema(s) that your project uses.

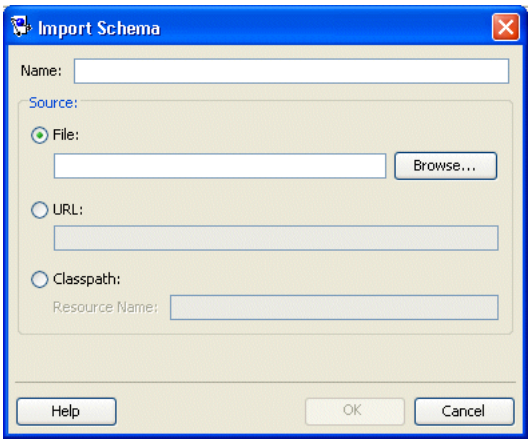
When you import a schema, you define a schema reference that gives TopLink the information it needs to locate the schema itself. Anytime after you import an XML schema, you can update the schema reference (see "Configuring XML Schema Reference" on page 4-36) if necessary.

After importing an XML schema, you can configure XML schema namespaces (see "Configuring XML Schema Namespace" on page 4-37).

To import an XML schema into an EIS project (using XML records) or an EIS project, use this procedure:

1. Right-click the schemas element in the **Navigator** and select **Import Schema** from the context menu. The Import Schema dialog box appears.

Figure 4–33 Import Schema Dialog Box



Use the following information to enter data in each field in the Import Shema dialog box:

Field	Description
Name	Specify the name of this schema. This is the display name that TopLink Workbench uses. It can be different than the name you specify when you configure **Source**.
Source	Select how TopLink Workbench should import the schema.
File	Specify that TopLink Workbench should import the schema from a file. Enter the fully qualified directory path and filename of the schema file.
URL	Specify that TopLink Workbench should import the schema using a URL. Enter the complete URL of the schema file. Note: When importing schemas by URL, ensure you have set your proxy information correctly. See "General Preferences" on page 4-13 for more information.
Classpath	Specify that TopLink Workbench should import the schema from the project classpath.
Resource Name	Enter the fully qualified name of the XML schema file including the name of the package of which it is a part. For example, if your XML schema mySchema.xsd is in `C:\project\config` and you add this directory to your project classpath (see "Configuring Project Classpath" on page 19-3, specify a resource name of `project.config.mySchema.xsd`.

To reimport *a specific schema*, right-click on the specific schema in the **Navigator** and select **Reimport Schema** from the context menu.

To reimport *all schemas in a project*, right-click on **Schemas** in the **Navigator** and select **Reimport All Schemas** from the context menu.

To change a schema's source, right-click on the specific schema in the Navigator window and select **Properties** from the context menu. The Schema Properties dialog appears.

Configuring XML Schema Reference

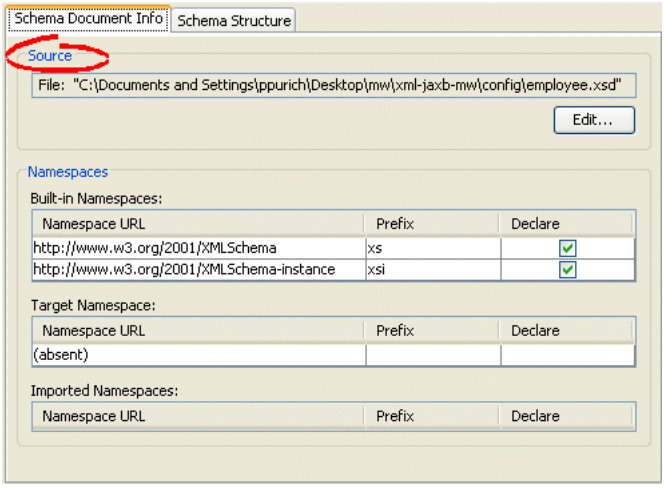
After you import an XML schema (see "Importing an XML Schema" on page 4-34), you can update its source by configuring the schema reference.

Using TopLink Workbench

To specify the source of a schema, use this procedure:

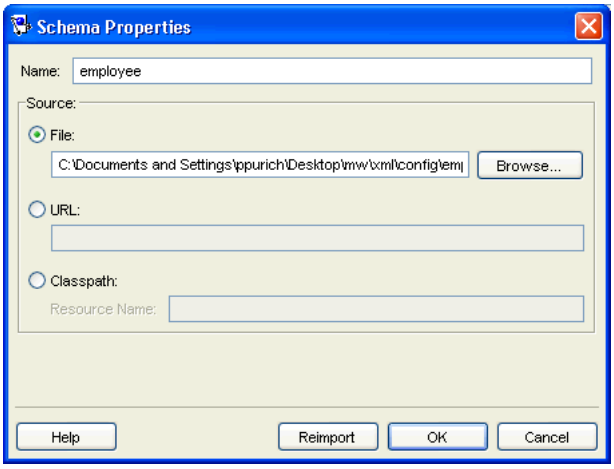
1. Select a schema element in the **Navigator**. Its properties appear in the **Editor**.
2. Click the **Schema Document Info** tab. The Schema Document Info tab appears.

Figure 4–34 Schema Document Info Tab–Source Field



3. Click **Edit** to select a new source for the selected schema. The Schema Properties dialog box appears.

Figure 4–35 Schema Properties Dialog Box



Use the following information to complete each field in the Schema Properties dialog box:

Field	Description
Name	Specify the name of this schema. This is the display name that TopLink Workbench uses. It can be different than the name you specify when you configure **Source**.
Source	Select how TopLink Workbench should import the schema.
File	Specify that TopLink Workbench should import the schema from a file. Enter the fully qualified directory path and filename of the schema file.
URL	Specify that TopLink Workbench should import the schema using a URL. Enter the complete URL of the schema file. **Note**: When importing schemas by URL, ensure you have set your proxy information correctly. See "General Preferences" on page 4-13 for more information.
Classpath	Specify that TopLink Workbench should import the schema from the project classpath.
Resource Name	Enter the fully qualified name of the XML schema file including the name of the package of which it is a part. For example, if your XML schema mySchema.xsd is in `C:\project\config` and you add this directory to your project classpath (see "Configuring Project Classpath" on page 19-3, specify a resource name of `project.config.mySchema.xsd`.

Using Java

Use Java to configure schema reference. Create a descriptor amendment method (see "Configuring Amendment Methods" on page 25-81) that instantiates the appropriate type of `XMLSchemaReference` (`XMLSchemaClassPathReference`, `XMLSchemaFileReference`, or `XMLSchemaURLReference`) and configures the descriptor with it, as follows:

- If you are using `EISDescriptors`, the TopLink runtime does not use the schema reference; no further configuration is required.
- If you are using `XMLDescriptors`, configure the descriptor with the `XMLSchemaReference` using `XMLDescriptor` method `setSchemaReference`.

Configuring XML Schema Namespace

As defined in `http://www.w3.org/TR/REC-xml-names/`, an **XML namespace** is a collection of names, identified by a URI reference, which are used in XML documents as element types and attribute names. To promote reusability and modularity, XML document constructs should have universal names, whose scope extends beyond their containing document. XML namespaces are the mechanism which accomplishes this.

When you import an XML schema (see "Importing an XML Schema" on page 4-34) such as the one that Example 4–3 shows, TopLink Workbench organizes the various namespaces that the XML schema identifies as Table 4–3 shows.

Example 4–3 XML Schema with Namespace Options

```
<xsd:schema
```

```
    xmlns:<prefix>="<URI>"            <!-- TopLink Workbench Built-in Namespace -->
    targetNamespace="<URI>"           <!-- TopLink Workbench Target Namespace -->
    elementFormDefault="qualified"
    attributeFormDefault="unqualified"
    version="10.1.3">
    <xsd:import                       <!-- TopLink Workbench Imported Namespace -->
        namespace="http://xmlns.oracle.com/ias/xsds/opm"
        schemaLocation="object-persistence_1_0.xsd"
    />
...
</xsd:schema>
```

Table 4–3 TopLink Workbench XML Schema Categories

TopLink Workbench Category	Defined By	Purpose	When Needed
Built-in	`xmlns:<prefix>="<URI>"`	Provides access to types defined in other XML schemas for use as is.	If your project uses more than one XML schema or if you want to use `xsi` or `xsd` types.
Target	`targetNamespace="<URI>"`	The namespace you use to qualify the types you define for your application. If set, all XML documents that use these types must use this namespace qualifier.	You may need to specify a target namespace depending on how element and attribute form options are set (see "Element and Attribute Form Options" on page 17-23).
Imported	`xsd:import`	Provides access to types defined in the corresponding built-in XML schema so that you can extend the built-in types. Extended types must be qualified by the target namespace.	If your project uses more than one XML schema and you want to extend one or more built-in types.

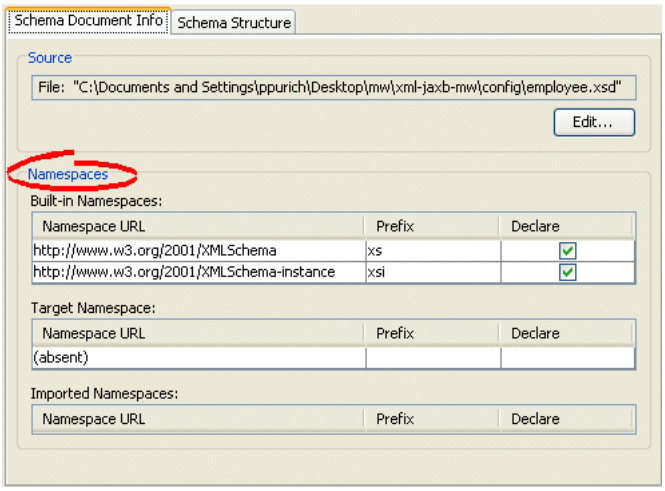
For more information, see "Understanding XML Namespaces" on page 17-22.

Using TopLink Workbench

To specify the namespaces of a schema, use this procedure:

1. Select a schema element in the **Navigator**. Its properties appear in the **Editor**.
2. Click the **Schema Document Info** tab. The Schema Document Info tab appears.

Figure 4–36 Schema Document Info Tab–Namespaces Field



Use the following information to complete each Namespaces field in the tab:

Field	Description
Built-in Namespaces	All namespaces defined by `xmlns:<prefix>="<URI>"`. Note that when a schema is imported to the TopLink Workbench (see "Importing an XML Schema" on page 4-34), none of the built-in namespaces' URLs are selected. If you are using inheritance, declare the built-in namespace with `xsi` prefix. Otherwise, TopLink will throw exceptions.
Target Namespaces	All namespaces defined by `targetNamespace="<URI>"`.
Imported Namespaces	All namespaces defined by `xsd:import`.
Prefix	Double-click in the **Prefix** field to specify the prefix that corresponds to the given namespace. When the TopLink runtime marshalls (writes) an object to an XML document, it uses the namespace prefixes you specify here. When the TopLink runtime unmarshalls (reads) an XML document, the document may use any prefix value as long as it corresponds to the appropriate namespace. For more information, see "TopLink Runtime Namespace Resolution" on page 17-26.
Declare	When selected, XML documents must use the corresponding URI qualifier when referring to types from this namespace. XML documents may use a different prefix value as long as that value is associated with the appropriate namespace URI. For more information, see "TopLink Runtime Namespace Resolution" on page 17-26.

Using Java

Using Java, to configure XML schema namespaces for an EIS descriptor (with XML records) or an XML descriptor, create a descriptor amendment method (see "Configuring Amendment Methods" on page 25-81) that uses `EISDescriptor` or `XMLDescriptor` method `getNamespaceResolver` to configure the descriptor's `NamespaceResolver` accordingly as Example 4–4 shows.

Example 4–4 Configuring Namespaces

```
public void addToDescriptor(ClassDescriptor descriptor) {
    descriptor.getNamespaceResolver.put(
        prefix,
        namespaceURI
    );
}
```

Working With Classes

Using TopLink Workbench, you can create Java classes and packages. This section includes information on the following:

- Creating Classes
- Configuring Classes
- Importing and Updating Classes
- Managing Nondescriptor Classes
- Renaming Packages

Creating Classes

Oracle recommends that you develop your Java classes using an IDE such as Oracle JDeveloper and import these existing classes into TopLink Workbench (see "Importing and Updating Classes" on page 4-50)

However, it is sometimes convenient to create and configure classes in TopLink Workbench: for example, when generating an object model from a database schema.

This section includes information on using TopLink Workbench to create Java classes.

For more information on using TopLink Workbench to edit classes, see "Creating Classes" on page 4-40.

Using TopLink Workbench

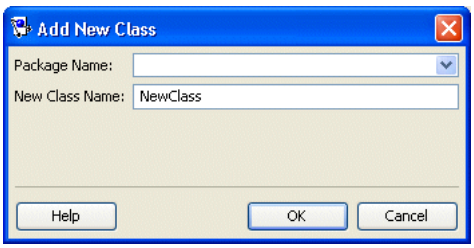
To create new classes and packages from within TopLink Workbench, use this procedure:



1. Select the project in the **Navigator** and click **Create New Class**.

   You can also right-click the project in the **Navigator** and choose **Create New Class** from the context menu or choose **Selected** > **Create New Class** from the menu.

Figure 4–37 Add New Class Dialog Box



Use the following information to enter data in each field on the Add New Class dialog box:

Field	Description
Package Name	Choose an existing package or enter a new package name. If blank, TopLink Workbench uses the default package name.
New Class Name	Enter a class name. The **New Class Name** must be unique within the package.

For more information on using TopLink Workbench to edit classes, see "Configuring Classes" on page 4-41.

Configuring Classes

Oracle recommends that you develop your Java classes using an IDE such as Oracle JDeveloper and import these existing classes into TopLink Workbench (see "Importing and Updating Classes" on page 4-50)

However, it is sometimes convenient to create (see "Creating Classes" on page 4-40) and configure classes in TopLink Workbench: for example, when generating an object model from a database schema.

This section describes using TopLink Workbench to edit classes, including the following:

- Configuring Class Information
- Configuring Class Modifiers
- Configuring Class Interfaces
- Adding Attributes
- Configuring Attribute Modifiers
- Configuring Attribute Type Information
- Configuring Attribute Accessing Methods
- Adding Methods
- Configuring Method Modifiers
- Configuring Method Type Information
- Configuring Method Parameters

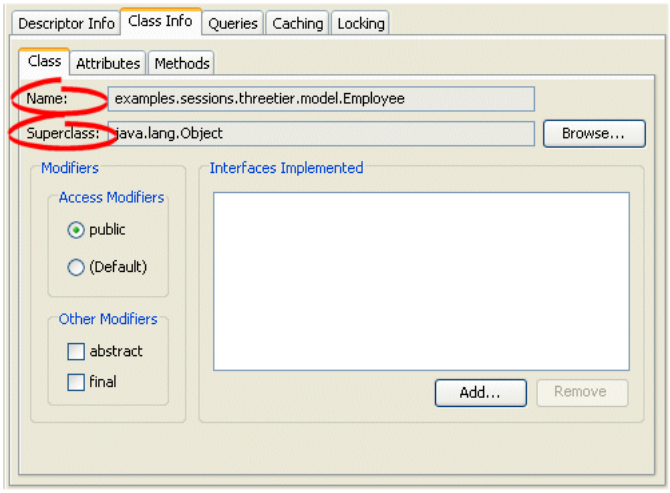
Configuring Class Information

This section includes information on Using TopLink Workbench to configure class information.

Using TopLink Workbench To configure class and superclass information, use this procedure:

1. Select a class in the **Navigator**. Its properties appear in the **Editor**.
2. Click the **Class Info** tab in the **Editor**.
3. Click the **Class** tab.

Figure 4–38 Class Tab, Class Information Fields



Use the following information to enter data in each field on the tab:

Field	Description
Name	The name of the class. This field is for display only.
Superclass	Click **Browse** and select a class and package that contains the class (that is, the superclass).

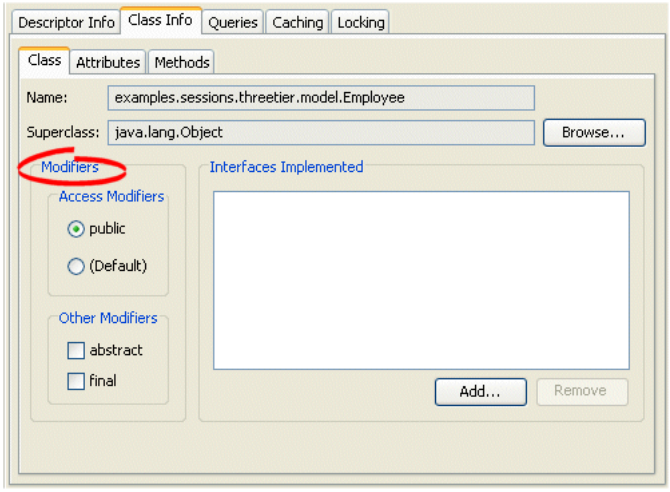
Configuring Class Modifiers

This section includes information on Using TopLink Workbench to configure class modifiers.

Using TopLink Workbench To configure class modifiers, use this procedure:

1. Select a class in the **Navigator**. Its properties appear in the **Editor**.
2. Click the **Class Info** tab in the **Editor**.
3. Click the **Class** tab.

Figure 4–39 Class Tab, Class Modifiers Fields



Use the following information to enter data in each field on the tab:

Field	Description
Access Modifiers	Use to specify whether the class is accessible publicly or not. Only public classes are visible to the Oracle TopLink Workbench.
Other Modifiers	Specify if the class is **Final** or **Abstract**, or both. Final classes are not included in the superclass selection lists for other classes to extend.

Configuring Class Interfaces

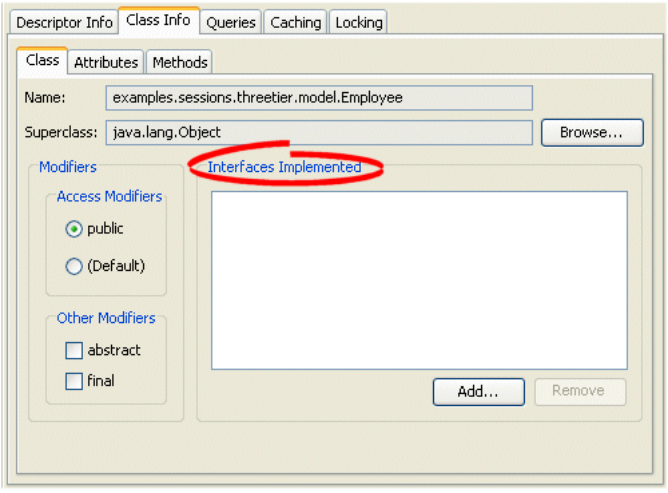
This section includes information on Using TopLink Workbench to specify the interfaces implemented by a class. You can choose any interface in the TopLink Workbench classpath (see "Configuring Project Classpath" on page 19-3).

Although you may add interfaces to a project directly (see "Importing and Updating Classes" on page 4-50), you do not need to do so in order to configure a class to implement an interface.

Using TopLink Workbench To implement interfaces, use this procedure:

1. Select a class in the **Navigator**. Its properties appear in the **Editor**.
2. Click the **Class Info** tab in the **Editor**.
3. Click the **Class** tab.

Figure 4–40 Class Tab, Interfaces Implemented Fields



Use the following information to enter data in the Interfaces Implemented field on the Class tab:

Field	Description
Interfaces Implemented	To add an interface, click **Add** and select the interface and package. To remove an interface, select the interface and click **Remove**

Adding Attributes

This section includes information on Using TopLink Workbench to add an attribute to a class.

Using TopLink Workbench To add a new attribute (field) to the descriptor, click **Add.**

To delete an existing attribute, select the attribute and click **Remove**.

To rename an existing attribute, select the attribute and click on **Rename**.

The **Attributes** tab contains the following tabs:

- General
- Accessors

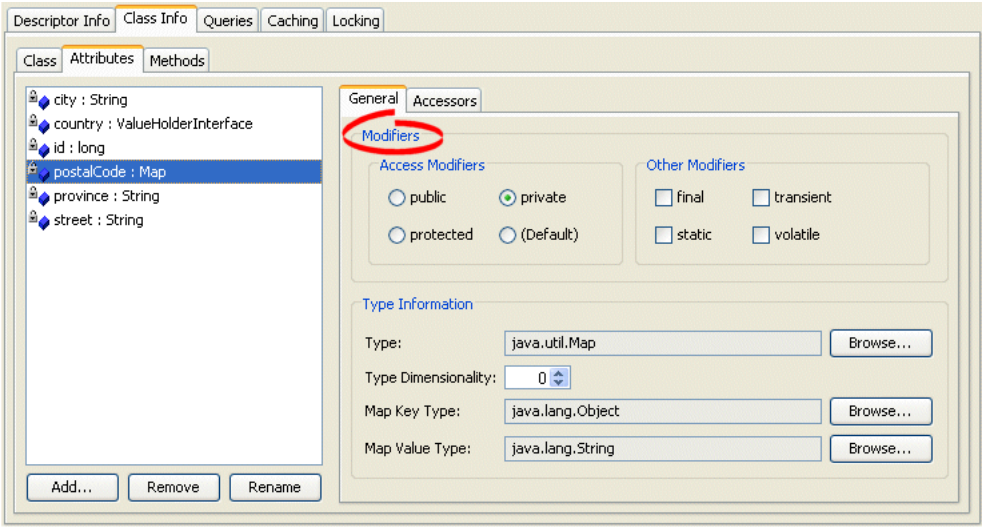
Configuring Attribute Modifiers

This section includes information on Using TopLink Workbench to configure attribute modifiers.

Using TopLink Workbench To specify access modifiers, use this procedure:

1. Select a class in the **Navigator**. Its properties appear in the **Editor**.
2. Click the **Class Info** tab in the **Editor**.
3. Click the **Attributes** tab. The Attributes tab contains two sub-tabs.
4. Click the **General** tab.

Figure 4–41 Attributes Tab, Modifiers Fields



Use the following information to enter data in the Modifiers fields on the Attributes tab:

Field	Description
Access Modifiers	Specify how the attribute is accessible: **Public** **Protected**–only visible within its own package and subclasses. **Private**–not visible for subclasses **Default**–only visible within its own package
Other Modifiers	Specify whether the attribute is **Final**, **Static**, **Transient**, or **Volatile**. Note: Selecting some modifiers may disable others.

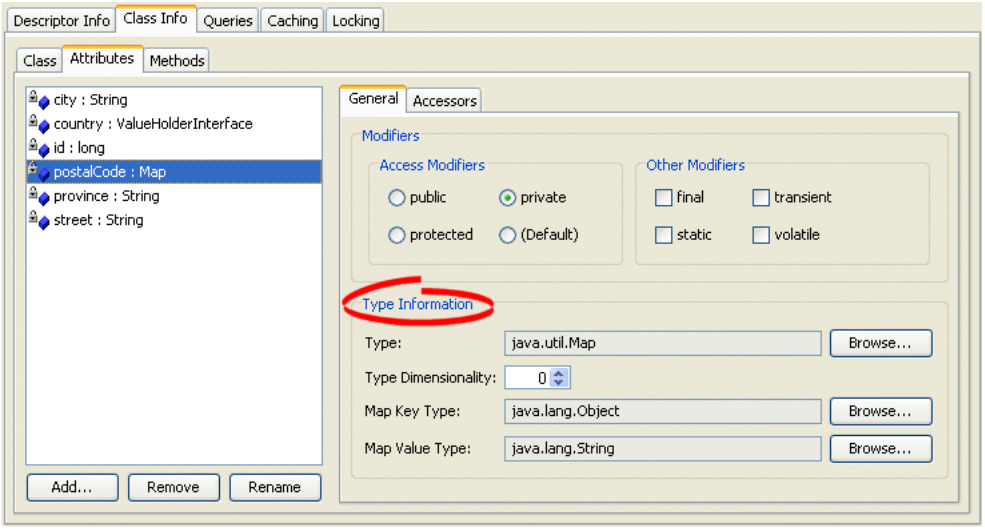
Configuring Attribute Type Information

This section includes information on Using TopLink Workbench to configure attribute type information.

Using TopLink Workbench To specify attribute type information, use this procedure:

1. Select a class in the **Navigator**. Its properties appear in the **Editor**.
2. Click the **Class Info** tab in the **Editor**.
3. Click the **Attributes** tab. The Attributes tab contains two sub-tabs.
4. Click the **General** tab.

Figure 4–42 Attributes Tab, Type Information Fields



Use the following information to enter data in Type Information fields on the Attributes tab:

Field	Description
Type	Click **Browse** and select a class and package for the attribute.
Type Dimensionality	Specify the length of an array. This field applies only if **Type** is an array.

Field	Description
Value Type	Click **Browse** and select a class and package for the attribute. This field applies for `ValueHolderInterface` types only.
Map Key Type	Click **Browse** and select a class and package for the attribute. This field applies for `Map` types only.
Map Value Type	Click **Browse** and select a class and package for the attribute. This field applies for `Map` types only.
Element Type	Click **Browse** and select a class and package for the attribute. This field applies for `List` types only.

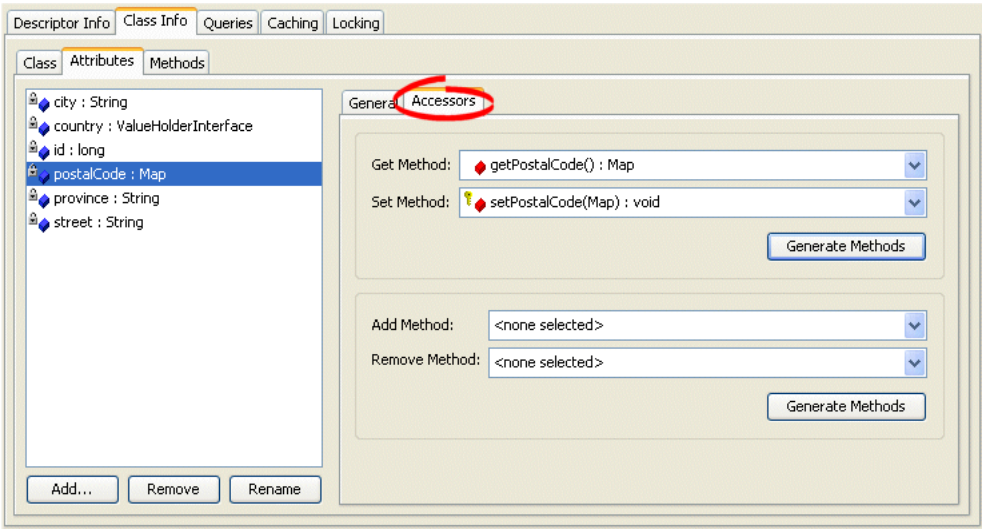
Configuring Attribute Accessing Methods

This section includes information on Using TopLink Workbench to configure attribute accessing methods. If you change an attribute and regenerate the accessing methods, TopLink *does not* remove any previously generated methods.

Using TopLink Workbench To specify attribute accessing methods, use this procedure:

1. Select a class in the **Navigator**. Its properties appear in the **Editor**.
2. Click the **Class Info** tab in the **Editor**.
3. Click the **Attributes** tab. The Attributes tab contains two sub-tabs.
4. Click the **Accessors** tab.

Figure 4–43 Attributes Tab, Accessors Fields



Use the following information to complete the Accessors fields on the Attributes tab:

Field	Description
Get Method	Choose the `get` method for the attribute. This field applies for non-`Collection` types only.
Set Method	Choose the `set` method for the attribute. This field applies for non-`Collection` types only.

Field	Description
Add Method	Choose the `add` method for the attribute. This field applies for `List` and `Map` types only.
Remove Method	Choose the `remove` method for the attribute. This field applies for `List` and `Map` types only.
Value Holder Get Method	Choose the method used to return the `ValueHolderInterface` type. This field applies for `ValueHolderInterface` types only.
Value Holder Set Method	Choose the method used to set the `ValueHolderInterface` type. This field applies for `ValueHolderInterface` types only.
Value Get Method	Choose the method used to return the actual value. This field applies for `ValueHolderInterface` types only.
Value Set Method	Choose the method used to set the actual value. This field applies for `ValueHolderInterface` types only.

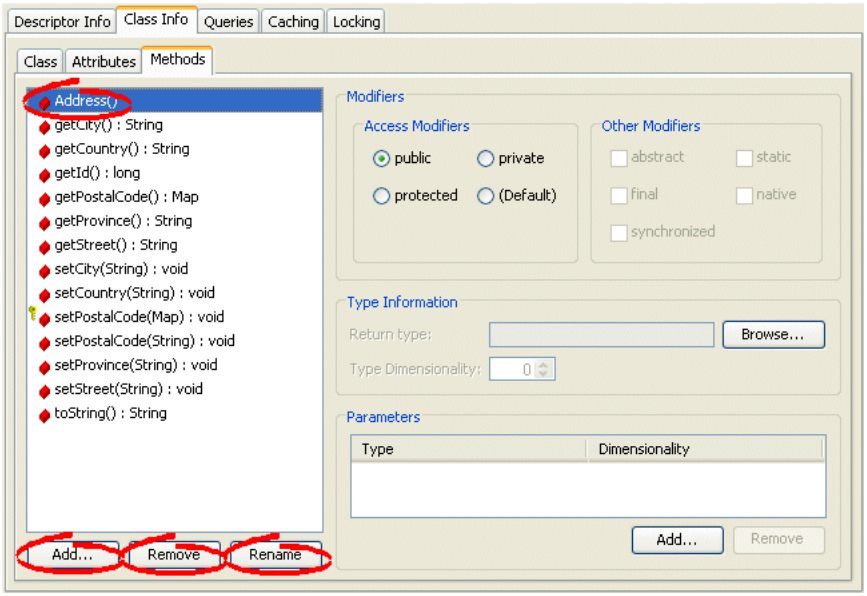
Adding Methods

This section includes information on Using TopLink Workbench to add a method to a class.

Using TopLink Workbench To add or remove methods, use this procedure:

1. Select a class in the **Navigator**. Its properties appear in the **Editor**.
2. Click the **Class Info** tab in the **Editor**.
3. Click the **Methods** tab.

Figure 4–44 Class Info–Methods Tab



To add a new method to the descriptor, click **Add.**

To delete an existing method, select the method and click **Remove**.

To rename an existing method, select the method and click **Rename**.

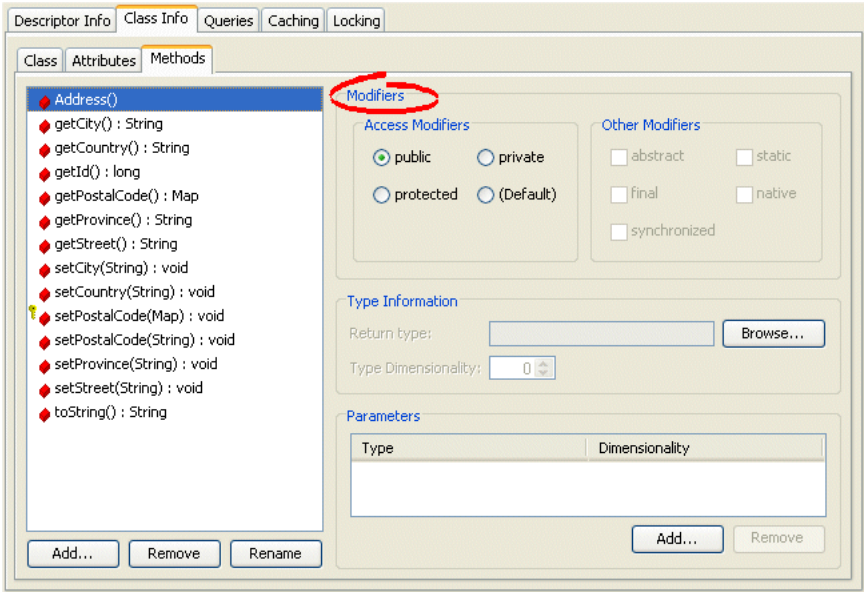
Configuring Method Modifiers

This section includes information on Using TopLink Workbench to configure method modifiers.

Using TopLink Workbench To specify access modifiers, use this procedure:

1. Select a class in the **Navigator**. Its properties appear in the **Editor**.
2. Click the **Class Info** tab in the **Editor**.
3. Click the **Methods** tab.

Figure 4–45 Methods Tab, Modifiers Fields



Use the following information to enter data in Modifiers fields on the Methods tab:

Field	Description
Access Modifiers	Specify how the method can be accessed: **Public** **Protected**–only visible within its own package and subclasses. **Private**–not visible for subclasses. **Default**–only visible within its own package.
Other Modifiers	Specify whether the method is **Abstract, Final**, **Synchronized, Static**, or **Native**. Note: Selecting some modifiers may disable others.

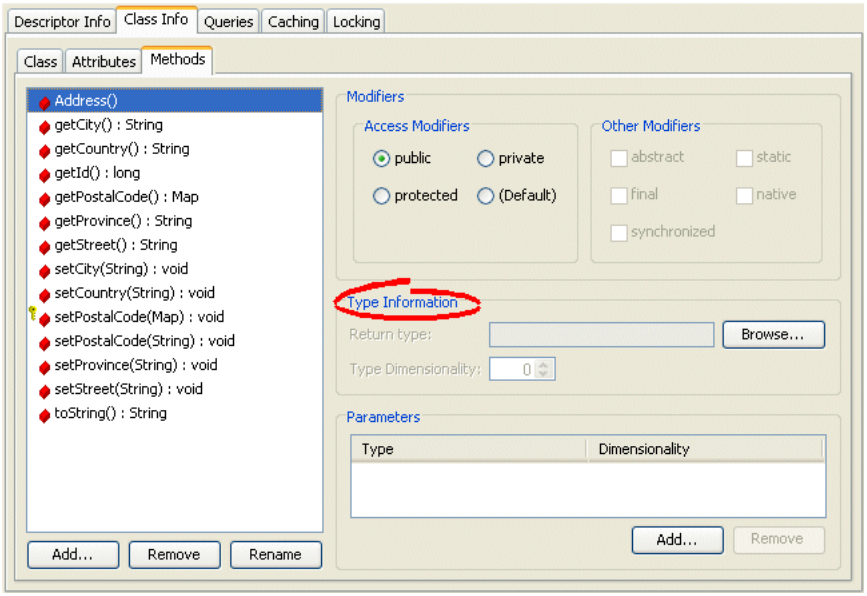
Configuring Method Type Information

This section includes information on Using TopLink Workbench to configure method type information.

Using TopLink Workbench To specify method type information, use this procedure:

1. Select a class in the **Navigator**. Its properties appear in the **Editor**.
2. Click the **Class Info** tab in the **Editor**.
3. Click the **Methods** tab.

Figure 4–46 Methods Tab, Type Information Fields



Use the following information to enter data in Type Information fields on the Methods tab:

Field	Description
Return Type	Click **Browse** and select a class and package for the method.
Type Dimensionality	Specify the length of an array. This field applies only if **Type** is an array.

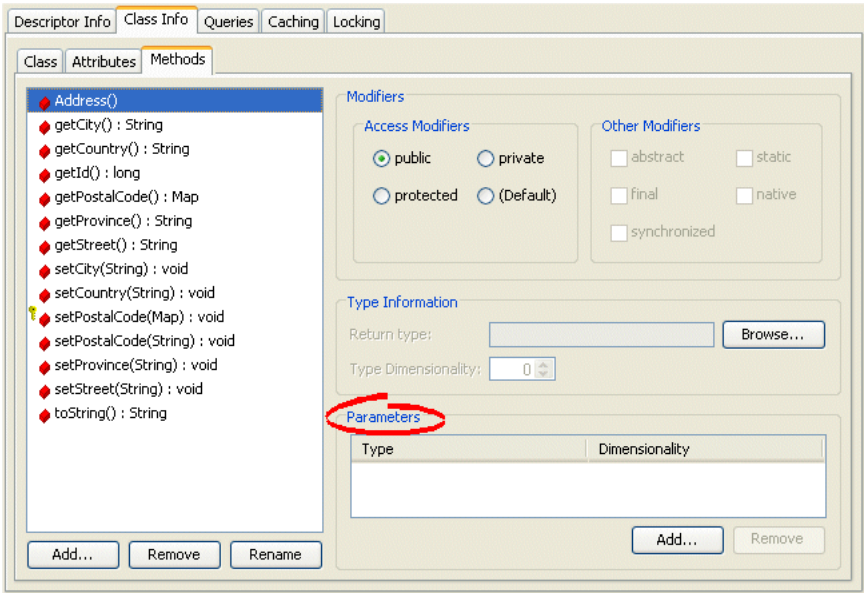
Configuring Method Parameters

This section includes information on Using TopLink Workbench to configure method parameters.

Using TopLink Workbench To specify additional method parameters, use this procedure:

1. Select a class in the **Navigator**. Its properties appear in the **Editor**.
2. Click the **Class Info** tab in the **Editor**.
3. Click the **Methods** tab.

Figure 4–47 Methods Tab, Method Parameters Fields



Use the following information to enter data in Parameters fields on the Methods tab:

Field	Description
Type	Click **Browse** and select a class and package for the method.
Dimensionality	Specify the length of an array. This field applies only if **Type** is an array.

Importing and Updating Classes

This section includes information on Using TopLink Workbench to import and update Java classes.

You can import Java classes and interfaces created in any IDE.

You can import any class on the system classpath or project classpath.

If a class exists on both the system classpath and the project classpath, TopLink Workbench will update the class from the system classpath. To update or refresh from the project classpath, remove the class from the system classpath and restart TopLink Workbench.

For more information, see "Configuring Project Classpath" on page 19-3.

Using TopLink Workbench

Use this procedure to update or refresh the classes in the TopLink Workbench project.

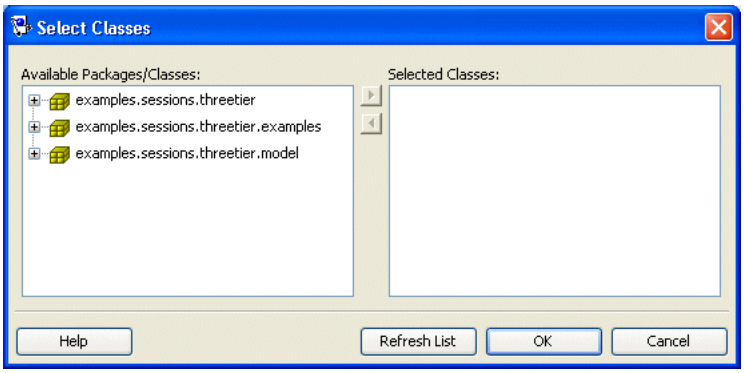
1. Define the available classes and packages for the project on the **General** tab. See "Configuring Project Classpath" on page 19-3 for information on classes and packages.



2. Click **Add or Refresh Class**. The Select Classes dialog box appears.

   You can also update the classes by choosing **Selected** > **Add or Refresh Classes** from the menu.

Figure 4–48 Select Classes Dialog Box



Select the packages or classes (or both) to import into the project and click **OK**. TopLink Workbench adds the new classes to your project in the **Navigator**.

By default, TopLink Workbench creates the following descriptor types for each package and class (depending on your project type):

- Relational projects–Relational class descriptors (see "Relational Class Descriptors" on page 24-2)
- EIS projects–EIS composite descriptors (see "EIS Composite Descriptors" on page 24-5)
- XML projects–XML descriptors (see "XML Descriptors" on page 23-12)

See Chapter 24, "Creating a Descriptor" for more information.

> **Note:** If the class exists on both the system classpath and the project classpath, TopLink Workbench will update the class from the system classpath. To update or refresh from the project classpath, remove the class from the system classpath and restart TopLink Workbench.

To Remove a Class From a Project, do the following:



Select the descriptor and click **Remove**, or choose **Selected** > **Remove** from the menu.

Managing Nondescriptor Classes

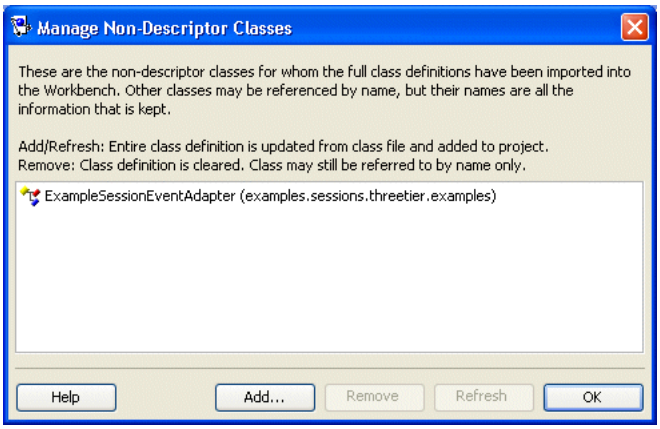
Some of the mappings in your TopLink project may reference classes that do not have TopLink descriptors or are not included in the project.

To add, remove, or refresh Java classes that do not have TopLink descriptors, use this procedure:

From the menu, select **Workbench > Manage Non-Descriptor Classes**. The Manage Non-Descriptor Classes dialog box appears.

You can access the dialog box by right-clicking the TopLink project icon in the **Navigator** and selecting **Manage Non-Descriptor Classes** from the context menu.

Figure 4–49 Manage Non-Descriptor Classes Dialog Box



Select one of the following options:

- To add new classes, click **Add**. The Select Classes dialog box appears.
- To add new classes, click **Add**. The Select Classes dialog box appears (see Figure 4–48, "Select Classes Dialog Box" on page 4-51.

  Only classes that have been added to the project's class path can be added as nondescriptor classes. See "Configuring Project Classpath" on page 19-3 for more information.
- To delete an existing class, select the class and click **Remove**.
- To refresh the classes (for example, if you edited the classes in an IDE), click **Refresh**.

Renaming Packages

When you add classes to a project, TopLink Workbench shows the classes contained in the package to which they belong (see "Using the Navigator" on page 4-9).

You can use TopLink Workbench to change the package statements in all the Java classes of a selected package (to move the all the classes contained by the selected package to a new package). This is useful if you are refactoring an existing TopLink Workbench project.

Note: The TopLink Workbench package rename feature is not intended for migrating projects from older versions of TopLink: for this, you must still use the TopLink Package Renamer. The Package Renamer updates import statements for TopLink classes: it does not change the package statements in user application classes.

For information on the TopLink Package Renamer, refer to *Oracle TopLink Release Notes* and *Oracle TopLink Getting Started Guide* for more information.

For more information on using TopLink Workbench to edit classes, see "Configuring Classes" on page 4-41.

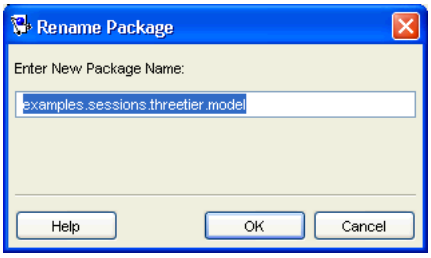
Using TopLink Workbench

To change the package of an existing class in TopLink Workbench, use this procedure:

1. Right-click the package in the **Navigator** and select **Rename**.

   You can also select the package and choose **Selected** > **Rename** from the menu.

Figure 4–50 Rename Package Dialog Box



Enter the package name and click **OK**. TopLink Workbench changes the name of the package in the Navigator window.

For more information on using TopLink Workbench to edit classes, see "Configuring Classes" on page 4-41.

Integrating TopLink Workbench With Apache Ant

If you use the Apache Ant Java-based build tool, you can use the Ant task and type definitions that TopLink provides to invoke certain TopLink Workbench functions from an Ant build file. Using these tasks, you can integrate TopLink Workbench into your automated build process.

This section describes the following:

- Configuring Ant to Use TopLink Workbench Tasks
- Understanding TopLink Workbench Ant Task API
- Creating TopLink Workbench Ant Tasks

For more information about Ant, see `http://ant.apache.org/manual/`.

Configuring Ant to Use TopLink Workbench Tasks

Before you can use TopLink Workbench tasks in your Ant build files, you must consider their library dependencies (see "Library Dependencies" on page 4-53).

To declare TopLink Workbench tasks in your Ant `build.xml` file, declare them directly (see "Declaring TopLink Workbench Tasks" on page 4-54).

Library Dependencies

In addition to the Ant library dependencies (see `http://ant.apache.org/manual/install.html#librarydependencies`), Table 4–4 lists the TopLink-specific JAR files that must be in your Ant classpath.

Table 4–4 TopLink Workbench Ant Task Library Dependencies

JAR Name	Needed For ...	Available At ...
`toplinkmw.jar`	TopLink Workbench Ant task and type definitions.	*<TOPLINK_HOME>*`/jlib`

Declaring TopLink Workbench Tasks

After you declare the TopLink Workbench task definitions (see Table 4–6) and data definitions (see Table 4–4) in the `toplink-ant-lib.xml` file as Example 4–5 shows, you can use a TopLink Workbench task in a `build.xml` file, as Example 4–6 shows:

Example 4–5 Declaring TopLink Workbench Ant Task and Data Types in a toplink-ant-lib.xml File

```
<?xml version="1.0"?>
<antlib>
    <taskdef name="mappings.export"
        classname="oracle.toplink.workbench.ant.taskdefs.ExportDeploymentXMLTask" />

    <taskdef name="mappings.validate"
        classname="oracle.toplink.workbench.ant.taskdefs.MappingsValidateTask" />

    <taskdef name="session.validate"
        classname="oracle.toplink.workbench.ant.taskdefs.SessionValidateTask" />

    <typedef name="ignoreerror"
        classname="oracle.toplink.workbench.ant.typedefs.IgnoreError" />

    <typedef name="ignoreerrorset"
        classname="oracle.toplink.workbench.ant.typedefs.IgnoreErrorSet" />

    <typedef name="loginspec"
        classname="oracle.toplink.workbench.ant.typedefs.LoginSpec" />
</antlib>
```

Example 4–6 Specifying the toplink-ant-lib.xml File in the build.xml File

```
<project name="MyBuild" default="validate.session" basedir="." xmlns:toplink="toplinklib">

    <typedef file = "toplink-ant-lib.xml" classpathref = "mw.classpath" uri = "toplinklib" />
...
</project>
```

Understanding TopLink Workbench Ant Task API

Table 4–5 lists the TopLink Workbench Ant task definitions that TopLink provides.

Table 4–5 TopLink Workbench Ant Task Definitions

Task Name	TopLink Class
mappings.validate	`oracle.toplink.workbench.ant.taskdefs.MappingsValidateTask`
session.validate	`oracle.toplink.workbench.ant.taskdefs.SessionValidateTask`
mappings.export	`oracle.toplink.workbench.ant.taskdefs.ExportDeploymentXMLTask`

Table 4–6 lists the TopLink Workbench Ant type definitions that TopLink provides.

Table 4–6 TopLink Workbench Ant Type Definitions

Type Name	TopLink Class
ignoreerror	`oracle.toplink.workbench.ant.typedefs.IgnoreError`
ignoreerrorset	`oracle.toplink.workbench.ant.typedefs.IgnoreErrorSet`
loginspec	`oracle.toplink.workbench.ant.typedefs.LoginSpec`

Creating TopLink Workbench Ant Tasks

Example 4–7 shows a typical Ant `build.xml` file that declares and uses the TopLink Workbench Ant task and type definitions.

Example 4–7 Example Ant Build File with TopLink Workbench Ant Tasks

```
<project name="MyBuild" default="validate.session" basedir="." xmlns:toplink="toplinklib">
    <!-- ================================================== -->
    <!--  Properties                                        -->
    <!-- ================================================== -->
    <target name="init">
        <property file="build.properties"/>

        <property name = "toplink.mwp.dir" value = "${basedir}/mw"/>
        <property name = "toplink.sessions.dir" value = "${basedir}/config"/>
        <property name = " myProject.classes" value = "${basedir}/classes "/>

        <path id = "database.classpath">
            <pathelement path = "${toplink.home}/jlib /dms.jar"/>
            <pathelement path = "${toplink.home}/jlib /OracleThinJDBC.jar"/>
        </path>
        <path id = "toplink.classpath">
            <pathelement path = "${toplink.home}/jlib /toplink.jar"/>
            <pathelement path = "${toplink.home}/jlib /ejb.jar"/>
            <pathelement path = "${toplink.home}/jlib /xmlparserv2.jar"/>
            <pathelement path = "${toplink.home}/jlib /antlr.jar"/>
        </path>
        <path id = "mw.classpath">
            <pathelement path = "${toplink.home}/jlib /tlmwcore.jar"/>
            <pathelement path = "${toplink.home}/jlib /toplinkmw.jar"/>
        </path>
        <path id = "mwplatforms.classpath">
            <pathelement path = "${toplink.home}/config"/>
        </path>

        <typedef file = "toplink-ant-lib.xml"

                 classpathref = "mw.classpath"
                 uri = "toplinklib" />
    </target>
    <!-- ================================================== -->
    <!--  Define task parameter                             -->
    <!-- ================================================== -->
    <target name="parameter.definition" depends="init">
        <toplink:ignoreerrorset id = "ignoreErrors">
        <toplink:ignoreerror code = "0233" />
        </toplink:ignoreerrorset>

        <toplink:loginspec id = "loginSpec"
        url = "jdbc:cloudscape:stagedb;create=true"
        driverclass = "COM.cloudscape.core.JDBCDriver"
        user = "scott"
        password="tiger" />
    </target>
     <!-- ================================================== -->
    <!--  Validate the MW Project                           -->
    <!-- ================================================== -->
    <target name="validate.project" depends="parameter.definition">

        <toplink:mappings.validate
            projectfile = "${toplink.mwp.dir}/myProject.mwp"
            reportfile = "${toplink.mwp.dir}/problem-report.html"
            reportformat = "html"
            property = "mw-valid"
            classpathref = "mwplatforms.classpath" >
```

```
            <toplink:classpath refid = "mw.classpath" />
            <toplink:classpath refid = "toplink.classpath" />

            <toplink:ignoreerrorset refid = "ignoreErrors"/>

        </toplink:mappings.validate>
    </target>
    <!-- ================================================== -->
    <!--  TopLink deployment descriptor XML generation      -->
    <!-- ================================================== -->
    <target name="export.deployment" depends="validate.project"  if="mw-valid">

        <toplink:mappings.export
            projectfile = "${toplink.mwp.dir}/myProject.mwp"
            deploymentfile = "${toplink.sessions.dir}/sessions.xml"
            property = "export-completed"
            failonerror = "true"
            classpathref = "toplink.classpath">

            <toplink:classpath refid = "mw.classpath" />
            <toplink:classpath refid = "mwplatforms.classpath" />

            <toplink:ignoreerrorset refid = "ignoreErrors"/>
            <toplink:loginspec refid = "loginSpec" />
        </toplink:mappings.export>
    </target>
    <!-- ================================================== -->
    <!--  TopLink Session Validate                          -->
    <!-- ================================================== -->
    <target name="validate.session"  depends="export.deployment" if="export-completed">

        <toplink:session.validate
            sessionsfile = "${toplink.sessions.dir}/sessions.xml"
            sessionname = "ThreeTierEmployee"
            property = "session-valid"
            classpathref = "toplink.classpath"
            classpath = "${ myProject.classes}" >

            <toplink:classpath refid = "mw.classpath" />
            <toplink:classpath refid = " database.classpath" />

            <toplink:loginspec refid = "loginSpec" />
        </toplink:session.validate>
    </target>
<project>
```

mappings.validate

The `mapings.validate` task is a testing task that you use to list of all the problems in a TopLink Workbench project (`.mwp`) file.

This task provides the ability to:

- log all the problems to a file in text or HTML format
- set an Ant property to indicate that the TopLink Workbench project is valid (has no errors)

Parameters

Table 4–7 mappings.validate Task Parameters

Attribute	Description	Required
projectfile	Fully qualified TopLink Workbench projects file name (.mwp).	Yes
reportfile	Fully qualified file name to which to write the output.	No
reportformat	The format of the generated output. Must be `html` or `text`.	No–default to `text`.
classpath	Project classpath.	No
classpathref	Reference to a path defined elsewhere.	No
property	The name of the property to set (true if there is no problem).	No

Parameters Specified as Nested Elements

You can specify the following parameters as nested elements of this task:

- classpath
- "ignoreerror" on page 4-60
- "ignoreerrorset" on page 4-61

Examples

Example 4–8 shows a typical `mappings.validate` task.

Example 4–8 A mappings.validate Task

```
<toplink:mappings.validate
    projectfile = "${toplink.mwp.dir}/myProject.mwp"
    reportfile = "${toplink.mwp.dir}/problem-report.html"
    reportformat = "html"
    property = "mw-valid"
    classpath = "${mwplatforms.classpath}" >

    <toplink:classpath refid = "mw.classpath" />
    <toplink:classpath refid = "toplink.classpath" />

    <toplink:ignoreerrorset refid = "ignoreErrors"/>
    <toplink:ignoreerror code = "0555" />
</toplink:mappings.validate>
```

session.validate

The `session.validate` task is a testing task that you use to test your TopLink deployment XML by running TopLink.

This task provides the ability to:

- specify the test type using a nested element
- set an Ant property to indicate that the TopLink Workbench project is valid (has no errors)

Parameters

Table 4–8 session.validate Task Parameters

Attribute	Description	Required
sessionsfile	Fully qualified `sessions.xml` file.	No–default to `sessions.xml` and to classpath.
sessionname	Name of the session to test.	Yes
classpath	Project classpath.	No
classpathref	Reference to a path defined elsewhere.	No
property	The name of the property to set (true if valid).	No

Parameters Specified as Nested Elements

You can specify the following parameters as nested elements of this task:

- classpath
- "loginspec" on page 4-62

Examples

Example 4–9 shows a typical `session.validate` task.

Example 4–9 A session.validate Task

```
<toplink:session.validate
    sessionsfile = "${toplink.sessions.dir}/sessions.xml"
    sessionname = "ThreeTierEmployee"
    property = "session-valid"
    classpathref = "toplink.classpath"
    classpath = "${ myProject.classes}" >

    <toplink:classpath refid = "mw.classpath" />
    <toplink:classpath refid = " database.classpath" />

    <toplink:loginspec refid = "loginSpec" />
</toplink:session.validate>
```

mappings.export

The `mappings.export` task is a generation task that you use to generate a TopLink deployment XML file for a given TopLink Workbench project (`.mwp`). The `mappings.export` task executes a `mappings.validate` (see "mappings.validate" on page 4-56) before executing. A `BuildException` is thrown if validation fails.

This task provides the ability to override the TopLink Workbench project database login information.

Parameters

Table 4–9 mappings.export Task Parameters

Attribute	Description	Required
projectfile	Fully qualified TopLink Workbench projects file name (`.mwp`).	Yes

Table 4–9 (Cont.) mappings.export Task Parameters

Attribute	Description	Required
deploymentfile	Fully qualified TopLink project deployment file name (`.xml`).	No–default to the name specified in the TopLink Workbench project (`.mwp`).
ejbjarxmldir	The directory that contains the `ejb-jar.xml` file (only applicable to J2EE project).	No–default to the directory specified in the TopLink Workbench project (`.mwp`).
classpath	Project classpath.	No
classpathref	Reference to a path defined elsewhere.	No
failonerror	Indicates whether the build will continue even if there are export errors; defaults to `true`.	No
property	The name of the property to set (`true` if export completed successfully).	No

Parameters Specified as Nested Elements

You can specify the following parameters as nested elements of this task:

- classpath
- "loginspec" on page 4-62
- "ignoreerror" on page 4-60
- "ignoreerrorset" on page 4-61

Examples

Example 4–9 shows a typical `mappings.export` task.

Example 4–10 A mappings.export Task

```
<toplink:mappings.export
    projectfile = "${toplink.mwp.dir}/myProject.mwp"
    deploymentfile = "${toplink.sessions.dir}/sessions.xml"
    property = "export-completed"
    failonerror = "true"
    classpathref = "toplink.classpath">

    <toplink:classpath refid = "mw.classpath" />
    <toplink:classpath refid = "mwplatforms.classpath" />

    <toplink:ignoreerrorset refid = "ignoreErrors"/>
    <toplink:ignoreerror code = "0545" />
    <toplink:loginspec
            url = "jdbc:cloudscape:stagedb;create=true"
            driverclass = "COM.cloudscape.core.JDBCDriver"
            user = "scott"
            password="tiger" />
</toplink:mappings.export>
```

classpath

Use the classpath element to define the Java classpath necessary to run a task. For more information, see `http://ant.apache.org/manual/using.html#path`.

Parameters

Table 4–10 classpath Element Parameters

Attribute	Description	Required
location	Specifies a single file or directory relative to the project's base directory (or an absolute filename).	No
path	Specifies one or multiple files or directories separated by a colon or semicolon.	No
refid	Reference to a path defined elsewhere.	No

Parameters Specified as Nested Elements

You can specify the following parameters as nested elements of this task:

- pathelement
- fileset
- dirset
- filelist

Examples

Example 4–11 shows a typical `classpath` element.

Example 4–11 A classpath Element

```
<classpath>
    <pathelement path="${classpath}"/>
        <fileset dir="lib">
            <include name="**/*.jar"/>
        </fileset>
    <pathelement location="classes"/>
        <dirset dir="${build.dir}">
            <include name="apps/**/classes"/>
            <exclude name="apps/**/*Test*"/>
        </dirset>
    <filelist refid="third-party_jars"/>
</classpath>
```

ignoreerror

Use the `ignoreerror` element to instruct a TopLink Ant task to ignore a specific TopLink Foundation Library (see "TopLink Exception Reference" on page 13-1) or TopLink Workbench (see "TopLink Workbench Error Reference" on page 14-1) run-time error code.

To instruct a TopLink Ant task to ignore multiple error codes, consider using an `ignoreerrorset` element (see "ignoreerrorset" on page 4-61).

Parameters

Table 4–11 ignoreerror Element Parameters

Attribute	Description	Required
code	Error code of the problem to ignore.	Yes

Parameters Specified as Nested Elements

You cannot specify parameters as nested elements of this element.

Examples

Example 4–12 shows a typical `ignoreerror` element. This element instructs a `mappings.export` task to ignore TopLink Workbench error code 0545.

Example 4–12 An ignoreerror Element

```
<toplink:mappings.export
    projectfile = "${toplink.mwp.dir}/myProject.mwp"
    deploymentfile = "${toplink.sessions.dir}/sessions.xml"
    classpathref = "toplink.classpath">

    <toplink:classpath refid = "mw.classpath" />
    <toplink:classpath refid = "mwplatforms.classpath" />

    <toplink:ignoreerror code = "0545" />

</toplink:mappings.export>
```

ignoreerrorset

Use the `ignoreerrorset` element to instruct a TopLink Ant task to ignore any of multiple TopLink Foundation Library (see "TopLink Exception Reference" on page 13-1) or TopLink Workbench (see "TopLink Workbench Error Reference" on page 14-1) run-time error codes.

Parameters

Table 4–12 ignoreerrorset Element Parameters

Attribute	Description	Required
id	Unique identifier for this type instance, can be used to reference this type in scripts.	No
refid	Reference to a ignoreerrorset defined elsewhere.	No

Parameters Specified as Nested Elements

You can specify the following parameters as nested elements of this element:

- "ignoreerror" on page 4-60

Examples

Example 4–13 shows a typical `ignoreerrorset` element. This element instructs a `mappings.export` task to ignore all of TopLink Workbench error codes 0402 and 0570. Note that the `mappings.export` task also uses an explicitly `ignoreerror` element: this means that the `mappings.export` task will ignore all of error codes 0402, 0570, and 0545.

Example 4–13 An ignoreerrorset Element

```
<toplink:ignoreerrorset id = "ignoreErrors">
    <toplink:ignoreerror code = "0402" />
    <toplink:ignoreerror code = "0570" />
</toplink:ignoreerrorset>
...
<toplink:mappings.export
    projectfile = "${toplink.mwp.dir}/myProject.mwp"
    deploymentfile = "${toplink.sessions.dir}/sessions.xml"
    classpathref = "toplink.classpath">
```

```
    <toplink:classpath refid = "mw.classpath" />
    <toplink:classpath refid = "mwplatforms.classpath" />

    <toplink:ignoreerrorset refid = "ignoreErrors"/>
    <toplink:ignoreerror code = "0545" />

</toplink:mappings.export>
```

loginspec

Use the `loginspec` element to instruct a TopLink Ant task to override the project database login information in a TopLink Workbench project. For more information, see Chapter 81, "Understanding Data Access".

> **Note:** You can only use this element with a relational project (see "Relational Projects" on page 17-6).
>
> You cannot use this element with a J2EE project.

Parameters

Table 4–13 loginspec Element Parameters

Attribute	Description	Required
id	Unique identifier for this type instance, can be used to reference this type in scripts.	No
refid	Reference to a loginspec defined elsewhere.	No
driverclass	Fully qualified class of the data source driver (see "Configuring Database Login Connection Options" on page 83-2).	No–default to the class that the TopLink Workbench project specifies.
url	URL of the driver see "Configuring Database Login Connection Options" on page 83-2).	Yes
user	Login user name (see "Configuring User Name and Password" on page 82-1).	No–default to the value that the TopLink Workbench project specifies
password	Login password (see "Configuring User Name and Password" on page 82-1).	No–default to the value that the TopLink Workbench project specifies

Parameters Specified as Nested Elements

You cannot specify parameters as nested elements of this element.

Examples

Example 4–14 shows a typical `loginspec` element.

Example 4–14 A loginspec Element

```
<toplink:mappings.export
    projectfile = "${toplink.mwp.dir}/myProject.mwp"
    deploymentfile = "${toplink.sessions.dir}/sessions.xml"
    classpathref = "toplink.classpath">

    <toplink:classpath refid = "mw.classpath" />
    <toplink:classpath refid = "mwplatforms.classpath" />
```

```
    <toplink:loginspec
        url = "jdbc:cloudscape:stagedb;create=true"
        driverclass = "COM.cloudscape.core.JDBCDriver"
        user = "scott"
        password="tiger" />
</toplink:mappings.export>
```

5

Using the Schema Manager

The `SchemaManager` and its related classes provide API that you can use from a Java application to specify database tables in a generic format, and then create and modify them in a specific relational database. This decouples your TopLink project from a particular database schema while giving you a programmatic means of creating a database schema based on your TopLink project. For example, you can use the schema manager to recreate a production database in a nonproduction environment. This lets you build models of your existing databases, and modify and test them during development.

> **Note:** You can also create database tables manually during development using TopLink Workbench (see "Creating New Tables" on page 4-22 and "Generating Tables on the Database" on page 4-32).

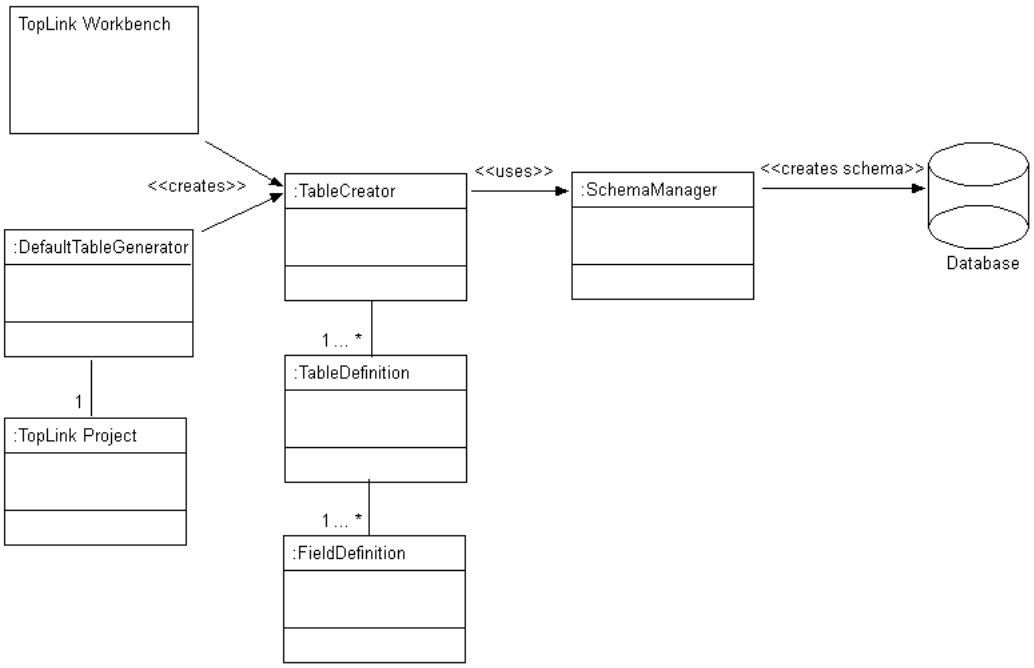
This chapter includes information on the following topics:

- Understanding the Schema Manager
- Creating a Table Creator
- Creating Tables With a Table Creator
- Automatic Database Table Creation

Understanding the Schema Manager

Figure 5–1 summarizes the important `SchemaManager` classes and the primary means of using them.

Figure 5–1 SchemaManager Usage



Although you can use the `SchemaManager` API directly, Oracle recommends that you create a `TableCreator` class and use its API (which, in turn, uses the `SchemaManager`).

You can automatically generate a `TableCreator` using:

- TopLink Workbench during development (see "Using TopLink Workbench During Development" on page 5-4)
- `DefaultTableGenerator` at run time (see "Using the Default Table Generator at Run Time" on page 5-4)

The `TableCreator` class owns one or more `TableDefinition` classes (one for each database table) and the `TableDefinition` class owns one or more `FieldDefinition` classes (one for each field).

The `TableDefinition` class lets you specify a database table schema in a generic format. At run time, TopLink uses the session associated with your TopLink project to determine the specific database type, and uses the generic schema to create the appropriate tables and fields for that database.

After creating a `TableCreator` class, you can use its API to create and drop tables (see "Creating Tables With a Table Creator" on page 5-6). You can also configure TopLink to do this automatically (see "Automatic Database Table Creation" on page 5-6).

Because the schema manager uses Java types rather than database types, it is database-independent. However, because it does not account for database-specific optimizations, it is best-suited for development purposes rather than production. For more information on how the schema manager maps Java types to database types, see "Schema Manager Java and Database Type Conversion" on page 5-3.

Although the schema manager can handle the sequencing configuration that you specify in your TopLink project, if you are using sequencing with non-Oracle databases, there are some sequencing restrictions you should be aware of (see "Sequencing" on page 5-3).

Schema Manager Java and Database Type Conversion

Table 5–1 lists the Java type to database type conversions that the schema manager supports depending on the database platform your TopLink project uses. This list is specific to the schema manager and does not apply to mappings. TopLink automatically performs conversions between any database types within mappings.

Table 5–1 Java and Database Field Type Conversion

Java Type	Oracle	DB2	Sybase	MySQL	MS Access
`java.lang.Boolean`	NUMBER	SMALLINT	BIT default 0	TINYINT(1)	SHORT
`java.lang.Byte`	NUMBER	SMALLINT	SMALLINT	TINYINT	SHORT
`java.lang.Byte[]`	LONG RAW	BLOB	IMAGE	BLOB	LONGBINARY
`java.lang.Character`	CHAR	CHAR	CHAR	CHAR	TEXT
`java.lang.Character[]`	LONG	CLOB	TEXT	TEXT	LONGTEXT
`java.lang.Double`	NUMBER	FLOAT	FLOAT(32)	DOUBLE	DOUBLE
`java.lang.Float`	NUMBER	FLOAT	FLOAT(16)	FLOAT	DOUBLE
`java.lang.Integer`	NUMBER	INTEGER	INTEGER	INTEGER	LONG
`java.lang.Long`	NUMBER	INTEGER	NUMERIC	BIGINT	DOUBLE
`java.lang.Short`	NUMBER	SMALLINT	SMALLINT	SMALLINT	SHORT
`java.lang.String`	VARCHAR2	VARCHAR	VARCHAR	VARCHAR	TEXT
`java.math.BigDecimal`	NUMBER	DECIMAL	NUMERIC	DECIMAL	DOUBLE
`java.math.BigInteger`	NUMBER	DECIMAL	NUMERIC	BIGINT	DOUBLE
`java.sql.Date`	DATE	DATE	DATETIME	DATE	DATETIME
`java.sql.Time`	DATE	TIME	DATETIME	TIME	DATETIME
`java.sql.Timestamp`	DATE	TIMESTAMP	DATETIME	DATETIME	DATETIME

For more information about database platforms that TopLink supports, see "Database Platforms" on page 81-3.

Sequencing

If you generate a `TableCreator` class using TopLink Workbench (see "Using TopLink Workbench During Development" on page 5-4) or `DefaultTableGenerator` (see "Using the Default Table Generator at Run Time" on page 5-4), then sequencing configuration is included in your `TableCreator` according to your TopLink project configuration. In this case, when you use `TableCreator` method `createTables`, it does the following:

- 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

You can use advanced API to handle special cases like Sybase or Microsoft SQL Server native sequencing (see "Using Java" on page 5-4).

For more information about sequencing, see "Understanding Sequencing in Relational Projects" on page 17-14.

Creating a Table Creator

You can automatically generate a `TableCreator` using:

- TopLink Workbench during development (see "Using TopLink Workbench During Development" on page 5-4)
- `DefaultTableGenerator` at run time (see "Using the Default Table Generator at Run Time" on page 5-4)

After creating a `TableCreator` class, you can use its API to create and drop tables (see "Creating Tables With a Table Creator" on page 5-6).

Using TopLink Workbench During Development

To create a `TableCreator` class that you can use in a Java application to recreate a database schema using the `SchemaManager`, use this procedure:

1. Right-click the project in the **Navigator** and choose **Export > Table Creator Java Source** from the context menu. The Table Creator dialog box appears.

   You can also select the table and choose **Selected** > **Export > Table Creator Java Source** from the menu.
2. Enter a name for the table creator class and click **OK**. The Save As dialog box appears.
3. Choose a location for your table creator class and click **OK**. TopLink Workbench exports the table creator Java class to the location you specify.

Using the Default Table Generator at Run Time

To create a `TableCreator` class in Java using the `DefaultTableGenerator`, use this procedure:

1. Create an instance of `DefaultTableGenerator`, passing in an instance of your TopLink project:

   ```
   DefaultTableGenerator myDefTblGen = new DefaultTableGenerator(toplinkProject);
   ```
2. Create a `TableCreator` instance:
   - If you want a `TableCreator` that can support any session, use:

     ```
     TableCreator myTblCre = myDefTblGen.generateDefaultTableCreator();
     ```
   - If you want a `TableCreator` customized for a specific TopLink session, use:

     ```
     TableCreator myTblCre =
     myDefTblGen.generateFilteredDefaultTableCreator(toplinkSession);
     ```

You can also configure TopLink to use the `DefaultTableGenerator` to automatically generate and execute a `TableCreator` at run time (see "Automatic Database Table Creation" on page 5-6).

Using Java

This section describes how to create a `TableCreator` class in Java, including the following:

- Creating a TableCreator Class
- Creating a TableDefinition Class

- Adding Fields to a TableDefinition
- Defining Sybase and Microsoft SQL Server Native Sequencing

Creating a TableCreator Class

To create your own `TableCreator` instance, you should extend `TableCreator` as Example 5–1 shows:

Example 5–1 Creating a TableCreator Class

```
public class MyTableCreator extends oracle.toplink.tools.schemaframework.TableCreator {

    public M7TableCreator() {
        setName("MyTableCreator");
        addTableDefinition(buildADDRESSTable());
    ...
    }

    public TableDefinition buildADDRESSTable() {
        TableDefinition table = new TableDefinition();
       ...
       return table;
    }
...
}
```

Creating a TableDefinition Class

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 TableDefinition

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. TopLink translates the Java class to the appropriate database field type at run time. 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`. For more information, see "Schema Manager Java and Database Type Conversion" on page 5-3.

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. TopLink automatically provides the required information, as necessary.

Defining Sybase and Microsoft SQL Server Native Sequencing

Use `FieldDefinition` method `addIdentityField` to add fields representing a generated sequence number from Sybase or Microsoft SQL Server native sequencing. See "Native Sequencing With a Non-Oracle Database Platform" on page 17-19 for detailed information on using sequencing.

Creating Tables With a Table Creator

After creating a `TableCreator` class (see "Creating a Table Creator" on page 5-4), you can use its API to create and drop tables. The important `TableCreator` methods are as follows (each method takes an instance of `DatabaseSession`):

- `createTables`–this method creates tables, adds constraints, and creates sequence tables and sequences (if sequence tables already exist, this method drops them and recreates them).
- `dropTables`–his method drops all constraints and drops all tables (except sequence tables) that the `TableCreator` defines.
- `createConstraints`–this method creates constraints on all pre-existing tables that the `TableCreator` defines.
- `dropConstraints`–this method drops constraints on all pre-existing tables that the `TableCreator` defines.
- `replaceTables`–this method drops and then creates all tables that the `TableCreator` defines.

Automatic Database Table Creation

If you deploy a CMP project to OC4J configured to use TopLink as the persistence manager, then you can configure OC4J to automatically create (and, optionally, delete) database tables for your persistent objects.

You can configure automatic database table creation at one of three levels as Table 5–2 shows. You can override the system level configuration at the application level and you can override system and application configuration at the EJB module level.

Table 5–2 Configuring Automatic Table Generation

Level	Configuration File	Setting	Values
System (global)	*<OC4J_HOME>*`/config/application.xml`	`autocreate-tables`	`True`[1] or `False`
		`autodelete-tables`	`True` or `False`[1]
Application (EAR)	`orion-application.xml`	`autocreate-tables`	`True`[1] or `False`
		`autodelete-tables`	`True` or `False`[1]
EJB Module (JAR)	`orion-ejb-jar.xml`	`pm-properties` sub-element `default-mapping` attribute `db-table-gen`[2]	`Create`, `DropAndCreate`, or `UseExisting`[3]

[1] Default.

[2] For more information, see "Configuring default-mapping Properties" on page 8-12.

[3] See Table 5–3.

If you configure automatic table generation at the EJB module level, the value you assign to the `db-table-gen` attribute corresponds to the `autocreate-tables` and `autodelete-tables` settings as Table 5–3 shows.

Table 5–3 Equivalent Settings for db-table-gen

db-table-gen Setting	autocreate-tables Setting	autodelete-tables Setting
`Create`	`True`	`False`
`DropAndCreate`	`True`	`True`
`UseExisting`	`False`	NA

You can use this feature in conjunction with default mapping (see "Default Mapping in CMP Projects Using OC4J at Run Time" on page 30-4).

6

Using an Integrated Development Environment

This chapter includes information on using TopLink with an integrated development environment (IDE). This chapter includes the following sections:

- Configuring TopLink for Oracle JDeveloper
- Configuring TopLink Workbench With Source Control Management Software

In addition to the development environment described here, TopLink can be used with *any* J2EE development environment and process.

Configuring TopLink for Oracle JDeveloper

This section contains information on how to configure TopLink for use with Oracle JDeveloper. JDeveloper is a J2EE development environment with end-to-end support to develop, debug, and deploy e-business applications and Web services.

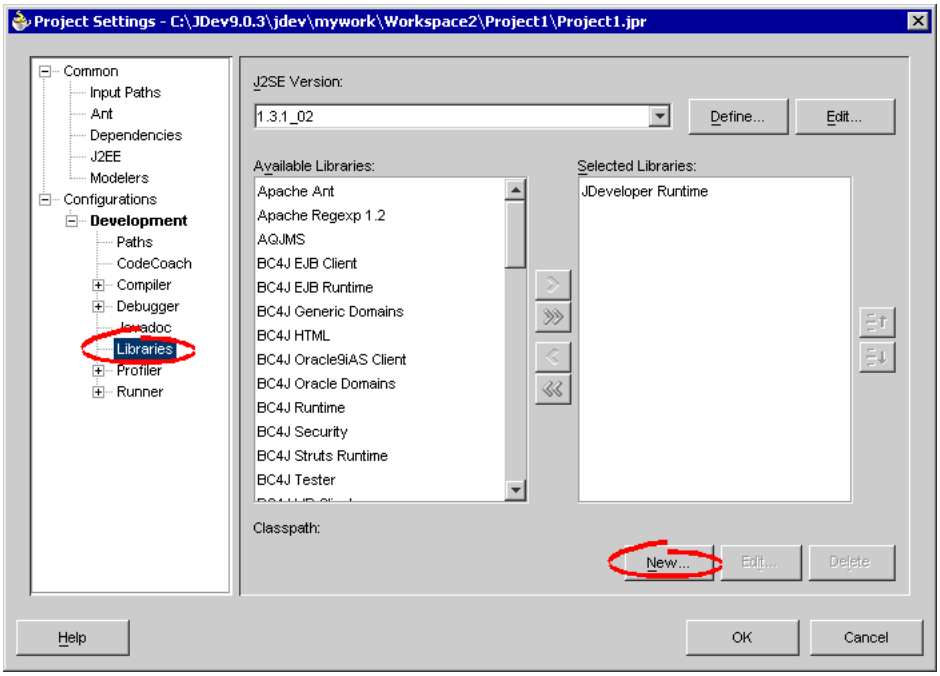
Using TopLink Mappings

Starting with Oracle JDeveloper 10*g*, the standard JDeveloper installation includes an embedded TopLink editor. Refer to the JDeveloper documentation for complete information.

To use TopLink with JDeveloper 9.0.4 (and earlier), use the following procedure to add the TopLink JAR files to your JDeveloper projects:

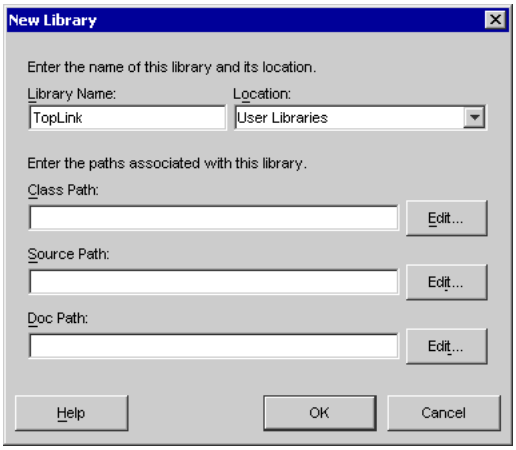
1. Select a JDeveloper project in the **System Navigator** window.
2. Choose **Project** > **Project Settings**. The **Project Settings** window appears.
3. Choose **Configurations** > **Development** > **Libraries**. A list of predefined and user-defined libraries appears.

Figure 6–1 List of Available Libraries



4. Click **New** to create a new library that will contain the TopLink `.jar` files. The New Library dialog box appears.
5. Enter a name for the new library–for example, **TopLink**. Ensure that the default choice for Location remains as **User Libraries**.

Figure 6–2 Creating a New Library Dialog Box



6. To edit the **Class Path** and add the TopLink `.jar` files, click **Edit**.

   Add the following to the beginning of your **Class Path**:

   ```
   <ORACLE_HOME>\toplink\jlib\toplink.jar
   <ORACLE_HOME>\toplink\jlib\antlr.jar
   <ORACLE_HOME>\lib\xmlparserv2.jar
   <ORACLE_HOME>\lib\xml.jar
   ```

7. Click **OK**. On the **Project Settings** window click **OK**.

Using an Existing User-Defined TopLink Library

After a user library is created, it can be referenced again by any other project. Revisit the **Libraries** window of the Project Settings, and add the TopLink Library to any project with which you want to use TopLink.

Using TopLink Sessions

When using TopLink 10*g* (10.1.3.1.0) with Oracle JDeveloper, you should be aware that you cannot configure all `sessions.xml` options with the JDeveloper mapping editor.

To configure supported options (prior to 10*g* (10.1.3.1.0)), use the JDeveloper mapping editor. For details, refer to the JDeveloper online help at:

`http://www.oracle.com/technology/documentation/9i_jdev.html`

To configure the new TopLink 10*g* (10.1.3.1.0) options, including, cache coordination options (see Chapter 88, "Configuring a Coordinated Cache"), historical sessions (see Chapter 78, "Configuring Historical Sessions"), and connection policy for server sessions (see "Configuring Connection Policy" on page 74-19) use one of the following methods:

- Use a `SessionEventListener.preLogin(SessionEvent)` method
- When accessing the session from the session manager, request it with the `loggedIn` option set to `false`. The returned session can then be customized and you can invoke login.

Configuring TopLink Workbench With Source Control Management Software

You can use TopLink Workbench with a source control management (SCM) system to facilitate enterprise-level team development (see "Using a Source Control Management System" on page 6-3). If you have a small development team, you can manage the changes from within XML files (see "Sharing Project Objects" on page 6-6).

When using a TopLink Workbench project in a team environment, you must synchronize your changes with other developers. See "Merging Files" on page 6-4 for more information.

Using a Source Control Management System

If you use an enterprise, file-based source control management system to manage your Java source files, you can use the same system with your TopLink Workbench project files. These project files are maintained by TopLink Workbench and written out in XML file format.

The *check in* and *check out* mechanism for the source control system defines how to manage the source (the XML source and TopLink Workbench project file) in a multiuser environment.

Checking Out and Checking In TopLink Workbench Project Files

Although your actual development process will vary depending on your SCM tool, a typical process involves the following steps:

1. Determine (based on your SCM system) which files to retrieve from the source management system.
2. Edit the project using TopLink Workbench.

3. Save the edited project. If TopLink Workbench displays the Read-Only Files dialog box, make a note of these files, they must be unlocked and possibly merged. See "Working With Locked Files" on page 6-6 for more information.
4. Merge the required project files. See "Merging Files" on page 6-4 for details.
5. Check in the modified files, then retrieve from the repository any files that have been added or modified for this TopLink Workbench project.

Merging Files

The most difficult aspect of application development is merging changes from two (or more) development team members that have simultaneously edited the same file. If one developer checks in his or her changes, a *merge* condition exists. Use a file comparison tool to determine the merged aspects of the project. The files to edit will vary, depending on the type of merge:

- Merging Project Files
- Merging Table, Descriptor, and Class Files

Example 6–1 and Example 6–2 demonstrate a *merge out* merging technique.

Merging Project Files

Project files contain references to the objects in the project. Generally, your project `<projectName>.mwp` contains the following elements:

- Database information: `<database>`
  - Database tables: `<tables>`
- Descriptors: `<descriptors>`
- Repository: `<repository>`
  - Classes: `<classpath-entries>`

Changes in these parts of the `.mwp` file are usually caused by adding, deleting, or renaming project elements.

To merge project files, you will generally need to merge a project file if another developer has added or removed a descriptor, table, or class, and checked in the project while you were adding or removing descriptors, tables, or classes from the same project. To merge the project's `.mwp` file, use this procedure:

1. Perform a file comparison between the `<projectName>.mwp` file in the repository and your local copy. The file comparison shows the addition or removal of a project element inside the owner (that is, `<database>`, `<descriptors>`, or `<repository>`).
2. Insert the XML script to, or delete from your local `<projectName>.mwp` file (inside the *corresponding owner element*). This brings your local code up-to-date to the current code in the code repository.
3. Retrieve any updated files, as indicated by your source control system.

Your local source now matches the repository.

Example 6–1 Merging Projects

Another developer has added and checked in a new **Employee** class descriptor to the `com.demo` package while you were working with the same TopLink Workbench project. To merge your work with the newly changed project, follow these steps:

1. Perform a file comparison on the <*projectName*>.mwp file to determine the differences between your local file and the file in the repository. Your SCM system may show the file in *merge* status.

   The file comparison shows the addition of the <package-descriptor> tag and a <name> element inside that tag:

   ```
   <package-descriptor>
       <name>com.demo.Employee.ClassDescriptor</name>
   </package-descriptor>
   ```

2. Insert this XML into your <*projectName*>.mwp file (inside the <descriptors> element) to bring it up-to-date with the current files in the source repository.
3. Retrieve any new or updated files from your source control system. This includes the newly added **Employee** class descriptor.
4. Check in files that you have modified.

Merging Table, Descriptor, and Class Files

Developers who concurrently modify the same existing table, descriptor, or class file will create a merge condition for the following files:

- Table: <*tableName*>.xml (one for each table)
- Descriptor: <*descriptorName.type*>.xml (one for each descriptor)
- Class: <*className*>.xml (one for each class)

TopLink Workbench changes these files when saving a project if you have changed any of the contents within them (such as adding a mapping to a descriptor, adding an attribute to a class, or a changing a field reference in a table).

If another developer has changed an attribute in a table, descriptor, or class, while you were changing a different mapping on that same descriptor, you will need to merge your project. To merge your project, use this procedure:

1. Perform a file comparison on the specific .xml files in merge status (that is, table, descriptor, or class). The file comparison shows the addition or removal of an XML element.
2. Insert the XML script to, or remove from your local .xml file to bring it up-to-date with the current files in the source repository.

Example 6–2 Merging Files

Another developer has added and checked in the **firstName** mapping to the **Employee** class descriptor while you were changing a different mapping on that same descriptor. To merge your work with the newly changed project, use this procedure:

1. Perform a file comparison on the com.demo.Employee.ClassDescriptor.xml file located in <*projectRoot*>/Descriptor/ directory that is in merge status.

   The file comparison shows the addition of the <mapping> tag and the elements inside that tag:

   ```
   <mapping>
       <uses-method-accessing>false</uses-method-accessing>
       <inherited>false</inherited>
       <read-only>false</read-only>
       <instance-variable-name>firstName</instance-variable-name>
       <default-field-names>
   ```

```
            <default-field-name>direct field=</default-field-name>
        </default-field-names>
        <field-handle>
            <field-handle>
                <table>EMPLOYEE</table>
                <field-name>F_NAME</field-name>
            </field-handle>
        </field-handle>
    <mapping-class>MWDirectToFieldMapping </mapping-class>
    </mapping>
```

2. Insert this XML block into your local `com.demo.Employee.ClassDescriptor.xml` file (inside the existing `<mapping>` element) to bring it up to date to the current files in the source repository.
3. Retrieve any new files noted as missing by your source control system. This includes any tables or descriptors that may be referenced by the new mapping.
4. Check in files that you have modified.

Sharing Project Objects

You can also share project objects by copying the table or descriptor files into the appropriate directories in the target project.

After copying the files, insert a reference to the table, descriptor, or class in the appropriate place in the `<projectName>.mwp` file. All references contained within the project file must refer to an existing object in the project.

Managing the ejb-jar.xml File

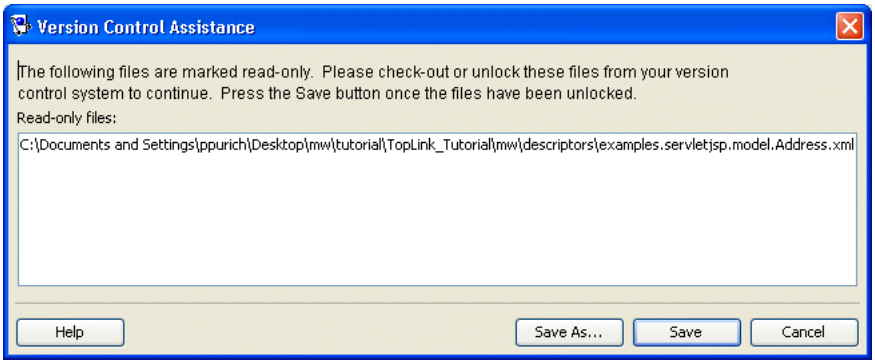
When working in a team environment, manage the `ejb-jar.xml` file similarly to the `.xml` project files. TopLink Workbench edits and updates the `ejb-jar.xml` file, if necessary, when working with an EJB project.

If you use a version control system, perform the same check in and check out procedures. For merge conditions, use a file comparison tool to determine which elements have been added or removed. Modify the file as necessary and check in the file to exercise version control on your work.

Working With Locked Files

When working in a team environment, your source control system may lock files when you retrieve them from the repository. If TopLink Workbench attempts to save a locked file, the Version Control Assistance dialog box appears, showing the locked files.

Figure 6–3 Version Control Assistance Dialog Box



Select one of the following methods to save your project:

- Use your source control system to unlock the files, and then click **Save**.
- Click **Save As** to save the project to a new location.

See "Saving Projects" on page 18-11 for more information.

Part III

Deploying a TopLink Application

This part describes how to package and deploy a TopLink application to an application server. It contains the following chapters.

- Chapter 7, "Integrating TopLink With an Application Server"

  This chapter contains information on software requirements for integrating TopLink with your specific application server.

- Chapter 8, "Creating TopLink Files for Deployment"

  This chapter describes how to create the necessary TopLink files for deployment to your application server.

- Chapter 9, "Packaging a TopLink Application"

  This chapter explains how to package the deployment files.

- Chapter 10, "Deploying a TopLink Application"

  This chapter provides procedures for deploying different types of TopLink applications in a variety of environments.

7

Integrating TopLink With an Application Server

This chapter describes how to configure Oracle TopLink for use with J2EE containers and application servers. It includes sections on the following:

- Application Server Support
- Application Server Integration Concepts

For more information, see the following:

- "Creating TopLink Files for Deployment" on page 8-1
- "Packaging a TopLink Application" on page 9-1
- "Deploying a TopLink Application" on page 10-1

Application Server Support

TopLink can be used with any J2EE application server.

Table 7–1 lists the application servers for which TopLink provides special EJB and CMP integration.

Table 7–1 TopLink Integration Support by Application Server Type

Application Server Type	TopLink Integration for Application Server Version	TopLink Integration for EJB Version
"Oracle Containers for J2EE (OC4J)" on page 7-5	■ 10.1.3 ■ 10.1.2 ■ 9.0.4 ■ 9.0.3	■ 2.*n*, 3.0[1] ■ 2.*n* ■ 2.*n* ■ 2.*n*
"BEA WebLogic Server" on page 7-14	■ 9.0 ■ 8.1 ■ 7.0 (Service Pack 2) ■ 6.1 (Service Pack 4)	■ 2.*n* ■ 2.*n* ■ 2.*n* ■ 2.*n*
"IBM WebSphere Application Server" on page 7-21	■ 6.0 ■ 5.1 ■ 5.0 ■ 4.0	■ 2.*n* ■ 2.*n* ■ 2.*n* ■ 2.*n*

[1] When using OC4J and Java 1.5, TopLink supports a subset of the persistent features anticipated in the final EJB 3.0 specification. EJB 3.0 feature support is subject to change and dependent upon the contents of the final specification. For more information on EJB 3.0 support, see Oracle Containers for J2EE Enterprise JavaBeans Developer's Guide.

Application Server Integration Concepts

This section describes concepts unique to TopLink application server integration, including the following:

- Software Requirements
- XML Parser Platform Configuration
- Security Permissions
- Persistence Manager Migration
- Clustering

Software Requirements

To run a TopLink application within a J2EE container, your system must meet the following software requirements:

- An application server or J2EE container (see Table 7–1)
- XML parser (see "XML Parser Platform Configuration" on page 7-2)
- 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 (WSAD)
  - Sun Java Development Kit (JDK) 1.3.1 or later. Oracle recommends using 1.4.2 (or later).
- Any other Java environment that is compatible with the Sun JDK 1.3.1 or later
- A command-line JVM executable (such as `java.exe` or `jre.exe`)

XML Parser Platform Configuration

The TopLink run-time environment uses an XML parser to do the following:

- Read and write XML configuration files (see "project.xml File" on page 8-2 and "sessions.xml File" on page 8-4)
- Read and write TopLink Workbench project files (see "Understanding TopLink Workbench" on page 4-1)
- Perform object-to-XML transformations in EIS projects using XML records (see Chapter 53, "Understanding EIS Mappings"
- Perform object-to-XML transformations in XML projects (see Chapter 62, "Understanding XML Mappings"

Application servers use an XML parser to read deployment files such as `ejb-jar.xml` and `<J2EE container>-ejb-jar.xml` (see "Creating TopLink Files for Deployment" on page 8-1).

To avoid XML parser conflicts, you must configure your TopLink application to use the same XML parser as that used by the application server on which you deploy your application.

Internally, TopLink accesses its XML parser using an instance of `oracle.toplink.platform.xml.XMLPlatform` class.

You can configure TopLink to use any XML parser for which an `XMLPlatform` class exists (see "Configuring XML Parser Platform" on page 7-3).

You can also create your own `XMLPlatform` to provide access to an XML parser not currently supported by TopLink (see "Creating an XML Parser Platform" on page 7-3).

Configuring XML Parser Platform

TopLink provides the `XMLPlatform` instances shown in Table 7–2.

Table 7–2 Supported XML Platforms

XMLPlatform...	Provides Access too...	Use With...
`oracle.toplink.platform.xml.xdk.XDKPlatform`[1]	`XDKParser`: this class provides access to the Oracle XML Developer's Kit (XDK) XML parser (see `http://www.oracle.com/technology/tech/xml/xdkhome.html`).	Oracle Containers for J2EE (OC4J)
`oracle.toplink.platform.xml.jaxp.JAXPPlatform`	`JAXPParser`: this class provides access to the Java SDK XML parser in the `javax.xml.parsers` package (see `http://java.sun.com/j2ee/1.4/docs/tutorial/doc/JAXPIntro2.html`).	BEA WebLogic Server IBM WebSphere Application Server

[1] Default:

> **Note:** To use an XML parser not listed in Table 7–2, create your own XMLPlatform (see "Creating an XML Parser Platform" on page 7-3).

To configure your TopLink application to use a particular instance of the `XMLPlatform` class, set system property `toplink.xml.platform` to the fully qualified name of your `XMLPlatform` class as Example 7–1 shows.

Example 7–1 Configuring XML Platform

```
toplink.xml.platform=oracle.toplink.platform.xml.jaxp.JAXPPlatform
```

Creating an XML Parser Platform

Using the `oracle.toplink.platform.xml` classes included in the public source files shipped with TopLink (see "Using Public Source" on page 12-2), you can create your own instance of the `oracle.toplink.platform.xml.XMLPlatform` class to specify an XML parser not listed in Table 7–2.

After creating your `XMLPlatform`, configure TopLink to use it (see "Configuring XML Parser Platform" on page 7-3).

XML Parser Limitations

Crimson (`http://xml.apache.org/crimson/`) is the XML parser supplied in the Java 2 Platform, Standard Edition (J2SE) and in some JAXP reference implementations.

If you use Crimson with the JAXP API to parse XML files whose system identifier is not a fully qualified URL, then XML parsing will fail with a *not valid URL* exception.

Other XML parsers defer validation of the system identifier URL until it is specifically referenced.

If you are experiencing this problem, consider one of the following alternatives:

- Ensure that your XML files use a fully qualified system identifier URL.
- Use another XML parser (such as the OracleAS XML Parser for Java v2).

Security Permissions

By default, when you run a TopLink-enabled application in a JVM configured with a nondefault `java.lang.SecurityManager`, the TopLink run-time environment executes certain internal functions by executing a `PrivilegedAction` with `java.security.AccessController` method `doPrivileged`. This ensures that you do not need to grant many permissions to TopLink for it to perform its most common operations. You need only grant certain permissions depending on the types of optional TopLink features you use.

For more information, see "Understanding Security Permissions" on page 7-22.

If you run a TopLink-enabled application in a JVM without a nondefault `SecurityManager`, you do not need to set any permissions.

Persistence Manager Migration

From the perspective of an application server, TopLink is a persistence manager. You can configure an application server to use TopLink as the default persistence manager.

You can only use one persistence manager for all the entity beans with container-managed persistence in a JAR file.

TopLink provides automated support for migrating an existing J2EE application to use TopLink as the persistence manager. For more information, see the following:

- "Migrating OC4J Orion Persistence to OC4J TopLink Persistence" on page 7-5
- "Migrating BEA WebLogic Persistence to OC4J TopLink Persistence" on page 7-16

Clustering

Most application servers include a clustering service that you can use with your TopLink application. To use TopLink with an application server cluster, use this procedure:

1. Install the `toplink.jar` file (and include it in the classpath) on each application server in the cluster to which you deploy TopLink applications.
2. Configure TopLink cache consistency options appropriate for your application.

   For more information, see Chapter 87, "Understanding the Cache".

   If you are deploying a CMP application, see also "Configuring cache-synchronization Properties" on page 8-11.
3. Configure clustering on each application server.

   For more information, see your application server documentation.

Oracle Containers for J2EE (OC4J)

To integrate a TopLink application with OC4J, you must consider the following:

- CMP Integration
- Migrating OC4J Orion Persistence to OC4J TopLink Persistence
- JTA Integration

In addition to configuring these OC4J specific options, you must also consider the general application server integration issues in "Application Server Integration Concepts" on page 7-2.

CMP Integration

To enable TopLink CMP integration in OC4J, use the following procedure (this procedure assumes you have already installed TopLink):

1. If necessary, migrate your CMP application using the TopLink migration tool (see "Migrating OC4J Orion Persistence to OC4J TopLink Persistence" on page 7-5).
2. Evaluate your choice of `UnitOfWork` change policy (see "Unit of Work and Change Policy" on page 97-6).

   Note: If you are using EJB 3.0, you can let TopLink configure your persistent classes with the most efficient change policy (see "Attribute Change Tracking Policy" on page 97-8) at class-loading time using byte code weaving.

3. Ensure that all necessary deployment descriptor files are in place (see "Creating TopLink Files for Deployment" on page 8-1 and "Packaging a TopLink Application" on page 9-1).

   Note: If you are using EJB 3.0, you can use annotations to specify most of what you formerly specified in deployment descriptors. Use deployment descriptors to override annotations or specify options not supported by annotations. For more information on what annotations are currently supported, see *Oracle Containers for J2EE Enterprise JavaBeans Developer's Guide*.

4. Optionally, consider the EJB customization options that TopLink provides (see "Configuring Miscellaneous EJB Options" on page 7-25).

Migrating OC4J Orion Persistence to OC4J TopLink Persistence

In 10*g* (10.1.3.1.0), OC4J is shipped configured to use TopLink as its default persistence manager.

If you upgrade your OC4J to this release, you must migrate persistence configuration from your original `orion-ejb-jar.xml` file to the `toplink-ejb-jar.xml` file.

In this release, Oracle provides a TopLink migration tool that you can use to automate this migration for Release 2 (9.0.4) or later OC4J installations.

After using the TopLink migration tool, you may need to make some additional changes as described in "Post-Migration Changes" on page 12.

If you encounter problems during migration, see "Troubleshooting Your Migration" on page 7-13.

This section explains how to use the TopLink migration tool, including:

- Overview
- Using the TopLink Migration Tool From TopLink Workbench
- Using the TopLink Migration Tool From the Command Line
- Post-Migration Changes
- Troubleshooting Your Migration

Overview

Before using the TopLink migration tool, review this section to understand how the TopLink migration tool works and to determine what OC4J persistence manager metadata is, and is not, migrated.

Input and Output

The TopLink migration tool takes the following files as input:

- `ejb-jar.xml`
- `orion-ejb-jar.xml`

It migrates as much OC4J-specific persistence configuration as possible to a new `toplink-ejb-jar.xml` file and creates the following new files in a target directory you specify:

- `orion-ejb-jar.xml`
- `toplink-ejb-jar.xml`
- TopLink Workbench project file `TLCmpProject.mwp`

The `ejb-jar.xml` and `orion-ejb-jar.xml` files may be in an EAR, JAR, or just standalone XML files. If you migrate from standalone XML files (rather than an EAR or JAR file), ensure that the domain classes are accessible and included in your classpath.

The TopLink migration tool creates a new `orion-ejb-jar.xml` and `toplink-ejb-jar.xml` file to the target directory you specify in the same format as it reads the original files. For example, if you specify an EAR file as input, then the TopLink migration tool stages and creates a new EAR file that contains both the new `orion-ejb-jar.xml` and the new `toplink-ejb-jar.xml` file, but is otherwise identical to the original.

The TopLink Workbench project file is always created as a separate file.

> **Note:** Oracle recommends that you make a backup copy of your `orion-ejb-jar.xml` file before using the TopLink migration tool.

Migration

As it operates, the TopLink migration tool logs all errors and diagnostic output to a log file named `oc4j_migration.log` in the output directory. If you use the TopLink

migration tool from TopLink Workbench, see also the TopLink Workbench log file `oracle.toplink.workbench.log` located in your user home directory (for example, `C:\Documents and Settings\<user-name>`).

The TopLink migration tool processes descriptor, mapping, and query information from the input files as follows:

- It builds a TopLink descriptor object for each entity bean and migrates native persistence metadata like mapped tables, primary keys, and mappings for CMP and CMR fields.
- It builds a TopLink mapping object for every CMP and CMR field of an entity bean and migrates native persistence metadata like foreign key references.
- It builds a TopLink query object for each `finder` or `ejbSelect` of an entity bean and migrates persistence metadata like customized query statements.

Table 7–3 lists OC4J `<entity-deployment>` attributes and subelements from the `orion-ejb-jar.xml` file and for each, indicates whether or not the TopLink migration tool:

- Retains it in the new `orion-ejb-jar.xml` file
- Migrates it to the new `toplink-ejb-jar.xml` file

In Table 7–3, elements are identified with angle brackets. Note that in some cases an attribute is migrated when set to one value, but discarded if set to another value (for example, `exclusive-write-access`).

Table 7–3 OC4J orion-ejb-jar.xml Feature Migration

orion-ejb-jar.xml Feature	Retained in New orion-ejb-jar.xml	Migrated to New toplink-ejb-jar.xml
`<entity-deployment>`		
`clustering-schema`	✔	
`copy-by-value`	✔	
`data-source`	✔	
`location`	✔	
`max-instances`	✔	
`min-instances`	✔	
`max-tx-retries`	✔	
`disable-wrapper-cache`	✔	
`name`	✔	
`pool-cache-timeout`	✔	
`wrapper`	✔	
`local-wrapper`	✔	
`call-timeout`		✔
`exclusive-write-access`		
`true`		✔
`false`		
`do-select-before-insert`		
`true`		
`false`		
`isolation`		✔
`locking-mode`		

Table 7–3 (Cont.) OC4J orion-ejb-jar.xml Feature Migration

orion-ejb-jar.xml Feature	Retained in New orion-ejb-jar.xml	Migrated to New toplink-ejb-jar.xml
pessimistic		✔
optimistic		
read-only		✔
old_pessimistic		
update-changed-fields-only		
true		✔
false		
table		✔
force-update		
true[1]		
false		✔
data-synchronization-option		
ejbCreate		
ejbPostCreate		
batch-size		
Any value greater than 1		
<ior-security-config>	✔	
<env-entry-mapping>	✔	
<resource-ref-mapping>	✔	
<resource-env-ref-mapping>	✔	
<primkey-mapping>		✔
<cmp-field-mapping>		✔
one-to-one-join		
inner		✔
outer[2]		
shared		✔
<finder-method>		✔
<persistence-type>[3]		✔

[1] You can enable `force-update` after migration. For more information, see "Configuring a Descriptor With EJB Information" on page 25-45.

[2] TopLink supports both `outer` and `inner` joins at run time.You can manually configure EJB descriptors with these options. For more information, see "Join Reading and Object-Level Read Queries" on page 93-12.

[3] The `persistence-type` attribute's table column size, if present, is discarded. For more information, see "Recovering persistence-type Table Column Size" on page 7-12.

Table 7–4 lists OC4J features and their TopLink equivalents configured by the TopLink migration tool.

Table 7–4 OC4J and TopLink Feature Comparison

Feature	orion-ejb-jar.xml	toplink-ejb-jar.xml
CMP field mapping	Direct	Direct-to-field
	Serialized object	Serialized object

Table 7–4 (Cont.) OC4J and TopLink Feature Comparison

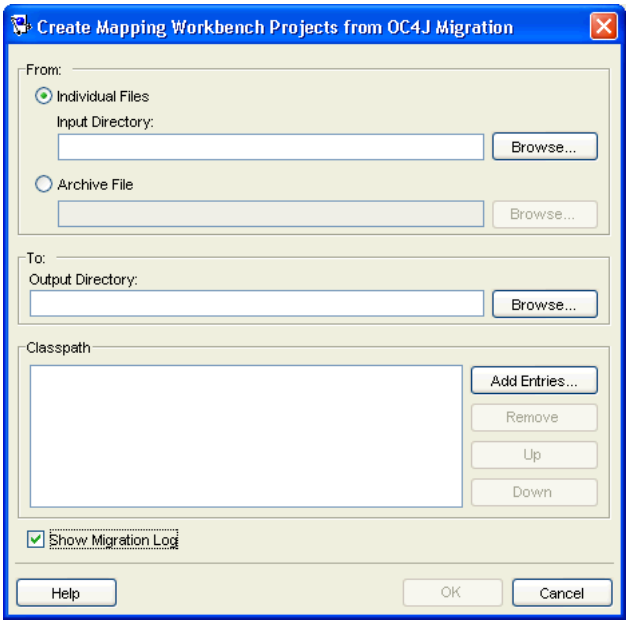
Feature	orion-ejb-jar.xml	toplink-ejb-jar.xml
CMR field mapping	One-to-one One-to-many Many-to-many	One-to-one One-to-many Many-to-many
Partial query	Full SQL statement	SQL Call
Finder	Oracle specific syntax	SQL Call or EJB QL
Lazy loading (fetch group)	Lazy loading of primary key and CMP fields	Not supported Alternatively, you can manually configure the TopLink equivalent, if appropriate (see "Fetch Groups" on page 23-5).
SQL statement caching	Cache static SQL	Not supported at the bean level. TopLink supports parameterized SQL and statement caching at the session and query level (see Chapter 93, "Understanding TopLink Queries").
Locking	Optimistic: database-level Pessimistic: bean instance-level	Optimistic: object-level Pessimistic: query lock at database-level
Read-only	Attempt to change throws `Exception`	Attempt to change throws `Exception`
Validity timeout	Read-only bean validity timeout before reloaded.	Cache timeout
Isolation level	Committed Serializable	Committed Serializable Not Committed Not Repeatable
Delay update until commit	Supported	Supported (see "Configuring a Descriptor With EJB Information" on page 25-45).
Exclusive write access on bean	Default value is `false`	Assume `true`
Insert without existence check	Supported	Supported
Update changed fields only	Supported	Supported (see "Attribute Change Tracking Policy" on page 97-8).
Force update	Invoke bean life cycle `ejbStore` method even though persistent fields have not changed	Supported

Using the TopLink Migration Tool From TopLink Workbench

To use the TopLink migration tool and create a new, mapped TopLink Workbench project from an OC4J application, use this procedure:

1. From TopLink Workbench, select **File > Migrate > From OC4J 9.0.x**.

Figure 7–1 Create Project from OC4J Migration Dialog Box



2. Continue with "Post-Migration Changes" on page 7-12.

Use the following information to enter data in each field of the Create Project from OC4J dialog box:

Field	Description
From	Use these fields to specify the location of the existing OC4J files. These files may be included as part of a JAR, EAR, or individual files.
Individual Files	Select to convert from individual `ejb-jar.xml` and `orion-ejb-jar.xml` files in the **Input Directory**. Click **Browse** and select the directory location that contains the XML files to convert from.
Archive File	Select to use a specific archive file. Click **Browse** and select the archive file to convert from..
To	Use these fields to specify the location to which migrated files are written.
Output Directory	Click **Browse** and select a directory location in which to create the new XML files and TopLink Workbench project.
Classpath	If you are migrating from individual files, ensure that the domain classes are accessible and included in your classpath.
Show Migration Log	Select to have migration log output displayed in a separate window.

Using the TopLink Migration Tool From the Command Line

To use the TopLink migration tool from the command line, you must perform the following steps:

1. Ensure that the following is in your classpath:
   - `<TOPLINK_HOME>/jlib/antlr.jar`
   - `<TOPLINK_HOME>/jlib/ejb.jar`

- *<TOPLINK_HOME>*/jlib/toplink.jar
- *<TOPLINK_HOME>*/jlib/cmpmigrator.jar
- *<TOPLINK_HOME>*/jlib/toplinkmw.jar
- *<TOPLINK_HOME>*/jlib/tlmwcore.jar
- *<TOPLINK_HOME>*/config
- *<ORACLE_HOME>*/lib/xmlparserv2.jar

2. If you intend to migrate from plain XML files (rather than an EAR or JAR file), ensure that the domain classes are accessible and included in your classpath.
3. Make a backup copy of your original XML files.
4. Execute the TopLink migration tool as Example 7–2 illustrates using the appropriate arguments listed in Table 7–5.

   The usage information for the TopLink migration tool is:

   ```
   java -Dtoplink.ejbjar.schemavalidation=<true|false>
   -Dtoplink.migrationtool.generateWorkbenchProject=<true|false>
   -Dhttp.proxyHost=<proxyHost> -Dhttp.proxyPort=<proxyPort>
   oracle.toplink.tools.migration.TopLinkCMPMigrator -s<nativePM> -i<inputDir>
   -a<ear>|<jar> -x -o<outputDir> -v
   ```

   To identify the input files, you must specify one of `-a` or `-x`.

   For troubleshooting information, see "Troubleshooting Your Migration" on page 13.

Example 7–2 Using the TopLink Migration Tool from the Command Line

```
java -Dhttp.proxyHost=www-proxy.us.oracle.com -Dhttp.proxyPort=80
oracle.toplink.tools.migration.TopLinkCMPMigrator -sOc4j-native -iC:/mywork/in
-aEmployee.ear -oC:/mywork/out -v
```

Table 7–5 TopLink Migration Tool Arguments

Argument	Description
`toplink.ejbjar.schemavalidation`	The system property used to turn on schema validation if `ejb-jar.xml` uses XML Schema (XSD) instead of DTD. The default value is `false`.
`toplink.migrationtool.generateWorkbenchProject`	The system property used enable generation of the TopLink Workbench project. The default value is `true`.
<proxyHost>	The address of your local HTTP proxy host
<proxyHost>	The port number on which your local HTTP proxy host receives HTTP requests.
`-s <source>`	The name of the native persistence manager from which you are migrating. For OC4J, use the name `Oc4j-native`.
`-i <input-directory>`	Fully qualified path to the input directory that contains both the OC4J `ejb-jar.xml` and `orion-ejb-jar.xml` files to migrate. Default: current working directory.
`-a <EAR-or-JAR>`	Fully qualified path to the archive file (either an EAR or JAR) that contains both the OC4J `ejb-jar.xml` and `orion-ejb-jar.xml` files to migrate.

Table 7–5 (Cont.) TopLink Migration Tool Arguments

Argument	Description
`-x`	Tells the TopLink migration tool that the OC4J files in the input directory to migrate from are plain XML files (not in an archive file). If you use this option, ensure that the domain classes are accessible and included in your classpath.
`-o <output-directory`	`<targetDir>` is the path to the directory into which the TopLink migration tool writes the new `orion-ejb-jar.xml`, `toplink-ejb-jar.xml`, and log files. The path may be absolute or relative to the current working directory. You must specify this argument value. Ensure that permissions are set on this directory to allow the TopLink migration tool to create files and subdirectories.
`-v`	Verbose mode. Tells the TopLink migration tool to log errors and diagnostic information to the console.

Post-Migration Changes

After you migrate the `orion-ejb-jar.xml` file persistence configuration to your `toplink-ejb-jar.xml` file, consider the following post-migration changes:

- Recovering persistence-type Table Column Size
- Updating the Unknown Primary Key Class Mapping Sequence Table
- Project Customization

Recovering persistence-type Table Column Size

In the `orion-ejb-jar.xml` file, you can specify this mapping, `cmp-field-mapping`, with a `persistence-type` attribute that provides both the type and column size as shown in Example 7–3.

Example 7–3 A cmp-field-mapping with persistence-type Specifying a Column Size

```
<cmp-field-mapping ejb-reference-home="MyOtherEntity" name="myField"
persistence-name="myField" persistence-type="VARCHAR2(30)">
```

The TopLink migration tool migrates the persistence type but not the column size because a TopLink project does not normally contain this size information.

To recover the `persistence-type` column size, do the following:

1. Perform the migration as described in "Using the TopLink Migration Tool From the Command Line" on page 10.
2. Launch the generated TopLink Workbench project file `TLCmpProject.mwp`.
3. Log in to your database (see "Logging In and Out of a Database" on page 4-21).

   TopLink Workbench retrieves all column sizes.

Updating the Unknown Primary Key Class Mapping Sequence Table

TopLink supports the use of an unknown primary key class (see "Unknown Primary Key Class Support" on page 7-25) and so the TopLink migration tool also supports this feature.

OC4J uses a native run-time key generator to generate unique keys for auto-id key fields. In contrast, TopLink uses a sequencing table.

If your OC4J persistence configuration includes the use of an unknown primary key class, then the TopLink migration tool will create a sequencing table for this purpose.

Before deploying your application after migration, you must do the following:

1. Determine the largest key value generated prior to migration.
2. Manually update the counter of the TopLink migration tool-generated sequence table to a number that must be one larger than the largest key value generated prior to migration.

Project Customization

You can customize the following components of your project:

- EJB 3.0 Persistence Manager Customization
- EJB 2.1 Persistence Manager Customization
- Session Event Listener

EJB 3.0 Persistence Manager Customization For an EJB 3.0 persistent application deployed to OC4J, you customize the TopLink persistence manager by creating a TopLink project XML file named `toplink-ejb-jar.xml` and a TopLink session XML file named `ejb3-toplink-sessions.xml` and packaging them in the `META-INF` directory of the EJB-JAR that contains your entities. For more information, see "Customizing the TopLink Persistence Manager in an EJB 3.0 Application" in the *Oracle Containers for J2EE Enterprise JavaBeans Developer's Guide*.

EJB 2.1 Persistence Manager Customization For an EJB 2.1 CMP application deployed to OC4J, you customize the TopLink persistence manager by configuring properties in the `orion-ejb-jar.xml` file. These properties are used to configure the TopLink session that the TopLink runtime uses internally for CMP projects. For more information, see "Configuring persistence-manager Entries" on page 8-10.

Session Event Listener After you applied the default settings to your project at deployment time, you may wish to customize the TopLink session by configuring the session event listener. The pre-login event that the session raises is particularly useful. It lets you define the custom (nondefault) specifics for the session just before the session initializes and acquires connections.

For more information, see the following:

- "Session Customization" on page 72-4
- "Managing Session Events With the Session Event Manager" on page 72-5
- "Configuring Session Event Listeners" on page 74-17

Troubleshooting Your Migration

This section describes solutions for problems you may encounter during migration, including the following:

- Log Messages
- Unexpected Relational Multiplicity

Log Messages

As it operates, the TopLink migration tool logs all errors and diagnostic output to a log file named `oc4j_migration.log` in the output directory. If you use the TopLink migration tool from TopLink Workbench, see also the TopLink Workbench log file

`oracle.toplink.workbench.log` located in your user home directory (for example, `C:\Documents and Settings\<user-name>`)

In addition to these warnings, the TopLink migration tool logs an error if it encounters a problem that prevents it from completing the migration. Table 7–6 lists these problems and suggests possible solutions.

Table 7–6 TopLink Migration Tool Error Messages

Error Message	Description
There is no `ejb-jar.xml` in the input file. You must provide the `ejb-jar.xml` in order for the migration process to work.	The `ejb-jar.xml` file is missing. The TopLink migration tool stops and copies the original input files into the target directory. Verify that the `ejb-jar.xml` file is present in the specified EAR, JAR, or as a plain XML file. Empty the target directory and execute the TopLink migration tool again.
There is not an `orion-ejb-jar.xml` with native persistent metadata defined, no migration needed.	The `orion-ejb-jar.xml` file is missing. The TopLink migration tool stops and copies the original input files into the target directory. Verify that the `orion-ejb-jar.xml` file is present in the specified EAR, JAR, or as a plain XML file. Empty the target directory and execute the TopLink migration tool again.
`toplink-ejb-jar.xml` is already defined in the archive, no migration needed.	A `toplink-ejb-jar.xml` file is already present in the target directory. The TopLink migration tool stops and copies the original input files into the target directory. Remove the `toplink-ejb-jar.xml` file from the target directory. Empty the target directory and execute the TopLink migration tool again.

Unexpected Relational Multiplicity

The TopLink migration tool retrieves relationship multiplicity from the `orion-ejb-jar.xml` file and not from the OC4J `ejb-jar.xml` file.

Thus, even though the OC4J `ejb-jar.xml` file defines a relationship to be one-to-many, if the `orion-ejb-jar.xml` file defines the same relationship as many-to-many, then the TopLink migration tool will migrate the relationship as many-to-many.

JTA Integration

For applications that require JTA integration, specify the external transaction controller when you configure the server platform in your session (see "Configuring the Server Platform" on page 74-14).

For more information, see "Integrating the Unit of Work With an External Transaction Service" on page 99-21.

BEA WebLogic Server

To integrate a TopLink application with BEA WebLogic Server, you must consider the following:

- Classpath
- CMP Integration
- Migrating BEA WebLogic Persistence to OC4J TopLink Persistence
- JTA Integration

- Security Manager

In addition to configuring these BEA WebLogic Server-specific options, you must also consider the general application server integration issues in "Application Server Integration Concepts" on page 7-2.

Classpath

To configure TopLink support for BEA WebLogic Server, do the following:

1. Add the following JAR files to the application server classpath:

   ```
   <ORACLE_HOME>\toplink\jlib\toplink.jar
   ```

2. Ensure that your TopLink application defines an XML parser platform (see "XML Parser Platform Configuration" on page 7-2).

CMP Integration

To enable TopLink CMP integration in BEA WebLogic Server, use the following procedure (assuming you have already installed TopLink):

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 the *<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 the `TopLink_CMP_Descriptor.xml` file.
   - Replace the WebLogic `persistence.install` file with the TopLink persistence.install file found in the *<ORACLE_HOME>*`\toplink\config` directory.

2. Add the following JAR files to the application server classpath:

   ```
   <ORACLE_HOME>\toplink\jlib\toplink.jar
   <ORACLE_HOME>\lib\xmlparserv2.jar
   ```

3. If necessary, migrate your CMP application using the TopLink migration tool ("Migrating BEA WebLogic Persistence to OC4J TopLink Persistence" on page 7-16).
4. Ensure that all necessary deployment descriptor files are in place (see "Creating TopLink Files for Deployment" on page 8-1 and "Packaging a TopLink Application" on page 9-1).
5. Optionally, consider the EJB customization options that TopLink provides ("Configuring Miscellaneous EJB Options" on page 7-25).
6. Start the container, and then start the TopLink application. Where supported, use a startup script to start the server. If you write your own startup script, ensure that the classpath includes the files listed in Step 2.

Migrating BEA WebLogic Persistence to OC4J TopLink Persistence

You can migrate a BEA WebLogic application that uses the default BEA WebLogic persistence manager to an Oracle Containers for J2EE (OC4J) application that uses TopLink as the persistence manager. In this release, Oracle provides a TopLink migration tool that you can use to automate this migration.

This section includes the following:

- Overview
- Using the TopLink Migration Tool From TopLink Workbench
- Using the TopLink Migration Tool From the Command Line

If you encounter problems during migration, see "Troubleshooting Your Migration" on page 7-13.

After using the TopLink migration tool, you may need to make some additional changes as described in "Post-Migration Changes" on page 12.

Overview

Before using the TopLink migration tool, review this section to understand how the TopLink migration tool works and to determine what BEA WebLogic persistence manager metadata is, and is not, migrated.

Input and Output

The TopLink migration tool takes the following files as input:

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

It migrates as much BEA WebLogic-specific persistence configuration as possible to a new `toplink-ejb-jar.xml` file and creates the following new files in a target directory you specify:

- `orion-ejb-jar.xml`
- `toplink-ejb-jar.xml`
- TopLink Workbench project file `TLCmpProject.mwp`

The input BEA WebLogic files may be in an EAR, JAR, or just standalone XML files. If you migrate from standalone XML files (rather than an EAR or JAR file), ensure that the domain classes are accessible and included in your classpath.

The TopLink migration tool creates a new `orion-ejb-jar.xml` and `toplink-ejb-jar.xml` file to the target directory you specify in the same format as it reads the original files. For example, if you specify an EAR file as input, then the TopLink migration tool stages and creates a new EAR file that contains both the new `orion-ejb-jar.xml` and the new `toplink-ejb-jar.xml` file, but is otherwise identical to the original.

The TopLink Workbench project file is always created as a separate file.

Note: Oracle recommends that you make a backup copy of your `weblogic-ejb-jar.xml`, `weblogic-cmp-rdbms-jar.xml`, and `toplink-ejb-jar.xml` files before using the TopLink migration tool.

For information on configuring the weblogic-ejb-jar.xml file, see "Configuring the weblogic-ejb-jar.xml File for BEA WebLogic Server" on page 8-13.

Migration

As it operates, the TopLink migration tool logs all errors and diagnostic output to a log file named `wls_migration.log` in the output directory. If you use the TopLink migration tool from TopLink Workbench, see also TopLink Workbench log file `oracle.toplink.workbench.log` located in your user home directory (for example, `C:\Documents and Settings\<user-name>`).

The TopLink migration tool processes descriptor, mapping, and query information from the input files. It performs the following:

- It builds a TopLink descriptor object for each entity bean and migrates native persistence metadata like mapped tables, primary keys, and mappings for CMP and CMR fields.
- It builds a TopLink mapping object for every CMP and CMR field of an entity bean and migrates native persistence metadata like foreign key references.
- It builds a TopLink query object for each `finder` or `ejbSelect` of an entity bean and migrates persistence metadata like customized query statements.

Table 7–7 lists BEA WebLogic features and their TopLink equivalents configured by the TopLink migration tool.

Table 7–7 BEA WebLogic and TopLink Feature Comparison

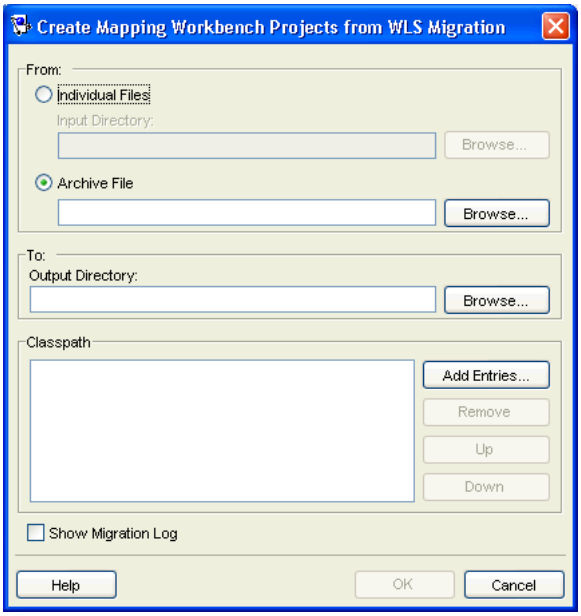
Feature	weblogic-cmp-rdbms-jar.xml	toplink-ejb-jar.xml
Direct mapping	Field mapping Table mapping	Direct-to-field
Relational mapping	WebLogic-RDBMS relation	One-to-one One-to-many Many-to-many
Multiple tables	Table mapping	Supported
Read-only	Read-only: concurrency strategy	Supported
Pessimistic locking	Enforce pessimistic concurrency on a per-bean basis	Supported
Large Objects (LOB) support	Map the current field to one of the following: 1. `CLOB` or `BLOB` in an Oracle database 2. `LongString` or `SybaseBinary` in a Sybase database	Supported for Oracle BLOB only
Optimistic locking	Optimistic: concurrency strategy Verify columns: optimistic concurrency strategy	Version Timestamp
Cascade delete	Database cascade delete	Privately owned
Sorting database dependency	Order database operations	unit of work
Lazy loading (fetch group)	Field group	Not supported. Alternatively, you can manually configure the TopLink equivalent, if appropriate (see "Fetch Groups" on page 23-5).

Using the TopLink Migration Tool From TopLink Workbench

To use the TopLink migration tool and create a new, mapped TopLink Workbench project from a BEA WebLogic application, use this procedure:

1. From TopLink Workbench, select **File > Migrate > From Weblogic Server**.

Figure 7–2 Create Projects from WebLogic Server Migration Dialog Box



2. Continue with "Post-Migration Changes" on page 7-12.

Use the following information to enter data in each field on the Create Project from WebLogic dialog box:

Field	Description
From	Use these fields to specify the location of the existing BEA WebLogic files. These files may be included as part of a JAR, EAR, or individual files.
Individual Files	Select to convert from individual `weblogic-ejb-jar.xml` and `weblogic-cmp-rdbms-jar.xml` files in the **Input Directory**. Click **Browse** and select the directory location that contains the XML files to convert from.
Archive File	Select to use a specific archive file. Click **Browse** and select the archive file to convert from..
To	Use this field to specify the location to which migrated files are written.
Output Directory	Click **Browse** and select a directory location in which to create the new XML files and TopLink Workbench project.
Classpath	If you are migrating from individual files, ensure that the domain classes are accessible and included in your classpath.
Show Migration Log	Select to have migration log output displayed in a separate window.

Using the TopLink Migration Tool From the Command Line

To use the TopLink migration tool from the command line, you must do the following:

1. Ensure that the following is in your classpath:
   - *<TOPLINK_HOME>*/jlib/antlr.jar
   - *<TOPLINK_HOME>*/jlib/ejb.jar
   - *<TOPLINK_HOME>*/jlib/toplink.jar
   - *<TOPLINK_HOME>*/jlib/cmpmigrator.jar
   - *<TOPLINK_HOME>*/jlib/toplinkmw.jar
   - *<TOPLINK_HOME>*/jlib/tlmwcore.jar
   - *<TOPLINK_HOME>*/config
   - *<ORACLE_HOME>*/lib/xmlparserv2.jar
2. If you intend to migrate from plain XML files (rather than an EAR or JAR file), ensure that the domain classes are accessible and included in your classpath.
3. Make a backup copy of your original XML files.
4. Execute the TopLink migration tool as Example 7–4 illustrates using the appropriate arguments listed in Table 7–8.

   The usage information for the TopLink migration tool is:

   ```
   java -Dtoplink.ejbjar.schemavalidation=<true|false>
   -Dtoplink.migrationtool.generateWorkbenchProject=<true|false>
   -Dhttp.proxyHost=<proxyHost> -Dhttp.proxyPort=<proxyPort>
   oracle.toplink.tools.migration.TopLinkCMPMigrator -s<nativePM> -i<inputDir>
   -a<ear>|<jar> -x -o<outputDir> -v
   ```

   To identify the input files, you must specify one of `-a` or `-x`.

   For troubleshooting information, see "Troubleshooting Your Migration" on page 13.

Example 7–4 Using the TopLink Migration Tool from the Command Line

```
java -Dhttp.proxyHost=www-proxy.us.oracle.com -Dhttp.proxyPort=80
oracle.toplink.tools.migration.TopLinkCMPMigrator -sWebLogic -iC:/mywork/in
-aEmployee.ear -oC:/mywork/out -v
```

Table 7–8 TopLink Migration Tool Arguments

Argument	Description
`toplink.ejbjar.schemavalidation`	The system property used to turn on schema validation if `ejb-jar.xml` uses XML Schema (XSD) instead of DTD. The default value is `false`.
`toplink.migrationtool.generateWorkbenchProject`	The system property used enable generation of the TopLink Workbench project. The default value is `true`.
<proxyHost>	The address of your local HTTP proxy host
<proxyHost>	The port number on which your local HTTP proxy host receives HTTP requests.
`-s <source>`	The name of the native persistence manager from which you are migrating. For BEA WebLogic, use the name `WebLogic`.
`-i <input-directory>`	Fully qualified path to the input directory that contains the BEA WebLogic `weblogic-ejb-jar.xml` and `weblogic-cmp-rdbms-jar.xml` files to migrate. Default: current working directory.

Table 7–8 (Cont.) TopLink Migration Tool Arguments

Argument	Description
`-a <EAR-or-JAR>`	Fully qualified path to the archive file (either an EAR or JAR) that contains both the BEA WebLogic `weblogic-ejb-jar.xml` and `weblogic-cmp-rdbms-jar.xml` files to migrate.
`-x`	Tells the TopLink migration tool that the BEA WebLogic files in the input directory to migrate from are plain XML files (not in an archive file). If you use this option, ensure that the domain classes are accessible and included in your classpath.
`-o <output-directory`	The path to the directory into which the TopLink migration tool writes the new `orion-ejb-jar.xml`, `toplink-ejb-jar.xml`, and log files. The path may be absolute or relative to the current working directory. You must specify this argument value. Ensure that permissions are set on this directory to allow the TopLink migration tool to create files and subdirectories.
`-v`	Verbose mode. Tells the TopLink migration tool to log errors and diagnostic information to the console.

JTA Integration

For applications that require JTA integration, specify the external transaction controller when you configure the server platform in your session (see "Configuring the Server Platform" on page 74-14).

For more information, see "Integrating the Unit of Work With an External Transaction Service" on page 99-21.

Security Manager

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 TopLink to use reflection.

The following example illustrates only the permissions that TopLink requires, but most `weblogic.policy` files contain more permissions than are shown in this example.

Example 7–5 A Subset of a "Grant" Section from a BEA WebLogic.policy File

```
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 TopLink to use reflection
    permission java.lang.reflect.ReflectPermission "suppressAccessChecks";
};
```

IBM WebSphere Application Server

To integrate a TopLink application with IBM WebSphere Application Server, you must consider the following:

- Classpath
- CMP Integration
- JTA Integration

In addition to configuring these IBM WebSphere application server-specific options, you must also consider the general application server integration issues in "Application Server Integration Concepts" on page 7-2.

Classpath

You configure the IBM WebSphere application server classpath differently depending on what version of this server you are using:

- Configuring Classpath for IBM WebSphere Application Server 4.0
- Configuring Classpath for IBM WebSphere Application Server 5.0 and Later

Configuring Classpath for IBM WebSphere Application Server 4.0

TopLink provides CMP support for IBM WebSphere application server 4.0. To configure the classpath for this version, do the following:

1. Add the following JAR files to the application server classpath directory (see Table 7–9):

   `<ORACLE_HOME>\toplink\jlib\toplink.jar`

2. Ensure that your TopLink application defines an XML parser platform (see "XML Parser Platform Configuration" on page 7-2).

Table 7–9 lists the default application classpath directories for IBM container components in IBM WebSphere application server 4.0.

Table 7–9 Classpath Directories for IBM WebSphere 4.0 Container Components

Container	Default Application Classpath
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 Classpath for IBM WebSphere Application Server 5.0 and Later

TopLink provides JTA and general integration support for IBM WebSphere application server 5.0 and later. To configure the classpath for this version, do the following:

1. Create a shared library that contains the following Toplink JAR files and associate the shared library with the application:

   `<ORACLE_HOME>\toplink\jlib\toplink.jar`

2. Ensure that your TopLink application defines an XML parser platform (see "XML Parser Platform Configuration" on page 7-2).

CMP Integration

To enable TopLink CMP integration in IBM WebSphere application server, use the following procedure (assuming you have already installed TopLink):

1. Ensure that all necessary deployment descriptor files are in place (see "Creating TopLink Files for Deployment" on page 8-1 and "Packaging a TopLink Application" on page 9-1).
2. Optionally, consider the EJB customization options that TopLink provides ("Configuring Miscellaneous EJB Options" on page 7-25).

JTA Integration

For applications that require JTA integration, specify the external transaction controller when you configure the server platform in your session (see "Configuring the Server Platform" on page 74-14).

For more information, see "Integrating the Unit of Work With an External Transaction Service" on page 99-21.

Clustering on IBM WebSphere Application Server

For more information on integrating a TopLink application with an application server cluster, see "Clustering" on page 7-4.

Understanding Security Permissions

By default, when you run a TopLink-enabled application in a JVM configured with a nondefault `java.lang.SecurityManager`, the TopLink run time executes certain internal functions by executing a `PrivilegedAction` with `java.security.AccessController` method `doPrivileged`. This ensures that you do not need to grant many permissions to TopLink for it to perform its most common operations. You need only grant certain permissions depending on the types of optional TopLink features you use (see "Permissions Required by TopLink Features" on page 7-23).

While using `doPrivileged` method provides enhanced security, it will severely impact overall performance. Alternatively, you can configure TopLink to disable the use of `doPrivileged` method even when a nondefault `SecurityManager` is present (see "Disabling doPrivileged Operation" on page 7-25). In this case, you must grant TopLink all required permissions (see "Permissions Required by TopLink Features" on page 7-23 and "Permissions Required When doPrivileged is Disabled" on page 7-24).

> **Note:** While enabling the use of `doPriviledged` method enhances TopLink application security, it does not guarantee that secure code cannot be called by application code in ways that the system did not intend. You must consider the use of `doPriviledged` method within the context of your overall application security strategy. For more information, see `http://java.sun.com/security/index.jsp`.

If you run a TopLink-enabled application in a JVM without a nondefault `SecurityManager`, you do not need to grant any permissions.

Permissions Required by TopLink Features

When you run a TopLink-enabled application in a JVM configured with a nondefault `java.lang.SecurityManager` and `doPrivileged` operation is enabled, you may need to grant additional permissions if your application requires any of the following:

- System Properties
- Loading project.xml or sessions.xml Files
- Cache Coordination
- Accessing a Data Source by Port
- Logging With java.util.logging
- J2EE Application Deployment

System Properties

By default, a TopLink-enabled application requires access to the system properties granted in the default `<JAVA_HOME>/lib/security/java.policy` file. If your application requires access to other platform-specific, environment, or custom properties, then grant further `PropertyPermission` permissions as Example 7–6 shows.

Example 7–6 Permissions for System Properties

```
permission java.util.PropertyPermission "my.property", "read";
```

Loading project.xml or sessions.xml Files

Most TopLink-enabled applications read in `project.xml` and `sessions.xml` files directly. Grant permissions to the specific files or file locations as Example 7–7 shows. This example assumes that both `project.xml` and `sessions.xml` files are located in the same directory (given by application-specific system property `deployment.xml.home`). Alternatively, you can specify a separate `FilePermission` for each file.

Example 7–7 Permissions for Loading Deployment XML Files

```
permission java.io.FilePermission "${deployment.xml.home}/*.xml", "read";
```

For information on `FilePermission` settings for J2EE applications, see "J2EE Application Deployment" on page 7-24.

Cache Coordination

If your application uses cache coordination (see "Understanding Cache Coordination" on page 87-10), then grant `accept`, `connect`, `listen`, and `resolve` permissions to the specific sockets used by your coordinated cache as Example 7–8 shows. This example assumes that the coordinated cache multicast port (see "Configuring a Multicast Port" on page 88-5) is 1024.

Example 7–8 Permissions for Cache Coordination

```
permission java.net.SocketPermission "localhost:1024-", "accept, connect, listen, resolve";
```

Accessing a Data Source by Port

If your TopLink-enabled application accesses a data source using a socket, then grant `connect` and `resolve` permissions for that socket as Example 7–9 shows. This example assumes that the host name (or IP address) of the remote host that provides the data source (such as a relational database server host) is given by application-specific system property `remote.data.source.host` and that this host accepts data source connections on port 1025.

Example 7–9 Permissions for non-J2EE Data Source Connections

```
permission java.net.SocketPermission "${remote.data.source.host}:1025-", "connect, resolve";
```

For J2EE applications, data source socket permissions are usually handled by the application server.

Logging With java.util.logging

If you configure your TopLink-enabled application to use `java.util.logging` package (see "Configuring Logging" on page 74-4), then grant your application `control` permissions as Example 7–10 shows.

Example 7–10 Permissions for java.util.logging

```
permission java.util.logging.LoggingPermission "control"
```

J2EE Application Deployment

If you are deploying a TopLink-enabled J2EE application, you must grant permissions for:

- The `toplink.jar` file. For example:

```
grant codeBase "file:<TOPLINK_HOME>/jlib/toplink.jar" {
  permission java.security.AllPermission;
};
```

If you are using an XML platform, you must also grant the following permissions:

- The `toplink.xml.platform` system property. For Example:

```
permission java.util.PropertyPermission "toplink.xml.platform", "read"
```

Permissions Required When doPrivileged is Disabled

If you disable `doPrivileged` operation when you run a TopLink-enabled application in a JVM configured with a nondefault `java.lang.SecurityManager`, you must grant the following permissions:

- `java.lang.reflect.RelectPermission "suppressAccessChecks"`
- `java.lang.RuntimePermission "accessDeclaredMembers"`
- `java.lang.RuntimePermission "getClassLoader"`

You may also have to grant additional permissions depending on the TopLink features your application uses. For more information, see "Permissions Required by TopLink Features" on page 7-23.

Disabling doPrivileged Operation

To disable `doPrivileged` operation when you run a TopLink-enabled application in a JVM configured with a nondefault `java.lang.SecurityManager`, set system property `oracle.j2ee.toplink.security.usedoprivileged` to `false`. If you are using OC4J, set system property `oracle.j2ee.security.usedoprivileged` to `false`.

To enable `doPrivileged` operation, set these system properties to `true`.

Configuring Miscellaneous EJB Options

TopLink provides system properties that you can use to customize the following EJB options:

- Setter Parameter Type Checking
- Unknown Primary Key Class Support
- Single-Object Finder Return Type Checking

Setter Parameter Type Checking

To make TopLink verify that the parameters to one-to-one and one-to-many relationship setters are of the same type as the corresponding CMR field, set system property `toplink.cts.collection.checkParameters` to a value of `true` (not case sensitive). If the setters are not the same type, then TopLink throws a `java.lang.IllegalArgumentException`.

> **Note:** Setting this property to `true` will affect performance. Use this setting only if necessary.

If you set the property to `false` (the default value), TopLink does not make this verification. In this case, it is up to your application to make sure the parameters are of the correct type.

For more information, see the EJB 2.1 specification, section 10.3.6.

Unknown Primary Key Class Support

In special situations, you may choose not to specify the primary key class or the primary key fields for an entity bean with container-managed persistence. For example, if the entity bean does not have a natural primary key or you want the deployer to select the primary key fields at deployment time, you may choose to defer primary key type specification.

If this is the case, you must declare the type of the argument of the `findByPrimaryKey` method as `java.lang.Object` and you must also specify the primary key class (`prim-key-class`) in the deployment descriptor (`ejb-jar.xml`) as `java.lang.Object`.

TopLink now provides run-time support for such deferred primary key type specification.

For more information, see the EJB 2.1 specification, section 10.8.3.

Single-Object Finder Return Type Checking

By setting system property `toplink.cts.checkMultipleRows` to `true`, you can configure TopLink to throw a `javax.ejb.FinderException` if multiple beans are returned from a single-object finder method.

For more information, see the EJB 2.1 specification, section 10.5.6.1.

8

Creating TopLink Files for Deployment

This chapter includes TopLink information you need when creating deployment files for the following types of applications:

- Java Applications
- JavaServer Pages and Servlet Applications
- Session Bean Applications
- CMP Applications
- BMP Applications

For more information on packaging and deployment, see the following:

- "Understanding TopLink Deployment File Creation" on page 8-1
- "Integrating TopLink With an Application Server" on page 7-1
- "Packaging a TopLink Application" on page 9-1
- "Deploying a TopLink Application" on page 10-1

Understanding TopLink Deployment File Creation

Depending on the type of application you are deploying, you may need to create any of the following deployment files:

- project.xml File
- sessions.xml File
- ejb-jar.xml File
- <J2EE-Container>-ejb-jar.xml File
- toplink-ejb-jar.xml File

TopLink Workbench provides the ability to create deployment files from a TopLink Workbench project (see "Exporting Project Information" on page 18-13). After you build a project, you have two options to create the deployment files:

- Create XML deployment files that require no compiling.
- Create Java source files, which you compile and deploy outside of TopLink Workbench.

Oracle recommends XML deployment because XML files are easier to deploy and troubleshoot than compiled Java files. 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.

Note: If you are using EJB 3.0, you can use annotations to specify most of what you formerly specified in deployment descriptors. Use deployment descriptors to override annotations or specify options not supported by annotations. For more information on what annotations are currently supported, see *Oracle Containers for J2EE Enterprise JavaBeans Developer's Guide*.

project.xml File

The `project.xml` file is the core of your application. It contains the descriptors and mappings you define and also includes any named queries or finders associated with your project.

This section describes the following:

- XSD File Format
- Non-CMP Applications and Project Metadata
- CMP Applications and Project Metadata
- Creating project.xml With TopLink Workbench
- Creating project.xml Programatically

XSD File Format

Starting with 10*g* (10.1.3.1.0), the `project.xml` file uses an XML schema file format (XSD file) instead of the old document type definition. This defines not only the elements and attributes, but also the rules that govern how the elements and attributes are used in a valid XML file. The XSD file is formatted as standard XML and fully compliant with Oracle namespaces. Although TopLink can read both the current XSD and older DTD formats, only the current XSD format is written out.

Previously formats were defined only by the DTDs. You can now generate deployment XML files based on the following XML schemas:

- `object-persistence_1_0.xsd`: This schema defines general persistence and mapping concepts.
- `toplink-object-persistence_10_1_3.xsd`: This schema extends the general concepts to include additional TopLink specific data.

For more information, refer to the appropriate XSD in the `<TOPLINK_HOME>\config\xsds` directory. The XSD files are also available on OTN at:

- `http://www.oracle.com/technology/oracleas/schema/object-persistence_1_0.xsd`
- `http://www.oracle.com/technology/oracleas/schema/toplink-object-persistence_10_1_3.xsd`

Non-CMP Applications and Project Metadata

For a non-CMP application, you define your project metadata in a `project.xml` file.

The `project.xml` file provides a simple and flexible way to configure, modify, and troubleshoot the project metadata. Because of these attributes, the `project.xml` file is the preferred way to configure a TopLink project.

TopLink Workbench provides a graphical tool to build and edit the `project.xml` file. For information on creating projects with TopLink Workbench, see "Creating project.xml With TopLink Workbench" on page 8-3.

CMP Applications and Project Metadata

For a CMP application, how you specify project metadata is dependent upon the J2EE application server you are deploying your application (see "toplink-ejb-jar.xml File" on page 8-6).

Creating project.xml With TopLink Workbench

Because you must synchronize the `project.xml` file with the classes and data source associated with your application, Oracle recommends that you not modify this file manually. TopLink Workbench ensures proper synchronization, and is the best way to make changes to the project. Simply modify the project in TopLink Workbench and redeploy the `project.xml` file. Using this option reduces development time by eliminating the need to regenerate and recompile Java code each time the project changes.

See "Exporting Project Information" on page 18-13 for detailed information on exporting the deployment XML information.

Note: You can name this file with a name other than `project.xml`; however, for clarity, this discussion assumes that the file has not been renamed.

Creating project.xml Programatically

Optionally, you can use the `DeploymentXMLGenerator` API to programatically generate the `project.xml` file in either of the following ways:

- From an application, instantiate the `DeploymentXMLGenerator` and your java source. Call the following method:

  ```
  generate (<MW_Project.mwp>, <output file.xml>)
  ```

- From the 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>
  ```

Before you use either method, ensure your the classpath includes the *<ORACLE_HOME>*`\toplink\config` directory.

Note: If you are using EJB 3.0, you can use annotations to specify most of what you formerly specified in the `project.xml` file. To override annotations or specify options not supported by annotations, you can still provide a `project.xml` file in your EJB 3.0 application. For more information on what annotations are currently supported, see *Oracle Containers for J2EE Enterprise JavaBeans Developer's Guide*

---

sessions.xml File

Each TopLink project belongs to a TopLink *session*. A session is the facade through which an application accesses TopLink functionality (for more information on sessions, see Part XV, "TopLink Sessions"). Where you define a session differs depending on whether or not your application uses CMP.

This section describes the following:

- XSD File Format
- Non-CMP Applications and Session Metadata
- CMP Applications and Session Metadata

XSD File Format

Starting with 10*g* (10.1.3.1.0), the `sessions.xml` file uses an XML schema file format (XSD file) instead of the old document type definition. In addition to gaining all the benefits of using an XSD, this change ensures that the TopLink run-time environment provides better diagnostics during `sessions.xml` file loading and validation.

The XSD files are also available on OTN at:
`http://www.oracle.com/technology/oracleas/schema/sessions_10_1_3.xsd`

When you use the XSD formatted `sessions.xml` file, the TopLink run time separates `sessions.xml` file validation from session instantiation. Separating XML file formatting problems from Session Manager session instantiation problems simplifies troubleshooting. Exceptions thrown during validation clearly indicate that the failure is due to an invalid `sessions.xml` file as Example 8–1 illustrates.

Example 8–1 Enhanced Validation Exceptions

```
Exception [TOPLINK-9010] (Oracle TopLink - 10g (10.0.3)(Build 040127Dev)):
oracle.toplink.exceptions.SessionLoaderException
Exception Description: A End tag does not match start tag 'session'. was thrown while parsing
the XML file against the XML schema.
Internal Exception: oracle.xml.parser.v2.XMLParseException: End tag does not match start tag
'session'.
```

Non-CMP Applications and Session Metadata

For a non-CMP application, you define your sessions in a `sessions.xml` file.

The `sessions.xml` file provides a simple and flexible way to configure, modify, and troubleshoot the application sessions. Because of these attributes, the `sessions.xml` file is the preferred way to configure a TopLink session.

TopLink Workbench provides a graphical tool to build and edit the `sessions.xml` file. For information on creating sessions with TopLink Workbench, see Chapter 73, "Creating Sessions".

CMP Applications and Session Metadata

For a CMP project, how you specify session metadata is dependent upon the J2EE application server to which you are deploying your application:

- For OC4J, the session configuration is done in the `orion-ejb-jar.xml` file. You can specify the `data-source`, some common session options, and a session customizer class (see "OC4J and the orion-ejb-jar.xml File" on page 8-6). In this

case, you name the TopLink project XML file as `toplink-ejb-jar.xml` (see "project.xml File" on page 8-2)

- For BEA WebLogic Server, the session configuration is done in the `toplink-ejb-jar.xml` file. You can specify the `data-source`, some common session options, and a session customizer class (see "toplink-ejb-jar.xml File" on page 8-6).
- For IBM WebSphere application server, the session configuration is done in a `sessions.xml` file which must be named `toplink-ejb-jar.xml` (see Chapter 73, "Creating Sessions").

> **Note:** If you are using EJB 3.0, you cannot use annotations to specify session configuration. You must provide a `sessions.xml` file, if one is applicable to your application type.

ejb-jar.xml File

Each EJB module contains one `ejb-jar.xml` file that describes all the EJB in the module.

Most IDEs provide facilities to create the `ejb-jar.xml` file. For more information about generating this file from your IDE, see your IDE documentation.

If you build an EJB application, Oracle recommends that you use TopLink Workbench to build the `ejb-jar.xml` file. Because TopLink Workbench can both read and write the `ejb-jar.xml file`, you can use TopLink Workbench to maintain your `ejb-jar.xml` file in the following ways:

- When you change the file manually outside of TopLink Workbench, reimport the `ejb-jar.xml` file into TopLink Workbench project to refresh the project.
- When you change the TopLink Workbench project, TopLink Workbench updates the `ejb-jar.xml` file automatically when you save the project.

For more information about managing the `ejb-jar.xml` file in TopLink Workbench, see "Working With the ejb-jar.xml File" on page 18-15 for more information.

> **Note:** If you are using EJB 3.0, you can use annotations to specify most of what you formerly specified in the `ejb-jar.xml` file. To override annotations or specify options not supported by annotations, you can still provide an `ejb-jar.xml` file in your EJB 3.0 application. For more information on what annotations are currently supported, see *Oracle Containers for J2EE Enterprise JavaBeans Developer's Guide*.

<*J2EE-Container*>-ejb-jar.xml File

The contents of the <*J2EE-Container*>`-ejb-jar.xml` file depend on the container to which you deploy your EJB. To create this file, use the tools that accompany your container.

In most cases, the <*J2EE-Container*>`-ejb-jar.xml` file integrates with TopLink without revision. However, in some cases, you must make some TopLink-specific modifications.

Note: If you are using EJB 3.0, you can use annotations to specify most of what you formerly specified in the `<J2EE-Container>-ejb-jar.xml` file. To override annotations or specify options not supported by annotations, you can still provide a `<J2EE-Container>-ejb-jar.xml` file in your EJB 3.0 application. For more information on what annotations are currently supported, see *Oracle Containers for J2EE Enterprise JavaBeans Developer's Guide*.

For more information, see the following:

- OC4J and the orion-ejb-jar.xml File
- BEA WebLogic Server and the weblogic-ejb-jar.xml File

OC4J and the orion-ejb-jar.xml File

Table 8–1 summarizes the scenarios in which you may choose to modify the `orion-ejb-jar.xml` file.

Table 8–1 When to Modify the orion-ejb-jar.xml File

CMP Type	Mapping Type	Action
Orion	Specified in `orion-ejb-jar.xml`	1. Deploy.
Orion	Default mappings	1. Edit the `orion-ejb-jar.xml` file to set `persistence-manager` attribute `name` to `orion`. 2. Deploy.
Toplink	Specified in `toplink-ejb-jar.xml` (default persistence manager properties)	1. Deploy.
Toplink	Specified in `toplink-ejb-jar.xml` (custom persistence manager properties)	1. Edit the `orion-ejb-jar.xml` file to set `persistence-manager` attribute `name` to `toplink`. 2. Edit additional `persistence-manager` subentries (see "Configuring the orion-ejb-jar.xml File for OC4J" on page 8-9). 3. Deploy.
Toplink	Default mappings (no `toplink-ejb-jar.xml`)	1. Deploy.

For more information on configuring the `orion-ejb.jar.xml` file, see "Configuring the orion-ejb-jar.xml File for OC4J" on page 8-9.

BEA WebLogic Server and the weblogic-ejb-jar.xml File

For more information on configuring the `weblogic-ejb-jar.xml`, see "Configuring the weblogic-ejb-jar.xml File for BEA WebLogic Server" on page 8-13.

toplink-ejb-jar.xml File

The `toplink-ejb-jar.xml` file is used only in CMP projects. The TopLink runtime uses properties set in the `<J2EE container>-ejb-jar.xml` file (see "<J2EE-Container>-ejb-jar.xml File" on page 8-5) to locate the `toplink-ejb-jar.xml` file and read it in.

> **Note:** If you are using EJB 3.0, you can use annotations to specify most of what you formerly specified in the `toplink-ejb-jar.xml` file. To override annotations or specify options not supported by annotations, you can still provide a `toplink-ejb-jar.xml` file in your EJB 3.0 application. For more information on what annotations are currently supported, see *Oracle Containers for J2EE Enterprise JavaBeans Developer's Guide*

The purpose of `toplink-ejb-jar.xml` file depends on the type of application server you are using:

- OC4J and the toplink-ejb-jar.xml File
- BEA WebLogic Server and the toplink-ejb-jar.xml File
- IBM WebSphere Application Server and the toplink-ejb-jar.xml File

OC4J and the toplink-ejb-jar.xml File

When deploying a CMP application to OC4J, the `toplink-ejb-jar.xml` file is the name used for the `project.xml` file.

To create the `toplink-ejb-jar.xml` file in this case, simply rename your `project.xml` file. For more information, see "project.xml File" on page 8-2.

BEA WebLogic Server and the toplink-ejb-jar.xml File

When deploying a CMP application to BEA WebLogic Server, the `toplink-ejb-jar.xml` file contains a reference to the `project.xml` file.

Example 8–2 shows a typical BEA WebLogic Server `toplink-ejb-jar.xml` file:

Example 8–2 BEA WebLogic Server toplink-ejb-jar.xml File

```
<?xml version="1.0"?>
<!DOCTYPE toplink-ejb-jar PUBLIC "-//Oracle Corp.//DTD TopLink CMP WebLogic 10.0.3 Developer
Preview//EN" "toplink-wls-ejb-jar_10_0_3.dtd">
<toplink-ejb-jar>
    <session>
        <name>ejb20_EmployeeDemo</name>
        <project-xml>Employee.xml</project-xml>
        <login>
            <datasource>jdbc/JTSTopLinkDS</datasource>
            <non-jts-datasource>jdbc/TopLinkDS</non-jts-datasource>
        </login>
        <customization-class>
           oracle.toplink.demos.ejb.cmp.wls.employee.EmployeeCustomizer
        </customization-class>
    </session>
</toplink-ejb-jar>
```

For BEA WegLogic Server, you can specify an optional deployment customization class (that implements `oracle.toplink.ejb.cmp.DeploymentCustomization` interface) used to allow deployment customization of TopLink mapping and run-time configuration. In Example 8–2, the deployment customization class is named `EmployeeCustomizer`. This deployment customization class must be fully qualified by its package name and included in the deployment JAR.

At deployment time, the TopLink runtime creates a new instance of this class and invokes its methods `beforeLoginCustomization` (before the TopLink runtime logs

in to the session) and `afterLoginCustomization` (after the TopLink runtime logs in to the session), passing in the TopLink session as a parameter.

Use your implementation of the `beforeLoginCustomization` method to configure session attributes not supported by the `pm-properties`, including the following:

- cache coordination
- parameterized SQL
- native SQL
- batch writing/batch size
- byte-array/string binding
- EIS login
- event listeners
- table qualifier
- sequencing

In Example 8–2, notice the `<non-jts-datasource>` element. The usage of this element is not limited to BEA WebLogic Server `toplink-ejb-jar.xml` file. Any time you use cache coordination with Java Transaction API (JTA), you need to configure this element through TopLink Workbench (see "Configuring a Nontransactional Read Login" on page 86-3). This enables TopLink to do the out-of-context read operations and optimization of the sequencing assignments (see "Externally Managed Transactional Data Sources" on page 81-1).

For more information about session configuration, see Chapter 74, "Configuring a Session".

IBM WebSphere Application Server and the toplink-ejb-jar.xml File

When deploying a CMP application to IBM WebSphere application server, the `toplink-ejb-jar.xml` file is the name used for the `sessions.xml` file and contains a reference to the `project.xml` file.

To create the `toplink-ejb-jar.xml` file in this case, simply rename your `sessions.xml` file. For more information, see "sessions.xml File" on page 8-4.

Java Applications

In a Java application, TopLink does not use a J2EE container for deployment. Instead, it relies on 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 file on the classpath.

Java applications require the following deployment files:

- project.xml File
- sessions.xml File

JavaServer Pages and Servlet Applications

Many designers build TopLink applications that use JavaServer Pages (JSP) and Java servlets. This type of design generally supports Web-based applications.

JSP and servlet applications require the following deployment files:

- project.xml File
- sessions.xml File

Session Bean Applications

Session beans generally model a process, operation, or service and as such, are not persistent. You can build TopLink applications that wrap interaction with TopLink in session beans. Session beans execute all TopLink-related operations on behalf of the client.

This type of design uses JTS and externally managed transactions, but does not incur the overhead associated with CMP applications. Session bean applications also scale and deploy easily.

Session bean applications require the following deployment files:

- project.xml File
- sessions.xml File

CMP Applications

Many applications use the persistence mechanisms a J2EE container offers. TopLink provides full support for this type of application.

You can only use one persistence manager for all the entity beans with container-managed persistence in a JAR file.

CMP applications require the following deployment files:

- ejb-jar.xml File
- <J2EE-Container>-ejb-jar.xml File
- toplink-ejb-jar.xml File

BMP Applications

If you choose to write your own persistence code with BMP, you can take advantage of the classes in `oracle.toplink.ejb.bmp` package. Whether or not you use these classes, BMP applications require the following deployment files:

- project.xml File
- sessions.xml File
- ejb-jar.xml File

Configuring the orion-ejb-jar.xml File for OC4J

To deploy a TopLink application to OC4J 10*g* (10.1.3.1.0) or later, modify the `orion-ejb-jar.xml` file as follows:

- Configuring persistence-manager Entries

If you are migrating an application from a previous release of OC4J, you can use the TopLink migration tool to automatically migrate persistence information from your `orion-ejb-jar.xml` file to a new `toplink-ejb-jar.xml`. For more information, see "Migrating OC4J Orion Persistence to OC4J TopLink Persistence" on page 7-5.

Configuring persistence-manager Entries

If you are using TopLink as your OC4J persistence manager, the default persistence manager in 10*g* (10.1.3.1.0), you can configure the `persistence-manager` subentry (see Table 8–2) in the `orion-ejb-jar.xml` file. For more information on the scenarios in which you would want to modify `orion-ejb-jar.xml`, see "OC4J and the orion-ejb-jar.xml File" on page 8-6.

If you are not using TopLink as your OC4J persistence manager, do not modify the `persistence-manager` subentries.

OC4J 10*g* (10.1.3.1.0) and later do not support `entity-deployment` attribute `pm-name`. Use `persistence-manager` attribute `name` instead (see Table 8–2). When OC4J parses the `orion-ejb.jar.xml` file, if it finds a `pm-name` attribute, OC4J ignores its value and logs the following warning message:

> **WARNING: Use of `pm-name` is unsupported and will be removed in a future release. Specify pm usage using `<persistence-manager>` 'name' instead.**

Table 8–2 orion-ejb-jar.xml File persistence-manager Entries

Entry	Description
`name`	The name of the persistence manager to use. Set this value to `toplink`. If you set the `name` property to `toplink`, you may also configure `pm-properties` (see "Configuring pm-properties" on page 8-10).
`class-name`	Do not configure this attribute. If `name` is set to `toplink`, then `class-name` is set correctly by default.
`descriptor`	This property applies only when `name` is set to `toplink`. If you export your TopLink mapping metadata to a deployment XML file, set this property to the name of the deployment XML file (default: `toplink-ejb-jar.xml`). Do not set this property if you are using a TopLink project class instead of a mapping metadata file (see `project-class` in Table 8–3).

Configuring pm-properties

When you select TopLink as the persistence manager (see `name` in Table 8–2), use the `persistence-manager` subentries for `pm-properties` (see Table 8–3) to configure the TopLink session that the TopLink run time creates and uses internally for CMP projects. The `persistence-manager` subentries take the place of a `sessions.xml` file in a CMP project.

> **Note:** You can only configure a subset of session features using these properties and in most cases, default configuration applies. To configure all session features and to override defaults, you must use a customization class (see `customization-class` in Table 8–3).

Table 8–3 orion-ejb-jar.xml File persistence-manager Subentries for pm-properties

Entry	Description
`session-name`	Unique name for this TopLink-persisted EJB deployment JAR file. Must be unique among all TopLink-persisted deployed JAR files in this application server instance. When the TopLink run time internally creates a TopLink session for this TopLink-persisted deployed JAR file, the TopLink session manager stores the session instance under this `session-name`. For more information about the session manager, see Chapter 75, "Acquiring and Using Sessions at Run Time"). If you do not specify a name, the TopLink runtime will generate a unique name.
`project-class`	If you export your TopLink mapping metadata to a Java class (that extends `oracle.toplink.sessions.Project`), set this property to the name of the class, fully qualified by its package name. Be sure to include the class file in the deployable JAR file. Do not set this property if you are using a mapping metadata file (see `descriptor` in Table 8–2).
`customization-class`	Optional Java class (that implements `oracle.toplink.ejb.cmp.DeploymentCustomization`) used to allow deployment customization of TopLink mapping and run-time configuration. At deployment time, the TopLink run time creates a new instance of this class and invokes its methods `beforeLoginCustomization` (before the TopLink run time logs into the session) and `afterLoginCustomization` (after the TopLink runtime logs into the session), passing in the TopLink session as a parameter. Use your implementation of the `beforeLoginCustomization` method to configure session attributes not supported by the `pm-properties` including: cache coordination (see also "Configuring cache-synchronization Properties" on page 8-11), parameterized SQL, native SQL, batch writing/batch size, byte-array/string binding, EIS login, event listeners, table qualifier, and sequencing. For more information about session configuration, see Chapter 74, "Configuring a Session". The class must be fully qualified by its package name and included in the deployment JAR file.
`db-platform-class`	Optional TopLink database platform class (instance of `oracle.toplink.platform.database` or `oracle.toplink.platform.database.oracle`) containing TopLink support specific to a particular database. Set this value to the database platform class that corresponds to the database that your application uses. The class must be fully qualified by its package name.
`remote-relationships`	Optional flag to allow relationships between remote objects. Valid values are: ▪ `true`: All relationships will be maintained through the remote interfaces of the entity beans ▪ `false`: Disables this feature.
`cache-synchronization`	See "Configuring cache-synchronization Properties" on page 8-11.
`default-mapping`	See "Configuring default-mapping Properties" on page 8-12.

Configuring cache-synchronization Properties

When you select TopLink as the persistence manager (see `name` in Table 8–2), use the `pm-properties` subentry for `cache-synchronization` (see Table 8–4) to configure TopLink cache coordination features of the session that the TopLink run time uses internally for CMP projects. For more information about TopLink cache coordination, see "Understanding Cache Coordination" on page 87-10.

When this subentry is present, you must use a customization class (see `customization-class` in Table 8–3) to complete cache coordination configuration.

For more information about TopLink cache coordination configuration, see Chapter 88, "Configuring a Coordinated Cache".

Table 8–4 orion-ejb-jar.xml File pm-properties Subentries for cache-synchronization

Entry	Description
`mode`	An indicator of whether or not cache coordination updates should be propagated to other servers synchronously or asynchronously. Valid values are as follows: ▪ `asynchronous` (default) ▪ `synchronous`

`server-url`	**For a JMS coordinated cache**: assuming that you are using the Oracle Application Server Containers for J2EE (OC4J) JNDI naming service and that all the hosts in your coordinated cache can communicate using OC4J proprietary RMI protocol ORMI, use a URL like:
	`ormi://<JMS-host-IP>:<JMS-host-port>`
	where `JMS-host-IP` is the IP address of the host on which the JMS service provider is running and `JMS-host-port` is the port on which the JMS service provider is listening for JMS requests.
	For an RMI or CORBA coordinated cache: assuming that you are using the OC4J JNDI naming service and that all the hosts in your coordinated cache can communicate using OC4J proprietary RMI protocol ORMI on OC4J default port 23791, use a URL like:
	`ormi://<session-host-IP>:23791`
	where `session-host-IP` is the IP address of the host on which this session is deployed.

`server-user`	Optional username required to log in to the JNDI naming service.

Configuring default-mapping Properties

When you select TopLink as the persistence manager (see `name` in Table 8–2), use the `pm-properties` subentry for `default-mapping` (see Table 8–5) to configure the TopLink default mapping and automatic table generation feature.

For more information about TopLink default mappings, see "Default Mapping in CMP Projects Using OC4J at Run Time" on page 30-4.

For more information about TopLink automatic table generation, see "Automatic Database Table Creation" on page 5-6.

Table 8–5 orion-ejb-jar.xml File pm-properties Subentries for default-mapping

Entry	Description
`db-table-gen`	Optional element that determines what TopLink will do to prepare the database tables that are being mapped to. Valid values are: ■ `Create` (default): This value tells TopLink to create the mapped tables during the deployment. If the tables already exist, TopLink will log an appropriate warning messages (such as "*Table already existed...*") and keeps processing the deployment. ■ `DropAndCreate`: This value tells TopLink to drop tables before creating them during deployment. If a table does not initially exist, the drop operation will cause anSQLException to be thrown through the driver. However, TopLink handles the exception (logs and ignores it) and moves on to process the table creation operation. The deployment fails only if both drop and create operations fail. ■ `UseExisting`: This value tells TopLink to perform no table manipulation. If the tables do not exist, deployment still goes through without error. If no `orion-ejb-jar.xml` file is defined in your EAR file, the OC4J container generates one during deployment. In this case, to specify a value for `db-table-gen`, use the TopLink system property `toplink.defaultmapping.dbTableGenSetting`. For example: `-Dtoplink.defaultmapping.dbTableGenSetting="DropAndCreate"`. The `orion-ejb-jar.xml` property overrides the system property. If both the `orion-ejb-jar.xml` property and the system property are present, TopLink retrieves the setting from the `orion-ejb-jar.xml` file. This setting overrides autocreate-tables and autodelete-tables configuration at the application (EAR) or system level. For more information, see "Automatic Database Table Creation" on page 5-6.
`extended-table-names`	An element used if the generated table names are not long enough to be unique. Values are restricted to `true` or `false` (default). When set to `true`, the TopLink run time will ensure that generated tables names are unique. In default mapping, each entity is mapped to one table. The only exception is in many-to-many mappings where there is one extra relation table involved in the source and target entities. When `extended-table-names` is set to `false` (the default), a simple table naming algorithm is used as follows: table names are defined as `TL_<bean_name>`. For example, if the bean name is `Employee`, the associated table name would be `TL_EMPLOYEE`. However, if the same entity is defined in multiple JAR files in an application, or across multiple applications, table-naming collision is inevitable. To address this problem, set `extended-table-names` to `true`. When set to `true`, TopLink uses an alternative table-naming algorithm as follows: table names are defined as *`<bean_name>_<jar_name>_<app_name>`*. This algorithm uses the combination of bean, JAR, and EAR names to form a table name unique across the application. For example, given a bean named `Employee`, which is in `Test.jar`, which is in `Demo.ear` (and the application name is "`Demo`"), then the corresponding table name will be `EMPLOYEE_TEST_DEMO`. If there is no `orion-ejb-jar.xml` file defined in the EAR file, the OC4J container generates one during deployment. In this case, to specify a value for `extended-table-names`, use the TopLink system property `toplink.defaultmapping.useExtendedTableNames`. For example: `-Dtoplink.defaultmapping.useExtendedTableNames="true"`. The `orion-ejb-jar.xml` property overrides the system property. If both the `orion-ejb-jar.xml` property and the system property are present, TopLink retrieves the setting from the `orion-ejb-jar.xml` file.

Configuring the weblogic-ejb-jar.xml File for BEA WebLogic Server

To deploy a TopLink application to a BEA WebLogic Server, modify the `weblogic-ejb-jar.xml` file as described in:

- Configuring persistence-descriptor Entries

Avoid the `weblogic-ejb-jar.xml` tags that TopLink either does not support or does not require (see "Unsupported weblogic-ejb-jar.xml File Tags" on page 8-15).

If you are migrating a BEA WEbLogic Server application to OC4J, you can use the TopLink migration tool to automatically migrate persistence information from your `weblogic-ejb-jar.xml` file to a new `toplink-ejb-jar.xml` file. For more information, see "Migrating BEA WebLogic Persistence to OC4J TopLink Persistence" on page 7-16.

Configuring persistence-descriptor Entries

Within the `weblogic-ejb-jar.xml` file, each bean must have a `persistence-descriptor` entry with subentries, as follows:

- Configure the `persistence-descriptor` entry with subentries that indicate 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 8–6 lists the options for the `type-identifier` tag, and Table 8–7 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 8–6 lists the options for the `type-identifier` tag, and Table 8–7 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 8–6 WebLogic type-identifier Settings

EJB Version	XML Elements
2.0	`<type-identifier>TopLink_CMP_2_0</type-identifier>`

Table 8–7 WebLogic type-version Settings

WebLogic EJB 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>`

Unsupported weblogic-ejb-jar.xml File Tags

The `weblogic-ejb-jar.xml` file includes several tags that TopLink either does not support or does not require:

- `concurrency-strategy`: This tag specifies how WebLogic manages concurrent users for a given bean. Because TopLink manages concurrent access internally, it does not require this tag.

  For more information about the TopLink concurrency strategy, see "Configuring Locking Policy" on page 25-64.

- `db-is-shared`: Because TopLink does not make any assumptions about the exclusivity of database access, TopLink does not require this tag. TopLink addresses multiuser access issues through various locking and refreshing policies.
- `delay-updates-until-end-of-tx`: TopLink always delays updates until the end of a transaction, and does not require this tag.
- `finders-load-bean`: TopLink always loads the bean upon execution of the finder, and does not require this tag.
- `pool`: 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 TopLink does not use a pooling strategy, TopLink ignores this tag.
- `is-modified-method-name`: TopLink does not require a bean developer-defined method to detect changes in the object state.
- `isolation-level`: Because isolation level settings for the cache or database transactions are specified in the TopLink project, TopLink ignores this tag.
- `cache`: Because you define TopLink cache properties in TopLink Workbench, this tag is unnecessary.

9

Packaging a TopLink Application

How you package the components of your application depends on the type of application and how you plan to deploy it.

This section describes TopLink-specific details applicable to the common packaging strategies used for the following types of application:

- Java Applications
- JavaServer Pages and Servlet Applications
- Session Bean Applications
- CMP Applications
- BMP Applications

> **Note:** If you are using EJB 3.0, you may be using annotations instead of some deployment files. Include deployment descriptors to override annotations or specify options not supported by annotations.

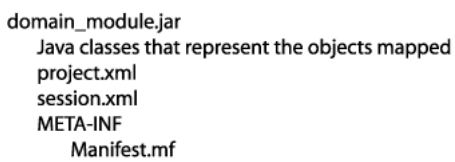
For information, see the following:

- "Integrating TopLink With an Application Server" on page 7-1
- "Creating TopLink Files for Deployment" on page 8-1
- "Deploying a TopLink Application" on page 10-1

Java Applications

For non-J2EE Java applications, it is common to package the application in a single JAR file as Figure 9–1 shows.

Figure 9–1 Packaging a non-J2EE Java Application



This JAR contains the 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)
- The mapped classes required by the application, in a fully-resolved directory structure

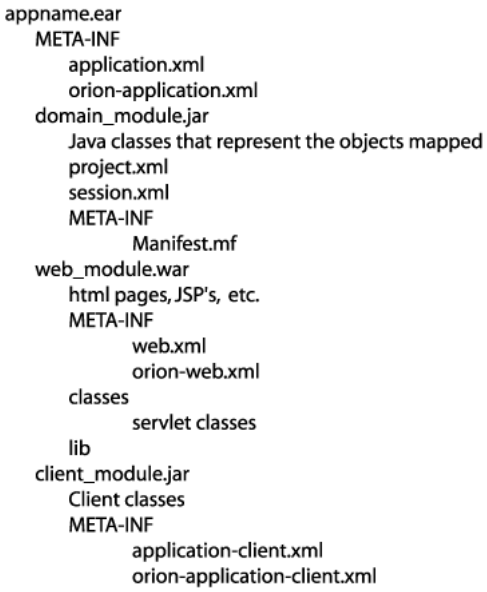
When you create the JAR file, the JAR building utility automatically creates a directory structure within the JAR. Ensure that the `sessions.xml` file and the `project.xml` file (or project class file) appear at the root of the JAR file. Ensure that the class directory structure starts at the root of the JAR.

If you do not store the `project.xml` or `sessions.xml` files at the root of the JAR file, see "Packaging With TopLink Metadata File Resource Paths" on page 9-7.

JavaServer Pages and Servlet Applications

For simple J2EE applications without EJB, it is common to package the application in an Enterprise Archive (EAR) file made up of various J2EE application component archives as Figure 9–2 shows.

Figure 9–2 Packaging a J2EE JSP or Servlet Application Without EJB



The component archives with TopLink dependencies include the following:

- TopLink Domain JAR

TopLink Domain JAR

The domain JAR contains the 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)

- The mapped classes required by the application, in a fully resolved directory structure

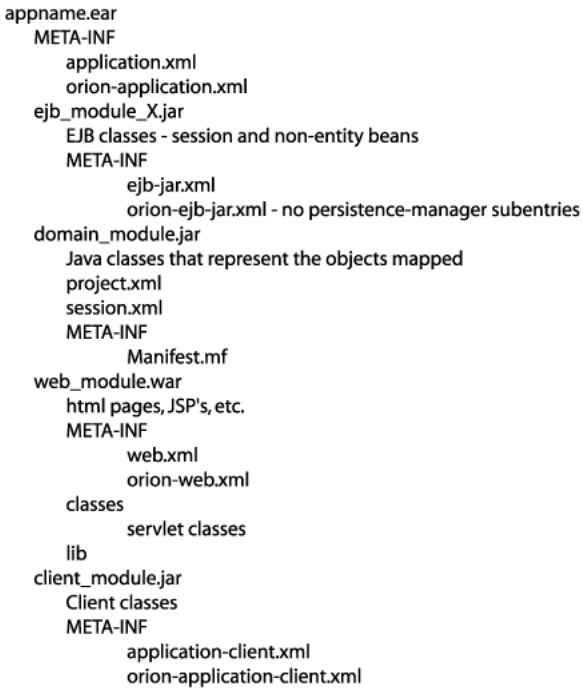
When you create the JAR file, the JAR building utility automatically creates a directory structure within the JAR. Ensure that the `sessions.xml` file and the `project.xml` file (or `project.class` file) appear at the root of the JAR file. Also ensure that the class directory structure starts at the root of the JAR.

If you do not store the `project.xml` or `sessions.xml` files at the root of the JAR file, see "Packaging With TopLink Metadata File Resource Paths" on page 9-7.

Session Bean Applications

For J2EE applications with session beans, it is common to package the application in an Enterprise Archive (EAR) file made up of various J2EE application component archives as Figure 9–3 shows.

Figure 9–3 Packaging a J2EE Application With Session Beans



The component archives with TopLink dependencies include the following:

- TopLink Domain JAR
- EJB JAR

TopLink Domain JAR

The domain JAR contains the TopLink files and domain objects required by the application, including the following:

- sessions.xml File
- project.xml File (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.

If you do not store the `project.xml` or `sessions.xml` files at the root of the JAR file, see "Packaging With TopLink Metadata File Resource Paths" on page 9-7.

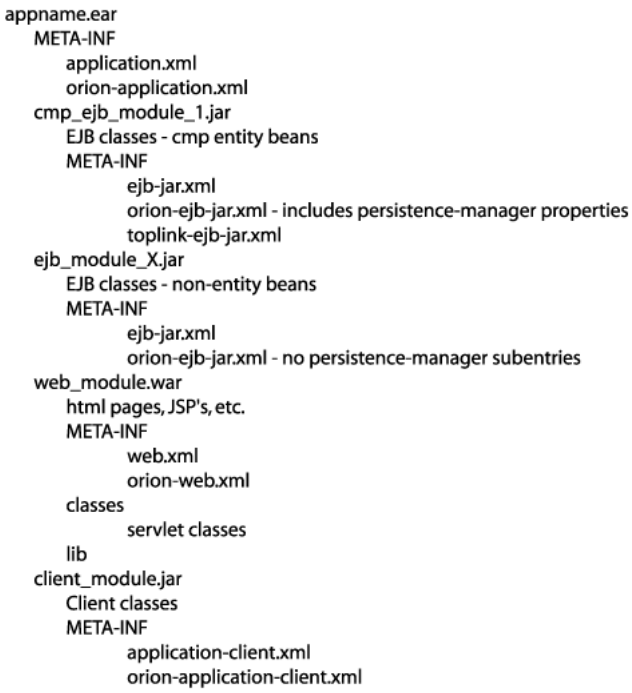
EJB JAR

In this type of application, the EJB JAR contains session beans. Consequently, its `orion-ejb-jar.xml` file does not contain `persistence-manager` or `pm-properties` entries. These entries apply only to CMP applications.

CMP Applications

For J2EE applications that use CMP to persist entity beans, it is common to package the application in an Enterprise Archive (EAR) file made up of various J2EE application component archives as Figure 9–4 shows.

Figure 9–4 Packaging a J2EE Application With Entity Beans With Container-Managed Persistence



The component archives with TopLink dependencies include the following:

- EJB JAR

EJB JAR

In this type of application, the EJB JAR file specifically service both non-entity and entity EJB. It includes the following:

- The home and remote, and all implementation code for all mapped beans in the application.
- All mapped non-EJB classes from the TopLink Workbench project
- The home and remote, and all implementation code for any session beans included in the application.
- Helper classes that contain TopLink amendment methods, and any other classes the application requires.

  For example, an instance of `oracle.toplink.ejb.cmp.DeploymentCustomization` (for more information, see `customization-class` in Table 8–3 in "Configuring persistence-manager Entries" on page 8-10).

Store the following XML files in the EJB JAR `\meta-inf` directory:

- ejb-jar.xml File
- <J2EE-Container>-ejb-jar.xml File
- toplink-ejb-jar.xml File

> **Note:** If you do not use XML files for deployment, include your compiled `oracle.toplink.sessions.Project` file at the root of the EJB JAR (not in the `\meta-inf` directory).

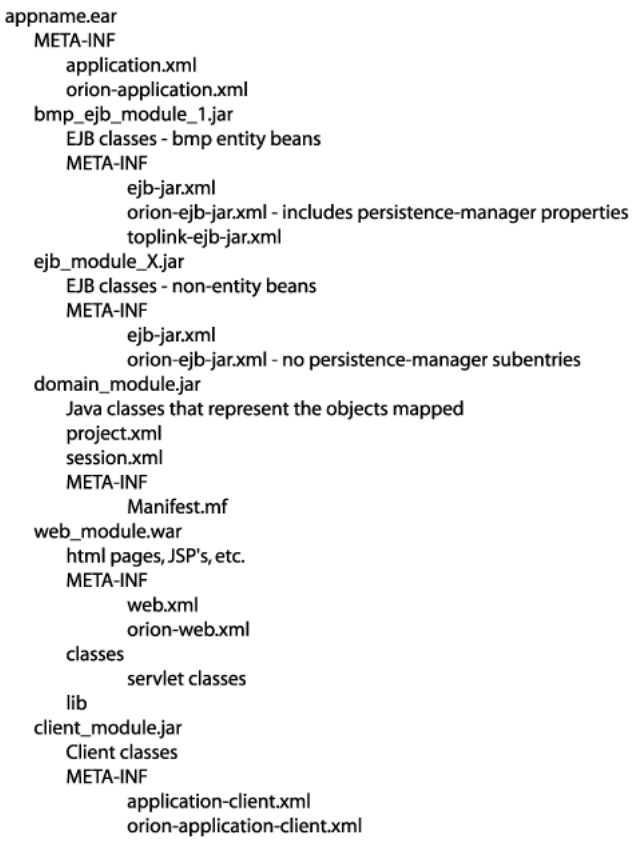
Because the EJB JAR contains both session and entity beans, if you are using OC4J or BEA WebLogic (WLS), the `<J2EE-Container>-ejb-jar.xml` file must contain `persistence-manager` and `pm-properties` entries. For more information, see "Configuring persistence-manager Entries" on page 8-10.

You must persist all of the entity beans to the same data source. For a CMP application, TopLink does not support the session broker functionality (see "Session Broker and Client Sessions" on page 72-26).

BMP Applications

For J2EE applications that use BMP to persist entity beans, it is common to package the application in an Enterprise Archive (EAR) file made up of various J2EE application component archives as Figure 9–5 shows.

Figure 9–5 Packaging a J2EE Application With Entity Beans With Bean-Managed Persistence



The component archives with TopLink dependencies include the following:

- TopLink Domain JAR
- EJB JAR

TopLink Domain JAR

The domain JAR contains the TopLink files and domain objects required by the application, including the following:

- sessions.xml File
- project.xml File (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.

If you do not store the `project.xml` or `sessions.xml` files at the root of the JAR file, see "Packaging With TopLink Metadata File Resource Paths" on page 9-7.

EJB JAR

In this type of application, the EJB JAR file specifically services both session and entity beans. It includes the following:

- The home and remote, and all implementation code for all mapped beans in the application
- All mapped non-EJB classes from the TopLink Workbench project
- The home and remote, and all implementation code for any session beans included in the application
- Helper classes that contain TopLink amendment methods, and any other classes the application requires

Store the following XML files in the EJB JAR `\meta-inf` directory:

- ejb-jar.xml File
- <J2EE-Container>-ejb-jar.xml File

Because the EJB JAR does not contain entity beans with container-managed persistence, its `orion-ejb-jar.xml` file must not contain `persistence-manager` or `pm-properties` entries.

For more information, see "Configuring persistence-manager Entries" on page 8-10.

Packaging With TopLink Metadata File Resource Paths

If you do not store the `project.xml` or `sessions.xml` files at the root of the JAR file, then you must provide the full resource path to the files when accessing them. Ensure that you use "/" in resources paths, not "\". Using "\" will not work in Java.

For example, in the `jar` element, reference the `project.xml` and `sessions.xml` files as follows:

```
<jar>/myapp/ordersys/persist/sessions.xml
<jar>/myapp/ordersys/persist/project.xml
```

In the `sessions.xml` file, reference the `project.xml` as follows:

```
myapp/ordersys/persist/project.xml
```

To acquire the session, use the following:

```
SessionManager.getManager().getSession(
    new XMLSessionConfigLoader("myapp/ordersys/persist/sessions.xml"),
    "OrdersysSession",
    getClass().getClassLoader()
);
```

For more information about acquiring sessions at run time, see "Acquiring a Session From the Session Manager" on page 75-3.

10

Deploying a TopLink Application

This chapter includes deployment information on the following TopLink applications:

- Java Applications
- JavaServer Pages and Servlets
- Session Bean Applications
- CMP Applications
- BMP Applications

For more information, see the following:

- "Integrating TopLink With an Application Server" on page 7-1
- "Creating TopLink Files for Deployment" on page 8-1
- "Packaging a TopLink Application" on page 9-1

Java Applications

Build the JAR file (see "Java Applications" on page 9-1) and place it on the classpath.

For more information on accessing TopLink from your client application, see Chapter 75, "Acquiring and Using Sessions at Run Time".

JavaServer Pages and Servlets

After you build the WAR and JAR files (see "JavaServer Pages and Servlet Applications" on page 9-2), build them into an EAR file for deployment. To deploy the EAR to your JSP servlet server, copy the EAR to a commonly used directory. You may also need to use server-specific deployment tools. For more information, see the server documentation.

For more information on accessing TopLink from your client application, see "Loading a Session From sessions.xml With an Alternative Class Loader" on page 75-4.

Session Bean Applications

After you build the WAR and JAR files (see "Session Bean Applications" on page 9-3), build them into an EAR file for deployment. To deploy the EAR file to your J2EE server, copy the EAR to a commonly used directory. You may also need to use server-specific deployment tools. For more information, see the server documentation.

For more information on accessing TopLink from your client application, see "Loading a Session From sessions.xml With an Alternative Class Loader" on page 75-4.

Optionally, you may also consider "Hot Deployment of EJB" on page 10-4.

CMP Applications

After you build the WAR and JAR files (see "CMP Applications" on page 9-4), build them into an EAR file for deployment. To deploy the EAR file to your J2EE server, copy the EAR to a commonly used directory. You may also need to use server-specific deployment tools. For more information, see the server documentation.

This section describes the following:

- Deploying a CMP Application to OC4J
- Deploying a CMP Application to BEA WebLogic Server
- Deploying a CMP Application to IBM WebSphere Application Server 4.0

For additional information on server-specific configuration, see Chapter 7, "Integrating TopLink With an Application Server".

Optionally, you may also consider "Hot Deployment of EJB" on page 10-4.

Deploying a CMP Application to OC4J

The most efficient way to deploy a CMP application to OC4J is using Oracle Enterprise Manager 10*g*. For more information, see *Oracle Application Server Administrator's Guide*.

When you deploy a CMP application to OC4J, the following happens:

- OC4J performs a partial EJB conformance check on the beans and their associated interfaces.
- OC4J builds the internal OC4J classes that manage security and transactions, as well as the RMI stubs and skeletons that enable client access to the beans.
- TopLink builds concrete bean subclasses and EJB finder method implementations.

Deploying a CMP Application to BEA WebLogic Server

TopLink CMP support includes integration for BEA WebLogic Server. To enable TopLink CMP for BEA WebLogic entity beans, use the WebLogic EJB Compiler (ejbc) to compile the EJB JAR file, as follows:

- Run ejbc from the command line. Include the EJB JAR file as a command line argument. ejbc creates an EJB JAR file that contains the original classes as well as all required generated classes and files.

When you run ejbc, the following happens:

- ejbc performs a partial EJB conformance check on the beans and their associated interfaces.
- ejbc 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.
- TopLink builds concrete bean subclasses and EJB finder method implementations.

For more information about running ejbc, see the BEA WebLogic Server 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 the following:

- Bean classes that do not conform with the EJB specification
- Classes missing from the classpath (all domain classes, required TopLink classes, and all required BEA WebLogic classes must be on the classpath)
- 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 RMI compiler (rmic))

Refer to Chapter 15, "Troubleshooting Application Deployment" for additional information.

Deploying a CMP Application to IBM WebSphere Application Server 4.0

TopLink CMP support includes an integration for IBM WebSphere application server 4.0. Use the following procedure to deploy your application to WebSphere applicaton server:

1. Use the TopLink Deploy Tool for WebSphere to compile the EJB JAR file. For more information, see "Using the WebSphere Deploy Tool" on page 10-5.
2. Start the WebSphere Administration Server.
3. Start the Administration Console and deploy the compiled JAR file.

For more information about deploying the JAR file, see the IBM WebSphere application server documentation.

> **Note:** When you deploy an application that contains an entity bean, set up a data source and associate it with the bean. For more information about how to create and associate data sources, see the IBM WebSphere application server documentation.

It is not necessary to deploy the EJB JAR file in WebSphere Studio Application Developer (WSAD), because deployment is carried out using the Deploy Tool (see "Using the WebSphere Deploy Tool" on page 10-5).

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, do the following:

1. Select the application that contains the entity beans.
2. Right-click and choose **Start**.

A message dialog box appears if the bean starts successfully. If an error occurs, consult Part V, "Troubleshooting a TopLink Application" for troubleshooting information.

To start the bean in WSAD, do the following:

1. 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.

BMP Applications

After you build the WAR and JAR files, build them into an EAR file for deployment. To deploy the EAR file to your J2EE server, copy the EAR to a commonly used directory. You may also need to use server-specific deployment tools. For more information, see the server documentation.

For additional information on server-specific configuration, see Chapter 7, "Integrating TopLink With an Application Server".

Optionally, you may also consider "Hot Deployment of EJB" on page 10-4.

Hot Deployment of EJB

Many J2EE containers support *hot deployment*, a feature that enables you to deploy EJB on a running server. Hot deployment allows you to do the following:

- Deploy newly developed EJB to a running production system.
- Remove (undeploy) deployed EJB from a running server.
- Modify (redeploy) the behavior of deployed EJB by updating the bean class definition.

The client receives deployment exceptions when attempting to access undeployed or re-deployed bean instances. The client application must catch and handle the exceptions.

How you configure hot deployment of EJB depends on the type of J2EE application you are deploying:

- Hot Deployment in a CMP Application
- Hot Deployment in a non-CMP Application

For more information about hot deployment, see the J2EE container documentation.

Hot Deployment in a CMP Application

When you take advantage of hot deployment in a CMP application, consider the following:

- You must deploy all related beans (all beans that share a common TopLink project) within the same EJB JAR file. Because 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 TopLink project. This flushes all object caches and rolls back any active object transactions associated with the project.

Hot Deployment in a non-CMP Application

When you take advantage of hot deployment in a non-CMP application, you must refresh the TopLink session using the `SessionManager` method `getSession` with the appropriate arguments (see "Refreshing a Session When the Class Loader Changes" on page 75-6.

If you do not use this `SessionManager` method, then your application is responsible for destroying or refreshing the session when a hot deployment (or hot redeployment) occurs.

Using the WebSphere Deploy Tool

TopLink integration for IBM WebSphere application 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 lets you deploy your project while bypassing the graphical interface element of the tool.

You can use the Deploy Tool on its own (see "Using the Deploy Tool on its Own" on page 10-5) or with the WebSphere Studio Application Developer (see "Using the Deploy Tool With WebSphere Studio Application Developer" on page 10-5).

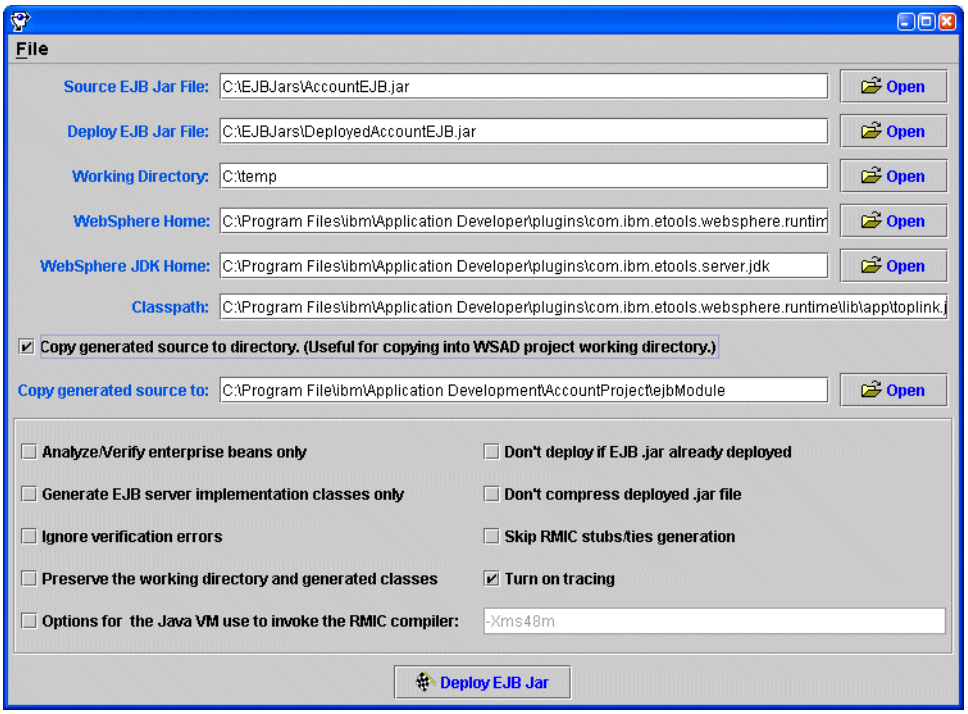
Using the Deploy Tool on its Own

To deploy a JAR file, use this procedure:

1. Start the WebSphere Deploy Tool by executing the appropriate `<TOPLINK_HOME>/bin/wasDeployTool` script: on Windows, execute `wasDeployTool.cmd`; on UNIX, execute `wasDeployTool.sh`.

   The WebSphere Deploy Tool dialog box appears (see Figure 10–1).

Figure 10–1 The Deploy Tool Set Up for Use with WSAD



2. 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.
3. Enable the **Turn on tracing** option if you want to see the details of the process.
4. Click **Deploy EJB**.

Using the Deploy Tool With WebSphere Studio Application Developer

The Deploy tool is compatible with the WSAD.

To deploy from the Deploy Tool to WSAD, do the following:

1. Select the EJB Project in WSAD and choose to generate **Deploy and RMIC Code**.
2. Export the EJB Project to an EJB JAR file, making sure that the TopLink project and `toplink-ejb-jar.xml` files are included in the EJB JAR file.
3. Start the WebSphere Deploy Tool by executing the appropriate `<TOPLINK_HOME>/bin/wasDeployTool` script: on Windows, execute `wasDeployTool.cmd`; on UNIX, execute `wasDeployTool.sh`.

   The WebSphere Deploy Tool dialog appears (see Figure 10–1).
4. Choose the EJB Project working directory so that TopLink overrides the WSAD deploy code with the 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 file.
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.

See Chapter 7, "Integrating TopLink With an Application Server" for additional information.

Part IV

Optimizing and Customizing a TopLink Application

This part describes how to optimize and customize a TopLink application. It contains the following chapters.

- Chapter 11, "Optimization"

  This chapter contains information on the diverse set of features TopLink provides to optimize performance.

- Chapter 12, "Customization"

  This chapter describes how to customize various aspects of TopLink, based on your application's specific needs.

11

Optimization

TopLink provides a diverse set of features to measure and optimize application performance. You can enable or disable most features in the descriptors or session, making any resulting performance gains global.

This chapter includes the following sections:

- Understanding Optimization
- Sources of Application Performance Problems
- Measuring TopLink Performance With the TopLink Profiler
- Measuring TopLink Performance With the Oracle Dynamic Monitoring System (DMS)

Understanding 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 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.

The most important concept associated with tuning your TopLink application is the idea of an iterative approach. The most effective way to tune your application is to do the following:

1. Measure application performance using the TopLink profiler (see "Measuring TopLink Performance With the TopLink Profiler" on page 11-2) or the Oracle Dynamic Monitoring System (DMS) profiler (see "Measuring TopLink Performance With the Oracle Dynamic Monitoring System (DMS)" on page 11-4)
2. Modify application components (see "Sources of Application Performance Problems")
3. 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 nonproduction environment before you deploy the application.

Sources of Application Performance Problems

For various parts of a TopLink enabled application, this section describes the performance problems most commonly encountered and provides suggestions for improving performance. Areas of an application where performance problems can occur include:

- General Performance Optimization
- Schema Optimization
- Mapping and Descriptor Optimization
- Session Optimization
- Cache Optimization
- Data Access Optimization
- Query Optimization
- Unit of Work Optimization
- Application Server and Database Optimization

Measuring TopLink Performance With the TopLink Profiler

The most important challenge to performance tuning is knowing what to optimize. To improve the performance of your application, identify the areas of your application that do not operate at peak efficiency. The TopLink performance profiler helps you identify performance problems by logging performance statistics for every executed query in a given session.

Note: You should also consider using general performance profilers such as JDeveloper or JProbe to analyze performance problems. These tools can provide more detail that may be required to properly diagnose a problem.

The TopLink performance Profiler logs the following information to the TopLink log file (for general information about TopLink logging, see "Logging" on page 72-7):

- 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 script (in milliseconds)
- SQL execute, the amount of time spent executing the SQL script (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 script before it is sent to the database (in milliseconds)

This section includes information on the following topics:

- Configuring the TopLink Performance Profiler
- Accessing the TopLink Profiler Results

Configuring the TopLink Performance Profiler

To enable the TopLink performance profiler, select the **TopLink** profiler option when configuring your session (see "Configuring a Performance Profiler" on page 74-10).

The TopLink performance profiler is an instance of `oracle.toplink.tools.profiler.PerformanceProfiler` class. It provides the following public API:

- `logProfile`–enables the profiler
- `dontLogProfile`–disables the profiler
- `logProfileSummary`–organizes the profiler log into a summary of all the individual operation profiles including operation statistics like the shortest time of all the operations that were profiled, the total time of all the operations, the number of objects returned by profiled queries, and the total time that was spent in each kind of operation that was profiled.
- `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 method`.

Accessing the TopLink Profiler Results

The simplest way to view TopLink profiler results is to read the TopLink log files with a text editor. For general information about TopLink logging, such as logging file location, see "Logging" on page 72-7.

Alternatively, you can use the graphical performance profiler that the TopLink Web client provides. For more information, refer to the Web client online Help and README files.

Example 11–1 shows an example of the TopLink profiler output.

Example 11–1 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 script.
- `prepare=495`: total time spent preparing the SQL script.
- `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.

Measuring TopLink Performance With the Oracle Dynamic Monitoring System (DMS)

Oracle DMS is a library that enables application and system developers to use a variety of DMS sensors to measure and export customized performance metrics for specific software components (called nouns).

TopLink includes DMS instrumentation in essential objects to provide efficient monitoring of run-time data in TopLink-enabled applications, including both J2EE and non-J2EE applications.

By enabling DMS profiling in a TopLink application (see "Configuring the Oracle DMS Profiler" on page 11-6), you can collect and easily access run-time data that can help you with application administration tasks and performance tuning.

Note: You should also consider using general performance profilers such as JDeveloper or JProbe to analyze performance problems. These tools can provide more detail that may be required to properly diagnose a problem.

Table 11–1 lists the many performance and status metrics TopLink provides through DMS.

Table 11–2 lists the various profiling levels you can use to adjust the level of profiling to the amount of monitoring information you require. Levels are listed in order of increasing system overhead.

You can easily access DMS data at run time using a management application that supports the Java Management Extensions (JMX) API (see "Accessing Oracle DMS Profiler Data Using JMX" on page 11-7) or using any Web browser and the DMS Spy servlet (see "Accessing Oracle DMS Profiler Data Using the DMS Spy Servlet" on page 11-7).

Table 11–1 TopLink DMS Metrics

DMS Noun Name[1]	Sensor Name	Level[2]	Description
`Cache`	`CacheHits`	`HEAVY`	Number of times an object looked up in the cache was found.
	`CacheMisses`	`HEAVY`	Number of times an object looked up in the cache was not found.
	`Caching`	`ALL`	Time spent adding, looking up, and removing objects in the cache.
`RCM`[3]	`ChangesNotProcessed`	`ALL`	Number of coordinated cache `ObjectChangeSets` discarded because the object was not found in the cache and was not merged.
	`ChangesProcessed`	`ALL`	Number of coordinated cache `ObjectChangeSets` for which an object was found in the cache and merged.
	`MessagesReceived`	`HEAVY`	Number of cache coordination messages received.
	`MessagesSent`	`HEAVY`	Number of cache coordination messages sent.
	`RemoteChangeSets`	`HEAVY`	Number of change sets received from remote machines and processed.
	`RCMStatus`	`HEAVY`	Cache coordination status: one of `not configured`, `started`, or `stopped`.
`Connection`	`ConnectCalls`	`HEAVY`	Total number of connect calls made.
	`ConnectionsInUse(`*`POOL_NAME`*`)`	`HEAVY`	Number of connections in use for the given connection pool.
	`DisconnectCalls`	`HEAVY`	Total number of disconnect calls made.
	`ConnectionManagement`	`ALL`	Time spent managing connections including connecting, reconnecting, and disconnecting from a data source.
`Query`	`DatabaseExecute`	`ALL`	Time spent in calls to the JDBC `Statement`. Includes time spent in calls to: `close`, `executeUpdate`, and `executeQuery`.
	`DeleteQueries`	`HEAVY`	Time spent executing delete queries, including time spent in JDBC calls.
	`ObjectBuilding`	`ALL`	Time spent building persistent objects from database rows.
	`QueryPreparation`	`ALL`	Time spent preparing a query. Does not include time spent doing SQL prepare.
	`ReadQueries`	`HEAVY`	Time spent executing read queries, including time spent in JDBC calls.
	`RowFetch`	`ALL`	Time spent fetching the JDBC result set from the database and building `DatabaseRecord` objects from the JDBC result set. Includes regular SQL calls and stored procedure calls.
	`SqlGeneration`	`ALL`	Time spent generating SQL. In the case of TopLink expressions, includes time spent converting `Expression` to SQL.
	`SqlPrepare`	`ALL`	Time spent in JDBC preparing the `Statement`. Includes the time spent in EIS creating an `Interaction` associated with a connection, and creating input and output `Record` objects.
	`UpdateQueries`	`HEAVY`	Time spent executing update queries, including time spent in JDBC calls.

Table 11–1 (Cont.) TopLink DMS Metrics

DMS Noun Name[1]	Sensor Name	Level[2]	Description
	`WriteQueries`	`HEAVY`	Time spent executing write queries, including time spent in JDBC calls.
`Session`	`ClientSession`	`HEAVY`	Number of `ClientSessions` currently logged in.
	`loginTime`	`NORMAL`	Time at which the session was logged in. Once the session is logged out, the sensor no longer appears.
	`SessionName`	`NORMAL`	Name of the session.
	`UnitOfWork`	`HEAVY`	Number of `UnitOfWork` objects acquired from this session.
`Transaction`	`DistributedMerge`	`ALL`	Time spent merging remote transaction changes into the local shared cache.
	`JtsAfterCompletion`	`ALL`	Time spent on JTS `afterCompletion` method.
	`JtsBeforeCompletion`	`ALL`	Time spent on JTS `beforeCompletion` method.
	`MergeTime`	`ALL`	Time spent merging changes into the shared cache.
	`OptimisticLocks`	`HEAVY`	Number of optimistic lock exceptions thrown.
	`Sequencing`	`ALL`	Time spent maintaining the sequence number mechanism and setting the sequence number on objects.
	`UnitOfWorkRegister`	`ALL`	Time spent in registering objects with the `UnitOfWork`.
	`UnitOfWorkCommits`	`ALL`	Time spent in the `UnitOfWork` commit process.
	`UnitOfWorkRollBacks`	`HEAVY`	Number of `UnitOfWork` commits that were rolled back.
`Miscellanous`	`DescriptorEvents`	`ALL`	Time spent by the `DescriptorEventManager` executing a descriptor event.
	`Logging`	`ALL`	Time spent logging TopLink activities.
	`SessionEvents`	`ALL`	Time spent by the SessionEvent manager executing a session event.

[1] DMS noun names are followed by the name of the session to which they belong. For example, `Cache(`*`SESSION_NAME`*`)`.

[2] See Table 11–2 for a description of each level setting.

[3] Cache Coordination

Table 11–2 DMS Metric Collection Levels

Level	Description
`NONE`	Disable collection of all DMS metrics.
`NORMAL`	Enable collection of TopLink DMS metrics. Adds very low overhead. This is the default setting.
`HEAVY`	Enable collection of basic TopLink DMS metrics. Adds about 1 percent overhead.
`ALL`	Enable all possible TopLink DMS metrics. Adds about 3 percent overhead.

Configuring the Oracle DMS Profiler

You configure DMS support in your TopLink application differently depending on the type of application it is:

- OC4J Applications

- Non-OC4J Applications

OC4J Applications

By default, DMS metric collection is enabled for TopLink CMP applications deployed to OC4J. For BMP or non-CMP applications deployed to OC4J, you must configure DMS metric collection (see "Configuring a Performance Profiler" on page 74-10).

TopLink EJB deployed in OC4J are subject to the DMS configuration specified by the OC4J command line-property `-Doracle.dms.sensors=<level>` where `<level>` is one of the values listed in Table 11–2.

Non-OC4J Applications

To enable DMS metric collection for TopLink applications deployed to an application sever other than OC4J do the following:

1. Ensure that the `dms.jar` file is in your application classpath.

   By default, the `dms.jar` file is located in `<ORACLE_HOME>\lib` directory.

2. Set system property `oracle.dms.sensors=<level>` where `<level>` is one of the values listed in Table 11–2.

3. To enable the DMS profiler, select the **DMS** profiler option when configuring your TopLink session (see "Configuring a Performance Profiler" on page 74-10).

Accessing Oracle DMS Profiler Data Using JMX

Using the Java Management Extensions (JMX) API, you can publish DMS profiler run-time data from a managed application (TopLink) to a JMX-compliant management application, by way of EJB-like MBean components.

When you configure your TopLink application to enable run-time services (see "Configuring a Performance Profiler" on page 74-10) and you deploy your application to OC4J, the TopLink runtime will deploy a JMX MBean so that a JMX management application can access the DMS profiler run-time data your application publishes.

For code examples that illustrates how to use DMS and JMX, see `http://www.oracle.com/technology/tech/java/oc4j/1003/how_to/jmx-enabled-demo.html`.

Accessing Oracle DMS Profiler Data Using the DMS Spy Servlet

Once your DMS enabled TopLink application is running, you can access the DMS data it is collecting.

The DMS Spy servlet is available in all Java processes that use DMS. It lets you monitor metrics for a single Java process from a Web browser.

To access DMS data directly using the DMS Spy servlet, do the following:

1. Ensure that the `dms.jar` file is in your application classpath.

   By default, the `dms.jar` file is located in `<ORACLE_HOME>\lib`.

2. Set the following system properties for the DMS enabled Java process you want to monitor:

```
oracle.dms.publisher.classes=oracle.dms.http.Httpd
oracle.dms.httpd.port.start=<port>
```

where <port> is the HTTP port on which DMS accepts requests (the default value is 46080).

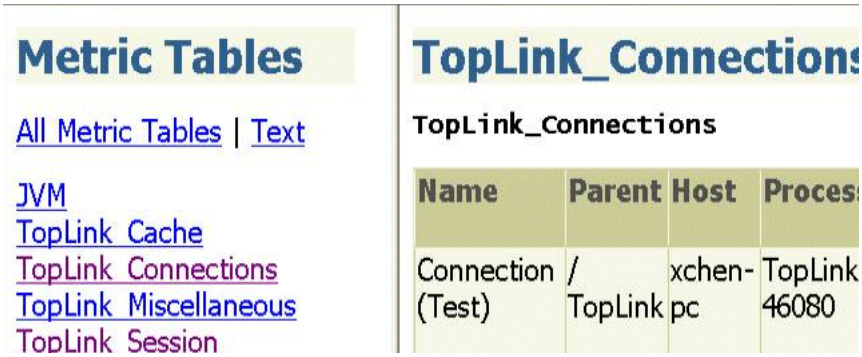
3. Apply the system property changes by restarting the Java process you want to monitor.
4. Using a Web browser, connect to the Java process and access the Spy servlet by entering the following URL:

   ```
   http://<host>:<port>/dms0/Spy
   ```

   where <host> is the host name of your Java process and <port> is the value specified by the `oracle.dms.httpd.port.start` system property.

   The Spy servlet displays all TopLink DMS-enabled objects appropriate for the current DMS level setting. Figure 11–1 shows an example of the DMS Spy servlet display.

Figure 11–1 DMS Spy Servlet Display



General Performance Optimization

Do not override TopLink default behavior unless your application absolutely requires it. Because 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.

Use TopLink Workbench rather than manual coding. TopLink Workbench is not only easy to use. The default configuration it exports to deployment XML (and the code it generates, if required) represents best practices optimized for most applications.

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, design the object model and database schema together. However, allow each model to be designed optimally: do not require a direct one-to-one correlation between the two.

This section includes the following schema optimization examples:

- Schema Case 1: Aggregation of Two Tables into One

- Schema Case 2: Splitting One Table Into Many
- Schema Case 3: Collapsed Hierarchy
- Schema Case 4: Choosing One out of Many

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 11–3 and Table 11–4 illustrate the table aggregation technique.

Table 11–3 Original Schema (Aggregation of Two Tables Case)

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 lets you read all information with a single operation, and doubles the update and insert speed, because only a single row in one table requires modifications.

Table 11–4 Optimized Schema (Aggregation of Two Tables Case)

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 11–5 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 11–5 Original Schema (Splitting One Table into Many Case)

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
	Employee	job	aggregate	JobClassification
	Employee	projects	many-to-many	Project

When you read a large volume of employee records 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 11–6.

Because you usually 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 percent.

Table 11–6 Optimized Schema (Splitting One Table into Many Case)

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			

Table 11–6 (Cont.) Optimized Schema (Splitting One Table into Many Case)

Elements	Details	Instance Variable	Mapping	Target
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 11–7 represents a system that assigns clients to a company's sales representatives. The managers also track the sales representatives that report to them.

Table 11–7 Original Schema (Collapsed Hierarchy Case)

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 11–8, you substantially reduce system complexity. You eliminate joins from the system, and simplify queries.

Table 11–8 Optimized Schema (Collapsed Hierarchy Case)

Elements	Details
Classes	Tables
Person	none
Employee	EMPLOYEE
SalesRep	EMPLOYEE
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 lets you access the object without instantiating the other objects in the collection.

Table 11–9 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 11–9 Original Schema (Choosing One out of Many Case)

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 11–9, 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 11–10 Optimized Schema (Choosing One out of Many Case)

Elements	Details	Instance Variable	Mapping	Target
Classes	Package, Location			
Tables	PACKAGE, LOCATION			
Relationships	Package	locations	one-to-many	Location
	Package	currentLocation	one-to-many	Location

Mapping and Descriptor Optimization

Always use indirection. It is not only critical in optimizing database access, but also allows TopLink to make several other optimizations including optimizing its cache access and unit of work processing. See "Configuring Indirection" on page 32-3.

Avoid using the existence checking option `checkCacheThenDatabase` on descriptors (see "Configuring Cache Existence Checking at the Descriptor Level" on page 25-43), unless required by the application. The default existence checking behavior offers better performance.

Avoid expensive initialization in the default constructor that TopLink uses to instantiate objects. Instead, use lazy initialization or use a TopLink instantiation policy (see "Configuring Instantiation Policy" on page 25-70) to configure the descriptor to use a different constructor.

Avoid using method access in your TopLink mappings (see "Configuring Method Accessing" on page 32-14), especially if you have expensive or potentially dangerous side-effect code in your get or set methods; use the default direct attribute access instead.

Session Optimization

Use a Server session in a server environment, not a `DatabaseSession`.

Use the TopLink client session instead of remote session. A client session is appropriate for most multiuser J2EE application server environments.

Do not pool client sessions. Pooling sessions offers no performance gains.

For more information, see "Server and Client Sessions" on page 72-14 and "OC4J Applications" on page 11-7.

Cache Optimization

Cache coordination (see "Understanding Cache Coordination" on page 87-10) is one way to allow multiple, possibly distributed, instances of a session to broadcast object changes among each other so that each session's cache can be kept up-to-date.

However, cache coordination is best suited to applications with specific characteristics (see "When to use Cache Coordination" on page 87-11). Before implementing cache coordination, tune the TopLink cache for each class using alternatives such as object identity type (see "Configuring Cache Type and Size at the Descriptor Level" on page 25-35), cache invalidation (see "Cache Invalidation" on page 87-8), or cache isolation (see "Cache Isolation" on page 87-9). Doing so lets you configure the optimal cache configuration for each type of class (see Table 11–11) and may eliminate the need for distributed cache coordination altogether.

Table 11–11 Identity Map and Cache Configuration by Class Type

Class Type	Identity Map Options	Cache Options
read-only	soft, hard, or full[1]	
read-mostly	soft or hard	cache invalidation or cache coordination
write-mostly	weak	cache invalidation

[1] If the number of instances is finite.

If you do use cache coordination, use JMS for cache coordination rather than RMI. JMS is more robust, easier to configure, and runs asynchronously. If you require synchronous cache coordination, use RMI.

You can configure a descriptor to control when the TopLink runtime will refresh the session cache when an instance of this object type is queried (see "Configuring Cache Refreshing" on page 25-27). Oracle does not recommend the use of **Always Refresh** or **Disable Cache Hits**.

Using **Always Refresh** may result in refreshing the cache on queries when not required or desired. As an alternative, consider configuring cache refresh on a query by query basis (see "Refreshing the Cache" on page 93-35).

Using **Disable Cache Hits** instructs TopLink to bypass the cache for object read queries based on primary key. This results in a database round trip every time an object read query based on primary key is executed on this object type, negating the performance advantage of the cache. When used in conjunction with **Always Refresh**, this option ensures that all queries go to the database. This can have a significant impact on performance. These options should only be used in specialized circumstances.

Data Access Optimization

Depending on the type of data source your application accesses, TopLink offers a variety of `Login` options that you can use to tune the performance of low level data reads and writes.

You can use several techniques to improve data access performance for your application. This section discusses some of the more common approaches, including:

- JDBC Driver Properties Optimization
- Data Format Optimization
- Batch Writing
- Parameterized SQL (Binding) and Prepared Statement Caching

JDBC Driver Properties Optimization

Consider the default behavior of the JDBC driver you choose for your application. Some JDBC driver options can affect data access performance.

Some important JDBC driver properties can be configured directly using TopLink Workbench or TopLink API (for example, see "JDBC Fetch Size" on page 11-18).

JDBC driver properties that are not supported directly by TopLink Workbench or TopLink API can still be configured as generic JDBC properties that TopLink passes to the JDBC driver.

For example, some JDBC drivers, such as Sybase JConnect, perform a database round trip to test whether or not a connection is closed: that is, calling the JDBC driver method `isClosed` results in a stored procedure call or SQL select. This database round-trip can cause a significant performance reduction. To avoid this, you can disable this behavior: for Sybase JConnect, you can set property name `CLOSED_TEST` to value `INTERNAL`.

For more information about configuring general JDBC driver properties from within your TopLink application, see "Configuring Properties" on page 82-4.

Data Format Optimization

By default, TopLink optimizes data access by accessing the data from JDBC in the format the application requires. For example, TopLink retrieves `long` data types from JDBC instead of having the driver return a `BigDecimal` that TopLink would then have to convert into a `long`.

Some older JDBC drivers do not perform data conversion correctly and conflict with this optimization. In this case, you can disable this optimization (see "Configuring Advanced Options" on page 83-11).

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.

When used without parameterized SQL, this is known as dynamic batch writing.

When used with parameterized SQL (see "Parameterized SQL (Binding) and Prepared Statement Caching" on page 11-15), this is known as parameterized batch writing. This allows a repeatedly executed statement, such as a group of inserts of the same type, to be executed as a single statement and a set of bind parameters. This can provide a large performance benefit as the database does not have to parse the batch.

When using batch writing, you can tune the maximum batch writing size using `setMaxBatchWritingSize` method of the Login interface. The meaning of this value depends on whether or not you are using parameterized SQL:

- If you are using parameterized SQL (you configure your `Login` by calling `Login` method `bindAllParameters`), the maximum batch writing size is the number of statements to batch (default: 100).
- If you are using dynamic SQL, the maximum batch writing size is the size of the SQL string buffer in characters (default: 32000).

By default, TopLink does not enable batch writing because not all databases and JDBC drivers support it. Oracle recommends that you enable batch writing for selected databases and JDBC drivers that support this option (see "Configuring JDBC Options" on page 83-9).

For a more detailed example of using batch writing to optimize write queries, see "Batch Writing and Parameterized SQL" on page 11-28.

Parameterized SQL (Binding) and Prepared Statement Caching

Using parameterized SQL, you can keep the overall length of an SQL query from exceeding the statement length limit that your JDBC driver or database server imposes.

Using parameterized SQL and prepared statement caching, you can improve performance by reducing the number of times the database SQL engine parses and prepares SQL for a frequently called query.

By default, TopLink does not enable parameterized SQL and prepared statement caching, because not all databases and JDBC drivers support it. Oracle recommends that you enable parameterized SQL and prepared statement caching for selected databases and JDBC drivers that support these options.

Not all JDBC drivers support all JDBC binding options (see "Configuring JDBC Options" on page 83-9). Selecting a combination of options may result in different

behavior from one driver to another. Before selecting JDBC options, consult your JDBC driver documentation. When choosing binding options, consider the following approach:

1. Try binding all parameters with all other binding options disabled.
2. If this fails to bind some large parameters, consider enabling one of the following options, depending on the parameter's data type and the binding options that your JDBC driver supports:
   a. To bind large `String` parameters, try enabling string binding.

      If large `String` parameters still fail to bind, consider adjusting the maximum `String` size. TopLink sets the maximum `String` size to 32000 characters by default.

   b. To bind large `Byte` array parameters, try enabling byte array binding.
3. If this fails to bind some large parameters, try enabling streams for binding.

   Typically, configuring string or byte array binding will invoke streams for binding. If not, explicitly configuring streams for binding may help.

For J2EE applications that use TopLink external connection pools, you must configure parameterized SQL in TopLink but you cannot configure prepared statement caching in TopLink. In this case, you must configure prepared statement caching in the application server connection pool. For example, in OC4J, if you configure your `data-source.xml` file with a managed `data-source` (where `connection-driver` is `oracle.jdbc.OracleDriver` and `class` is `oracle.j2ee.sql.DriverManagerDataSource`), you can configure a non-zero `num-cached-statements` that enables JDBC statement caching and defines the maximum number of statements cached.

For applications that use TopLink internal connection pools, you can configure parameterized SQL and prepared statement caching.

You can configure parameterized SQL and prepared statement caching at the following levels:

- project level–applies to all named queries (see "Configuring Named Query Parameterized SQL and Statement Caching at the Project Level" on page 20-7);
- descriptor level–applies on a per-named-query basis (see "Configuring Named Query Options" on page 25-22);
- database login level–applies to all queries (see "Configuring JDBC Options" on page 83-9) and provides additional parameter binding API to alleviate the limit imposed by some drivers on SQL statement size;
- query level–applies on a per-query basis (see "Using Parameterized SQL and Statement Caching in a DatabaseQuery" on page 94-17).

Query Optimization

TopLink provides an extensive query API for reading, writing, and updating data. This section describes ways of optimizing query performance in various circumstances.

Before optimizing queries, consider the optimization suggestions in "Data Access Optimization" on page 11-14.

This section includes information on the following:

- Parameterized SQL and Prepared Statement Caching
- Named Queries
- Batch and Join Reading
- Partial Object Queries and Fetch Groups
- JDBC Fetch Size
- Cursored Streams and Scrollable Cursors
- Read Optimization Examples
- Write Optimization Examples

Parameterized SQL and Prepared Statement Caching

These features lets you cache and reuse a query's pre-parsed database statement when the query is re-executed.

For more information, see"Parameterized SQL (Binding) and Prepared Statement Caching" on page 11-15.

Named Queries

Whenever 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. Using named queries also allows for the query to be prepared once, and for the SQL generation to be cached.

For more information, see "Named Queries" on page 93-16.

Batch and Join Reading

To optimize database read operations, 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, see the following:

- "Using Batch Reading" on page 94-10
- "Join Reading and Object-Level Read Queries" on page 93-12
- "Batch Writing" on page 11-15

Partial Object Queries and Fetch Groups

Partial object queries lets you retrieve partially populated objects from the database rather than complete objects.

For CMP applications, you can use fetch groups to accomplish the same performance optimization.

For more information about partial object reading, see "Partial Object Queries" on page 93-11.

For more information about fetch groups, see "Fetch Groups" on page 23-5.

JDBC Fetch Size

The JDBC fetch size gives the JDBC driver a hint as to the number of rows that should be fetched from the database when more rows are needed.

For large queries that return a large number of objects you can configure the row fetch size used in the query to improve performance by reducing the number database hits required to satisfy the selection criteria.

Most JDBC drivers default to a fetch size of 10, so if you are reading 1000 objects, increasing the fetch size to 256 can significantly reduce the time required to fetch the query's results. The optimal fetch size is not always obvious. Usually, a fetch size of one half or one quarter of the total expected result size is optimal. Note that if you are unsure of the result set size, incorrectly setting a fetch size too large or too small can decrease performance.

Set the query fetch size with `ReadQuery` method `setFetchSize` as Example 11–2 shows. Alternatively, you can use `ReadQuery` method `setMaxRows` to set the limit for the maximum number of rows that any `ResultSet` can contain.

Example 11–2 JDBC Driver Fetch Size

```
// Create query and set Employee as its reference class
ReadAllQuery query = new ReadAllQuery(Employee.class);
ExpressionBuilder builder = query.getExpressionBuilder();
query.setSelectionCriteria(builder.get("id").greaterThan(100));

// Set the JDBC fetch size
query.setFetchSize(50);

// Configure the query to return results as a ScrollableCursor
query.useScrollableCursor();

// Execute the query
ScrollableCursor cursor = (ScrollableCursor) session.executeQuery(query);

// Iterate over the results
while (cursor.hasNext()) {
    System.out.println(cursor.next().toString());
}
cursor.close();
```

In this example, when you execute the query, the JDBC driver retrieves the first 50 rows from the database (or all rows if less than 50 rows satisfy the selection criteria). As you iterate over the first 50 rows, each time you call `cursor.next()`, the JDBC driver returns a row from local memory–it does not need to retrieve the row from the database. When you try to access the fifty first row (assuming there are more than 50 rows that satisfy the selection criteria), the JDBC driver again goes to the database and retrieves another 50 rows. In this way, 100 rows are returned with only two database hits.

If you specify a value of zero (default; means the fetch size is not set), then the hint is ignored and the JDBC driver's default is used.

For more information about configuring JDBC driver properties from within your TopLink application, see "Configuring Properties" on page 82-4.

Cursored Streams and Scrollable Cursors

You can configure a query to retrieve data from the database using a cursored Java stream or scrollable cursor. This lets you view a result set in manageable increments rather than as a complete collection. This is useful when you have a large result set. You can further tune performance by configuring the JDBC driver fetch size used (see "JDBC Fetch Size" on page 11-18).

For more information about scrollable cursors, see "Handling Cursor and Stream Query Results" on page 96-15.

Read Optimization Examples

TopLink provides the read optimization features listed in Table 11–12.

This section includes the following read optimization examples:

- Reading Case 1: Displaying Names in a List
- Reading Case 2: Batch Reading Objects
- Reading Case 3: Using Complex Custom SQL Queries
- Reading Case 4: Using View Objects
- Reading Case 5: Inheritance Views

Table 11–12 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, registers only those objects that will change. For more information, see Chapter 97, "Understanding TopLink Transactions".
Indirection	Uses indirection objects to defer the loading and processing of relationships.	Provides a major performance benefit. It allows database access to be optimized and allows TopLink to internally make several optimizations in caching and unit of work.
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. For more information, see "Cache Type and Object Identity" on page 87-3.
Weak identity map	Offers client-side caching for objects.	Reduces database access and maintains a cache of all referenced objects. For more information, see "Cache Type and Object Identity" on page 87-3.
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. For more information, see "Using Batch Reading" on page 94-10 and "Using Join Reading" on page 94-11.

Table 11–12 (Cont.) Read Optimization Features

Feature	Function	Performance Technique
Partial object reading and fetch groups.	Allows reading of a subset of a result set of the object's attributes.	Reduces the amount of data read from the database. For more information, see "Partial Object Queries" on page 93-11. For more information about fetch groups, see "Fetch Groups" on page 23-5.
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 lets you compute complex results on the database, instead of reading the objects into the application and computing the results locally. For more information, see "Report Query" on page 93-15.
JDBC fetch size and ReadQuery maximum rows	Reduces the number of database hits required to return all the rows that satisfy selection criteria.	For more information, see "JDBC Fetch Size" on page 11-18.
Cursors	Lets you view a large result set in manageable increments rather than as a complete collection	For more information, see "Cursored Streams and Scrollable Cursors" on page 11-19
Inheritance views	Allows a view to be used for queries against an inheritance superclass that can read all of its subclasses in a single query, instead of multiple queries	For more information, see "Reading Case 5: Inheritance Views" on page 11-26.

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.

TopLink features that optimize these types of operations include:

- Partial Object Reading
- Report Query
- Fetch Groups (CMP projects only)

These features let you 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.

For more information about partial object queries, see "Partial Object Queries" on page 93-11.

In Example 11–3, the query builds complete employee objects, even though the list displays only employee last names. With no optimization, the query reads all the employee data.

Example 11–3 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 11–4 demonstrates the use of partial object reading. It reads only the last name and primary key for the employee data. This reduces the amount of data read from the database.

Example 11–4 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 names of the employees.
Since TopLink automatically includes the primary key of the object, the full
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");

// 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;
```

Report Query Report query lets you retrieve data from a set of objects and their related objects. Report query supports database reporting functions and features.

For more information, see "Report Query Results" on page 93-8.

Example 11–5 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 11–3, and avoids instantiating employee instances.

Example 11–5 Optimization Through Report Query

```
/* Read all the employees from the database, ask the user to choose one and return
it. The report query is used to read just the last name of the employees. Then 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 11–3) and the report query optimization in Example 11–5 appear to be minor, report queries offer a substantial performance improvement.

Fetch Groups Fetch groups, applicable only to CMP projects, are similar to partial object reading, but does allow caching of the objects read. For objects with many attributes or reference attributes to complex graphs (or both), you can define a fetch group that determines what attributes are returned when an object is read. Because TopLink will automatically execute additional queries when the `get` method is called for attributes not in the fetch group, ensure that the unfetched data is not required: refetching data can become a performance issue.

For more information about querying with fetch groups, see "Using Queries With Fetch Groups" on page 96-2.

Example 11–6 demonstrates the use of a static fetch group.

Example 11–6 Configuring a Query with a FetchGroup Using the FetchGroupManager

```
// Create static fetch group at the descriptor level
FetchGroup group = new FetchGroup("nameOnly");
group.addAttribute("firstName");
group.addAttribute("lastName");
descriptor.getFetchGroupManager().addFetchGroup(group);

// Use static fetch group at query level
ReadAllQuery query = new ReadAllQuery(Employee.class);
query.setFetchGroupName("nameOnly");

// Only Employee attributes firstName and lastName are fetched.
// If you call the Employee get method for any other attribute, TopLink executes
// another query to retrieve all unfetched attribute values. Thereafter, calling that get
// method will return the value directly from the object
```

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 typically 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 read operations required, use join and batch reading. Example 11–7 illustrates the unoptimized code required to retrieve a collection of objects with a one-to-one reference to another object. Example 11–8 and Example 11–9 illustrate the use of joins and batch reading to improve efficiency.

Example 11–7 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 11–8 Optimization Through Joining

```
/* Read all the employees, and collect their address' cities. Although the code
   is almost identical because joining optimization is used it takes only 1
   query */

// Read all the employees from the database, using joining.
// This requires 1 SQL call
ReadAllQuery query = new ReadAllQuery(Employee.class);
ExpressionBuilder builder = query.getExpressionBuilder();
query.setSelectionCriteria(builder.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 11–9 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 takes only
   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(Employee.class);
ExpressionBuilder builder = query.getExpressionBuilder();
query.setSelectionCriteria(bulder.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 11–8 and Example 11–9) accesses the database only twice, it is significantly faster than the approach illustrated in Example 11–7.

Joins offer a significant performance increase under most circumstances. Batch reading offers a further performance advantage in that it allows for delayed loading through value holders, and has much better performance where the target objects are shared.

For example, if employees in Example 11–7, Example 11–8, and Example 11–9 are 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 available for one-to-one, one-to-many, many-to-many, direct collection, direct map and aggregate collection mappings. Joining is only available for one-to-one and one-to-many mappings. Note that one-to-many joining will return a large amount of duplicate data and so is normally less efficient than batch reading.

WARNING: Allowing an unverified SQL string to be passed into methods (for example: `readAllObjects(Class class, String sql)` method) makes your application vulnerable to SQL injection attacks.

Reading Case 3: Using Complex Custom SQL Queries

TopLink provides a high-level query mechanism. However, if your application requires a complex query, a direct SQL or stored procedure call may be the best solution.

For more information about executing SQL calls, see "SQLCall" on page 93-17.

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 11–10 illustrates unoptimized code that reads information from several objects.

Example 11–10 No Optimization

```
/* Gather the information to report on an employee and return the summary of the
   information. In this situation, a hash table 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 11–11 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 as follows:

- Define the descriptor as usual, but specify the `tableName` as `EMPLOYEE_VIEW`.
- Map only the attributes required for the report. In the case of the `numberOfProjects` and associations, use a transformation mapping to retrieve the required data.

You can now query the report from the database in the same way as any other object enabled by TopLink.

Example 11–12 View the Report from Example 11–11

```
/* 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;
}
```

WARNING: Allowing an unverified SQL string to be passed into methods (for example: `readAllObjects(Class class, String sql)` and `readObject(Class class, String sql)` method) makes your application vulnerable to SQL injection attacks.

Reading Case 5: Inheritance Views

If you have an inheritance hierarchy that spans multiple tables and frequently query for the root class, consider defining an inheritance all-subclasses view. This allows a view to be used for queries against an inheritance superclass that can read all of its subclasses in a single query instead of multiple queries.

For more information about inheritance, see "Descriptors and Inheritance" on page 23-3.

For more information about querying on inheritance, see "Querying on an Inheritance Hierarchy" on page 96-4.

Write Optimization Examples

TopLink provides the write optimization features listed in Table 11–13.

This section includes the following write optimization examples:

- Writing Case: Batch Writes

Table 11–13 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. For more information, see Chapter 97, "Understanding TopLink Transactions"). Note: The Unit of Work supports marking classes as read-only (see "Configuring Read-Only Descriptors" on page 25-5 and "Declaring Read-Only Classes" on page 99-6). This avoids tracking of objects that do not change.
Batch writing	Lets you 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 (see "Cursors" on page 11-28 and "Batch Writing and Parameterized SQL" on page 11-28).
Parameterized SQL	Improves performance for frequently executed SQL statements (see "Parameterized SQL and Prepared Statement Caching" on page 11-17).
Sequence number preallocation	Dramatically improves insert performance. (see "Sequence Number Preallocation" on page 11-29).

Table 11–13 (Cont.) Write Optimization Features

Feature	Effect on Performance
Multiprocessing	Splitting a batch job across threads lets you synchronize reads from a cursored stream and use parallel Units of Work for performance improvements even on a single machine (see "Multiprocessing" on page 11-29).
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 (see "Configuring Existence Checking at the Project Level" on page 19-8 or "Configuring Cache Existence Checking at the Descriptor Level" on page 25-43 and "Using Registration and Existence Checking" on page 99-5).

Writing Case: 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 employee records 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 11–13 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.

TopLink offers several optimization features to improve the performance of this batch job.

To improve this operation, do the following:

- Use TopLink batch read operations and cursor support (see "Cursors" on page 11-28).
- Use batch writing or parameterized batch writing to write to the database (see "Batch Writing and Parameterized SQL" on page 11-28).

  If your database does not support batch writing, use parameterized SQL to implement the write query.
- Implement sequence number preallocation (see "Sequence Number Preallocation" on page 11-29).
- Implement multiprocessing (see "Multiprocessing" on page 11-29).

Cursors To optimize the query in Example 11–13, use a cursored stream to read the Employees from the source database. You can also employ a weak identity map instead of a hard or soft cache 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.

For more information, see "Cursored Streams and Scrollable Cursors" on page 11-19.

Batch Writing and Parameterized SQL Batch writing lets you 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 alone (dynamic batch writing) using `Login` method `useBatchWriting`. If you add batch writing to Example 11–13, you execute each batch of 100 employees as a single SQL execution. This reduces the number of SQL executions from 20,200 to 300.

You can also enable batch writing and parameterized SQL (parameterized batch writing) and prepared statement caching. Parameterized SQL avoids the prepare component of SQL execution. This improves write performance because it avoids the prepare cost of an SQL execution. For parameterized batch writing you would get one statement per Employee, and one for Address: this reduces the number of SQL executions from 20,200 to 400. Although this is more than dynamic batch writing alone, parameterized batch writing also avoids all parsing, so it is much more efficient overall.

Although parameterized SQL avoids the prepare component of SQL execution, it does not reduce the number of executions. Because of this, parameterized SQL alone may not offer as big of a gain as batch writing. However, if your database does not support batch writing, parameterized SQL will improve performance. If you add parameterized SQL in Example 11–13, you must still execute 20,200 SQL executions, but parameterized SQL reduces the number of SQL PREPAREs to 4.

For more information, see "Batch Writing" on page 11-15.

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 11–13 uses the select calls to acquire sequence numbers. You can substantially improve performance if you use sequence number preallocation.

In TopLink, you can configure the sequence preallocation size on the login object (the default size is 50). Example 11–13 uses a preallocation size of 1 to demonstrate this point. If you stream the data in batches of 100 as suggested in "Cursors", 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.

For more information about sequencing preallocation, see "Sequencing and Preallocation Size" on page 17-20.

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 11–14 illustrates the optimized code for this example. Note that it does not illustrate multiprocessing.

Example 11–14 Fully Optimized

```
/* Read each batch of employees, acquire a Unit of Work, and register them */
targetSession.getLogin().useBatchWriting();
targetSession.getLogin().setSequencePreallocationSize(200);
targetSession.getLogin().bindAllParameters();
targetSession.getLogin().cacheAllStatements();
targetSession.getLogin().setMaxBatchWritingSize(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(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
```

Unit of Work Optimization

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 99-5).
- 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 99-6).
- 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 99-6).
- Before using conforming queries, be sure that it is necessary. For alternatives, see "Using Conforming Queries and Descriptors" on page 99-8.

If your performance measurements show that you have a performance problem during unit of work commit, consider using object level or attribute level change tracking, depending on the type of objects involved and how they typically change. For more information, see "Unit of Work and Change Policy" on page 97-6.

Application Server and Database Optimization

Configuring your application server and database correctly can have a big impact on performance and scalabilty. Ensure that you correctly optimize these key components of your application in addition to your TopLink application and persistence.

For your application or J2EE server, ensure your memory, thread pool and connection pool sizes are sufficient for your server's expected load, and that your JVM has been configured optimally.

Ensure that your database has been configured correctly for optimal performance and its expected load.

12

Customization

This chapter includes the following sections:

- Overview
- Creating Custom Data Types
- Using Public Source

Overview

By design, TopLink can adapt to a variety of relational and nonrelational data sources.

To integrate TopLink with a data source that is not directly supported by the TopLink API, Oracle recommends that you use an EIS project (see "EIS Projects" on page 17-7) or a XML project (see "XML Projects" on page 17-9).

Using an EIS project, you can integrate your TopLink-enabled application with any nonrelational data source that supports a J2C) adapter and any supported EIS record type, including indexed, mapped, or XML. If no J2C adapter exists for your target data source, you can concentrate your integration efforts on creating an adapter. Simultaneously, you can build your application according to J2C specifications. Although this still requires custom development effort, it is more efficient than trying to extend TopLink classes and provides you with a J2C adapter that you can leverage in any other project (making it a better value).

Using an XML project, you can integrate your TopLink-enabled application with Web services or other XML-message based designs.

The remainder of this chapter describes other customization options provided by the TopLink API.

Creating Custom Data Types

TopLink provides support for all the most common Java data types. Table 12–1 lists the TopLink mapping extensions that you can use to support custom data types. You can also create your object converter to allow conversion between a data type and your own Java type.

Table 12–1 Mapping Extensions for Custom Data Types

Extension	Description
"Object Type Converter" on page 30-12	An extension of direct and direct collection mappings that lets you match a fixed number of data values to Java objects. Use this converter when the values in the schema differ from those in Java

Table 12–1 (Cont.) Mapping Extensions for Custom Data Types

Extension	Description
"Serialized Object Converter" on page 30-10	An extension of direct and direct collection mappings that lets you map serializable objects, such as multimedia data, to a binary format in a data source, such as a base64 element in an XML document or Binary Large Object (BLOB) field in a database
"Type Conversion Converter" on page 30-11	An extension of direct and direct collection mappings that lets you explicitly map a data source type to a Java type. For example, a `java.util.Date` in Java can be mapped to a `java.sql.Date` in the data source.
"Simple Type Translator" on page 30-12	An extension of direct and direct collection mappings that lets you automatically translate an XML element value to an appropriate Java type based on the element's `<type>` attribute as defined in your XML schema.

Using Public Source

The source code to most public classes is available in `<TOPLINK_HOME>\jlib\source.jar`.

This is provided for debugging purposes.

Part V

Troubleshooting a TopLink Application

This part describes how to troubleshoot a TopLink application at time of deployment and at run time. It contains the following chapters.

- Chapter 13, "TopLink Exception Reference"

  This chapter contains detailed information on each TopLink error code and message.

- Chapter 14, "TopLink Workbench Error Reference"

  This chapter describes common problems and their solutions when using TopLink Workbench.

- Chapter 15, "Troubleshooting Application Deployment"

  This chapter explains common problems and solutions when deploying TopLink applications.

13

TopLink Exception Reference

This chapter lists the TopLink exception error codes, information about the likely cause of the problem and a possible corrective action. Each exception 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 raised exception

This chapter contains information on the following exceptions:

- Descriptor Exceptions (1 – 201) on page 13-2
- Concurrency Exceptions (2001 – 2009) on page 13-25
- Conversion Exceptions (3001– 3008) on page 13-26
- Database Exceptions (4002 – 4018) on page 13-27
- Optimistic Lock Exceptions (5001 – 5009) on page 13-29
- Query Exceptions (6001 – 6129) on page 13-31
- Validation Exceptions (7001 – 7200), on page 13-43
- EJB QL Exceptions (8001 – 8010) on page 13-63
- Session Loader Exceptions (9000 - 9010) on page 13-64
- Communication Exceptions (12000 - 12003) on page 13-65
- EIS Exceptions (17007 – 17025), 90000, 91000 on page 13-66
- JMS Processing Exceptions (18001 - 18004) on page 13-69
- Default Mapping Exceptions (20001 - 20008) on page 13-69
- Discovery Exceptions (22001 - 22004) on page 13-70
- Remote Command Manager Exceptions (22101 - 22111) on page 13-71
- Transaction Exceptions (23001 - 23015) on page 13-73
- XML Marshal Exceptions (25001 – 25020) on page 13-81
- XML Conversion Exceptions (25501) on page 13-74
- Migration Utility Exceptions (26001 - 26020) on page 13-74
- XML Platform Exceptions (27001 – 27006, 27101 – 27103, 27201 – 27202) on page 13-83
- Entity Manager Setup Exceptions (28001 – 28007) on page 13-80
- EJB JAR XML Exceptions (72000 – 72023) on page 13-77

Descriptor Exceptions (1 – 201)

`DescriptorException` is a development exception that is raised when insufficient information is 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, as shown in Example 13–1.

Internal exception, mapping and descriptor appear only if 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 13–1 Descriptor Exception

```
EXCEPTION [TOPLINK – 75]: oracle.toplink.exceptions.DescriptorException
EXCEPTION DESCRIPTION: The reference class is not specified.
```

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 raised 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.

2: ATTRIBUTE_AND_MAPPING_WITHOUT_INDIRECTION_ MISMATCH

Cause: `attributeName` is declared as type `ValueHolderInterface`, but 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 raised on foreign reference mappings.

Action: If you do not want to use indirection on the mapping, change the attribute so it is not of type `ValueHolderInterface`. Otherwise, change the mapping associated with the attribute to use indirection.

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 the method `setAttributeName(String attributeName)`.

7: ATTRIBUTE_TYPE_NOT_VALID

Cause: When using Java 2, the specified `attributeName` is not defined as type `vector`, or a type that implements `the Map` or `Collection` interface. 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`.

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.

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 the method `setDirectFieldName(String fieldName)`.

10: FIELD_NAME_NOT_SET_IN_MAPPING

Cause: The field name is not set in the mapping. It is raised from direct to field mapping, array mapping, and structure mapping.

Action: Specify the field name by calling the method `setFieldName(String fieldName)`.

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 the method `setForeignKeyFieldName(String fieldName)`.

Action: Use the method `addForeignKeyFieldName(String sourceForeignKeyName, String targetPrimaryKeyFieldName)` to add multiple foreign key fields.

12: IDENTITY_MAP_NOT_SPECIFIED

Cause: The descriptor has been set not to use identity map, but the existence checking is set to be performed on identity map: the descriptor must use an identity map to use the **Check cache does exist** option.

Action: Either use identity map, or set the existence checking to some other option.

13: ILLEGAL_ACCESS_WHILE_GETTING_VALUE_THRU_ INSTANCE_ VARIABLE_ACCESSOR

Cause: 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 raised when TopLink tries to access the instance variable using the `java.lang.reflect` Java package. The error is a purely Java exception, and TopLink wraps only the reflection exception.

Action: Inspect the internal exception, and refer to the Java documentation.

14: ILLEGAL_ACCESS_WHILE_CLONING

Cause: 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 TopLink using Java reflection. The error is a purely Java exception, and TopLink wraps only the reflection exception.

Action: Inspect the internal exception, and refer to the Java documentation.

15: ILLEGAL_ACCESS_WHILE_CONSTRUCTOR_INSTANTIATION

Cause: The domain class does not define a public default constructor, which TopLink needs to create new instances of the domain class.

Action: Define a public default constructor or use a different instantiation policy.

16: ILLEGAL_ACCESS_WHILE_EVENT_EXECUTION

Cause: The descriptor callback method `eventMethodName` with `DescriptorEvent` as an argument is not accessible. This exception is raised when TopLink tries to access the event method using Java reflection. The error is a purely Java exception, and TopLink wraps only the reflection exception.

Action: Inspect the internal exception, and refer to the Java documentation.

17: ILLEGAL_ACCESS_WHILE_GETTING_VALUE_THRU_ METHOD_ ACCESSOR

Cause: Attempt to invoke inaccessible `methodName` on the object `objectName`. The underlying getter method to access an attribute in the domain object is not accessible. This exception is raised when TopLink tries to access an attribute through a method using the `java.lang.reflect` Java package. The error is a purely Java exception, and TopLink wraps only the reflection exception.

Action: Inspect the internal exception, and refer to the Java documentation.

18: ILLEGAL_ACCESS_WHILE_INSTANTIATING_METHOD_ BASED_PROXY

Cause: The method used by the transformation mapping using a value holder is invalid. This exception is raised when 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 refer to the Java documentation.

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 refer to the Java documentation.

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 TopLink wraps only the reflection exception.

Action: Inspect the internal exception, and refer to the Java documentation.

21: ILLEGAL_ACCESS_WHILE_INVOKING_ROW_EXTRACTION_ METHOD

Cause: TopLink is unable to extract data `row`, because TopLink cannot 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 TopLink wraps only the reflection exception.

Action: Inspect the internal exception, and refer to the Java documentation.

22: ILLEGAL_ACCESS_WHILE_METHOD_INSTANTIATION

Cause: 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 TopLink wraps only the reflection exception.

Action: Inspect the internal exception, and refer to the Java documentation.

23: ILLEGAL_ACCESS_WHILE_OBSOLETE_EVENT_EXECUTION

Cause: The descriptor callback method `eventMethodName` with `Session` as an argument is inaccessible. This exception is raised when TopLink tries to access the event method using Java reflection. The error is a purely Java exception, and TopLink wraps only the reflection exception.

Action: Inspect the internal exception, and refer to the Java documentation.

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 raised by Java, and TopLink wraps only the reflection exception.

Action: Inspect the internal exception, and refer to the Java documentation.

25: ILLEGAL_ACCESS_WHILE_SETTING_VALUE_THRU_ METHOD_ ACCESSOR

Cause: 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 raised by Java and TopLink wraps only the reflection exception.

Action: Inspect the internal exception, and refer to the Java documentation.

26: ILLEGAL_ARGUMENT_WHILE_GETTING_VALUE_ THRU_INSTANCE_ VARIABLE_ACCESSOR

Cause: 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 refer to the Java documentation.

27: ILLEGAL_ARGUMENT_WHILE_GETTING_VALUE_THRU_ METHOD_ ACCESSOR

Cause: 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 refer to the Java documentation.

28: ILLEGAL_ARGUMENT_WHILE_INSTANTIATING_METHOD_ BASED_ PROXY

Cause: The method that the method-based proxy uses in a transformation mapping is receiving invalid arguments when the valueholder is being instantiated. This exception is raised when TopLink tries to access the method using the `java.lang.reflect` Java package.

Action: Inspect the internal exception, and refer to the Java documentation.

29: ILLEGAL_ARGUMENT_WHILE_INVOKING_ATTRIBUTE_ METHOD

Cause: The number of actual and formal parameters differs, or an unwrapping conversion has failed. On transformation mapping, the method used to retrieve values from the database row while reading the transformation mapped attribute is getting an invalid argument.

Action: Inspect the internal exception, and refer to the Java documentation.

30: ILLEGAL_ARGUMENT_WHILE_INVOKING_FIELD_TO_ METHOD

Cause: The number of actual and formal parameters differs 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 invalid argument. The error is a purely Java exception, and TopLink wraps only the reflection exception.

Action: Inspect the internal exception, and refer to the Java documentation.

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 invalid argument. This exception is raised when TopLink tries to invoke the event method using Java reflection. The error is a purely Java exception, and TopLink wraps only the reflection exception.

Action: Inspect the internal exception, and refer to the Java documentation.

32: ILLEGAL_ARGUMENT_WHILE_SETTING_VALUE_THRU_ INSTANCE_ VARIABLE_ACCESSOR

Cause: An invalid value is being assigned to the attribute instance variable. 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.

TopLink assigns value by using Java reflection. Java raises the error and TopLink wraps only the reflection exception.

Action: Inspect the internal exception, and refer to the Java documentation.

33: ILLEGAL_ARGUMENT_WHILE_SETTING_VALUE_THRU _METHOD_ ACCESSOR

Cause: An illegal argument is being passed to the attribute's set accessor method. 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 raises the error and TopLink wraps only the reflection exception.

Action: Inspect the internal exception, and refer to the Java documentation.

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 raises the error and TopLink wraps only the reflection exception.

Action: Inspect the internal exception, and refer to the Java documentation.

35: INVALID_DATA_MODIFICATION_EVENT

Cause: Applications should never encounter this exception. This exception usually occurs at the time of developing TopLink, although in cases, where you write 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 raised.

Action: Contact Oracle Support Services.

36: INVALID_DATA_MODIFICATION_EVENT_CODE

Cause: An application should never encounter this exception. This exception usually occurs at the time of developing TopLink, although in cases, where you 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 raised.

Action: Contact Oracle Support Services.

37: INVALID_DESCRIPTOR_EVENT_CODE

Cause: An application should never encounter this exception. This exception usually occurs at the time of developing TopLink. The exception means that the descriptor event manager does not support the event code passed in the event.

Action: Contact Oracle Support Services.

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 that is raised by a Java reflection when TopLink instantiates the identity map class. TopLink wraps only the Java exception.

Action: Inspect the internal exception, and refer to the Java documentation.

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.

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.

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 TopLink can put and extract values for the primary key.

Action: Define a mapping.

43: MISSING_CLASS_FOR_INDICATOR_FIELD_VALUE

Cause: 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` are stored in a hash table, and later the class is extracted from the hash table 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.

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.

45: MISSING_MAPPING_FOR_FIELD

Cause: TopLink is missing a mapping for `field`; a mapping for the field is not specified.

Action: Define a mapping for the field.

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.

47: MULTIPLE_TABLE_PRIMARY_KEY_NOT_SPECIFIED

Cause: The multiple table primary key mapping is not 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 the method `addMultipleTablePrimaryKeyFieldName(String fieldNameInPrimaryTable, String fieldNameInSecondaryTable)` on the descriptor to set the primary keys.

48: MULTIPLE_WRITE_MAPPINGS_FOR_FIELD

Cause: Multiple writable mappings for the field `fieldName` are defined in the descriptor. Exactly one must be defined as writable; the others must be specified as read-only. When multiple write mappings are defined for the field, TopLink is unable to choose the appropriate mapping for writing the value of the field in the database row. Therefore, the exception is raised during the validation process of descriptors.

The most common cause of this problem is 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.

49: NO_ATTRIBUTE_TRANSFORMATION_METHOD

Cause: The attribute transformation method name in the transformation mapping is not specified. This method is invoked internally by TopLink to retrieve value to store in the domain object.

Action: Define a method and set the method name on the mapping by calling the method `setAttributeTransformation(String methodName)`.

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)`.

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 TopLink tries to build one from the foreign keys specified in the mapping.

Action: Specify the fields.

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 the method `setReferenceKeyFieldName(String fieldName)`.

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 the method `setRelationTableName(String tableName)`.

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.

55: NO_SUCH_METHOD_ON_FIND_OBSOLETE_METHOD

Cause: TopLink cannot find the descriptor callback method on the domain class. It must take a `Session` or a `DescriptorEvent` as its argument. TopLink tries to invoke the method using Java reflection. It is a Java exception and TopLink is wrapping only the main exception.

Action: Inspect the internal exception, and refer to the Java documentation.

56: NO_SUCH_METHOD_ON_INITIALIZING_ATTRIBUTE_METHOD

Cause: TopLink cannot find the method `methodName(Record databaseRow)` or methodName(Record databaseRow, Session `session`). 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 refer to the Java documentation.

57: NO_SUCH_METHOD_WHILE_CONSTRUCTOR_INSTANTIATION

Cause: The constructor is inaccessible to TopLink. 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 refer to the Java documentation.

58: NO_SUCH_METHOD_WHILE_CONVERTING_TO_METHOD

Cause: TopLink failed to find a method with signature `methodName()` or `methodName(oracle.toplink.sessions.Session)`. TopLink wraps the Java reflection exception that was raised by its attempt to create a `Method` type (`java.lang.reflect`) from the method names in the transformation mapping.

Action: Ensure that the method `methodName` is defined on the domain class that owns the attribute mapped by the transformation mapping.

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. TopLink wraps the Java reflection exception that is caused when it is creating a `Field` type (`java.lang.reflect.Field`) from the attribute name.

Action: Inspect the internal exception, and refer to the Java documentation.

60: NO_SUCH_METHOD_WHILE_INITIALIZING_ ATTRIBUTES_IN_METHOD_ ACCESSOR

Cause: The method `setMethodName` or `getMethodName` is not defined for the attribute in the domain class `javaClassName`, or it is not accessible. 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 refer to the Java documentation.

61: NO_SUCH_METHOD_WHILE_INITIALIZING_CLASS_EXTRACTION_ METHOD

Cause: The static class extraction method `methodName(Record databaseRow)` does not exist, or is not accessible. A Java reflection exception wrapped in a TopLink exception is raised when a class extraction method is being created from the method name in the inheritance policy.

Action: Inspect the internal exception, and refer to the Java documentation.

62: NO_SUCH_METHOD_WHILE_INITIALIZING_COPY_POLICY

Cause: The clone method `methodName()` does not exist, or is not accessible. A Java reflection exception wrapped in a TopLink exception is raised when a method to create clones is being created from the method name in the copy policy.

Action: Inspect the internal exception, and refer to the Java documentation.

63: NO_SUCH_METHOD_WHILE_INITIALIZING_INSTANTIATION_POLICY

Cause: The instance creation method `methodName()` does not exist, or is not accessible. A Java reflection exception wrapped in a TopLink exception is raised 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 refer to the Java documentation.

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.

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.

66: NOT_DESERIALIZABLE

Cause: Attempt to deserialize an object from the byte array that is read from the database. The exception is raised when the serialized object mapping is converting the byte array into an object.

Action: Inspect the internal exception, and refer to the Java documentation.

67: NOT_SERIALIZABLE

Cause: Attempt to serialize an object into a byte array. The exception is raised when a serialized object mapping is converting the object into a byte array.

Action: Inspect the internal exception, and refer to the Java documentation.

68: NULL_FOR_NON_NULL_AGGREGATE

Cause: The value of the 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 the mapping method `allowNull`. Provide parameters only if you are making a distinction between `foo()` and `foo(integer)`.

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 raised only on some JVMs.

Action: Inspect the internal exception, and refer to the Java documentation.

70: NULL_POINTER_WHILE_GETTING_VALUE_THRU_METHOD_ACCESSOR

Cause: The getter method is invoked to get the value of an attribute through Java reflection. This exception is raised only on some JVM.

Action: Inspect the internal exception, and refer to the Java documentation.

71: NULL_POINTER_WHILE_SETTING_VALUE_THRU_INSTANCE_VARIABLE_ACCESSOR

Cause: A `NullPointerException` has been raised 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 raised only on some JVMs.

Action: Inspect the internal exception, and refer to the Java documentation.

72: NULL_POINTER_WHILE_SETTING_VALUE_THRU_METHOD_ACCESSOR

Cause: A NullPointerException has been raised 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 raised only on some JVMs.

Action: Inspect the internal exception, and refer to the Java documentation.

73: PARENT_DESCRIPTOR_NOT_SPECIFIED

Cause: TopLink is unable to find the descriptor for the parent class. The descriptor of a subclass has no parent descriptor.

Action: The method `setParentClass(Class parentClass)` must be called on the subclass descriptor.

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)`.

75: REFERENCE_CLASS_NOT_SPECIFIED

Cause: The reference class is not specified in the foreign reference mapping.

Action: Set the reference class by calling the method `setReferenceClass(Class aClass)`.

77: REFERENCE_DESCRIPTOR_IS_NOT_AGGREGATE

Cause: The referenced descriptor for class `className` is not set to an aggregate descriptor. An aggregate mapping should always reference a descriptor that is aggregate.

Action: Call the method `descriptorIsAggregate` on the referenced descriptor.

78: REFERENCE_KEY_FIELD_NOT_PROPERLY_SPECIFIED

Cause: The table for the reference field is not 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.

79: REFERENCE_TABLE_NOT_SPECIFIED

Cause: The reference table name in the direct collection mapping is not specified.

Action: Use the method `setReferenceTableName(String tableName)` on the mapping to set the table name.

80: RELATION_KEY_FIELD_NOT_PROPERLY_SPECIFIED

Cause: The table for the relation key field is not the relation table. If the source and target relation fields names that are 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.

81: RETURN_TYPE_IN_GET_ATTRIBUTE_ACCESSOR

Cause: The method `attributeMethodName` that is specified in the transformation mapping does not have a return type set in the attribute, as it should because this method is used to extract value from the database row.

Action: Verify the method and make appropriate changes.

82: SECURITY_ON_FIND_METHOD

Cause: The descriptor callback method with `DescriptorEvent` as an argument is not accessible. Java raises 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 refer to the Java documentation.

83: SECURITY_ON_FIND_OBSOLETE_METHOD

Cause: The descriptor callback method with `Session` as an argument is not accessible. Java raises 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 refer to the Java documentation.

84: SECURITY_ON_INITIALIZING_ATTRIBUTE_METHOD

Cause: Access to the method `methodName(Record databaseRow)` or `methodName(Record databaseRow,` Session session) has been denied. Java raises 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 refer to the Java documentation.

85: SECURITY_WHILE_CONVERTING_TO_METHOD

Cause: TopLink failed to find a method with signature `methodName()` or `methodName(oracle.toplink.sessions.Session)`. Java raises 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 refer to the Java documentation.

86: SECURITY_WHILE_INITIALIZING_ATTRIBUTES_IN_INSTANCE_ VARIABLE_ACCESSOR

Cause: Access to the instance variable `attributeName` in the class `javaClassName` is denied. Java raises a security exception when creating a `Field` type from the given attribute name using Java reflection.

Action: Inspect the internal exception, and refer to the Java documentation.

87: SECURITY_WHILE_INITIALIZING_ATTRIBUTES_IN_METHOD_ ACCESSOR

Cause: The methods `setMethodName` and `getMethodName` in the object `javaClassName` are inaccessible. Java raises a security exception when creating a `Method` type from the given attribute accessor method name using Java reflection.

Action: Inspect the internal exception, and refer to the Java documentation.

88: SECURITY_WHILE_INITIALIZING_CLASS_ EXTRACTION_METHOD

Cause: The static class extraction method `methodName(Record databaseRow)` is not accessible. Java raises 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 the inheritance policy.

Action: Inspect the internal exception, and refer to the Java documentation.

89: SECURITY_WHILE_INITIALIZING_COPY_POLICY

Cause: The clone method `methodName()` is inaccessible. Using `ClassDescriptor` method `useCloneCopyPolicy` (`java.lang.String methodName`), you can specify that the creation of clones within a unit of work is done by sending the `methodName` method to the original object. If the clone method `methodName` with no arguments is inaccessible (your application does not have sufficient privileges to call the method), Java raises a security exception when reflectively accessing the method with the given method name using the `java.lang.reflect` Java package.

Action: Inspect the internal exception, and refer to the Java documentation.

90: SECURITY_WHILE_INITIALIZING_INSTANTIATION_POLICY

Cause: The instance creation method `methodName()` is inaccessible. Using any of the `ClassDescriptor` methods `useFactoryInstantiationPolicy` (`java.lang.Class factoryClass, java.lang.String methodName`), `useFactoryInstantiationPolicy` (`java.lang.Class factoryClass`, `java.lang.String methodName`, `java.lang.String factoryMethodName`), `useFactoryInstantiationPolicy` (`java.lang.Object factory`, `java.lang.String methodName`), or `useMethodInstantiationPolicy(java.lang.String staticMethodName`), you can specify how new instances are created. If any of the methods or factory methods are inaccessible (your application does not have sufficient privileges to call the method), Java raises a security exception when

reflectively accessing the method with the given method name using the `java.lang.reflect` Java package.

Action: Inspect the internal exception, and refer to the Java documentation.

91: SEQUENCE_NUMBER_PROPERTY_NOT_SPECIFIED

Cause: Either the `sequenceNumberName` or the `sequenceNumberFieldName` property is not set. To use sequence-generated IDs, both the sequenceNumberName and sequenceNumberFieldName properties must be set for the descriptor.

Action: To use sequence-generated IDs, set both the sequence number name and field name properties.

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.

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.

94: TABLE_NOT_SPECIFIED

Cause: No table is specified in the descriptor. The descriptor must have a table name defined.

Action: Call the method `addTableName(String tableName)` or `setTableName(String tableName)` to set the tables on the descriptor.

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.

97: TARGET_INVOCATION_WHILE_CLONING

Cause: TopLink has encountered a problem in cloning the object `domainObject` clone method. The `methodName` triggered an exception. Java raises 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 refer to the Java documentation.

98: TARGET_INVOCATION_WHILE_EVENT_EXECUTION

Cause: A descriptor callback method `eventMethodName(DescriptorEvent event)` is not accessible. The exception occurs when the descriptor event method is invoked using Java reflection.

Action: Inspect the internal exception, and refer to the Java documentation.

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 refer to the Java documentation.

100: TARGET_INVOCATION_WHILE_INSTANTIATING_METHOD_BASED_PROXY

Cause: A method has raised an exception. Java raises this exception while instantiating a method based proxy and instantiating transformation mapping.

Action: Inspect the internal exception, and refer to the Java documentation.

101: TARGET_INVOCATION_WHILE_INVOKING_ATTRIBUTE_METHOD

Cause: The underlying method raises 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 refer to the Java documentation.

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 refer to the Java documentation.

103: TARGET_INVOCATION_WHILE_INVOKING_ROW_EXTRACTION_METHOD

Cause: TopLink encountered a problem extracting the class type from row `rowName` while invoking a class extraction method.

Action: Inspect the internal exception, and refer to the Java documentation.

104: TARGET_INVOCATION_WHILE_METHOD_INSTANTIATION

Cause: TopLink is unable to create a new instance. The creation method `methodName` caused an exception.

Action: Inspect the internal exception, and refer to the Java documentation.

105: TARGET_INVOCATION_WHILE_OBSOLETE_EVENT_EXECUTION

Cause: The underlying descriptor callback method `eventMethodName(Session session)` raises 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 refer to the Java documentation.

106: TARGET_INVOCATION_WHILE_SETTING_VALUE_THRU_METHOD_ACESSOR

Cause: The method `setMethodName` on the object raises 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 refer to the Java documentation.

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.

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.

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.

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.

112: ONLY_ONE_TABLE_CAN_BE_ADDED_WITH_THIS_METHOD

Cause: Attempt to enter more than one table through this method.

Action: Use the method `addTableName(String tableName)` to add multiple tables to the descriptor.

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 refer to the Java documentation.

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 refer to the Java documentation.

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.

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.

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.

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.

120: CHILD_DOES_NOT_DEFINE_ABSTRACT_QUERY_KEY

Cause: The query key `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 class that implements the interface descriptor by the abstract query key in the interface descriptor.

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: Contact Oracle Support Services.

125: VALUE_HOLDER_INSTANTIATION_MISMATCH

Cause: The mapping for the attribute `getAttributeName()` uses indirection and must be initialized to a new value holder.

Action: Ensure that the mapping uses indirection and that the attribute is initialized to a new value holder.

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.

127: RETURN_AND_MAPPING_WITH_INDIRECTION_MISMATCH

Cause: The return type of the method used to get the attribute `getAttributeName()` of a mapping is not declared as type `ValueHolderInterface`, but the mapping is using indirection.

Action: Verify that the method used to get the attribute named `getAttributeName()` of `DatabaseMapping` returns a value holder, or change the mapping so it does not use indirection.

128: RETURN_AND_MAPPING_WITHOUT_INDIRECTION_MISMATCH

Cause: The return type of the method used to get the attribute `getAttributeName()` of DatabaseMapping 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 value holder.

129: PARAMETER_AND_MAPPING_WITH_INDIRECTION_MISMATCH

Cause: The return type of the method used to set the attribute `getAttributeName()` of DatabaseMapping 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 so it does not use indirection.

130: PARAMETER_AND_MAPPING_WITHOUT_INDIRECTION_MISMATCH

Cause: The return type of the method used to set the attribute `getAttributeName()` of DatabaseMapping 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 value holder.

131: GET_METHOD_RETURN_TYPE_NOT_VALID

Cause: The return type of the method used to get the attribute `getAttributeName()` of DatabaseMapping is not declared as type `Vector` (or a type that implements the `Map` or `Collection` interface if using Java 2).

Action: Declare the return type of the method used to get the attribute `getAttributeName()` of DatabaseMapping as type `Vector` (or a type that implements the map or collection interface if using Java 2).

133: SET_METHOD_PARAMETER_TYPE_NOT_VALID

Cause: The parameter type of the method used to set the attribute `getAttributeName()` of DatabaseMapping is not declared as type `Vector` (or a type that implements the map or collection interface, if using Java 2).

Action: Declare the parameter type of the method used to set the attribute `getAttributeName()` of DatabaseMapping as type `Vector` (or a type that implements the `Map` or `Collection` interface, if using Java 2).

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.

138: ATTRIBUTE_AND_MAPPING_WITH_TRANSPARENT_INDIRECTION_MISMATCH

Cause: The attribute `getAttributeName()` of DatabaseMapping is not declared as a supertype of `validTypeName`, but the mapping is using transparent indirection.

Action: Verify the attribute's type and the mapping setup.

139: RETURN_AND_MAPPING_WITH_TRANSPARENT_INDIRECTION_MISMATCH

Cause: The return type of the method used to get the attribute `getAttributeName()` of DatabaseMapping 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.

140: PARAMETER_AND_MAPPING_WITH_TRANSPARENT_INDIRECTION_MISMATCH

Cause: The parameter type of the method used to set the attribute `getAttributeName()` of DatabaseMapping is not declared as a supertype of `validTypeName`, but the mapping is using transparent indirection.

Action: Verify the attribute's type and the mapping setup.

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.

142: TABLE_IS_NOT_PRESENT_IN_DATABASE

Cause: The table whose name is provided by the `Descriptor` method `getTableName` is not present in the database.

Action: Verify the table name for the descriptor.

143: MULTIPLE_TABLE_INSERT_ORDER_MISMATCH

Cause: The multiple table insert order vector specified the `Descriptor` method `getMultipleTableInsertOrder` has fewer or more tables than are specified in the `Descriptor` method `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.

144: INVALID_USE_OF_TRANSPARENT_INDIRECTION

Cause: Transparent indirection is being used with a mapping other than a `CollectionMapping`.

Action: Verify the mapping. It must be a collection mapping.

145: MISSING_INDIRECT_CONTAINER_CONSTRUCTOR

Cause: The indirect container class does not implement the constructor.

Action: Implement the constructor for the container.

146: COULD_NOT_INSTANTIATE_INDIRECT_CONTAINER_CLASS

Cause: TopLink is unable to instantiate the indirect container class using the constructor.

Action: Validate the constructor for the indirect container class.

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 with JDK 1.3.1 or later.

Action: Validate the container policy being used.

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.

149: INVALID_USE_OF_NO_INDIRECTION

Cause: `NoIndirectionPolicy` object calls this method.

Action: Contact Oracle Support Services.

150: INDIRECT_CONTAINER_INSTANTIATION_MISMATCH

Cause: The mapping for the attribute `getAttributeName()` of DatabaseMapping uses transparent indirection and must be initialized to an appropriate container.

Action: Initialize the mapping to an appropriate container.

151: INVALID_MAPPING_OPERATION

Cause: An invalid mapping operation has been used.

Action: See the documentation for valid mapping operations.

152: INVALID_INDIRECTION_POLICY_OPERATION

Cause: An invalid indirection policy operation has been used.

Action: See the documentation for valid indirection policy operations.

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.

154: INVALID_INDIRECTION_CONTAINER_CLASS

Cause: An invalid indirection container class has been used.

Action: Verify the container class.

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.

156: STRUCTURE_NAME_NOT_SET_IN_MAPPING

Cause: The structure name is not set.

Action: Set the structure name appropriately.

157: NORMAL_DESCRIPTORS_DO_NOT_SUPPORT_ NON_RELATIONAL_ EXTENSIONS

Cause: Relational descriptors do not support nonrelational extensions.

Action: Contact Oracle Support Services.

158: PARENT_CLASS_IS_SELF

Cause: The descriptor's parent class has been set to itself.

Action: Contact Oracle Support Services.

159: PROXY_INDIRECTION_NOT_AVAILABLE

Cause: An attempt to use proxy indirection has been made, but JDK 1.3.1 or later is not being used.

Action: Use JDK 1.3.1 or later.

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.

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 getter method is correct for the indirection policy.

162: INVALID_SET_PARAMETER_TYPE_FOR_PROXY_ INDIRECTION

Cause: The parameter for the setter method is incorrect for the indirection type.

Action: Ensure that the parameter type of the setter method is correct for the indirection policy.

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.

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`.

165: ERROR_OCCURRED_IN_AMENDMENT_METHOD

Cause: The specified amendment method threw an exception.

Action: Examine the returned exception for further information.

166: VARIABLE_ONE_TO_ONE_MAPPING_IS_NOT_DEFINED

Cause: There is no mapping for the attribute.

Action: Validate the mapping and attribute.

167: NO_CONSTRUCTOR_INDIRECT_COLLECTION_CLASS

Cause: A valid constructor was not found for the indirection container class.

Action: Add a default constructor or a constructor with a `ValueHolderInterface` in the container class.

168: TARGET_INVOCATION_WHILE_CONSTRUCTOR_INSTANTIATION

Cause: The constructor is missing.

Action: Create the required constructor.

169: TARGET_INVOCATION_WHILE_CONSTRUCTOR_INSTANTIATION_OF_FACTORY

Cause: The constructor is missing.

Action: Create the required constructor.

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.

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.

172: NO_SUCH_METHOD_WHILE_CONSTRUCTOR_INSTANTIATION_OF_FACTORY

Cause: A method call from inside the constructor is invalid because this method does not exist.

Action: Ensure that the factory has a default constructor for the called method.

173: NULL_POINTER_WHILE_CONSTRUCTOR_INSTANTIATION_OF_FACTORY

Cause: A method on a `null` object was called from inside a constructor. The factory constructor was inaccessible.

Action: Examine the internal exception and take the appropriate action.

174: ILLEGAL_ACCESS_WHILE_METHOD_INSTANTIATION_OF_FACTORY

Cause: A method was called on an object from inside a factory instantiation, and Java has determined this method to be invalid.

Action: Determine why the method is invalid, and replace the method with a valid one.

175: TARGET_INVOCATION_WHILE_METHOD_INSTANTIATION_OF_FACTORY

Cause: A problem was encountered creating factory using creation method. The creation method triggered an exception.

Action: Examine the exception and take the corresponding action.

176: NULL_POINTER_WHILE_METHOD_INSTANTIATION_OF_FACTORY

Cause: A method called to instantiate a factory threw a `NullPointerException`. The creation method is not accessible.

Action: Do not use that method to instantiate a factory.

177: NO_MAPPING_FOR_ATTRIBUTENAME

Cause: Mapping is missing for the attribute `attributeName`.

Action: The attribute must be mapped.

178: NO_MAPPING_FOR_ATTRIBUTENAME_IN_ENTITY_BEAN

Cause: Cannot find mapping for an attribute `attributeName` in an entity bean `beanName`.

Action: Map the attribute.

179: UNSUPPORTED_TYPE_FOR_BIDIRECTIONAL_RELATIONSHIP_MAINTENANCE

Cause: The attribute uses bidirectional relationship maintenance, but has `ContainerPolicy`, which does not support it.

Action: The attribute must be mapped with a different collection type.

180: REFERENCE_DESCRIPTOR_CANNOT_BE_AGGREGATE

Cause: No string is defined in the `DescriptorExceptionResource.java` file.

Action: The reference descriptor for a relationship mapping (except `AggregateCollectionMapping`) cannot be aggregate. You must change the reference descriptor to non-aggregate.

181: ATTRIBUTE_TRANSFORMER_CLASS_NOT_FOUND

Cause: The `AttributeTransformer` class cannot be found.

Action: Ensure that the `AttributeTransformer` class exists and is on the classpath.

182: FIELD_TRANSFORMER_CLASS_NOT_FOUND

Cause: The `FieldTransformer` class cannot be found.

Action: Ensure that the `FieldTransformer` class exists and is on the classpath.

183: ATTRIBUTE_TRANSFORMER_CLASS_INVALID

Cause: Invalid use of a class `className` as an `AttributeTransformer`.

Action: Examine the internal exception stack trace and make the appropriate correction.

184: FIELD_TRANSFORMER_CLASS_INVALID

Cause: Invalid use of a class `className` as a `FieldTransformer`.

Action: Do not use the class as a `FieldTransformer`.

185: RETURNING_POLICY_FIELD_TYPE_CONFLICT

Cause: `ReturningPolicy` contains field with two different types.

Action: The field was added to `ReturningPolicy` twice with different types. The field must be added to `ReturningPolicy` once. You must remove excessive `addFieldForInsert` and/or `addInsertField` calls.

186: RETURNING_POLICY_FIELD_INSERT_CONFLICT

Cause: `ReturningPolicy` contains field that has been added twice using `addInsertField` and `addInsertFieldReturnOnly`.

Action: A field must be added to `ReturningPolicy` only once. You must remove excessive `addField` calls.

187: RETURNING_POLICY_AND_DESCRIPTOR_FIELD_TYPE_CONFLICT

Cause: `ReturningPolicy` contains field with type `Type`, but the same field in descriptor has type `differentType`.

Action: Specify field type in `addField` method only in the event that it cannot be obtained from the descriptor.

188: RETURNING_POLICY_UNMAPPED_FIELD_TYPE_NOT_SET

Cause: `ReturningPolicy` contains unmapped field `fieldName` that requires type.

Action: You must specify field type in the `addField` method.

189: RETURNING_POLICY_MAPPED_FIELD_TYPE_NOT_SET

Cause: `ReturningPolicy` contains mapped field `fieldName` that requires type.

Action: You must specify field type in the `addField` method.

190: RETURNING_POLICY_MAPPING_NOT_SUPPORTED

Cause: `ReturningPolicy` contains a field that is mapped with unsupported mapping.

Action: You cannot use `ReturningPolicy` with this field. Do not add it to `ReturningPolicy`.

191: RETURNING_POLICY_FIELD_NOT_SUPPORTED

Cause: `ReturningPolicy` contains a field `fieldName` that is not supported. Field is either sequence field, class type indicator, or used for locking.

Action: You cannot use `ReturningPolicy` with this field. Do not add it to `ReturningPolicy`.

192: CUSTOM_QUERY_AND_RETURNING_POLICY_CONFLICT

Cause: `ReturningPolicy` contains a field fieldName, but custom query `queryName` does not output it.

Action: Update the custom query so that it outputs a value for this field.

193: NO_CUSTOM_QUERY_FOR_RETURNING_POLICY

Cause: There is no custom query set, but `ReturningPolicy` contains one or more fields to be returned and doesn't support generating call with return.

Action: Specify a custom `InsertObjectQuery` or `UpdateObjectQuery` through `DescriptorQueryManager` `setInsertQuery`, `setInsertCall`, `setUpdateQuery`, or `setUpdateCall` methods that outputs values for fields added to `ReturningPolicy`.

194: CLASS_EXTRACTION_METHOD_MUST_BE_STATIC

Cause: The class extraction method must be a static method on the descriptor's class.

Action: Make the class extraction method a static method on the descriptor's class.

195: ISOLATED_DESCRIPTOR_REFERENCED_BY_SHARED_DESCRIPTOR

Cause: The shared class must not reference the isolated class.

Action: Ensure that the shared class does not reference the isolated class.

196: UPDATE_ALL_FIELDS_NOT_SET

Cause: `updateAllFields` flag has not been set or has been set to `false`. When using `setForceUpdate(true)` method of `CMPPolicy` you must also call `setUpdateAllFields(true)` method of `CMPPolicy`.

Action: Ensure that `updateAllFields` is set to `true` if `forceUpdate` is `true`.

197: INVALID_MAPPING_TYPE

Cause: A mapping of an inappropriate type has been set for this descriptor.

Action: The mapping type has to map the descriptor type, e.g. relational mapping for relational descriptor, EIS mapping for EIS descriptor, and XML mapping for XML descriptor.

198: NEED_TO_IMPLEMENT_CHANGETRACKER

Cause: The object does not implement the `ChangeTracker` interface.

Action: Ensure that the object implements `ChangeTrackerInterface` in order to use `ObjectChangeTrackingPolicy` or `AttributeChangeTrackingPolicy`.

199: NEED_TO_IMPLEMENT_FETCHGROUPTRACKER

Cause: The domain class does not implement the `FetchGroupTracker` interface.

Action: Ensure that the domain class implements the `FetchGroupTracker` interface in order to use the fetch group.

200: ATTEMPT_TO_REGISTER_DEAD_INDIRECTION

Cause: Attempt to register an object with dead indirection as a new object. Possibly, the object was deleted or removed from the cache during a merge of a serialized clone or did not exist in the cache at the time of the merge. This is a concurrency violation.

Action: Ensure that the object exists in the cache before attempting to merge a deserialized version into the cache. Consider a locking strategy. For more information, see "Merging Changes in Working Copy Clones" on page 99-13 and "Indirection, Serialization, and Detachment" on page 30-9.

201: UNIT_OF_WORK_ISOLATED_OBJECTS_ACCESSED_IN_SESSION

Cause: Attempt to built and object in the session cache, but the descriptor is marked as isolated in the unit of work.

Action: Ensure that the descriptor is never accessed outside of a unit of work.

For more information, see Part VIII, "Descriptors".

Concurrency Exceptions (2001 – 2009)

`ConcurrencyException` is a development exception that is raised when a Java concurrency violation occurs. Only when a running thread is interrupted, causing the JVM to throw an `InterruptedException`, is an internal exception information displayed with the error message, as shown in Example 13–2.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
INTERNAL EXCEPTION: Message
```

Example 13–2 Concurrency Exception

```
EXCEPTION [TOPLINK – 2004]: oracle.toplink.exceptions.ConcurrencyException
EXCEPTION DESCRIPTION: Signal attempted before wait on concurrency manager.
This usually means that an attempt was made to commit or roll back a transaction
before being started, or rolled back twice.
```

2001: WAIT_WAS_INTERRUPTED

Cause: In a multi threaded environment, one of the waiting threads was interrupted.

Action: Such exceptions are application-dependent.

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 application-dependent.

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 application-dependent.

2004: SIGNAL_ATTEMPTED_BEFORE_WAIT

Cause: A signal was attempted before a wait on concurrency manager. This usually means that an attempt was made to commit or roll back a transaction before it was started, or to rollback a transaction twice.

Action: Verify transactions in the application.

2005: WAIT_FAILURE_SEQ_DATABASE_SESSION

Cause: An `InterruptedException` was raised while `DatabaseSession` sequencing waited for a separate connection to become available.

Action: Examine concurrency issues involving object creation with your `DatabaseSession`.

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`.

2007: MAX_TRIES_EXCEDED_FOR_LOCK_ON_CLONE

Cause: Maximum number of attempts to lock object was exceed resulting in a failure to clone the object.

Action: Ensure that the number of attempts is within the limit.

2008: MAX_TRIES_EXCEDED_FOR_LOCK_ON_MERGE

Cause: Maximum number of attempts to lock object was exceed resulting in a failure to merge the transaction.

Action: Ensure that the number of attempts is within the limit.

2009: MAX_TRIES_EXCEDED_FOR_LOCK_ON_BUILD_OBJECT

Cause: Maximum number of attempts to lock object was exceed resulting in a failure to build the object: thread `threadNumber` has a lock on the object, but thread `anotherThreadNumber` is building the object

Action: Ensure that the number of attempts is within the limit.

For more information, see "Concurrency" on page 2-18.

Conversion Exceptions (3001– 3008)

`ConversionException` 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, as shown in Example 13–3.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
INTERNAL EXCEPTION: Message
```

Example 13–3 Conversion Exception

```
EXCEPTION [TOPLINK – 3006]: oracle.toplink.exceptions.ConversionException
EXCEPTION DESCRIPTION: object must be of even length to be converted to a
ByteArray
```

3001: COULD_NOT_BE_CONVERTED

Cause: Attempt to convert an object `object` of class `ObjectClass` to `JavaClass`. The object cannot be converted to a given type.

Action: Ensure that the object being converted is of the right type.

3002: COULD_NOT_BE_CONVERTED_EXTENDED

Cause: Attempt to convert an object `object` of class `ObjectClass` from mapping `mappingType` to `JavaClass`. The object cannot be converted to a given type.

Action: Ensure that the object being converted is of the right type.

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.

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.

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.

3006: COULD_NOT_CONVERT_TO_BYTE_ARRAY

Cause: Attempt to convert `String` object of uneven length to a `ByteArray`. This object cannot be converted to a `ByteArray`.

Action: Verify the object being converted.

3007: COULD_NOT_BE_CONVERTED_TO_CLASS

Cause: Attempt to convert an object `object` of class `ObjectClass` to `JavaClass`. The class `JavaClass` is not on the classpath.

Action: Ensure that the class `JavaClass` is on the classpath.

3008: INCORRECT_DATE_TIME_FORMAT

Cause: Incorrect date-time format object. The expected format is YYYY-MM-DD'T'HH:MM:SS.

Action: Ensure that the date-time object is in the expected format of YYYY-MM-DD'T'HH:MM:SS.

For more information, see the following:

- Schema Manager Java and Database Type Conversion on page 5-3
- Type Conversion Converter on page 30-11

Database Exceptions (4002 – 4018)

`DatabaseException` is a run-time 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, as shown in Example 13–4.

This exception can occur on any database operation. If an execution of a SQL script is involved in a database operation causing DatabaseException, the exception's message, accessible through the `getMessage` method, contains the SQL that caused this exception.

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 13–4 Database Exception

```
EXCEPTION [TOPLINK – 4002]: oracle.toplink.exceptions.DatabaseException
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
```

4002: SQL_EXCEPTION

Cause: A SQL exception was encountered, raised by the underlying JDBC bridge. TopLink wraps only that exception.

Action: Inspect the internal exception that was raised.

4003: CONFIGURATION_ERROR_CLASS_NOT_FOUND

Cause: The driver class name was not found.

Action: Verify the class name given in `JDBCLogin`.

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 log in again because the connection to the database might have been lost.

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 TopLink cannot process the `BLOB` data using `ByteArrayOutputStream`.

Action: Verify whether the underlying driver supports `BLOB`s properly. If it does, then report this problem to Oracle Support Services.

4007: COULD_NOT_CONVERT_OBJECT_TYPE

Cause: Attempt to convert an object type on internal `error.java.sql.TYPES = type`. The object from the result set cannot be converted to the type that was returned from the metadata information.

Action: Verify whether the underlying driver supports the conversion type properly. If it does, then report this problem to Oracle Support Services.

4008: LOGOUT_WHILE_TRANSACTION_IN_PROGRESS

Cause: Attempt to log out while the transaction is still in progress. You cannot log out while a transaction is in progress.

Action: Wait until the transaction is finished.

4009: SEQUENCE_TABLE_INFORMATION_NOT_COMPLETE

Cause: The sequence information given to TopLink is not sufficiently complete to get the set of sequence numbers from the database. This usually happens on native sequencing (see "Understanding Sequencing in Relational Projects" on page 17-14) on an Oracle database.

Action: Verify the data provided, especially the sequence name provided in TopLink.

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.

4014: CANNOT_REGISTER_SYNCHRONIZATIONLISTENER_FOR_ UNITOFWORK

Cause: TopLink cannot register the synchronization listener: *underlying_exception_string*. When the 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.

4015 SYNCHRONIZED_UNITOFWORK_DOES_NOT_ SUPPORT_ COMMITANDRESUME

Cause: A synchronized `UnitOfWork` does not support the `commitAndResume` operation. When the 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 4014: CANNOT_REGISTER_SYNCHRONIZATIONLISTENER_FOR_UNITOFWORK). 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.

4016: CONFIGURATION_ERROR_NEW_INSTANCE_ INSTANTIATION_ EXCEPTION

Cause: A configuration error occurred when TopLink attempted to instantiate the given driver class. TopLink cannot instantiate the driver.

Action: Check the driver.

4017: CONFIGURATION_ERROR_NEW_INSTANCE_ILLEGAL_ACCESS_ EXCEPTION

Cause: A configuration error occurred when TopLink attempted to instantiate the given driver class. TopLink cannot instantiate the driver.

Action: Check the driver.

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 Exceptions (5001 – 5009)

`OptimisticLockException` is a run-time 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, as shown in Example 13–5.

For more information about optimistic locking, see the section on Optimistic locking in a stateless environment in Chapter 2, "Understanding TopLink Application Development". These exceptions should be handled in a try-catch block.

Format

```
EXCEPTION [TOPLINK – error code]: Exception Name
EXCEPTION DESCRIPTION: Message
```

Example 13–5 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.
```

5001: NO_VERSION_NUMBER_WHEN_DELETING

Cause: Attempt to delete the object `object` that does not have a 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 operation shows the object being deleted when the exception was raised.

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.

5004: NO_VERSION_NUMBER_WHEN_UPDATING

Cause: An attempt has been made to update the object `object` that does not have a version number in the identity map. It may not have been read before being updated, or it has been deleted.

Action: Use SQL logging to determine the reason for the exception. The last update operation shows the object being updated when the exception was raised.

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.

5007: MUST_HAVE_MAPPING_WHEN_IN_OBJECT

Cause: The object `object` does not have a non-read-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.

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`.

5009: UNWRAPPING_OBJECT_DELETED_SINCE_LAST_READ

Cause: Attempt to unwrapped an object of class `className` with primary key `key`–the object was deleted since it had been last read.

Action: Ensure the existence of the object being upwrapped.

Query Exceptions (6001 – 6129)

`QueryException` 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 TopLink is able to determine the query that caused this exception, as shown in Example 13–6.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
QUERY:
```

Example 13–6 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 the query is null.
```

6001: ADDITIONAL_SIZE_QUERY_NOT_SPECIFIED

Cause: Additional size-retrieving query was not specified: 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.

6002: AGGREGATE_OBJECT_CANNOT_BE_DELETED

Cause: An aggregated object was deleted independently of its owner: aggregate 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.

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.

6004: BACKUP_CLONE_IS_ORIGINAL_FROM_PARENT

Cause: The object `clone` of class `clone.getClass()` with identity hash code `(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 `validateObjectSpace` method of the UnitOfWork to help debug where the error occurred.

6005: BACKUP_CLONE_IS_ORIGINAL_FROM_SELF

Cause: The object clone of class `clone.getClass()` with identity hash code `(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 referenced by another object. If you do not want the new object to be cloned, use the `registerNewObject(Object)` method of the UnitOfWork.

Action: Verify that you are correctly registering your objects. If you are still having problems, use the `validateObjectSpace` method of the UnitOfWork to help debug where the error occurred.

6006: BATCH_READING_NOT_SUPPORTED

Cause: The mapping that does not support batch reading was used. The optimization of batch reading all the target rows is not supported for the mapping.

Action: The problem is a TopLink development problem, and you should never encounter this error code unless the mapping is a new custom mapping. Contact Oracle Support Services.

6007: DESCRIPTOR_IS_MISSING

Cause: The descriptor for the reference class is missing. The descriptor related to the class or the object is not found in the session.

Action: Verify whether or not the related descriptor was added to the session, and whether or not the query is performed on the right object or class.

6008: DESCRIPTOR_IS_MISSING_FOR_NAMED_QUERY

Cause: The descriptor `DomainClassName` for the query named `queryName` is missing. The descriptor where named query is defined is not added to the session.

Action: Verify whether or not the related descriptor was added to the session, and whether or not the query is performed on the right class.

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, then check the size of the query that was specified, or report this problem to Oracle Support Services.

6014: INVALID_QUERY

Cause: Attempt to write an object in a unit of work using modify queries. These objects must be registered.

Action: Prior to modification, register objects in the unit of work, so during commit the unit of work can perform the required changes to the database.

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.

6016: INVALID_QUERY_ON_SERVER_SESSION

Cause: Attempt to change an object or a database 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.

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 hash table or if class extraction method was used, it did not return any concrete class type. The exception is raised when subclasses are being read.

Action: Check the class extraction method, if specified, or check the descriptor to verify all the type indicator values were specified.

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.

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.

6024: OBJECT_TO_MODIFY_NOT_SPECIFIED

Cause: An object to modify has not been specified for a modify query.

Action: Verify that the query contains an object before executing.

6026: QUERY_NOT_DEFINED

Cause: The query is not defined. When executing a query on the session, the parameter that takes the query is `null`.

Action: Verify that the query is passed properly.

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 the `commitAndResume` method is called.

6028: READ_BEYOND_QUERY

Cause: Attempt 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.

6029: REFERENCE_CLASS_MISSING

Cause: The reference class in the query is not specified: a reference class must be provided.

Action: Ensure that the query is correct.

6030: REFRESH_NOT_POSSIBLE_WITHOUT_CACHE

Cause: Attempt to refresh while the 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: Ensure that the query is correct.

6031: SIZE_ONLY_SUPPORTED_ON_EXPRESSION_QUERIES

Cause: 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.

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 Oracle Support Services.

6034: INVALID_QUERY_ITEM

Cause: TopLink is unable to validate a query item expression.

Action: Validate the expression being used.

6041: SELECTION_OBJECT_CANNOT_BE_NULL

Cause: The selection object that was passed to a `ReadObjectQuery` (or refresh) was `null`.

Action: Check `setSelectionObject` method on read query.

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.

6043: REPORT_RESULT_WITHOUT_PKS

Cause: Attempt to read the object by a `ReportQuery` without a primary key: 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` method on report query.

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.

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.

6046: CANNOT_DELETE_READ_ONLY_OBJECT

Cause: Attempt to delete a read-only class.

Action: Contact Oracle Support Services.

6047: INVALID_OPERATOR

Cause: The operator used in the expression is not valid.

Action: Check `ExpressionOperator` class to see a list of all the operators that are supported.

6048: ILLEGAL_USE_OF_GETFIELD

Cause: Invalid use of `getField` method's data in the expression. This is a TopLink development exception that you should not encounter.

Action: Report this problem to Oracle Support Services.

6049: ILLEGAL_USE_OF_GETTABLE

Cause: Invalid use of `getTable` method's data in the expression. This is a TopLink development exception that you should not encounter.

Action: Report this problem to Oracle Support Services.

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 Oracle Support Services.

6051: CANNOT_CACHE_PARTIAL_OBJECT

Cause: Attempt to cache a partial object: partial objects are never put in the cache. Partial object queries are not allowed to maintain the cache or to be edited. Use the `dontMaintainCache` method.

Action: Call the `dontMaintainCache` method before executing the query.

6052: OUTER_JOIN_ONLY_VALID_FOR_ONE_TO_ONE

Cause: Invalid use of an outer join: an outer join (`getAllowingNull` method) is valid only for one-to-one mappings and cannot be used for the mapping.

Action: Do not attempt to use the `getAllowingNull` method for mappings other than one-to-one.

6054: CANNOT_ADD_TO_CONTAINER

Cause: TopLink is unable to add an object to a container class using `policy`. This is TopLink development exception, and you should never encounter this problem unless a custom container policy has been written.

Action: Contact Oracle Support Services.

6055: METHOD_INVOCATION_FAILED

Cause: The invocation of a method on the object `anObject` threw a Java reflection exception while accessing the method.

Action: Inspect the internal exception, and refer to the Java documentation.

6056: CANNOT_CREATE_CLONE

Cause: Attempt to create a clone of the object `anObject` using `policy.` This is a TopLink development exception, and you should never encounter this problem unless a custom container policy has been written.

Action: Report this problem to Oracle Support Services.

6057: METHOD_NOT_VALID

Cause: Invalid call of the method `methodName` on object `aReceiver`. This is a TopLink development exception, and you should never encounter this problem unless a custom container policy has been written.

Action: Contact Oracle Support Services.

6058: METHOD_DOES_NOT_EXIST_IN_CONTAINER_CLASS

Cause: The method named `methodName` was not found in the class `aClass`. This exception is raised when looking for a clone method on the container class. The clone is needed to create clones of the container in unit of work.

Action: Define a clone method on the container class.

6059: COULD_NOT_INSTANTIATE_CONTAINER_CLASS

Cause: Attempt to instantiate a class `aClass` as the container for the results of a query–this class cannot be instantiated as a container. The exception is a Java exception that is raised when a new interface container policy is being created using Java reflection. TopLink wraps only the Java exception.

Action: Inspect the internal exception, and refer to the Java documentation.

6060: MAP_KEY_NOT_COMPARABLE

Cause: Attempt to 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. This raises a Java reflection exception while accessing the method. TopLink wraps only the Java exception.

Action: Inspect the internal exception, and refer to the Java documentation.

6061: CANNOT_ACCESS_METHOD_ON_OBJECT

Cause: Attempt to reflectively access the method `aMethod` for object: `anObject` of type `ObjectClass`. This raises a Java reflection exception while accessing the method. TopLink wraps only the Java exception.

Action: Inspect the internal exception, and refer to the Java documentation.

6062: CALLED_METHOD_THREW_EXCEPTION

Cause: The method `aMethod` was called reflectively on `objectClass` and threw an exception. The method `aMethod` raises a Java reflection exception while accessing a method. TopLink wraps only the Java exception.

Action: Inspect the internal exception, and refer to the Java documentation.

6063: INVALID_OPERATION

Cause: Invalid operation `operationName` on the cursor. The operation is not supported.

Action: Check the class documentation and look for the corresponding method to use.

6064: CANNOT_REMOVE_FROM_CONTAINER

Cause: Attempt to remove `anObject` of type `anObjectClass` from `aContainerClass` using `policy`. This is a TopLink development exception and you should never encounter this problem unless a custom container policy has been written.

Action: Contact Oracle Support Services.

6065: CANNOT_ADD_ELEMENT

Cause: Attempt to add an element to the collection container policy (cannot add object `anObject` of type `ObjectClass` to a `ContainerClass`).

Action: Inspect the internal exception, and refer to the Java documentation.

6066: BACKUP_CLONE_DELETED

Cause: Object references remained after the deletion of the object: the object clone of class `clone.getClass` with identity hash code `(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 `validateObjectSpace()` method of the UnitOfWork to help identify where the error occurred.

6068: CANNOT_COMPARE_TABLES_IN_EXPRESSION

Cause: Attempt to compare table reference to `data` in expression.

Action: Check the expression.

6069: INVALID_TABLE_FOR_FIELD_IN_EXPRESSION

Cause: Field has invalid table in this context for field `fieldName` in expression.

Action: Check the expression.

6070: INVALID_USE_OF_TO_MANY_QUERY_KEY_IN_EXPRESSION

Cause: Invalid use of a query key representing a one-to-many relationship in expression.

Action: Use the `anyOf` operator instead of the `get` operator.

6071: INVALID_USE_OF_ANY_OF_IN_EXPRESSION

Cause: Invalid use of `anyOf` for a query key not representing a to-many relationship in expression.

Action: Use the `get` operator instead of the `anyOf` operator.

6072: CANNOT_QUERY_ACROSS_VARIABLE_ONE_TO_ONE_MAPPING

Cause: Attempt to query across a variable one-to-one mapping. This is not supported.

Action: Change the expression such that the query is not performed across a variable one-to-one mapping.

6073: ILL_FORMED_EXPRESSION

Cause: Ill-formed expression in query that is attempting to print an object reference into a SQL statement for `queryKey`.

Action: Contact Oracle Support Services.

6074: CANNOT_CONFORM_EXPRESSION

Cause: Expression cannot determine if the object conforms in memory.

Action: Set the query to check the database; change the query such that it does not attempt to conform to the results of the query.

6075: INVALID_OPERATOR_FOR_OBJECT_EXPRESSION

Cause: Invalid operator was used for the object comparison: object comparisons can use only 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.

6076: UNSUPPORTED_MAPPING_FOR_OBJECT_COMPARISON

Cause: Unsupported type of mapping was used for the object comparison: object comparisons can be used only 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.

6077: OBJECT_COMPARISON_CANNOT_BE_PARAMETERIZED

Cause: Object comparison was parameterized: object comparisons cannot be used in parameter queries.

Action: Change the query so that it does not attempt to use objects when using parameterized queries.

6078: INCORRECT_CLASS_FOR_OBJECT_COMPARISON

Cause: An incorrect class of the argument was used for the object comparison.

Action: Ensure the class for the query is correct.

6079: CANNOT_COMPARE_TARGET_FOREIGN_KEYS_ TO_NULL

Cause: Object comparison was used for target foreign key relationships: object comparisons cannot be used for target foreign key relationships

Action: Query on the source primary key.

6080: INVALID_DATABASE_CALL

Cause: Invalid database call: the call must be an instance of `DatabaseCall`.

Action: Ensure the call being used is a `DatabaseCall`.

6081: INVALID_DATABASE_ACCESSOR

Cause: Invalid database accessor: the accessor must be an instance of `DatabaseAccessor`.

Action: Ensure the accessor being used is a `DatabaseAccessor`.

6082: METHOD_DOES_NOT_EXIST_ON_EXPRESSION

Cause: The nonexisting method `methodName` with argument type `argTypes` was invoked on an expression.

Action: Ensure the method being used is a supported method.

6083: IN_CANNOT_BE_PARAMETERIZED

Cause: The query that was using `IN` was parameterized: queries using `IN` cannot be parameterized.

Action: Disable the query prepare or binding.

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.

6085: REDIRECTION_METHOD_NOT_DEFINED_CORRECTLY

Cause: The redirection query's method is not defined or it is defined 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.

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.

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.

6088: NO_ATTRIBUTES_FOR _REPORT_QUERY

Cause: A `ReportQuery` has been built with no attributes specified.

Action: Specify the attribute for the query.

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 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 Oracle Support Services.

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 TopLink cache is built with objects.

Action: Do not use a `ReportQuery` in this case.

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 Oracle Support Services.

6092: MUST_INSTANTIATE_VALUEHOLDERS

Cause: Uninstantiated value holders have been detected.

Action: Instantiate the value holders for the collection on which you want to query.

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: Use the `buildSelectionCriteria` method only with one-to-one and one-to-many mappings.

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.

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`.

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` is raised when the delegate's clone method is 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 the `invoke` method of java.lang.reflect.Method.

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 raises a `java.lang.reflect.InvocationTargetException` when invoked.

Action: Verify the implementation of the delegate's clone method.

6098: UNEXPECTED_INVOCATION

Cause: A proxy object method raises 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 raised.

6099: JOINING_ACROSS_INHERITANCE_WITH_MULTIPLE_TABLES

Cause: Joining with query across inheritance class with multiple table subclasses. This is not supported: joining cannot be used on relationships with inheritance classes that have subclasses that span multiple tables as this requires multiple separate queries. The multiple queries cannot be joined into a single query.

Action: Use batch reading on the relationship instead, as this will provide equivalent or better performance.

6100: MULTIPLE_ROWS_DETECTED_FROM_SINGLE_OBJECT_READ

Cause: Multiple values detected for single-object read query. This is a CMP compliance option that ensures the finder methods for a single object only return a single row.

Action: Set the system property `toplink.cts.checkMultipleRows` to `false`, or ensure that the finder query only returns a single row from the database.

6101: HISTORICAL_QUERIES_MUST_PRESERVE_GLOBAL_CACHE

Cause: Executing this query could violate the integrity of the global session cache which must contain only the latest versions of objects.

Action: To execute a query that returns objects of a historical nature, you must do one of the following:

1. Use a `HistoricalSession` (`acquireSessionAsOf`). All objects read will be cached and automatically read at that time. This applies also to triggering object relationships.

2. Set `shouldMaintainCache` to `false`. You may make any object expression as of a previous time, provided none of its fields are represented in the result set (i.e. used in the `where` clause).

6102: HISTORICAL_QUERIES_ONLY_SUPPORTED_ON_ORACLE

Cause: Invalid database was used: historical queries only work with Oracle 9.2.0.4 or later databases, as it uses the Oracle database Flashback feature.

Action: Ensure that historical queries are only used with an Oracle 9.2.0.4 or later database.

6103: INVALID_QUERY_ON_HISTORICAL_SESSION

Cause: Invalid query was executed on a historical session: you may not execute a `WriteQuery` from inside a read-only `HistoricalSession`. To restore historical objects, try the following: read the same object as it is now with a `UnitOfWork` and commit the `UnitOfWork`.

Action: To restore historical objects, read the same object as it is now with a `UnitOfWork` and commit the `UnitOfWork`.

6104: OBJECT_DOES_NOT_EXIST_IN_CACHE

Cause: The object does not exist in the cache.

Action: Ensure that the object exists in the cache.

6105: MUST_USE_CURSOR_STREAM_POLICY

Cause: The cursor stream policy was not used on the query instantiation.

Action: Reinitialize the query with a cursor stream policy.

6106: CLASS_PK_DOES_NOT_EXIST_IN_CACHE

Cause: The object with primary key does not exist in the cache.

Action: Ensure that the object exists in the cache.

6107: UPDATE_STATEMENTS_NOT_SPECIFIED

Cause: Missing update statements on `UpdateAllQuery`.

Action: Add update statements using the `addUpdate` method.

6108: INHERITANCE_WITH_MULTIPLE_TABLES_NOT_SUPPORTED

Cause: For UpdateAllQuery, inheritance was used with multiple tables: `UpdateAllQuery` does not support inheritance with multiple tables.

Action: Do not use `UpdateAllQuery` in this situation.

6109: QUERY_FETCHGROUP_NOT_DEFINED_IN_DESCRIPTOR

Cause: The named fetch group is not defined at the descriptor level.

Action: Ensure the fetch group is defined in the descriptor.

6110: CANNOT_CONFORM_UNFETCHED_ATTRIBUTE

Cause: Read query cannot conform to the unfetched attribute of the partially fetched object in the unit of work identity map.

Action: Do not use unfetched attribute conforming, or explicitly fetch the attribute before conforming.

6111: FETCH_GROUP_ATTRIBUTE_NOT_MAPPED

Cause: The fetch group attribute is not defined or mapped.

Action: Ensure that any attribute defined in a fetch group is defined in the class and mapped.

6112: FETCH_GROUP_NOT_SUPPORT_ON_REPORT_QUERY

Cause: A fetch group was set on report query: fetch groups cannot be set on report queries.

Action: Remove the fetch group setting on `ReportQuery`, or use `ReadObjectQuery` or `ReadObjectQuery` instead.

6113: FETCH_GROUP_NOT_SUPPORT_ON_PARTIAL_ATTRIBUTE_READING

Cause: Fetch group was used together with partial attribute reading: fetch groups cannot be used together with partial attribute reading.

Action: Remove the partial attribute reading setting in the query.

6114: FETCHGROUP_VALID_ONLY_IF_FETCHGROUP_MANAGER_IN_DESCRIPTOR

Cause: A fetch group manager is not defined at the descriptor while attempting to set a fetch group on a query.

Action: You must define a fetch group manager at the descriptor in order to set a fetch group on the query.

6115: ISOLATED_QUERY_EXECUTED_ON_SERVER_SESSION

Cause: An isolated query was executed on a server session: queries on isolated classes, or queries set to use exclusive connections, must not be executed on a `ServerSession` or in CMP outside of a transaction.

Action: Do not execute queries on isolated classes or queries set to use exclusive connections on a `ServerSession` or in CMP outside of a transaction.

6116: NO_CALL_OR_INTERACTION_SPECIFIED

Cause: No call or interaction method was specified for the attempted operation.

Action: Specify a call or interaction method.

6117: CANNOT_CACHE_CURSOR_RESULTS_ON_QUERY

Cause: A query that uses a cursored result to cache query results was set.

Action: Do not cache query results or do not use a cursor policy.

6118: CANNOT_CACHE_ISOLATED_DATA_ON_QUERY

Cause: Query on an isolated class attempted to cache query results on the query.

Action: Do not cache query results for a query on an isolated class.

6119: MAPPING_FOR_EXPRESSION_DOES_NOT_SUPPORT_JOINING

Cause: The join expression is not valid, or is for a mapping type that does not support joining.

Action: Joining is supported only for one-one and one-many mappings.

6120: SPECIFIED_PARTIAL_ATTRIBUTE_DOES_NOT_EXIST

Cause: The partial attribute `attributeName` is not a valid attribute of the class `className`.

Action: Ensure that this attribute exists, and is mapped.

6121: INVALID_BUILDER_IN_QUERY

Cause: The query has not been defined correctly: the expression builder is missing.

Action: Ensure the queries builder is always on the left for sub queries and parallel queries.

6122: INVALID_EXPRESSION

Cause: Attempt to use an invalid expression `expression`.

Action: Ensure the correctness of the expression.

6123: INVALID_CONTAINER_CLASS

Cause: Invalid container class specified: the container class `className` cannot be used, because the container needs to implement `interfaceName`.

Action: Ensure that the class specified as a container implements the correct interface.

6124: INCORRECT_QUERY_FOUND

Cause: TopLink was expecting to find the query queryName, but instead found an incorrect query `queryName`.

Action: Provide the correct query.

6125: CLEAR_QUERY_RESULTS_NOT_SUPPORTED

Cause: The `ReadQuery` method `clearQueryResults()` was called: this method cannot be called anymore. The call to the `clearQueryResults` method now requires that the session be provided.

Action: Call the `ReadQuery` method `clearQueryResults(session)`.

6126: CANNOT_CONFORM_AND_CACHE_QUERY_RESULTS

Cause: A query is being executed that uses both conforming and cached query results. These two settings are incompatible.

Action: Ensure that the query uses either conforming or cached query results.

6127: REFLECTIVE_CALL_ON_TOPLINK_CLASS_FAILED

Cause: A reflective call failed on the TopLink class `classNAME`.

Action: Set up your environment to allow Java reflection

6128: BATCH_READING_NOT_SUPPORTED_WITH_CALL

Cause: Attempt to use batch reading on a query using a custom call: batch reading is not supported on queries using custom calls.

Action: Do not use batch reading on queries using custom calls.

6129: REFRESH_NOT_POSSIBLE_WITH_CHECK_CACHE_ONLY

Cause: Attempt to refreshing the query that does not go to the database. This is not possible.

Action: Avoid refreshing queries that do not go to the database.

For more information, see Part XVIII, "Queries".

Validation Exceptions (7001 – 7200)

`ValidationException` 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 13–7 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.
```

7001: LOGIN_BEFORE_ALLOCATING_CLIENT_SESSIONS

Cause: Attempt to allocate client sessions before logging in to the server.

Action: Ensure you have called `login` method on your server session or database session. This error also appears in multi-threaded environments as a result of concurrency issues. Check that all your threads are synchronized.

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.

7003: MAX_SIZE_LESS_THAN_MIN_SIZE

Cause: The maximum number of connections in a connection pool is less than the minimum number of connections. The connection pool size must be greater than the minimum number of connections.

Action: Check `addConnectionPool(String poolName, JDBCLogin login, int minNumberOfConnections, int maxNumberOfConnections)` method on the server session.

7004: POOLS_MUST_BE_CONFIGURED_BEFORE_LOGIN

Cause: Attempt to add connection pools after logging in to the server session: 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.

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 the table or stored procedure definition.

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.

7010: START_INDEX_OUT_OF_RANGE

Cause: This is a TopLink development exception and you 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 Oracle Support Services.

7011: STOP_INDEX_OUT_OF_RANGE

Cause: This is a TopLink development exception and you 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 Oracle Support Services.

7012: FATAL_ERROR_OCCURRED

Cause: This is a TopLink development exception and you should never encounter this problem. It happens when test cases are executed.

Action: Report this problem to Oracle Support Services. This error commonly occurs if you attempt to call the `commit` method on an invalid (or previously committed) unit of work.

If `cannotCommitUOWAgain` method of `ValidationException` appears in the stack trace, verify that the `commit` method was called on valid `UnitOfWork` instances.

7013: NO_PROPERTIES_FILE_FOUND

Cause: The `toplink.properties` file cannot be found on the system classpath.

Action: Ensure that there is a `toplink.properties` file located on the system classpath.

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 the child descriptor and remove the identity map from it.

7018: FILE_ERROR

Cause: You should never encounter this problem. It happens when test cases are executed.

Action: Contact Oracle Support Services.

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.

7024: INVALID_MERGE_POLICY

Cause: This is a TopLink development exception and you should never encounter it.

Action: Contact Oracle Support Services.

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 Oracle Support Services.

7027: SEQUENCE_SETUP_INCORRECTLY

Cause: The sequence `sequenceName` is set up incorrectly: increment does not match preallocation size.

Action: Contact Oracle Support Services.

7028: WRITE_OBJECT_NOT_ALLOWED_IN_UNIT_OF_WORK

Cause: Attempt to use `writeObject()` method in a `UnitOfWork`.

Action: Ensure that a `writeObject()` method is not in the `UnitOfWork`.

7030: CANNOT_SET_READ_POOL_SIZE_AFTER_LOGIN

Cause: Attempt to set read pool size after the server session has already been logged in.

Action: Set the pool size before login.

7031: CANNOT_ADD_DESCRIPTORS_TO_SESSION_BROKER

Cause: Attempt to add a descriptor directly to a session broker.

Action: Descriptors are added to the sessions contained in the session broker.

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.

7033: NO_SESSION_REGISTERED_FOR_NAME

Cause: A session `sessionName` is not registered in the session broker.

Action: Check the session broker.

7038: LOG_IO_ERROR

Cause: Error while logging message to session's log.

Action: Check the internal exception.

7039: CANNOT_REMOVE_FROM_READ_ONLY_CLASSES_ IN_NESTED_UNIT_ OF_WORK

Cause: Attempt 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 be a superset of its parent's set of read-only classes.

Action: Contact Oracle Support Services.

7040: CANNOT_MODIFY_READ_ONLY_CLASSES_SET_ AFTER_USING_UNIT_ OF_WORK

Cause: Attempt 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 Oracle Support Services.

7042: PLATFORM_CLASS_NOT_FOUND

Cause: The platform class `className` was not found and a reflection exception is raised.

Action: Check the internal exception.

7043: NO_TABLES_TO_CREATE

Cause: A project `project` does not have any tables to create on the database.

Action: Validate the project and tables you are attempting to create.

7044: ILLEGAL_CONTAINER_CLASS

Cause: Illegal attempt to use the class `className` as the container–this class does not implement the `Collection` or `Map` interfaces.

Action: Implement either the `Collection` or `Map` interfaces in the container class.

7047: CONTAINER_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 the method `methodName`.

Action: Use map class that implements the `Map` interface.

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 the way 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 getter method in order to extract it from each object using reflection. For instance, a user might call `policy.setKeyMethodName("getId")`.

Action: Check the second parameter of the `useMapClass` method of `DatabaseQuery`.

7051: MISSING_MAPPING

Cause: Missing attribute `attributeName` for descriptor `descriptor` called from `methodName`. This is a TopLink development exception, and you should never encounter it.

Action: Contact Oracle Support Services.

7052: ILLEGAL_USE_OF_MAP_IN_DIRECTCOLLECTION

Cause: The method `useMapClass` was called on a `DirectCollectionMapping`. It is invalid to call the `useMapClass` method on a `DirectCollectionMapping`. TopLink cannot instantiate Java attributes mapped using a `DirectCollectionMapping` with a map. The `useMapClass` method is supported for `OneToManyMapping` and `ManyToManyMapping`. The Java 2 `Collection` interface is supported using the `useCollectionClass` method.

Action: Use the `useCollectionClass` method. Do not call the `useMapClass` method on `DirectCollectionMapping`.

7053: CANNOT_RELEASE_NON_CLIENTSESSION

Cause: 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.

7054: CANNOT_ACQUIRE_CLIENTSESSION_FROM_SESSION

Cause: TopLink is unable to acquire a session that is not a client session. Client sessions can be acquired only from server sessions.

Action: Modify the code to ensure an acquire session is attempted only from server sessions.

7055: OPTIMISTIC_LOCKING_NOT_SUPPORTED

Cause: Attempt to use optimistic locking with stored procedure generation. This is not supported.

Action: Do not use optimistic locking with stored procedure generation.

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.

7058: INVALID_CONNECTOR

Cause: The connector selected is invalid and must be of type `DefaultConnector`.

Action: Ensure that the connector is of type `DefaultConnector`.

7059: INVALID_DATA_SOURCE_NAME

Cause: Invalid data source name `dsName`.

Action: Verify the data source name.

7060: CANNOT_ACQUIRE_DATA_SOURCE

Cause: TopLink is unable to acquire the data source with the name `dsName`, or an error has occurred in setting up the data source.

Action: Verify the data source 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` file.
- The driver is not on the classpath.
- The user or password is incorrect.
- The database server URL or driver name is not properly specified.

7061: JTS_EXCEPTION_RAISED

Cause: An exception occurred within the Java Transaction Service (JTS).

Action: Examine the JTS exception and see the JTS documentation.

7062: FIELD_LEVEL_LOCKING_NOTSUPPORTED_ OUTSIDE_A_UNIT_OF_ WORK

Cause: Attempt to use `FieldLevelLocking` outside a unit of work. This is not supported. In order to use field-level locking, a unit of work must be used for *all* write operations.

Action: Use a unit of work for writing.

7063: EJB_CONTAINER_EXCEPTION_RAISED

Cause: An exception occurred within the EJB container.

Action: Examine the EJB exception and see the JTS documentation.

7064: EJB_PRIMARY_KEY_REFLECTION_EXCEPTION

Cause: An exception occurred in the reflective enterprise bean primary key extraction.

Action: Ensure that your primary key object is defined correctly.

7065: EJB_CANNOT_LOAD_REMOTE_CLASS

Cause: The remote class for the bean cannot be loaded or found.

Action: Ensure that the correct class loader is set properly.

7066: EJB_MUST_BE_IN_TRANSACTION

Cause: Attempt to create or remove an enterprise bean outside of a transaction: TopLink is unable to create or remove enterprise beans unless a JTS transaction is present, `bean=bean`.

Action: Ensure that the JTS transaction is present.

7068: EJB_INVALID_PROJECT_CLASS

Cause: The platform class `platformName` was not found for the project `projectName` using the default class loader.

Action: Validate the project and platform.

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.

7070: EJB_TOPLINK_PROPERTIES_NOT_FOUND

Cause: Attempt to load `toplink.properties` resource bundle outside the classpath: the resource bundle must be located on the classpath in a TopLink directory.

Action: Validate the classpath and the location of the TopLink resource bundle.

7071: CANT_HAVE_UNBOUND_IN_OUTPUT_ARGUMENTS

Cause: Attempt to use input or output parameters without using binding.

Action: Use binding on the `StoredProcedureCall`.

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 TopLink session common properties from the TopLink global properties file (`sessions.xml` for non-EJB applications, or `toplink-ejb-jar.xml` for EJB applications).

Action: Ensure that your TopLink global properties file is correctly configured. Pay particular attention to the `platform-class` and `external-transaction-controller-class` properties.

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.

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.

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`.

7076: DESCRIPTOR_MUST_NOT_BE_INITIALIZED

Cause: Attempt to generate a project class while descriptors have already been initialized: when generating the project class, the descriptors must not be initialized.

Action: Ensure that the descriptors are not initialized before generating the project class.

7077: EJB_INVALID_FINDER_ON_HOME

Cause: The home interface `toString` method 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.

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 Oracle Support Services.

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.

7080: EJB_FINDER_EXCEPTION

Cause: A `FinderException` is raised when attempting to load an object from the class with the primary key.

Action: Contact Oracle Support Services.

7081: CANNOT_REGISTER_AGGREGATE_OBJECT_IN_ UNIT_OF_ WORK

Cause: Attempt to register an aggregate object directly in the unit of work. This is not supported. The aggregate object must be associated with the source (owner) object.

Action: Contact Oracle Support Services.

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`.

7083: NO_PROJECT_SPECIFIED_IN_PROPERTIES

Cause: The `toplink.properties` file does not include any information on the TopLink project to use for the server. One project file must be specified.

Action: Specify either `projectClass`, `projectFile`, or `xmlProjectFile`.

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 Oracle Support Services.

7085: SUB_SESSION_NOT_DEFINED_FOR_BROKER

Cause: TopLink is unable to create an instance of the external transaction controller specified in the properties file.

Action: Contact Oracle Support Services.

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.

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.

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.

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.

7091: SET_LISTENER_CLASSES_EXCEPTION

Cause: TopLink is unable to create the listener class that implements `SessionEventListener` for the internal use of `SessionXMLProject`.

Action: Contact Oracle Support Services.

7092: EXISTING_QUERY_TYPE_CONFLICT

Cause: 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.

7093: QUERY_ARGUMENT_TYPE_NOT_FOUND

Cause: 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 classpath environment.

7094: ERROR_IN_SESSION_XML

Cause: The `sessions.xml` or `toplink.properties` file cannot be loaded.

Action: Ensure that the path to either of the files exists on the classpath environment.

7095: NO_SESSIONS_XML_FOUND

Cause: The `sessions.xml` or `toplink.properties` file cannot be loaded.

Action: Ensure that the path to either of the files exists on the classpath environment. The `sessions.xml` should be included in the root of the deployed JAR file. When using a WAR file, the `sessions.xml` file should be located in the

`WEB-INF\classes` directory. When using EJB 3.0, TopLink automatically loads the `ejb3-toplink-sessions.xml` file.

7096: CANNOT_COMMIT_UOW_AGAIN

Cause: TopLink cannot invoke `commit` method on an inactive unit of work that was committed or released.

Action: Ensure you invoke `commit` method on a new unit of work or invoke `commitAndResume` method so that the unit of work can be reused. For more information about the `commitAndResume` method, see *Oracle TopLink API Reference.*

7097: OPERATION_NOT_SUPPORTED

Cause: TopLink cannot invoke an unsupported operation on an object.

Action: Do not use the operation indicated in the stack trace.

7099: PROJECT_XML_NOT_FOUND

Cause: The file name specified for the XML-based project is incorrect.

Action: Verify the name and location of the file.

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 classpath.

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 `removeFromIdentityMap` method of the `Session` is intended to allow garbage collection, which has already been done.

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: Check that this object is not `null` before invoking its methods.

7104: INVALID_SEQUENCING_LOGIN

Cause: A separate connection(s) for sequencing was requested, but the sequencing login uses the 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.

7105: INVALID_ENCRYPTION_CLASS

Cause: Error encountered while converting encryption class.

Action: Ensure the encryption class name is correctly specified in the `sessions.xml` file and that the encryption class specified is available on the classpath. A common reason for this exception is the usage of JDK 1.3 and earlier versions. The TopLink JCE encryption mechanism requires JDK 1.4 and later (or JDK 1.3 configured with the JCE plug-in) to function properly.

7106: ERROR_ENCRYPTING_PASSWORD

Cause: Error encountered during password string encryption.

Action: An error is raised while trying to encrypt the password string. A common reason for this exception is the usage of JDK 1.3 and earlier versions. The TopLink JCE encryption mechanism requires JDK 1.4 and later (or JDK 1.3 configured with the JCE plug-in) to function properly.

7107: ERROR_DECRYPTING_PASSWORD

Cause: Error encountered during password string decryption.

Action: An exception was raised while trying to decrypt the password string. A common reason for this exception is the usage of JDK 1.3 and earlier versions. The TopLink JCE encryption mechanism requires JDK 1.4 and later (or JDK 1.3 configured with the JCE plug-in) to function properly.

7108: NOT_SUPPORTED_FOR_DATASOURCE

Cause: Attempt to use an operation that is not supported for nonrelational platforms.

Action: Do not use this operation on the current platform, or use a relational database platform.

7109: PROJECT_LOGIN_IS_NULL

Cause: The login in the project used to create the session is `null` meaning no login was specified for the TopLink project. The login used for the project must be a valid login.

Action: Add login information using TopLink Workbench or using Java code.

7110: HISTORICAL_SESSION_ONLY_SUPPORTED_ON_ORACLE

Cause: Attempt to use a HistoricalSession with a non-Oracle database. At present, `HistoricalSession` only works with Oracle databases that have a Flashback feature (Oracle database 9.2.0.4 or later).

Generic history support (see `oracle.toplink.history.HistoryPolicy`) works for any database. If a `HistoryPolicy` is incorrectly set, TopLink may be defaulting to using flashback instead. An `AsOfSCNClause` is implicitly flashback only.

Action: Ensure that the `HistoryPolicy` is set correctly.

7111: CANNOT_ACQUIRE_HISTORICAL_SESSION

Cause: Invalid attempt to acquire a HistoricalSession: you may not acquire a `HistoricalSession` from a unit of work, another `HistoricalSession`, a `ServerSession`, or a `ServerSessionBroker`. You may acquire a HistoricalSession from a regular session, a `ClientSession`, or a `ClientSessionBroker`.

Action: To recover objects, read the objects in both a `HistoricalSession` and `UnitOfWork`, and call `mergeCloneWithReferences(historicalObject)` method on the `UnitOfWork`.

7112: FEATURE_NOT_SUPPORTED_IN_JDK_VERSION

Cause: Attempt to use a TopLink feature that is not available in the current JDK version.

Action: You must use the version of the JDK that supports this feature.

7113: PLATFORM_DOES_NOT_SUPPORT_CALL_WITH_RETURNING

Cause: Attempt to use an unsupported call with returning for a platform.

Action: Set stored procedures with output parameters in `setInsertQuery`, `setInsertCall`, `setUpdateQuery`, or `setUpdateCall` methods of the `DescriptorQueryManager`.

7114: ISOLATED_DATA_NOT_SUPPORTED_IN_CLIENTSESSIONBROKER

Cause: Attempt to use isolated data within a `ClientSessionBroker`: isolated data is not currently supported within a `ClientSessionBroker`. Session contains descriptors representing isolated data.

Action: Ensure that isolated data is not used.

7115: CLIENT_SESSION_CANNOT_USE_EXCLUSIVE_CONNECTION

Cause: Attempt to use `ExclusiveConnection` for `ClientSession` reads without isolated data. This is not supported.

Action: You must update the `ConnectionPolicy` used to remove `ExclusiveConnection` configuration, or the project to set certain data to be exclusive.

7116: INVALID_METHOD_ARGUMENTS

Cause: Invalid arguments are used in the method.

Action: Refer to the public API of the calling method and use valid values for the arguments.

7117: MULTIPLE_CURSORS_NOT_SUPPORTED

Cause: Attempt to use more than one cursor in a `SQLCall`.

Action: TopLink currently supports only one cursor per call.

7118: WRONG_USAGE_OF_SET_CUSTOM_SQL_ARGUMENT_TYPE_METOD

Cause: The `setCustomSQLArgumentType` method was invoked on `SQLCall`, but this method does not use custom SQL.

Action: Don't call this method on `SQLCall` that does not use custom SQL.

7119: CANNOT_TRANSLATE_UNPREPARED_CALL

Cause: Unprepared `SQLCall` attempted translation.

Action: `SQLCall` must be prepared before translation.

7120: CANNOT_SET_CURSOR_FOR_PARAMETER_TYPE_OTHER_THAN_OUT

Cause: Attempt to use parameter in `SQLCall` as a cursor, but this parameter is of type other than `OUT`.

Action: Ensure that the parameter used as a cursor has parameter type `OUT`.

7121: PLATFORM_DOES_NOT_SUPPORT_STORED_FUNCTIONS

Cause: Attempt to use stored functions for a platform that does not support stored functions.

Action: Do not define stored functions on this platform.

7122: EXCLUSIVE_CONNECTION_NO_LONGER_AVAILABLE

Cause: The exclusive connection associated with the session is unavailable for the query on the object.

Isolated objects with indirection read through an `ExclusiveIsolatedClientSession` must not have indirection triggered after the `ExclusiveIsolatedClientSession` has been released.

Action: Reread the objects through the current `ExclusiveIsolatedClientSession`.

7123: UNIT_OF_WORK_IN_TRANSACTION_COMMIT_PENDING

Cause: Attempt to perform an operation that is not allowed at this time: a successful `writeChanges` operation has been called on this `UnitOfWork`. As the commit process has been started, but not yet finalized, the only supported operations now are `commit`, `commitAndResume`, `release`, any nonobject level query, or `SQLCall` execution.

Action: Execute one of the supported operations to continue.

7124: UNIT_OF_WORK_AFTER_WRITE_CHANGES_FAILED

Cause: Attempt to perform an operation that is not allowed at this time: an unsuccessful `writeChanges` operation has been called on this `UnitOfWork`. Given the possibility that partial changes have been written to the data store but not rolled back (if inside external transaction), the only supported operations now are release, global transaction rollback, any nonobject level query or `SQLCall` execution.

Action: Determine the cause of the original failure and retry in a new `UnitOfWork`.

7125: INACTIVE_UNIT_OF_WORK

Cause: Attempt to perform an operation on an inactive unit of work: once the `UnitOfWork` has been committed and/or released, no further operation should be performed on it.

Action: Acquire a new `UnitOfWork`, or use the `commitAndResume` method instead of `commit` method in the future.

7126: CANNOT_WRITE_CHANGES_ON_NESTED_UNIT_OF_WORK

Cause: Attempt to call a `writeChanges` method on a `NestedUnitOfWork`. This is not supported: a nested `UnitOfWork` never writes changes directly to the data store, only the parent `UnitOfWork` does.

Action: Call the `commit` method instead, and then the `writeChanges` method on the parent `UnitOfWork`.

7127: CANNOT_WRITE_CHANGES_TWICE

Cause: Attempt to write changes to the data store more than once: you can only write changes to the data store once.

Action: You must either roll back the transaction, or call the `commit` method on this `UnitOfWork` and start a new transaction.

7128: ALREADY_LOGGED_IN

Cause: Attempt to log in to a session more than once.

Action: Do not try to login again.

7129: INVALID_NULL_METHOD_ARGUMENTS

Cause: Attempt to use null values for arguments in a method that cannot have null values.

Action: Ensure that the method's arguments do not have a null value.

7130: NESTED_UOW_NOT_SUPPORTED_FOR_ATTRIBUTE_TRACKING

Cause: Attempt to use a nested `UnitOfWork` with attribute change tracking. This is not supported.

Action: Do not use a nested `UnitOfWork` with attribute change tracking.

7131: WRONG_COLLECTION_CHANGE_EVENT_TYPE

Cause: The collection change event is of the wrong type. The collection change event type has to be added or removed.

Action: Ensure that the collection change event type used is defined in `CollectionChangeEvent`.

7132: WRONG_CHANGE_EVENT

Cause: Wrong event class. Only `PropertyChangeEvent` and `CollectionChangeEvent` classes are supported.

Action: Ensure that the event class is either `PropertyChangeEvent` or `CollectionChangeEvent`.

7133: OLD_COMMIT_NOT_SUPPORTED_FOR_ATTRIBUTE_TRACKING

Cause: Attempt to use old commit for attribute change tracking. This is not supported.

Action: Do not try to use attribute change tracking with an old commit.

7134: SERVER_PLATFORM_IS_READ_ONLY_AFTER_LOGIN

Cause: Attempt to make changes to the server platform after login: the server platform is read-only after login.

Action: Changes to the server platform must be made before login. You must either:

1. CMP application: define a class that implements `oracle.toplink.ejb.cmp.DeploymentCustomization`, and customize its `public String beforeLoginCustomization(Session session)` method to change your server platform. Consult the documentation for defining a customization class in your `orion-ejb-jar.xml` file.
2. Non-CMP/POJO application: define a subclass of `oracle.toplink.sessions.SessionEventAdapter`, and override the `public void preLogin(SessionEvent event)` method to change your server platform. The session is contained in the event. Consult the documentation for `sessions.xml` and using `SessionEventAdapter`.

7135: CANNOT_COMMIT_AND_RESUME_UOW_WITH_UPDATE_ALL_QUERIES

Cause: Attempt to commit and resume a `UnitOfWork` containing an `UpdateAllQuery`. This is not supported.

Action: You must either commit and continue in a new `UnitOfWork`, or do not use `UpdateAllQuery`.

7136: NESTED_UOW_NOT_SUPPORTED_FOR_UPDATE_ALL_QUERY

Cause: Attempt to use a nested `UnitOfWork` for an UpdateAll query. This is not supported.

Action: Do not use a nested `UnitOfWork` for an `UpdateAllQuery`.

7137: UNFETCHED_ATTRIBUTE_NOT_EDITABLE

Cause: Attempt to edit an unfetched attribute: the object is partially fetched (using fetch group), the unfetched attribute is not editable.

Action: Do not edit the unfetched attribute, or explicitly fetch the attribute before editing it.

7138: OBJECT_NEED_IMPL_TRACKER_FOR_FETCH_GROUP_USAGE

Cause: Attempt to use a fetch group with the object that does not implement FetchGroupTracker: the object must implement `FetchGroupTracker` in order to use fetch group.

Action: The domain call must implement `FetchGroupTracker` in order to use fetch group.

7139: UPDATE_ALL_QUERIES_NOT_SUPPORTED_WITH_OTHER_WRITES

Cause: Attempt to issue an UpdateAll query within a `UnitOfWork` containing other write operations.

Action: Do not use `UpdateAllQuery` within a `UnitOfWork` containing other write operations.

7140: WRONG_SEQUENCE_TYP

Cause: Sequence type does not have the method.

Action: Do not call this method on this type of sequence.

7144: PLATFORM_DOES_NOT_SUPPORT_SEQUENCE

Cause: Platform does not support sequence.

Action: Do not use this sequence type on this platform.

7145: SEQUENCE_CANNOT_BE_CONNECTED_TO_TWO_PLATFORMS

Cause: Two attempts have been made to connect to sequence, but the sequence is already connected to a platform. Likely the two sessions share the `DatasourcePlatform` object.

Action: Ensure that the sequence is used by a single session only.

7146: QUERY_SEQUENCE_DOES_NOT_HAVE_SELECT_QUERY

Cause: `QuerySequence` does not have select query.

Action: Ensure that the sequence has a select query.

7147: CREATE_PLATFORM_DEFAULT_SEQUENCE_UNDEFINED

Cause: Attempt to create platform default sequence by a platform that does not override the `createPlatformDefaultSequence` method.

Action: You must either override the `createPlatformDefaultSequence` method on the platform, or explicitly set default sequence by calling `setDefaultSequence` on `DatasourceLogin`.

7148: CANNOT_RESUME_SYNCHRONIZED_UOW

Cause: Attempt to use `commitAndResume()` method with a JTA or a synchronized unit of work.

Action: Do not use commitAndResume() with a JTA or a synchronized unit of work.

7149: INVALID_COMPOSITE_PK_SPECIFICATION

Cause: Attempt to provide an invalid specification of the composite primary key: the names of the primary key fields or properties in the primary key class `PKClassName` and those of the entity bean class `ClassName` must correspond and their types must be the same.

Action: Ensure that the names of the primary key fields or properties in the primary key class `PKClassName` and those of the entity bean class `ClassName` correspond, and their types are the same.

7150: INVALID_FETCH_LAZY_TYPE

Cause: Attempt to specify an invalid `LAZY` fetch type on an attribute `attributeName` within entity class `ClassName`.

Action: Provide the fetch type other than `LAZY` on an attribute `attributeName` within entity class `ClassName`.

7151: INVALID_ANNOTATION_SPECIFIED

Cause: An invalid EJB 3.0 annotation `annotation` has been specified in the entity class `ClassName` on accessor method `methodName`. It is not supported.

Action: Specify a valid annotation on this accessor method.

7152: TABLE_PER_CLASS_INHERITANCE_NOT_SUPPORTED

Cause: Table per class inheritance is not supported. Entity class `ClassName`.

Action: Use different approach.

7153: NAMED_QUERY_ALREADY_EXISTS

Cause: The descriptor named query `query` already exists. Named query names must be unique.

Action: Provide a unique name for the query.

7154: ASSOCIATION_TABLE_ANNOTATION_NOT_FOUND

Cause: An annotation `@AssociationTable` was not found on either the owning mapping class `ClassName`, or nonowning mapping class `DifferentClassName` for the `@ManyToMany` annotation.

Action: Ensure that the annotation `@AssociationTable` exists.

7155: ATTRIBUTE_OVERRIDE_COLUMNS_NOT_SPECIFIED

Cause: An annotation `@AttributeOverride` columns was not specified on a mapping `mapping` from entity class `ClassName`.

Action: Specify this annotation.

7156: UNABLE_TO_LOAD_CLASS

Cause: TopLink was unable to find the class `ClassName`. Ensure the class name/path is correct and available to the class loader.

Action: Ensure the class name and/or path is correct and available to the class loader.

7157: INVALID_COLUMN_ANNOTATION_ON_RELATIONSHIP

Cause: Attempt to use an invalid`@Column` annotation by the entity class `ClassName` to map its relationship attribute `attribute`.

Action: Ensure the entity class `ClassName` uses a `@JoinColumn` attribute instead of a `@Column` attribute to map its relationship attribute attributeName.

7158: ERROR_PROCESSING_NAMED_QUERY_ANNOTATION

Cause: Error encountered when building the `@NamedQuery` annotation from entity class `ClassName`.

Action: Ensure the correctness of the annotation specification.

7159: INVALID_NAMED_QUERY_ANNOTATION_NAME

Cause: Invalid use of an unnamed `@NamedQuery` annotation with the query string `string` by an entity class `ClassName` .

Action: Specify a name for `@NamedQuery`.

7160: ONE_TO_MANY_HAS_BOTH_ASSOCIATIONTABLE_AND_JOINCOLUMN_ANNOTATIONS

Cause: `@OneToMany` annotation for attribute `attributeName` in entity class `ClassName` has both an `@AssocationTable` and `@JoinColumn`(s) annotations specified.

Action: Specify one annotation only: `@AssocationTable` or `@JoinColumn`(s).

7161: NO_PK_ANNOTATIONS_FOUND

Cause: Primary key annotation is not specified for the entity class ClassName.

Action: Define either an `@Id`, `@EmbeddedId` or an `@IdClass` annotation.

7162: MULTIPLE_EMBEDDED_ID_ANNOTATIONS_FOUND

Cause: Multiple `@EmbeddedId` annotations are specified (on `attributeName` and `differentAttributeName`) for entity class `ClassName`.

Action: Specify only one `@EmbeddedId` annotation for this entity.

7163: EMBEDDED_ID_AND_ID_ANNOTATIONS_FOUND

Cause: Entity class `ClassName` has both an `@EmbdeddedId` (on attribute `attributeName`) and an `@Id` (on attribute `differentAttributeName`).

Action: Specify only one ID type on this entity: either `@EmbdeddedId` or `@Id`.

7164: INVALID_CLASS_TYPE_FOR_BLOB_ANNOTATION

Cause: Attribute `attributeName` in an entity class `ClassName` has an invalid type for a `@Lob` annotation of type `BLOB`.

Action: Define this attribute as a `java.sql.Blob`, `byte[]`, `Byte[]` or a `Serializable` type.

7165: INVALID_CLASS_TYPE_FOR_CLOB_ANNOTATION

Cause: Attribute `attributeName` in entity class `ClassName` has an invalid type for a `@Lob` annotation of type `CLOB`.

Action: Define this attribute as a `java.sql.Clob`, `char[]`, `Character[]` or `String` type.

7166: ANNOTATIONS_CONFLICT

Cause: Conflict between annotations `@Annotation` and `@DifferentAnnotation`.

Action: Ensure that these two annotations are not conflicting.

7167: RESERVED_NAME

Cause: Attempt to use a reserved annotation name `AnnotationName` by an entity class `ClassName`: cannot use this name, because it is reserved for name.

Action: Specify a different name for this annotation.

7168: ID_ANNOTATION_CANNOT_SPECIFY_GENERATOR

Cause: Attempt to specify a generator with type `Type` by an annotation `@Annotation`.

Action: Ensure that this annotation specifies a generator of a different type.

7169: INVALID_ENTITY_LISTENER_CALLBACK_METHOD_ARGUMENTS

Cause: The callback method `methodName` on the entity listener `ListenerClassName` has an incorrect signature.

Action: Provide only one parameter of type `Object` for this method.

7170: INVALID_ENTITY_CALLBACK_METHOD_ARGUMENTS

Cause: The callback method `methodName` on the entity class `ClassName` has an incorrect signature. It should not have any parameters.

Action: Do not provide any parameters for this method.

7171: INVALID_CALLBACK_METHOD

Cause: Attempt to use an invalid callback method `methodName` on the listener class `ListenerClassName`.

Action: Ensure the validity of the method.

7172: ERROR_INSTANTIATING_ENTITY_LISTENER

Cause: TopLink was unable to instantiate the entity listener using the `@EntityListener` annotation.

Action: Ensure that the annotation is correctly specified.

7173: WRONG_PROPERTY_NAME_IN_CHANGE_EVENT

Cause: Attempt to fire a property change event on a nonexisting property `propertyName` in a change event `EventClassName`.

Action: Ensure that this property exists.

7174: NO_CORRESPONDING_SETTER_METHOD_DEFINED

Cause: The getter method `methodName` on an entity class `ClassName` does not have a corresponding setter method defined.

Action: Define a corresponding setter method.

7175: UNSUPPORTED_CASCADE_LOCKING_MAPPING

Cause: Attempt to use the mapping `MappingType` that does not support cascading version optimistic locking.

Action: Use a different mapping.

7176: UNSUPPORTED_CASCADE_LOCKING_MAPPING_WITH_CUSTOM_QUERY

Cause: Attempt to use the mapping `MappingType` that does not support cascading version optimistic locking. This mapping has a custom query.

Action: Use a different mapping.

7177: UNSUPPORTED_CASCADE_LOCKING_DESCRIPTOR

Cause: Attempt to use an aggregate descriptor `DescriptorName` that has privately-owned mappings: aggregate descriptors do not support cascading version optimistic locking.

Action: Use a different descriptor.

7178: ORACLEOCIPROXYCONNECTOR_REQUIRES_ORACLEOCICONNECTIONPOOL

Cause: Attempt to use invalid arguments in an entity callback method `methodName` on a `OracleOCIProxyConnector`: it requires the `OracleOCIConnectionPool` data source.

Action: Provide `OracleOCIProxyConnector` with the `OracleOCIConnectionPool` argument.

7179: ORACLEJDBC10_1_0_2PROXYCONNECTOR_REQUIRES_ORACLECONNECTION

Cause: Attempt to use invalid arguments in an entity callback method `methodName` on a `OracleJDBC10_1_0_2ProxyConnector`.

Action: Provide `OracleJDBC10_1_0_2ProxyConnector` with data source producing Oracle Connections.

7180: ORACLEJDBC10_1_0_2PROXYCONNECTOR_REQUIRES_ORACLECONNECTION_VERSION

Cause: Attempt to use the `OracleJDBC10_1_0_2ProxyConnector` with Oracle JDBC version 10.1.0.1 or earlier: `OracleJDBC10_1_0_2ProxyConnector` requires Oracle JDBC version 10.1.0.2 or later.

Action: Provide `OracleJDBC10_1_0_2ProxyConnector` with Oracle JDBC version 10.1.0.2 or later in order for the OracleConnection to declare the `openProxySession` method.

7181: ORACLEJDBC10_1_0_2PROXYCONNECTOR_REQUIRES_INT_PROXYTYPE

Cause: Attempt to use an invalid type for the `proxytype` property on the `OracleJDBC10_1_0_2ProxyConnector`: it requires the `proxytype` property to be an `int` converted to a `String`.

Action: On the `OracleJDBC10_1_0_2ProxyConnector`, provide the proxytype property of type `int` converted to a `String`. For example, `Integer.toString(OracleConnection.PROXYTYPE_USER_NAME)`.

7182: COULD_NOT_FIND_DRIVER_CLASS

Cause: TopLink could not find the driver class.

Action: Ensure that the driver class exists and is on the classpath.

7183: ERROR_CLOSING_PERSISTENCE_XML

Cause: TopLink cannot close `persistence.xml` file.

Action: If you modified the `persistence.xml`, ensure that this file is not read-only.

7184: CONFIG_FACTORY_NAME_PROPERTY_NOT_SPECIFIED

Cause: The system property propertyName is not specified.

Action: Set this property to a class that defines a `getContainerConfig()` method.

7185: CONFIG_FACTORY_NAME_PROPERTY_NOT_FOUND

Cause: TopLink cannot find class ClassName.

Action: Ensure that the class exists and is properly specified.

7186: CANNOT_INVOKE_METHOD_ON_CONFIG_CLASS

Cause: TopLink cannot invoke method methodName on configuration class ClassName.

Action: Ensure that this method exists in this class.

7187: CONFIG_METHOD_NOT_DEFINED

Cause: Class ClassName does not define a public static method methodName.

Action: In the class ClassName, define a public static method methodName that has no parameters and returns a `Collection`.

7188: CLASS_LIST_MUST_NOT_BE_NULL

Cause: Class list is `null`.

Action: Provide a non-`null` class list.

7189: CURRENT_LOADER_NOT_VALID

Cause: TopLink cannot create a temporary class loader from the current loader `ClassLoaderName`.

Action: Ensure the validity of the current class loader.

7190: METHOD_FAILED

Cause: Failure in the execution of the methodName method.

Action: Ensure the correctness of the method.

7191: ENTITY_CLASS_NOT_FOUND

Cause: An entity class ClassName was not found using class loader ClassLoaderName.

Action: Ensure that the entity class exists and is on the classpath.

7192: CLASS_FILE_TRANSFORMER_THROWS_EXCEPTION

Cause: Class file transformer `TransformerName` threw an exception when performing `transform()` method on class `ClassName`.

Action: Ensure the correctness of the `transform()` method. Check the internal exception for details on the root cause of this exception

7193: JAR_FILES_IN_PERSISTENCE_XML_NOT_SUPPORTED

Cause: Attempt to use JAR files in the `persistence.xml` file: JAR files are not supported in this version of TopLink.

Action: Do not use JAR files.

7194: FLUSH_MODE_NOT_SUPPORTED

Cause: Attempt to use an unsupported method `setFlushMode()`.

Action: Do not use this method.

7195: COULD_NOT_BIND_JNDI

Cause: TopLink could not bind `name` to `anotherName`.

Action: Ensure the correct use of this binding.

7196: EXCEPTION_CONFIGURING_EM_FACTORY

Cause: Exception configuring `EntityManagerFactory`.

Action: Ensure the correct configuration of this factory class. Check the internal exception for details on the root cause of this exception

7197: NULL_PK_IN_UOW_CLONE

Cause: TopLink could not calculate changes, because the primary key is set to `null`.

Action: Set the primary key to a non-`null` value.

7198: CANNOT_CAST_TO_CLASS

Cause: Attempt to cast an instance of class `ClassName` to `AnotherClassName`. This is an invalid cast.

Action: Provide a valid cast.

7199: NO_MAPPED_BY_ATTRIBUTE_FOUND

Cause: An attribute `attributeName` in an entity class ClassName has a `mappedBy` value of `value`. This value does not exist in its owning entity class ClassName.

Action: If the owning entity class is an `@EmbeddableSuperclass`, this is invalid. Make your attribute reference the correct subclass.

7200: NAME_AND_REF_NAME_MUST_BE_SPECIFIED

Cause: Either name or referencedColumnName element is not specified: if there is more than one join column, both the `name` and the `referencedColumnName` elements must be specified in each `@JoinColumn` annotation.

Action: Ensure that both the `name` and the `referencedColumnName` elements are specified in each `@JoinColumn` annotation.

EJB QL Exceptions (8001 – 8010)

`EJBQLException` is a run-time exception that is raised when the EJB QL string does not parse properly, or the contents cannot be resolved within the context of the TopLink session. The associated message typically includes a reference to the EJB QL string that caused the problem.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
```

Example 13–8 EJB QL Exception

```
EXCEPTION [TOPLINK – 8002]: oracle.toplink.exceptions.EJBQLException
EXCEPTION DESCRIPTION: TopLink has encountered a problem while parsing the EJB QL
string.
```

8001: recognitionException

Cause: The TopLink EJB QL parser does not recognize a clause in the EJB QL string.

Action: Validate the EJB QL string.

8002: generalParsingException

Cause: 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.

8003: classNotFoundException

Cause: The class specified in the EJB QL string was not found.

Action: Ensure that the class is on the appropriate classpath.

8004: aliasResolutionException

Cause: TopLink was unable to resolve the alias used in the EJB QL string.

Action: Validate the identifiers used in the EJB QL string.

8005: resolutionClassNotFoundException

Cause: 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 classpath.

8006: missingDescriptorException

Cause: The class specified in the query does not have a TopLink descriptor.

Action: Ensure that the class has been mapped and is specified correctly in the EJB QL string.

8009: expressionNotSupported

Cause: An unsupported expression was used in the EJB QL.

Action: Change the query to use only supported expressions.

8010: generalParsingException2

Cause: 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 Exceptions (9000 - 9010)

`SessionLoaderException` is a run-time exception that is raised 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 13–9 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.
```

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.

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 TopLink XSD. See the directory where you installed TopLink (for example, <*ORACLE_HOME*>`/toplink/config/xsds`)

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.

9003: UNABLE_TO_PROCESS_TAG

Cause: The session loader caught an exception while either parsing the value of the specified tag or calling the setter method associated with the specified tag.

Action: Verify the value shown for the specified tag.

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 file system.

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 file system.

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 file system.

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. Oracle TopLink 10*g* supports only UTF-8 encoding. The TopLink `SAXParseException` occurs if you attempt to read a non-UTF-8 formatted XML file.

Action: Verify that the XML is correctly formatted at the given line and column. Alternatively, ensure the Oracle parser is in your classpath and that it appears before any other XML parser.

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.

9008: UN_EXPECTED_VALUE_OF_TAG

Cause: The value of an XML tag does not correspond to any known TopLink required values.

Action: Please verify the list of values for this tag.

9009: UNKNOWN_ATTRIBUTE_OF_TAG

Cause: There is an incorrect name value pair when processing transport properties for the XSD tag.

Action: Please verify that all properties have both the name and the value filled in, in the session configuration XML file.

9010: XML_SCHEMA_PARSING_ERROR

Cause: An exception was raised while parsing the XML file against the XML schema.

Action: Examine the exception and take the appropriate action.

Communication Exceptions (12000 - 12003)

`CommunicationException` is a run-time exception that wraps all RMI, CORBA, or input and output exceptions that occur.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
```

Example 13–10 Communication Exception

```
EXCEPTION [TOPLINK – 12000]: oracle.toplink.exceptions.CommunicationException
EXCEPTION DESCRIPTION: Error Sending connection service to myService.
```

12000: ERROR_SENDING_CONNECTION_SERVICE

Cause: Failed to add a connection to `CacheSynchronizationManager` or `RemoteCommandManager`.

Action: See the generated exception for the root cause.

12001: UNABLE_TO_CONNECT

Cause: `CacheSynronizationManager` failed to connect to the specified service.

Action: See the generated exception for the root cause.

12002: UNABLE_TO_PROPAGATE_CHANGES

Cause: `CacheSynronizationManager` failed to propagate changes to the specified service.

Action: See the generated exception for the root cause.

12003: ERROR_IN_INVOCATION

Cause: Error invoking a remote call.

Action: See the generated exception for the root cause.

EIS Exceptions (17007 – 17025), 90000, 91000

`EISException` is a run-time exception that is raised when invoking EIS interactions. For more information on EIS interactions, see "Enterprise Information System (EIS) Interactions" on page 93-19.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
```

Example 13–11 JMS Processing Exception

```
EXCEPTION [TOPLINK – 17010]: oracle.toplink.eis.EISException
EXCEPTION DESCRIPTION: Output record contains an unsupported message type.
```

17007: PROP_NOT_SET

Cause: The specified property is not set.

Action: Verify your interaction and ensure that the specified property is set (see "Configuring Custom EIS Interactions for Basic Persistence Operations" on page 28-6 or "Creating an EIS Interaction for a Named Query" on page 25-20).

17008: INVALID_PROP

Cause: Invalid property encountered.

Action: Verify your interaction and remove the specified property (see "Configuring Custom EIS Interactions for Basic Persistence Operations" on page 28-6 or "Creating an EIS Interaction for a Named Query" on page 25-20).

17009: PROPS_NOT_SET

Cause: The specified properties are not set.

Action: Verify your interaction and ensure that the specified properties are set (see "Configuring Custom EIS Interactions for Basic Persistence Operations" on page 28-6 or "Creating an EIS Interaction for a Named Query" on page 25-20).

17010: OUTPUT_UNSUPPORTED_MSG_TYPE

Cause: The output record contains an unsupported message type.

Action: Verify your interaction and ensure that you specify a supported message type. Ensure that the required connector JAR file is on your classpath (see "EIS Projects" on page 17-7).

17011: NO_CONN_FACTORY

Cause: The connection factory is not specified.

Action: Verify your interaction and ensure that you specify a connection factory (see "Configuring EIS Connection Specification Options at the Project Level" on page 21-2 or "Configuring EIS Connection Specification Options at the Session Level" on page 84-2).

17012: INVALID_INTERACTION_SPEC_TYPE

Cause: `InteractionSpec` is not a `CciJMSInteractionSpec`.

Action: Verify your interaction and ensure that you specify a valid interaction specification type (`CciJMSInteractionSpec`). Ensure that the required connector JAR file is on your classpath (see "EIS Projects" on page 17-7).

17013: INVALID_RECORD_TYPE

Cause: `Record` is not a `CciJMSRecord`.

Action: Verify your interaction and ensure that you specify a valid record type (CciJMSRecord). Ensure that the required connector JAR file is on your classpath (see "EIS Projects" on page 17-7).

17014: UNKNOWN_INTERACTION_SPEC_TYPE

Cause: Unknown interaction specification type.

Action: Verify your interaction and ensure that you specify a valid interaction specification type. Ensure that the required connector JAR file is on your classpath (see "EIS Projects" on page 17-7).

17015: INVALID_INPUT

Cause: Invalid input: input must contain a single text element.

Action: Verify your interaction and ensure that you specify valid input–a single text element (see "Configuring Custom EIS Interactions for Basic Persistence Operations" on page 28-6 or "Creating an EIS Interaction for a Named Query" on page 25-20).

17016: TIMEOUT

Cause: A time-out occurred–no message was received.

Action: Verify your interaction and the EIS on which you invoked it.

17017: INPUT_UNSUPPORTED_MSG_TYPE

Cause: Input record contains an unsupported message type.

Action: Verify your interaction and ensure that you specify a valid message type. Ensure that the required connector JAR file is on your classpath (see "EIS Projects" on page 17-7).

17018: INVALID_METHOD_INVOCATION

Cause: TopLink cannot invoke `begin` method on a nontransacted session.

Action: To be determined.

17019: TX_SESSION_TEST_ERROR

Cause: Problem testing for transacted session.

Action: To be determined.

17020: INVALID_AQ_INTERACTION_SPEC_TYPE

Cause: `InteractionSpec` is not an `AQInteractionSpec`.

Action: Verify your interaction and ensure that you specify a valid Oracle AQ interaction specification type (`AQInteractionSpec`). Ensure that the required connector JAR file is on your classpath (see "EIS Projects" on page 17-7).

17021: INVALID_AQ_RECORD_TYPE

Cause: `Record` is not an `AQRecord`.

Action: Verify your interaction and ensure that you specify a valid Oracle AQ record type (`AQRecord`). Ensure that the required connector JAR file is on your classpath (see "EIS Projects" on page 17-7).

17022: INVALID_AQ_INPUT

Cause: Invalid input: input must contain a single raw element.

Action: Verify your Oracle AQ interaction and ensure that you specify valid input–a single raw element (see "Configuring Custom EIS Interactions for Basic Persistence Operations" on page 28-6 or "Creating an EIS Interaction for a Named Query" on page 25-20).

17023: INVALID_FACTORY_ATTRIBUTES

Cause: An exception occurred setting `MQQueueConnectionFactory` attributes.

Action: Verify your interaction and ensure that you specify an appropriate IBM `MQSeries` connection factory (see "Configuring EIS Connection Specification Options at the Project Level" on page 21-2 or "Configuring EIS Connection Specification Options at the Session Level" on page 84-2).

17024: COULD_NOT_DELETE_FILE

Cause: TopLink cannot delete the specified file.

Action: To be determined.

17025: GROUPING_ELEMENT_REQUIRED

Cause: No grouping element is specified: this mapping requires a foreign key grouping element, as mulitple foreign keys exist.

Action: Verify your EIS mappings (see Chapter 53, "Understanding EIS Mappings".

90000: RESOURCE_EXCEPTION

Cause: Unknown resource.

Action: See the generated exception for the root cause.

91000: EIS_EXCEPTION

Cause: General EIS-related failure.

Action: See the generated exception for the root cause.

JMS Processing Exceptions (18001 - 18004)

`JMSProcessingException` is a run-time exception that is raised when processing JMS messages.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
```

Example 13–12 JMS Processing Exception

```
EXCEPTION [TOPLINK – 18001]: oracle.toplink.exceptions.JMSProcessingException
EXCEPTION DESCRIPTION: Error while processing incomming JMS message.
```

18001: DEFAULT

Cause: Failed to process the incoming JMS message.

Action: See the generated exception for the root cause.

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`.

18003: MDB_ERROR_LOOKUP_SESSION_NAME_ENV

Cause: Failure in looking up the session's name defined it the `env-entry` element of the message-driven bean.

Action: Verify the correctness of the `env-entry` element definition of the Message Driven Bean.

18004: MDB_FOUND_NO_SESSION

Cause: The message-driven bean (MDB) cannot find the session.

Action: Ensure that the session that this MDB's `getSession()` method returns is not `null` session.

Default Mapping Exceptions (20001 - 20008)

`DefaultMappingException` is a run-time exception that is raised when an error occurs during OC4J CMP default mapping.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
```

Example 13–13 Default Mapping Exception

```
EXCEPTION [TOPLINK – 20002]: oracle.toplink.exceptions.DefaultMappingException
EXCEPTION DESCRIPTION: The finder method with the parameters as defined in the
```

```
ejb-jar.xml file, is not found in the home of bean.
```

20001: FINDER_PARAMETER_TYPE_NOT_FOUND

Cause: TopLink could not find the parameter type, defined in the `ejb-jar.xml` file of the finder in the entity bean.

Action: Check the finder definition in the `ejb-jar.xml` file and the bean home to ensure that the named finder has the specified method parameter type(s).

20002: FINDER_NOT_DEFINED_IN_HOME

Cause: The finder method with the parameters as defined in the `ejb-jar.xml` file is not found in the home of bean.

Action: Check the `ejb-jar.xml` file and the bean home to ensure that the finder with the specified name and method parameter type(s) are defined.

20003: EJB_SELECT_NOT_DEFINED_IN_BEAN

Cause: The `ejbSelect` method with the parameters, as defined in the `ejb-jar.xml` file, is not found in the bean class.

Action: Check the `ejb-jar.xml` file and the bean class to ensure that the `ejbSelect` method is defined with the specified name and method parameter type(s).

20004: FINDER_NOT_START_WITH_FIND_OR_EJBSELECT

Cause: The finder method of bean in the `ejb-jar.xml` file is not well defined. It should start with either `find` or `ejbSelect`.

Action: Check the `ejb-jar.xml` file to ensure that the query name starts with `find` or `ejbSelect`, according to the EJB specification.

20005: GETTER_NOT_FOUND

Cause: The abstract `get` method is not defined in the bean.

Action: Check the `ejb-jar.xml` file and the bean class to ensure that the abstract `get` method is well defined.

20006: FIELD_NOT_FOUND

Cause: The CMP field is not defined in the bean.

Action: Check the `ejb-jar.xml` file and the bean class to ensure that the CMP field defined in the `ejb-jar.xml` file is also defined in the bean class.

For more information, see"Default Mapping in CMP Projects Using OC4J at Run Time" on page 30-4.

Discovery Exceptions (22001 - 22004)

`DiscoveryException` is a run-time exception that is raised when `DiscoveryManager` is operating.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
```

Example 13–14 Discovery Exception

```
EXCEPTION [TOPLINK – 22001]: oracle.toplink.exception.DiscoveryException
```

```
EXCEPTION DESCRIPTION: Could not join multicast group.
```

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 `joingGroup` method of the MulticastSocket failed.

Action: See the generated exception for the root cause.

22002: ERROR_SENDING_ANNOUNCEMENT

Cause: `DiscoveryManager` failed to inform other services that its service had 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 `setAnnouncementDelay` method of the `DiscoveryManager`.

22003: ERROR_LOOKING_UP_LOCAL_HOST

Cause: DiscoveryManager failed to do the lookup of a local host.

Action: See the generated exception for the root cause.

22004: ERROR_RECEIVING_ANNOUNCEMENT

Cause: `DiscoveryManager` caught a `java.io.IOException` while blocking for announcements from other `DiscoveryManagers`.

Action: See the generated exception for the root cause.

Remote Command Manager Exceptions (22101 - 22111)

`RemoteCommandManagerException` is a run-time exception that is raised when the remote command module is used.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
```

Example 13–15 Remote Command Manager Exception

```
EXCEPTION [TOPLINK – 22104]:
oracle.toplink.exceptions.RemoteCommandManagerException
EXCEPTION DESCRIPTION: Could not look up host name.
```

22101: ERROR_OBTAINING_CONTEXT_FOR_JNDI

Cause: Failure to get a JNDI context with the specified properties due to a `javax.naming.NamingException`.

Action: See the generated exception for root cause. Verify that the properties for looking up the context are correct.

22102: ERROR_BINDING_CONNECTION

Cause: Failure to post a connection in the local naming service with the name `serviceName`.

Action: See the generated exception for the root cause.

22103: ERROR_LOOKING_UP_REMOTE_CONNECTION

Cause: Failure to look up a remote connection with the specified name and URL.

Action: See the generated exception for the root cause. Verify that the remote connection and URL are correct.

22104: ERROR_GETTING_HOST_NAME

Cause: The `getLocalHost` method of the `java.net.InetAddress` failed to look up the specified host name.

Action: See the generated exception for the root cause. Verify that the host is online and reachable.

22105: ERROR_PROPAGATING_COMMAND

Cause: Failure to propagate a command to the specified connection.

Action: See the generated exception for the root cause. Verify that the remote host of the specified connection is online and reachable if the generated exception included a `CommunicationException`.

22106: ERROR_CREATING_JMS_CONNECTION

Cause: Failure to create JMS connection with `Topic`, `Topic Factory`, and `Context` properties.

Action: Verify that the JMS Topic is configured correctly with the application server and the specified Topic. Topic Factory and Context properties info can be used by a Java client to the JMS Topic.

22107: ERROR_UNBINDING_LOCAL_CONNECTION

Cause: Failure to establish a remove local connection in local naming service under name.

Action: Restart the application server or use the application server tool to remove name from its JNDI if this tool is available.

22108: ERROR_SERIALIZE_OR_DESERIALIZE_COMMAND

Cause: Failure to serialize or deserialize command.

Action: If the command is a user defined command, make sure it is serializable. If it is a TopLink command, file a bug report including the stack trace.

22109: ERROR_RECEIVING_JMS_MESSAGE

Cause: Failure to receive JMS message from JMS provider.

Action: Make sure that TopLink sessions are the only publishers to the JMS Topic and that the TopLink sessions use the same project.

22110: ERROR_DISCOVERING_IP_ADDRESS

Cause: Failure to discover local host IP address.

Action: Replace the `$HOST` string of the URL with the known host name or IP address.

22111: ERROR_GETTING_SERVERPLATFORM

Cause: Failure to get server platform. The server platform must be set on `Session` or `RemoteCommandManager`.

Action: Set the correct `ServerPlatform` on the `RemoteCommandManager`. This exception is raised when the user does not use a TopLink Session and implements the `CommandProcessor`.

Transaction Exceptions (23001 - 23015)

`TransactionException` is a run-time exception that is raised when an error is encountered during a transaction. When this occurs, the message contains a reference to the error code and error message.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
```

Example 13–16 Transaction Exception

```
EXCEPTION [TOPLINK – 23001]: oracle.toplink.exceptions.TransactionException
EXCEPTION DESCRIPTION: Error looking up external Transaction resource under JNDI
name.
```

23001: ERROR_DOING_JNDI_LOOKUP

Cause: Error looking up external transaction resource under JNDI name `name`.

Action: Examine the internal exception and take the appropriate action.

23002: ERROR_GETTING_TRANSACTION_STATUS

Cause: Error obtaining the status of the current externally-managed transaction.

Action: Examine the internal exception and take the appropriate action.

23003: ERROR_GETTING_TRANSACTION

Cause: Error obtaining the current externally managed transaction.

Action: Examine the internal exception and take the appropriate action.

23004: ERROR_BINDING_TO_TRANSACTION

Cause: Error obtaining the transaction manager.

Action: Examine the internal exception and take the appropriate action.

23005: ERROR_BEGINNING_TRANSACTION

Cause: Error binding to the externally managed transaction.

Action: Examine the internal exception and take the appropriate action.

23006: ERROR_COMMITTING_TRANSACTION

Cause: Error beginning a new externally managed transaction.

Action: Examine the internal exception and take the appropriate action.

23007: ERROR_ROLLING_BACK_TRANSACTION

Cause: Error committing the externally managed transaction.

Action: Examine the internal exception and take the appropriate action.

23008: ERROR_MARKING_TRANSACTION_FOR_ROLLBACK

Cause: Error rolling back the externally managed transaction.

Action: Examine the internal exception and take the appropriate action.

23009: ERROR_NO_TRANSACTION_ACTIVE

Cause: Error marking the externally managed transaction for rollback.

Action: Examine the internal exception and take the appropriate action.

23010: ERROR_INACTIVE_UOW

Cause: No externally managed transaction is currently active for this thread.

Action: Ensure that the transaction is still active.

23011: ERROR_OBTAINING_TRANSACTION_MANAGER

Cause: A `UnitOfWork` was rendered inactive before the associated externally managed transaction was complete.

Action: Examine the internal exception and take the appropriate action.

23012: ENTITY_TRANSACTION_WITH_JTA_NOT_ALLOWED

Cause: Attempt to use an `EntityTransaction` while using JTA. This is an invalid operation.

Action: Examine the internal exception and take the appropriate action.

23013: CANNOT_ENLIST_MULTIPLE_DATASOURCES

Cause: Attempt to enlist multiple data sources in the transaction.

Action: Enlist only one data source per transaction.

23014: EXCEPTION_IN_PROXY_EXECUTION

Cause: Exception occurred in `Proxy` execution.

Action: Examine the internal exception and take the appropriate action.

23015: ERROR_NO_ENTITY_TRANSACTION_ACTIVE

Cause: No entity transaction is currently active for this thread.

Action: Ensure that the transaction is active.

For more information, see Part XIX, "Transactions".

XML Conversion Exceptions (25501)

`XMLConversionException` is a run-time exception that is raised when a conversion between TopLink instances and XML fails. This exception is used in cache coordination that uses XML change sets.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
```

Example 13–17 XML Conversion Exception

```
EXCEPTION [TOPLINK – 25001]: oracle.toplink.exceptions.XMLConversionException
EXCEPTION DESCRIPTION: Cannot create URL for file  [\\FILE_SERVER\command.xml].
```

25501: ERROR_CREATE_URL

Cause: Failure to create a URL for the specified file.

Action: Ensure that all specified XPath strings on mappings are valid and correct.

Migration Utility Exceptions (26001 - 26020)

`MigrationUtilityException` is a run-time exception that is raised when an error is encountered during the use of the TopLink migration utility.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
```

Example 13–18 Migration Utility Exception

```
EXCEPTION [TOPLINK – 26002]: oracle.toplink.exceptions.MigrationUtilityException
EXCEPTION DESCRIPTION: The program security manager prevents the migration utility
from creating a JAR class loader for the JAR file.
```

26001: WLS_MULTIPLE_JARS_WITH_INPUT_ORION_NOT_SUPPORTED

Cause: WLS->OC4J TopLink migration exception: There are multiple migratable JARs in the input EAR file, and the `orion-ejb-jar.xml` is also provided in the input. The migration utility only supports a single migratable JAR file in an EAR file if the `orion-ejb-jar.xml` file is also in the input.

Action: Remove the `orion-ejb-jar.xml` file from the input directory, or keep only one migratable JAR file in the input EAR file.

26002: FAILED_TO_CREATE_JAR_CLASSLOADER

Cause: The program security manager prevents the migration utility from creating a JAR class loader for the JAR file.

Action: Configure the program security manager to grant permission on the program to create the JAR class loader.

26003: FAILED_TO_CREATE_DIRECTORY

Cause: The program security manager prevents the migration utility from creating a directory.

Action: Configure the program security manager to grant permission on the program to create a local directory.

26004: FILE_NOT_ACCESSIBLE

Cause: The file is not accessible by the migration utility.

Action: Check to ensure that the file is on the specified path and is readable.

26005: FILE_NOT_DELETABLE

Cause: The program security manager prevents the migration utility from deleting a directory.

Action: Configure the program security manager to grant permission to the program to delete files.

26006: FAILED_TO_READ_INPUTSTREAM

Cause: A `java.io.IOException` occurred when reading data from an input stream.

Action: Check to ensure that the file is not locked and is accessible.

26007: FAILED_TO_CLOSE_STREAM

Cause: A `java.io.IOException` occurred when closing data from an input or output stream.

Action: Check to ensure that the file is not locked and is accessible.

26008: FAILED_TO_CLOSE_ZIPFILE

Cause: A `java.io.IOException` occurred when closing data from a zip file.

Action: Check to ensure that the zip file is not locked and is accessible.

26009: JAR_FILE_NOT_ACCESSIBLE

Cause: The JAR file was not found or was not accessible on the specified path.

Action: Check to ensure that the JAR file is not locked and is accessible.

26010: QUERY_NOT_WELL_DEFINED

Cause: Query method with parameters of the entity defined in the `ejb-jar.xml` file is not well defined as it does not start with `find` or `ejbSelect`.

Action: Check the `ejb-jar.xml` file to ensure that the finder name starts with either `find` or `ejbSelect` according to the EJB spec.

26011: FAIL_TO_LOAD_CLASS_FOR_QUERY

Cause: The class defined as a parameter type of the query of the entity in `ejb-jar.xml` could not be loaded.

Action: Ensure that the bean class is included on the classpath.

26012: FINDER_NOT_DEFINED_IN_ENTITY_HOME

Cause: The `finder` method with parameters defined in the `ejb-jar.xml` file is not defined at the entity's local and/or remote home interface.

Action: Check the `ejb-jar.xml` file and the bean home to ensure that the `finder` method parameter types are properly defined.

26013: EJB_SELECT_NOT_DEFINED_IN_ENTITY_BEAN_CLASS

Cause: The `ejbSelect` method with parameters defined in the `ejb-jar.xml` file is not defined at the entity's bean class.

Action: Check the `ejb-jar.xml` file and the bean class to ensure that the `ejbSelect` method parameter types are properly defined.

26014: ENTITY_IN_WLS_CMP_JAR_NOT_DEFINED_IN_WLS_EJB_JAR

Cause: The entity specified in the `weblogic-cmp-jar.xml` file is not defined in the `weblogic-ejb-jar.xml` file.

Action: Ensure that the entity is consistently defined in the deployment descriptor files.

26015: NO_ENTITY_DEFINED_IN_WLS_CMP_JAR

Cause: No entity element is defined in the `weblogic-cmp-rdbms-jar.xml` file.

Action: Ensure that all entities defined in the `ejb-jar.xml` file are mapped in the WebLogic CMP descriptor file.

26016: WLS_CMP_DESCRIPTOR_FILE_NOT_FOUND

Cause: The WebLogic CMP descriptor file specified in the `weblogic-ejb-jar.xml` file was not found in the directory.

Action: Ensure that the WebLogic CMP descriptor file, specified in the `weblogic-ejb-jar.xml` file, is included in the input directory.

26017: CMP_DESCRIPTOR_NOT_FOUND_IN_JAR

Cause: The CMP descriptor file is not found in the JAR file to be migrated.

Action: Ensure that the WebLogic CMP descriptor file, specified in the `weblogic-ejb-jar.xml` file, is included in the input archive JAR file.

26018: NOT_ALL_ENTITIES_IN_EJB_JAR_MAPPED_IN_ORION

Cause: Some of the entity beans `EntityBeanClassName`-s defined in the `ejb-jar.xml` file in the jar `jarName` are not explicitly mapped in the `orion-ejb-jar.xml` file.

Action: Provide the completely mapped `orion-ejb-jar.xml` file to the migration tool. You can obtain the completely mapped `orion-ejb-jar.xml` file from the `/application-deployment` directory after deploying the application.

26019: ENTITY_NOT_MAPPED_IN_ORION

Cause: The entity `EntityName` in the `orion-ejb-jar.xml` file is not mapped, because no table is specified.

Action: Provide the completely mapped `orion-ejb-jar.xml` file to the migration tool. You can obtain the completely mapped `orion-ejb-jar.xml` file from the `/application-deployment` directory after deploying the application.

26020: ENTITY_IN_ORION_NOT_IN_EJB_JAR

Cause: The entity EntityName specified in the `orion-ejb-jar.xml` file is not defined in the `ejb-jar.xml` file.

Action: Define this entity in the `ejb-jar.xml` file.

For more information, see Chapter 7, "Integrating TopLink With an Application Server" on page 7-1.

EJB JAR XML Exceptions (72000 – 72023)

`EJBJARXMLException` is a run-time exception that is raised at deployment time when the `ejb-jar.xml` file is read and the required concrete EJB classes code is generated.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
```

Example 13–19 EJB JAR XML Exception

```
EXCEPTION [TOPLINK – 72000]: oracle.toplink.exceptions.EJBJarXMLException
EXCEPTION DESCRIPTION: Error reading ejb-jar.xml file.
```

72000: READ_EXCEPTION

Cause: Failure to read an `ejb-jar.xml` file due to a `java.io.IOException` or `javax.xml.parsers.ParserConfigurationException`.

Action: See the generated exception for the root cause.

72001: INVALID_DOC_TYPE

Cause: Failure 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 documentation type.

72002: CONCRETE_INSTANCE_VARIALBE_EXISTS

Cause: There was a concrete instance variable `variableName` already defined on the class `ClassName`.

Action: Either remove this instance variable from the class, refresh in the TopLink Workbench, and update from the `ejb-jar.xml` file again, or remove the field from the `ejb-jar.xml` file and update again.

72003: EJB_2_0_ATTRIBUTE_NOT_EXIST

Cause: A mapping for a field `fieldName` could not be created on the descriptor `DescriptorName`.

Action: Check to ensure that there are abstract `get` and `set` methods accessible for this field.

72004: NEITHER_HOME_NOR_REMOTE_INTERFACE_FOUND

Cause: Neither local home nor remote home interface for the class ClassName was found on the classpath. The finders for this class were not updated

Action: Update the finders for this class; ensure that either local home or remote home interface for the class is on the classpath.

72005: FINDER_NOT_EXIST_ON_REMOTE_HOME_AND_LOCAL_HOME

Cause: The finder method methodName was not defined on a home interface for the entity `EntityName`. The finder was not updated

Action: Define the finder method.

72006: NO_PERSISTENCE_TYPE_SPECIFIED

Cause: TopLink Workbench could not determine a project persistence type from the XML file.

Action: Specify the persistence type for your TopLink project.

72007: SELECT_NOT_DEFINED_IN_BEANCLASS

Cause: Failure to find a corresponding `ejbSelect` method in the bean class `ClassName`. No information was updated for this query.

Action: Define `ejbSelect` method in your bean class.

72008: EJB_DESCRIPTOR_MUST_HAVE_EJB_NAME

Cause: The EJB descriptor `DescriptorName` does not have an EJB name specified.

Action: In your EJB descriptor, specify an EJB name.

72009: EJB_DESCRIPTOR_MUST_HAVE_PRIMARYKEY_CLASS

Cause: The EJB descriptor `DescriptorName` does not have a primary key class specified.

Action: In your EJB descriptor, specify a primary key class.

72010: EJB_NAME_MUST_BE_UNIQUE

Cause: The EJB name `EJBName` was used by more than one descriptor. EJB names must be unique

Action: Ensure that the EJB name is specified in one and only one descriptor.

72011: EMPTY_TEXT_ATTRIBUTE

Cause: An element `elementName` has an empty text attribute `attributeName`.

Action: Use a non-empty text attribute for the subelement.

72012: MULTIPLE_ENTITIES_FOUND_FOR_EJB_NAME

Cause: There are multiple entities with the same EJB bean name `BeanName` in the XML file.

Action: Use a unique EJB bean name for each entity.

72013: INVALID_CMP_VERSION

Cause: The entity EntityName does not have a valid `cmp-version` definition. The `cmp-version` must be `1.x` or `2.x`.

Action: Use a `cmp-version` value of `1.x` or `2.x` only.

72014: INVALID_EJB_NAME_FOR_RELATIONSHIP_ROLE

Cause: The EJB bean name BeanName in the `ejb-relationship-role` element is not found in the XML file.

Action: Use the EJB bean name of an existing entity for the `ejb-relationship-role`.

72015: INVALID_MULTIPLICITY

Cause: There was an invalid multiplicity value defined for a relationship involving the mapping `Mapping` Name in a descriptor `DescriptorName`. The multiplicity must be either `One` or `Many`.

Action: Use a multiplicity value of `One` or `Many` only.

72016: INVALID_PERSISTENCE_TYPE

Cause: The entity EntityName does not have a valid persistence type. The type must be either `Bean` or `Container`.

Action: Use a `persistence-type` value of `Bean` or `Container` only.

72017: INVALID_QUERY_METHOD_NAME

Cause: The query method `methodName` does not start with `find` or `ejbSelect`. No information was updated for this query.

Action: Verify that the query method name starts with either `find` or `ejbSelect`.

72018: NOT_SINGLE_PERSISTENCE_TYPE

Cause: Attempt to use a nonuniform persistence type for the TopLink Workbench project: TopLink Workbench requires that all entities in the XML file for this project have the same `persistence-type` and/or `cmp-version`. The project will be set according to the `persistence-type` and/or `cmp-version` of the first entity in the file.

Action: Provide all entities in the XML file for this project with the same `persistence-type` and/or `cmp-version`.

72019: PROJECT_MUST_HAVE_AT_LEAST_ONE_EJB_DESCRIPTOR

Cause: The project `ProjectName` does not have an EJB descriptor: it must have at least one EJB descriptor.

Action: Provide an EJB descriptor for the project.

72020: ATTRIBUTE_NOT_EXIST

Cause: The attribute `attributeName` is not found.

Action: Define this attribute.

72021: EJB_CLASS_NOT_FOUND

Cause: The class `ClassName` could not be found on the classpath. A descriptor could not be created for it.

Action: Ensure that the class `ClassName` is on the classpath.

72022: REQUIRED_ATTRIBUTE_NOT_EXIST

Cause: There is an element that does not have a required attribute.

Action: Add the subelement to the parent element.

72023: NO_CMR_FIELD_FOR_BEAN_ABSTRACT_SETTER

Cause: Entity bean BeanName has an abstract setter method with an `EntityBean` attribute type for attribute `attributeName`, but does not have a corresponding CMR field.

Action: Verify that all CMP entities have the same `cmp-version` value.

Entity Manager Setup Exceptions (28001 – 28007)

`EntityManagerSetupException` is raised when an error is encountered during the process of setting up an entity manager.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
```

Example 13–20 Entity Manager Setup Exception

```
EXCEPTION [TOPLINK – 28001]: oracle.toplink.exceptions.EntityManagerSetupException
EXCEPTION DESCRIPTION: Error while trying to create session.
```

28001: SESSIONS_XML_VALIDATION_EXCEPTION

Cause: A `ValidationException` was thrown while trying to create a session `sessionName`.

Action: Ensure that your `fileName` file is on the classpath. If it is on the classpath, ensure that this file contains the session `sessionName`.

28002: WRONG_SESSION_TYPE_EXCEPTION

Cause: Failure to load a `ServerSession` `sessionName` from `sourceName`.

Action: Ensure that the server session's type is correct.

28003: MISSING_SERVER_PLATFORM_EXCEPTION

Cause: Server platform specification was not found: TopLink has loaded a session `sessionName` from `sourceName`, but this session either does not have a server platform specified, or specifies a server platform that does not use an external transaction controller.

Action: If you plan to use JTA, specify an appropriate server platform.

28004: ERROR_IN_SETUP_OF_EM

Cause: Error while setting up an `oracle.toplink.sessions.entitymanager.EntityManagerFactory`.

Action: Ensure the correct setup of the `EntityManagerFactory`: `initializeFromMain` method of the `EntityContainer` should return `true`.

28005: EXCEPTION_IN_SETUP_OF_EM

Cause: Error during the setup of `oracle.toplink.sessions.entitymanager.EntityManagerFactory`.

Action: See the generated exception for the root cause.

28006: CLASS_NOT_FOUND_FOR_PROPERTY

Cause: Failure to find the class `ClassName` specified in the property `propertyName`.

Action: Ensure that the class exists and is on the classpath.

28007: FAILED_TO_INSTANTIATE_SERVER_PLATFORM

Cause: Failure to instantiate a server platform of type `ServerPlatformType` specified in the property `propertyName`.

Action: See the generated exception for the root cause.

XML Marshal Exceptions (25001 – 25020)

`XMLMarshalException` is raised when an error is encountered during the XML marshalling process.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
```

Example 13–21 XML Marshal Exception

```
EXCEPTION [TOPLINK – 25001]: oracle.toplink.exceptions.XMLMarshalException
EXCEPTION DESCRIPTION: Error while trying to create session.
```

25001: INVALID_XPATH_STRING

Cause: Attempt to use an invalid XPath `string`.

Action: Ensure that the XPath is specified correctly.

25002: INVALID_XPATH_INDEX_STRING

Cause: Attempt to parse an integer index from the XPath string `xpathString`.

Action: Ensure that the XPath is specified correctly.

25003: MARSHAL_EXCEPTION

Cause: Error occurred during the object marshalling.

Action: See the generated exception for the root cause.

25004: UNMARSHAL_EXCEPTION

Cause: Error while setting up an `oracle.toplink.sessions.entitymanager.EntityManagerFactory`.

Action: Ensure the correct setup of the `EntityManagerFactory`: `initializeFromMain` method of the `EntityContainer` should return `true`.

25005: VALIDATE_EXCEPTION

Cause: Validation error.

Action: See the generated exception for the root cause.

25006: DEFAULT_ROOT_ELEMENT_NOT_SPECIFIED

Cause: A default root element is not specified for the `XMLDescriptor` mapped to `target`.

Action: Specify the default root element for the descriptor.

25007: DESCRIPTOR_NOT_FOUND_IN_PROJECT

Cause: A descriptor for class `ClassName` was not found in the project.

Action: Ensure that the descriptor for this class exists.

25008: NO_DESCRIPTOR_WITH_MATCHING_ROOT_ELEMENT

Cause: A descriptor with the default root element `element` was not found in the project.

Action: Ensure that the descriptor for this class exists and that its default root element is specified.

25010: SCHEMA_REFERENCE_NOT_SET

Cause: A schema reference is not specified for the XMLDescriptor mapped to target.

Action: Specify the schema reference for the descriptor.

25011: NULL_ARGUMENT

Cause: A `null` argument was encountered.

Action: Ensure that this argument is not `null`.

25012: ERROR_RESOLVING_XML_SCHEMA

Cause: Failure to resolve the XML schema.

Action: See the generated exception for the root cause.

25013: ERROR_SETTING_SCHEMAS

Cause: Failure to set schemas.

Action: See the generated exception for the root cause.

25014: ERROR_INSTANTIATING_SCHEMA_PLATFORM

Cause: Error during the schema platform instantiation.

Action: See the generated exception for the root cause.

25015: NAMESPACE_RESOLVER_NOT_SPECIFIED

Cause: Failure to resolve the namespace URI for target.

Action: Specify a namespace resolver on the descriptor

25016: NAMESPACE_NOT_FOUND

Cause: A namespace for the prefix `prefix` was not found in the namespace resolver.

Action: Ensure that the namespace for this prefix exists in the namespace resolver.

25017: ENUM_CLASS_NOT_SPECIFIED

Cause: Enumeration class is not specified.

Action: Set either `enumClass` or `enumClassName` on the `JAXBTypesafeEnumConverter`.

25018: FROMSTRING_METHOD_ERROR

Cause: Error during the invocation of the `fromString` method on the enumeration class `ClassName`: the method does not exist or could not be invoked.

Action: See the generated exception for the root cause.

25019: INVALID_ENUM_CLASS_SPECIFIED

Cause: The specified enumeration class `ClassName` was not found.

Action: Ensure the correct specification of the enumeration class.

25020: ILLEGAL_STATE_XML_UNMARSHALLER_HANDLER

Cause: Attempt to call the method `getResult` before the call to the `endDocument` event.

Action: Call the `getResult` method after the `endDocument` event has been called.

For more information, see "Understanding XML Mappings" on page 62-1.

XML Platform Exceptions (27001 – 27006, 27101 – 27103, 27201 – 27202)

`XMLPlatformException` is raised when an error related to XML platform is encountered.

Format

```
EXCEPTION [TOPLINK – error code]: Exception name
EXCEPTION DESCRIPTION: Message
```

Example 13–22 EJB JAR XML Exception

```
EXCEPTION [TOPLINK – 27001]: oracle.toplink.platform.xml.XMLPlatformException
EXCEPTION DESCRIPTION: The XML platform class ClassName was not found.
```

27001: XML_PLATFORM_CLASS_NOT_FOUND

Cause: The XML platform class `ClassName` was not found.

Action: Ensure that this class is on the classpath.

27002: XML_PLATFORM_COULD_NOT_INSTANTIATE

Cause: Failure to instantiate the XML platform `ClassName.`

Action: See the generated exception for the root cause.

27003: XML_PLATFORM_COULD_NOT_CREATE_DOCUMENT

Cause: The XML platform failed to create a new XML document.

Action: See the generated exception for the root cause.

27004: XML_PLATFORM_INVALID_XPATH

Cause: The XPath is invalid.

Action: See the generated exception for the root cause.

27005: XML_PLATFORM_VALIDATION_EXCEPTION

Cause: Error during the document validation.

Action: See the generated exception for the root cause.

27006: XML_PLATFORM_PARSER_ERROR_RESOLVING_XML_SCHEMA

Cause: Failure to resolve an XML Schema `schema`: XML platform parser error occurred.

Action: See the generated exception for the root cause.

27101: XML_PLATFORM_PARSE_EXCEPTION

Cause: Failure to parse the document.

Action: See the generated exception for the root cause.

27102: XML_PLATFORM_PARSER_FILE_NOT_FOUND_EXCEPTION

Cause: Failure to find the XML platform parser file.

Action: Ensure that the parser file exists and is accessible (on the classpath).

27103 XML_PLATFORM_PARSER_SAX_PARSE_EXCEPTION

Cause: SAX parser error at line `lineNumber`, URI `uri`.

Action: See the generated exception for the root cause.

27201: XML_PLATFORM_TRANSFORM_EXCEPTION

Cause: Error during the transforming of the document.

Action: See the generated exception for the root cause.

27202: XML_PLATFORM_INVALID_TYPE

Cause: Unknown type `type` encountered.

Action: See the generated exception for the root cause.

14

TopLink Workbench Error Reference

TopLink checks each project, descriptor, and mapping to ensure that you have properly defined the required settings. Errors and warnings are displayed in the Problems window (see "Using the Problems Window" on page 4-11).

You can also create a project status report (see "Generating the Project Status Report" on page 18-12) that contains all errors in a specific project.

This chapter contains information on the following:

- Oracle TopLink Workbench Error Messages:
  - Miscellaneous Errors (1 – 89, 106 – 133), on page 14-1
  - Project Errors (100 – 102), on page 14-3
  - Descriptor Errors (200 – 399), on page 14-3
  - Mapping Errors (400 – 483), on page 14-13
  - Table Errors (500 – 610), on page 14-17
  - XML Schema Errors (700 – 706), on page 14-23
  - Session Errors (800 – 812), on page 14-23
- Common Classpath Problems, on page 14-25
- Data Source Problems, on page 14-25

Miscellaneous Errors (1 – 89, 106 – 133)

This section lists TopLink Workbench error codes, information about the likely **Cause** of the problem, and a possible corrective **Action**.

13: No class indicator field should be defined for the abstract class *[class]*.

Cause: Abstract classes should not be included or contain an **Indicator Value** on a descriptor's **Inheritance** tab.

Action: You must either remove the **Include** option for the class on the **Inheritance** tab, or remove the **abstract** modifier option on the descriptor's **Class Info – Class** tab. See "Descriptors and Inheritance" on page 23-3 and Chapter 25, "Configuring a Descriptor" on page 25-1.

54: No class indicator field is selected for this root class.

Cause: You selected the **Use Class Indicator Field** option for the root descriptor in the inheritance hierarchy, but did not specify an indicator field for the root and its children.

Action: Use the **Field Selection** list on the **Inheritance** tab for the root class. See "Descriptors and Inheritance" on page 23-3 and Chapter 25, "Configuring a Descriptor" on page 25-1.

55: No class indicator value is defined for this included descriptor *[class]*

Cause: You selected the **Use Class Indicator Dictionary** option for the root descriptor in the inheritance hierarchy, but did not specify an indicator value for the root and its children.

Action: Use the **Indicator Type** list on the **Inheritance** tab for the root class. See "Descriptors and Inheritance" on page 23-3 and Chapter 25, "Configuring a Descriptor" on page 25-1.

89: Root class does not include an indicator mapping for this descriptor.

Cause: The root class in the inheritance hierarchy is set to the **Use Class Indicator Dictionary** option. The dictionary does not contain an indicator value for this child class.

Action: Select an **Indicator Type** on the **Inheritance** tab of the root class that includes the child types. See "Descriptors and Inheritance" on page 23-3 and Chapter 25, "Configuring a Descriptor" on page 25-1.

106: Mulitple mapping *[mappings]* write to the database field *[db field]* .

Cause: One database field is populated by more than one mapping

Action: Ensure the "one mapping per field" ratio for write operations.

118: The selected parent descriptor for this descriptor's inheritance policy does not have an associated inheritance policy.

Cause: Missing descriptor's inheritance policy.

Action: Ensure that the descriptor you are using has a valid associated inheritance policy (`InheritancePolicy`). See "Descriptors and Inheritance" on page 23-3 and "Configuring Inheritance for a Parent (Root) Descriptor" on page 25-52.

123: This root class has no class indicator mappings for its hierarchy.

Cause: You created an inheritance policy with the **Use Class Indicator Dictionary** option but did not specify the indicator values for all subclasses.

Action: Specify the indicator values for all subclasses on the descriptor's **Inheritance** tab. See "Descriptors and Inheritance" on page 23-3 and "Configuring Inheritance for a Child (Branch or Leaf) Class Descriptor" on page 25-51.

Note: TopLink displays a list of each subclass and indicator value if you have identified the subclasses' parent descriptor.

126: Writable mappings defined for the class indicator field *[field name]*.

Cause: You selected the **Use Class Indicator Field** option for the root descriptor in the inheritance hierarchy, but the mappings for this field are writable.

Action: Select a **Use Class Indicator Field** on the descriptor's **Inheritance** tab that does not contain any writable mappings. See the following:

- "Descriptors and Inheritance" on page 23-3
- Chapter 25, "Configuring a Descriptor" on page 25-1
- "Configuring Inheritance for a Child (Branch or Leaf) Class Descriptor" on page 25-51

- "Configuring Inherited Attribute Mapping in a Subclass" on page 25-58

132: The implemented interface *[interface]* is not an interface.

Cause: You selected a noninterface (a class) as an implemented interface.

Action: Ensure that you select an interface. See "Descriptors and Inheritance" on page 23-3 and Chapter 25, "Configuring a Descriptor" on page 25-1.

133: The superclass for [class] is an interface, classes cannot extend interfaces.

Cause: You selected an interface instead of a class as a parent for your child class.

Action: Use the **Inheritance** tab for the root class. See "Descriptors and Inheritance" on page 23-3 and Chapter 25, "Configuring a Descriptor" on page 25-1.

Project Errors (100 – 102)

This section lists TopLink Workbench project errors.

100: The project caches all statments by default for queries, but does not bind all parameters.

Cause: A named query that caches statements must also bind parameters.

Action: On the **Named Queries – Options** tab, change the **Cache Statement** field to **False**, or change the **Bind Parameters** field to **True**. See "Configuring Named Query Options" on page 25-22.

101: The project uses a custom sequence table, but the counter field is not specified.

Cause: On the project's **Sequencing** tab, you selected **Use Custom Sequence Table**, but did not complete the **Counter Field** field.

Action: Select the field to use as the Counter Field for this sequence table. See "Configuring Sequencing at the Project Level" on page 20-3 for details.

102: The project uses a custom sequence table, but the name field is not specified.

Cause: On the project's **Sequencing** tab, you selected **Use Custom Sequence Table**, but did not complete the **Name Field** field.

Action: Select the field to use as the **Name Field** for this sequence table. See "Configuring Sequencing at the Project Level" on page 20-3 for details.

Descriptor Errors (200 – 399)

This section lists TopLink Workbench descriptor errors.

200: The descriptor's class is not public.

Cause: The descriptor must use a public access modifier.

Action: On the descriptor's **Class Info – Class** tab, change the **Access Modifier** option to **Public**. See "Configuring Classes" on page 4-41 and "Configuring Class Modifiers" on page 4-42.

201: This class is a subclass of a final class.

Cause: If you select the **Final** option on the descriptor's **Class Info – Class** tab for a class, then the class cannot contain subclasses.

Action: See "Configuring Classes" on page 4-41 and "Configuring Class Information" on page 4-41.

210: Two methods *[method name1] [method name2]* cannot have the same signature.

Cause: You created methods with identical signatures.

Action: On the **Class Info – Methods** tab, change the information for one of the methods. See "Configuring Classes" on page 4-41 and "Adding Methods" on page 4-47.

211: The format for *[date]* must be in the format HH-MM-SS or HH:MM:SS. Literal argument of expression *[line number]* on query *[query name]* is invalid.

Cause: You attempted to use an invalid argument on a query.

Action: Use HH-MM-SS or HH:MM:SS format. See "Type Conversion Converter" on page 30-11 and "Configuring a Type Conversion Converter" on page 32-20.

212: The format for *[date]* must be in the format YYYY/MM/DD HH:MM:SS or YYYY-MM-DD HH:MM:SS. Literal argument of expression *[line number]* on query *[query name]* is invalid.

Cause: You attempted to use an invalid argument on a query.

Action: Use YYYY/MM/DD HH:MM:SS or YYYY-MM-DD HH:MM:SS format. See "Type Conversion Converter" on page 30-11 and "Configuring a Type Conversion Converter" on page 32-20.

213: The format for *[date]* must be in the format YYYY/MM/DD or YYYY-MM-DD. Literal argument of expression *[line number]* on query *[query name]* is invalid.

Cause: You attempted to use an invalid argument on a query.

Action: Use YYYY/MM/DD or YYYY-MM-DD format. See "Type Conversion Converter" on page 30-11 and "Configuring a Type Conversion Converter" on page 32-20.

214: The format for *[date]* must be in the format YYYY/MM/DD HH:MM:SS, YYYY/MM/DD, or YYYY-MM-DD. Literal argument of expression *[line number]* on query *[query name]* is invalid.

Cause: You attempted to use an invalid argument on a query.

Action: Use YYYY/MM/DD HH:MM:SS, YYYY/MM/DD, or YYYY-MM-DD format. See "Type Conversion Converter" on page 30-11 and "Configuring a Type Conversion Converter" on page 32-20.

215: The format for *[argument]* must be an even length HEX string. Literal argument of expression *[line number]* on query *[query name]* is invalid.

Cause: You attempted to use an invalid argument on a query.

Action: Use HEX format. See "Type Conversion Converter" on page 30-11.

216: The format for *[argument]* must be a string. Literal argument of expression *[line number]* on query *[query name]* is invalid.

Cause: You attempted to use an invalid argument on a query.

Action: Use a `String`. See "Type Conversion Converter" on page 30-11.

217: Literal argument of expression *[line number]* on query *[query name]* is invalid. The format is illegal.

Cause: You attempted to use an invalid argument on a query.

Action: Use a valid format.

220: An aggregate shared by multiple source descriptors cannot have one-to-many or many-to-many mappings.

Cause: You attempted to create multiple one-to-many and many-to-many, or one-to-one mappings in which the target is the aggregate. Aggregate descriptors

that are shared by multiple source descriptors cannot have mappings that contain a target object that references the descriptor.

Action: For aggregate descriptors that are shared by multiple source descriptors, remove mappings that contain a target object that references the descriptor. See Part VIII, "Descriptors" and "Relational Aggregate Descriptors" on page 24-2.

221: Classes cannot reference an aggregate target with one-to-one, one-to-many, or many-to-many mappings.

Cause: You tried to select an aggregate descriptor as a reference.

Action: Do not select an aggregate descriptor as the **Reference Descriptor** for a one-to-one, one-to-many, or many-to-many mapping. See Part VIII, "Descriptors" and "Relational Aggregate Descriptors" on page 24-2.

225: The implementor *[implementor name]* no longer implements this interface.

Cause: One descriptor listed as an implementation method for this interface descriptor no longer implements this descriptor's interface.

Action: Either remove the descriptor from the list of implementation methods or alter the descriptor's class so that it implements this descriptor's interface. See Part VIII, "Descriptors" and "Relational Interface Descriptors" on page 24-2.

230: No primary table is specified.

Cause: The descriptor is not associated with a database table.

Action: On the descriptor's **Descriptor Info** tab, use the **Associated Table** field to select a primary table. See "Configuring Associated Tables" on page 26-2.

231: No primary key(s) specified in *[table name]* table.

Cause: You did not specify a primary key for each database table. When importing tables from a database into TopLink Workbench, the primary key information will be retained only if the JDBC driver supports the `getPrimaryKeys` method.

Action: Ensure that a primary key is specified for each descriptor on the **Descriptor Info** tab. See "Configuring Associated Tables" on page 26-2.

232: The following primary key field is unmapped *[field name]*.

Cause: The primary key field does not have a writable mapping.

Action: Ensure that the primary key(s) are mapped. See "Configuring Associated Tables" on page 26-2.

233: The number of primary keys does not match the number of primary keys on the parent.

Cause: In an inheritance hierarchy, the child class does not have the same number of primary keys as the parent class.

Action: Ensure that the parent and child class have the same number of primary keys on the descriptor's **Descriptor Info** tab. See "Configuring Associated Tables" on page 26-2.

234: The primary keys do not match parent's primary keys.

Cause: In an inheritance hierarchy, the child's primary key(s) does not match the root's primary key(s).

Action: Ensure that each child's **Primary Key** on the **Descriptor Info** tab matches the parent's primary key. Ensure that the parent and child class have the same primary keys on the descriptor's **Descriptor Info** tab. See "Configuring Associated Tables" on page 26-2.

235: The following primary field field does not have writable mappings: *[field name]*.

Cause: You attempted to have multiple mappings write to the same database field.

Action: Ensure that each database field has a single, writable mapping. See Chapter 26, "Configuring a Relational Descriptor".

236: No sequence field is selected.

Cause: You selected **Use Sequencing** on a descriptor's **Descriptor Info** tab, but did not specify the sequence information.

Action: Specify a **Name**, **Table**, and **Field**. See "Configuring Sequencing at the Descriptor Level" on page 26-3.

237: No sequence name is selected.

Cause: You selected **Use Sequencing** on a descriptor's **Descriptor Info** tab but did not specify the sequence information.

Action: Specify a **Name**, **Table**, and **Field**. See "Configuring Sequencing at the Descriptor Level" on page 26-3.

238: No sequence table is selected.

Cause: You selected **Use Sequencing** on a descriptor's **Descriptor Info** tab but did not specify the sequence information.

Action: Specify a **Name**, **Table**, and **Field**. See "Configuring Sequencing at the Descriptor Level" on page 26-3.

239: The selected sequence table is not one of the descriptor's associated tables.

Cause: The tabled used for sequencing is not associated with the descriptor.

Action: You must either associate the sequencing table with the descriptor, or select a table that is already associated with the descriptor. See"Configuring Sequencing at the Descriptor Level" on page 26-3 and "Configuring Associated Tables" on page 26-2.

240: Two queries *[query name1] [query name2]* cannot have the same signature.

Cause: Two queries for this descriptor share the same signature (query name + parameter names). This is not allowed.

Action: You must either remove one of the queries, rename one of the queries, or change the parameters so that the signatures no longer match.

241: The query *[query name]* has Cache Statement set to true, but does not bind parameters.

Cause: A named query that caches statements does not bind parameters. It must do so.

Action: On the **Named Queries – Options** tab, either change the **Cache Statements** field to **False**, or change the **Bind Parameters** field to **True**. See "Configuring Named Query Options" on page 25-22.

242: The query *[query name]* does not maintain cache but does refresh the remote identity map results.

Cause: The query has **Refresh Remote Identity Map** selected, but does not have **Maintain Cache** selected.

Action: You must either select **Maintain Cache** for the descriptor, or deselect **Refresh Remote Identity Map**. See "Configuring Named Query Options" on page 25-22.

243: The query *[query name]* does not maintain cache but does refresh the identity map results.

Cause: The query has **Refresh Identity Map** selected but does not have **Maintain Cache** selected.

Action: You must either select **Maintain Cache** for the descriptor, or deselect **Refresh Identity Map**. See "Configuring Named Query Options" on page 25-22.

245: The query *[query name]* refreshes identity map results but does not refresh remote identity map results.

Cause: **Refresh Identity Map Results** is selected for the query, but **Refresh Remote Identity Map Results** is not.

Action: You must either select **Refresh Remote Identity Maps** or deselect **Refresh Identity Maps**. See "Configuring Named Query Options" on page 25-22.

246: The query key *[query key]* does not have an associated database field.

Cause: The query key is missing an associated database field. Each query key must be associated with a database field.

Action: On the **Query Keys** tab, use the **Field** option to select a database field for the query key. See "Configuring Query Keys" on page 25-30.

247: The database field selected for query key *[query key]* does not exist on this descriptor's associated tables.

Cause: The database field selected for this query key does not exist on this descriptor's associated tables. Each database field associated with a query key must exist on database table associated with the query key's descriptor.

Action: You must either change the database field associated with the query key, or associate the descriptor with a database table that includes the database field associated with the query key. See "Configuring Query Keys" on page 25-30.

248: The expression *[line number]* on query *[query name]* is invalid because a parameter has not been specified.

Cause: One of the arguments in the query expression is missing or invalid.

Action: Edit the query and ensure that all query keys and parameters have been specified. See "Configuring Query Keys" on page 25-30.

249: The expression *[line number]* on query *[query name]* is invalid because a query key has not been specified.

Cause: One of the arguments in the query expression is missing or invalid.

Action: Edit the query and ensure that all query keys and parameters have been specified. See "Configuring Query Keys" on page 25-30.

250: The expression *[line number]* on query *[query name]* is invalid because the chosen query key is not a valid mapping type in an expression.

Cause: One of the arguments in the query expression is invalid.

Action: Edit the query and ensure that all query keys and parameters have been specified. See "Configuring Query Keys" on page 25-30.

251: The expression *[line number]* on query *[query name]* is invalid. When querying on a reference mapping, only unary operators (Is Null, Not Null) are supported.

Cause: You created an expression node that includes a reference mapping with an invalid operator.

Action: On the **Expression Builder** dialog box, select the node and change the **Operator** field to **IS NULL** or **NOT NULL**.

252: The query *[query name]* has no attribute chosen for the ordering attribute at index *[index]*.

Cause: To be determined.

Action: To be determined.

253: The ordering attribute {0} for query {1} is not valid. ReadAllQuery ordering items must be either query keys or direct mappings.

Cause: To be determined.

Action: To be determined.

254: The query {0} has no attribute chosen for the joined attribute at index {1}.

Cause: To be determined.

Action: To be determined.

255: The joined attribute {0} for query {1} is not valid. Joined attributes must be 1-1 or 1-many mappings.

Cause: To be determined.

Action: To be determined.

256: The query {0} has no attribute chosen for the batch read attribute at index {1}

Cause: To be determined.

Action: To be determined.

257: The batch read attribute {0} for query {1} is not valid. Batch read attributes must be 1-1, 1-m, m-m, direct collection, or direct map mappings.

Cause: To be determined.

Action: To be determined.

258: The query {0} has no attribute chosen for the grouping attribute at index {1}.

Cause: To be determined.

Action: To be determined.

259: The query {0} has no attribute chosen for the report attribute {1}.

Cause: To be determined.

Action: To be determined.

260: The report attribute {0} for query {1} is not valid. Report query attributes must be either query keys or direct mappings.

Cause: To be determined.

Action: To be determined.

261: The expression *[line number]* on query *[query name]* is invalid. When querying on a reference mapping, only unary operators (Is Null, Not Null) are supported.

Cause: To be determined.

Action: To be determined.

263: The format for {2} must be either 'true' or 'false'. Literal argument of expression (line {0}) on query {1} is invalid.

Cause: To be determined.

Action: To be determined.

264: The format for {2} must be a single character. Literal argument of expression (line {0}) on query {1} is invalid.

Cause: To be determined.

Action: To be determined.

265: The format for {2} must be between {3} and {4}. Literal argument of expression (line {0}) on query {1} is invalid.

Cause: To be determined.

Action: To be determined.

266: The format for {2} must be a string. Literal argument of expression (line {0}) on query {1} is invalid.

Cause: To be determined.

Action: To be determined.

267: The format for {2} must contain only digits, '-', and '.'. Literal argument of expression (line {0}) on query {1} is invalid.

Cause: To be determined.

Action: To be determined.

268: The format for {2} must contain only digits, '-', and '.'. Literal argument of expression (line {0}) on query {1} is invalid.

Cause: To be determined.

Action: To be determined.

269: The format for {2} must be in the format YYYY/MM/DD or YYYY-MM-DD. Literal argument of expression (line {0}) on query {1} is invalid.

Cause: To be determined.

Action: To be determined.

270: No schema context is specified.

Cause: Each descriptor in an XML or EIS project must be associated with an XML schema context.

Action: Select the EIS or XML descriptor in the **Navigator** and complete the **Schema Context** field on the **Descriptor Info** tab.

271: The descriptor represents a document root object, but no default root element is chosen.

Cause: Each root descriptor must have a default root element.

Action: On the descriptor's **Descriptor Info** tab, complete the **Default Root Element** field.

280: A descriptor that represents \"anyType\" cannot support inheritance.

Cause: To be determined.

Action: To be determined.

281: A descriptor that represents \"anyType\" may contain only a single Any (Object or Collection) mapping.

Cause: To be determined.

Action: To be determined.

290: No primary keys specified.

Cause: Although you have associated the descriptor with a database table, you have not identified the primary keys.

Action: Use the **Primary Keys** area of the descriptor's **Descriptor Info** tab to select the primary keys for the descriptor.

291–304: The event policy's *[method type]* method is no longer a visible member of this descriptor's associated class.

Cause: You changed the class hierarchy within the project, causing the method to no longer be visible to the class.

Action: Ensure that the selected method is visible to the class.

305: The write-lock field is stored in an object, but there is not a writable mapping to the field.

Cause: If the write lock field is stored in object, there must be a non-read-only mapping to it.

Action: On the mapping's **General** tab, ensure that **Read-Only** is *not* selected.

306: Database fields specified for Selected Fields type Locking Policy must be mapped: *[field name]*

Cause: You selected an unmapped database field for a descriptor's locking policy.

Action: On the descriptor's **Locking** tab, ensure that you have selected a mapped database field as the **Selected Field**. See "Configuring Locking Policy" on page 25-64.

307: Database fields specified for Selected Fields type Locking Policy must not be primary key fields: *[field name]*

Cause: The database fields you selected for the optimistic locking policy (by fields) contains the primary keys for the database table.

Action: In the **By Fields** area of the descriptor's **Locking** tab, select different fields. See "Configuring Locking Policy" on page 25-64.

308: Version locking is chosen as the Locking Policy, but the field is not specified.

Cause: If you select to use version locking with an optimistic locking policy, you must identify which database field to use for version control.

Action: Use the **Database Field** field on the descriptor's **Locking** tab to select a field to use for version control. See "Configuring Locking Policy" on page 25-64.

309: The Version Locking database field selected does not exist on this descriptor's associated tables.

Cause: The database field you selected for optimistic version locking does not exist on the descriptor's associated table.

Action: You must either select a different database field on the descriptor's **Locking** tab, or associate the descriptor with a different database table. See "Configuring Locking Policy" on page 25-64.

310: Database fields specified for Selected Fields type Locking Policy do not exist on this descriptor's associated tables: *[field name]*

Cause: The database fields you selected for the optimistic locking policy (by fields) do not exist on the descriptor's associated table.

Action: You must either select a different database field on the descriptor's **Locking** tab, or associate the descriptor with a different database table. See "Configuring Locking Policy" on page 25-64.

311: The method you have specified for the instantiation policy's method on this descriptor is no longer a visible member of this class.

Cause: The method selected as the instantiation method has either been removed, or its visibility has been reduced so that it is no longer publicly visible.

Action: Deselect this method as the instantiation method. See "Configuring Locking Policy" on page 25-64.

312: The method you have specified for the instantiation policy's factory instantiation method on this descriptor is no longer a visible member of this class.

Cause: The method selected as the factory instantiation method has either been removed, or its visibility reduced so that it is no longer publicly visible.

Action: Deselect this method as the factory instantiation method. See "Configuring Instantiation Policy" on page 25-70.

313: The method you have specified for the instantiation policy's factory method on this descriptor is no longer a visible member of this class.

Cause: The method selected as the factory method has either been removed, or its visibility reduced so that it is no longer publicly visible.

Action: Deselect this method as the factory method. See "Configuring Instantiation Policy" on page 25-70.

314: "Use factory" is specified for the Instantiation policy, but all required information is not specified.

Cause: You selected the **Use Factory** option on the descriptor's **Instantiation Policy** tab, but did not specify the **Factory Class**, **Factory Method**, or **Instantiation Method** fields.

Action: Complete the **Factory Class**, **Factory Method**, or **Instantiation Method** fields on the descriptor's **Instantiation** tab. See "Configuring Instantiation Policy" on page 25-70.

315: "Use method" is selected for the Instantiation policy, but no method is selected.

Cause: You selected the **Use Method** option on the descriptor's **Instantiation Policy** tab, but did not specify the field.

Action: Select the **Method** on the descriptor's **Instantiation** tab. See "Configuring Instantiation Policy" on page 25-70.

316: The class does not have an accessible zero argument constructor.

Cause: No accessible zero argument constructor exists for the class associated with this descriptor.

Action: Make the zero argument constructor accessible if it exists, or create a accessible zero argument constructor if it doesn't exist.

317: No method was specified for the copying policy.

Cause: You specified that the descriptor should use a specific clone method for copying, but you did not select a method.

Action: Complete the **Use Clone Method** field on the descriptor's **Copying** tab to select a method.

318: The method specified for the copy policy on this descriptor is no longer a visible member of this class.

Cause: You changed the class hierarchy within the project, causing the copy policy to no longer be visible to the class.

Action: Ensure that the copy policy is visible to the class.

319: Primary keys do not match across associated tables and no reference(s) specified in multiple table policy information.

Cause: You attempted to associate multiple tables using a primary key.

Action: Primary key field names must match across associated tables, or references must be defined from the base table to each derived table.

320: The multiple table reference should be defined from the base table [table name] to the derived table.

Cause: This descriptor has **Inheritance** and **Multitable** advanced properties defined on it.

Action: The multiple table relationship that is defined between the base class' table and this derived class' table must be defined from base to derived.

321: The multiple table reference should not be defined on the database.

Cause: When using multitables with differently named primary keys, you must set a reference from the TOP table to the BOTTOM table. This reference must not be an actual constraint on the database.

Action: Select the table in which this is defined, and deselect the **On Database** option.

322: A class containing the desired after loading method should be specified.

Cause: You added an after-load method to a descriptor, but you did not specify a class.

Action: Complete the **After Load** tab. See "Configuring Amendment Methods" on page 25-81.

323: An after-load method must be specified.

Cause: You added an after-load method to a descriptor, but did not select an amendment method.

Action: Complete the **After Load** tab. See "Configuring Amendment Methods" on page 25-81.

324: An interface class must be specified for the interface alias.

Cause: You added an interface alias to a descriptor, but did not select an amendment method.

Action: Complete the **Interface Alias** tab.

325: The inheritance hierarchy originating in this descriptor cannot contain both aggregate and nonaggregate child descriptors.

Cause: Aggregate and class descriptors cannot be in the same inheritance hierarchy.

Action: Ensure that the inheritance hierarchy contains either aggregate *or* nonaggregate children, but not both.

326: The inheritance hierarchy originating in this descriptor cannot contain both root and composite child descriptors.

Cause: There is a mixture of root and composite descriptors among the descendents of this descriptor.

Action: Make all descendents of this descriptor the same type by either making them all root, or making them all composite. You can do this by removing the differing descriptor from the hierarchy, or changing their type to be consistent with the other descriptors in the hierarchy.

330: The returning policy insert fields do not exist on this descriptor's associated tables: *[field name]*

Cause: The field you selected on the descriptor's **Returning** tab does not exist on the database table associated with the descriptor.

Action: Select a different database table in the **Insert** area of the descriptor's **Returning** tab.

331: The returning policy update field *[field name]* does not exist on this descriptor's associated tables.

Cause: The field you selected on the descriptor's **Returning** tab does not exist on the database table associated with the descriptor.

Action: Select a different database table in the **Update** area of the descriptor's **Returning** tab.

350: Descriptors with Unknown Primary Keys must use sequencing.

Cause: **Unknown Primary Key Class** is selected for this descriptor, but the descriptor does not use sequencing.

Action: Change the descriptor so that it uses sequencing, or so that it no longer uses an unknown primary key class.

Mapping Errors (400 – 483)

This section lists TopLink Workbench mapping errors.

400: Method accessors have not been selected.

Cause: You selected **Use Method Accessing** for a mapping, but you did not select a method.

Action: You must select a **Get** and **Set** method on the mapping's **General** tab. See "Configuring Method Accessing" on page 32-14.

401, 402: The *[get/set access method]* method for this mapping's method accessing field is no longer visible to this descriptor.

Cause: You changed the class hierarchy within the project, causing the method access type (`get` or `set`) to no longer be visible to the class.

Action: Ensure that the selected method is visible to the class.

403: Mappings for EJB 2.0 CMP descriptors that use Value Holder Indirection must not use method accessing.

Cause: You cannot use method accessing on mappings for EJB 2.0 CMP descriptors that use **ValueHolder Indirection**.

Action: Because EJB attributes are code-generated, reference mappings *should not* be set to use method access. The attributes are code-generated to be of type **ValueHolder** but the abstract methods are defined to return the local interface type of the related bean.

404: Mapping references a write-lock field, but it is not read-only.

Cause: You specified a locking policy for a descriptor, but one of the attribute mappings is not read-only.

Action: Select the **Read Only** option on the mapping's **General** tab.

410: No direct field is specified.

Cause: For direct collection mappings, you must specify the direct collection information.

Action: Select a **Target Table** and **Direct Field** that the direct collection specifies.

415: No direct key field is specified.

Cause: For direct map mappings, you must specify a direct key field in the reference table that stores the primitive data value of the map key.

Action: On the direct map mapping's **General** tab, select a **Direct Key Field**. See "Configuring Direct Key Field" on page 44-2.

420: No database field is selected.

Cause: You created a direct-to-field or type conversion mapping without selecting a database field.

Action: For attributes with direct-to-field mappings, you must specify a **Database Field** on the mapping's **General** tab. For attributes with type conversion mappings, you must specify a **Database Field** on the mapping's **General** tab.

421: The selected database field does not exist on this descriptor's associated tables.

Cause: The database field mapped to an attribute is not included in the table associated with the attribute's descriptor.

Action: Ensure that the **Database Field** field on a mappings **General** tab is included in the table that you associated with the attribute's descriptor. See "Configuring Associated Tables" on page 26-2 and "Configuring a Database Field" on page 34-2.

430, 431: No null value type has been selected.

Cause: You selected to **Use Default Value When Database Field is Null** for a mapping, but did not specify the value.

Action: Specify a default **Type** or **Value**, or both on the mapping's **General** tab. See "Configuring a Default Null Value at the Mapping Level" on page 32-12.

This message may also appear after using the Package Rename tool when upgrading an older TopLink Workbench project.

440: XML type mappings are supported only on the Oracle9*i* Platform.

Cause: You created a Direct to XML Type mapping in relational project that uses a non-Oracle9*i* database.

Action: Select an Oracle9*i* platform as the database platform for the data source. See "Configuring Relational Database Platform at the Project Level" on page 20-2.

450: No reference descriptor is selected.

Cause: You created a mapping, but did not specify the reference descriptor

Action: You must select a **Reference Descriptor** for each relationship mapping on the mapping's **General** tab.

451: *[descriptor name]*references *[descriptor name]*, which is not active.

Cause: You tried to select an inactive descriptor as a **Reference Descriptor** on the mapping's **General** tab.

Action: You must either select a new **Reference Descriptor**, or make the descriptor active.

460: No table reference is selected.

Cause: You created a relationship mapping, but did not specify a reference table.

Action: Select (or create) a table reference for each relationship mapping on the mapping's **Table Reference** tab.

461: Table reference is invalid.

Cause: The table reference selected for this mapping is invalid.

Action: Select a different table reference for this mapping.

462: The reference *[table reference]* does not have any field associations.

Cause: You selected a table reference for a mapping, but did not add a key pair.

Action: You must specify source and target key pairs for the reference.

463: A key pair has not been completely specified for a reference.

Cause: You created a table reference without a key pair.

Action: You must specify a foreign key reference for the database table. Use the database table's **Reference** tab to add a key pair.

464: No relationship partner is specified.

Cause: You selected the **Maintains Bidirectional Relationship** option for a relationship mapping, but did not select a mapping to use as the relationship partner.

Action: Select a mapped attribute (from the reference descriptor) for this relationship. See "Configuring Bidirectional Relationship" on page 32-34.

465: The relationship partner must be a one-to-one, one-to-many, or many-to-many mapping.

Cause: You selected an invalid attribute as the Relationship Partner in a bidirectional relationship.

Action: In the **Relationship Partner** field, select a one-to-one, one-to-many, or many-to-many mapping. See "Configuring Bidirectional Relationship" on page 32-34.

466: The specified relationship partner mapping does not specify this mapping as its own relationship partner.

Cause: **Maintains Bidirectional Relationship** is selected for this mapping, but the mapping selected as the relationship partner does not have this mapping selected as its relationship partner.

Action: You must either select a different mapping for this mappings relationship partner, which has this mapping selected as it bidirectional relationship partner, or select this mapping as the bidirectional relationship partner of the mapping selected as the bidirectional relationship partner for this mapping.

467: The chosen reference descriptor is not a valid reference descriptor for this mapping.

Cause: The descriptor selected as the reference descriptor for this mapping is not a valid reference descriptor.

Action: Select a valid reference descriptor for this mapping.

470: No container class is selected.

Cause: No container class has been selected for this collection mapping.

Action: Select a `Container` class for this `Collection` mapping.

471: The container policy uses a Collection class, but the container class is not a Collection.

Cause: The selected container class for this collection mapping is not a Collection, but **Use Collection Class** is selected.

Action: Select a `Container` class that is a `Collection` for this mapping.

472: The container policy uses a Map class, but the container class is not a Map.

Cause: The selected `Container` class for this `Collection` mapping is not a `Map` class, but **Use Map Class** is selected.

Action: Select a `Container` class that is a `Map` class.

473: The container class must be instantiable.

Cause: The selected `Container` class for this `Collection` mapping is not instantiable.

Action: Select a `Container` class this is instantiable, (not an `Interface`, `Abstract` class, or `Primitive` class).

474: The container class does not agree with the instance variable.

Cause: The selected `Container` class for this `Collection` mapping, does not agree with the instance variable that is associated with the mapping. Either the variable is a `Map` class and the selected `Container` class is a `Collection` or vice versa.

Action: You must either select a `Container` class that agrees with the type of instance variable with which it is associated, or change the instance variable to agree with the selected `Container` class.

475: The container class is a Map, but the key method is not selected.

Cause: Use Map Class is selected for the `Container` policy for this `Collection` mapping, but a key method has not been selected.

Action: You must either select a key method for this `Container` policy, or change the `Container` policy to not use a map class.

4 76: The key method specified for this mapping is no longer visible to the owning descriptor's class.

Cause: The selected key method for the `Container` policy for this `Collection` mapping policy is not visible to the descriptor's class.

Action: You must either select a different method that is visible to the descriptor's class, or change the selected method so that it is visible.

477: The key method specified for this mapping is not valid.

Cause: The selected key method for the `Container` policy for this `Collection` mapping is invalid because it does not have the correct return type, or it does not accept more than zero parameters.

Action: You must either select a different method that is valid, or change the selected method so that it will return the correct type and accept more than zero parameters.

478: One-to-Many and Many-to-Many mappings in EJB 2.0 CMP descriptors may not use ValueHolder indirection.

Cause: A one-to-many or many-to-many mapping in an EJB 2.0 CMP descriptor is using `ValueHolder` indirection.

Action: You must either change the mapping to use no indirection or non-`ValueHolder` indirection.

480: No relation table is selected.

Cause: You created a many-to-many mapping, but did not specify a relation table. The relation table represents the relationship between the primary keys of the source table and target table.

Action: Select or create a **Relation Table** on the mapping's **General** tab.

481: The relation table is not dedicated to single, writable many-to-many mapping.

Cause: More than one many-to-many mapping in the project are using the same relation table.

Action: Each relation table should be used in one and only one many-to-many mapping.

482: No source reference is selected.

Cause: You created a many-to-many mapping, but did not select (or create) a source table reference on the mapping's **Source Reference** tab.

Action: The source table reference must contain a **Source** field (from the mapping's relation table) and a **Target** field (from one of the descriptor's associated tables).

483: No target reference is selected.

Cause: You created a many-to-many mapping, but did not select (or create) a target table reference on the mapping's **Source Reference** tab.

Action: The target table reference must contain a **Source** field (from the mapping's relation table) and a **Target** field (from one of the descriptor's associated tables).

Table Errors (500 – 610)

This section lists TopLink Workbench table errors.

500: You cannot use joining because the source and target (reference) descriptors are the same type.

Cause: You selected the **Use Joining** option on a one-to-one mapping in which the source and reference descriptors are the same.

Action: You must either deselect the **Use Joining** option or select a difference **Reference Descriptor** on the **One-to-One Mapping General** tab.

510: No query key associations have been defined.

Cause: You created a variable one-to-one mapping, but did not define a key pair.

Action: Create or select a key pair on the mapping's **Query Key Association** tab.

511: Not all query key associations have foreign key fields specified.

Cause: You created a query key association without a foreign key.

Action: You must specify a foreign key field for each query key association on the **Query Key Association** tab for variable one-to-one mapping.

512: The following specified query key names are no longer valid: *[query key]*

Cause: The query keys listed for this mapping no longer refer to the reference descriptor for this mapping. The query keys are now invalid.

Action: You must either remove the invalid query keys, or change the reference descriptor so that it corresponds with the query keys.

513: No indicator field is selected.

Cause: You created a variable one-to-one mapping, but did not specify a database field in which to store indicator values.

Action: Select the **Class Indicator Field** on the **Class Indicator Info** tab.

514: No indicator values are specified.

Cause: You created a variable one-to-one mapping, but did not specify indicator values for each object type.

Action: Select the **Indicator Type** on the **Class Indicator Info** tab.

515: *[descriptor name]* is not an implementor of the *[descriptor name]* interface, so it cannot have an indicator value.

Cause: You included a descriptor on the **Variable One-to-One Class Indicator Info** tab that is an implementor.

Action: Unselect the descriptor on the **Variable One-to-One Class Indicator Info** tab or add the descriptor to the **Implementor** tab.

516: The chosen reference descriptor is not an interface descriptor.

Cause: This variable one-to-one mapping has a reference descriptor selected which is not an interface descriptor. The reference descriptor for a variable one-to-one mapping must be an interface descriptor for the mapping to be valid.

Action: You must either choose a reference descriptor that is an interface descriptor, or change the mapping to no longer be variable.

520: No attribute transformer is specified.

Cause: No attribute transformer is specified for this transformation mapping.

Action: Select an attribute transformer for this transformation mapping.

521: The attribute transformer class is missing.

Cause: No class has been specified for the attribute transformer for this transformation mapping.

Action: Select a class for the attribute transformer.

522: The attribute transformer class *[class name]* is not a valid transformer class.

Cause: The attribute transformer class that is selected is not a valid attribute transformer class.

Action: Select a valid attribute transformer class for the transformation mapping.

523: The attribute transformer method is missing.

Cause: No method has been selected for the attribute transformer for the transformation mapping.

Action: Select a method for the attribute transformer.

524: The attribute transformer method *[method name]* is not visible to the parent descriptor's class.

Cause: The selected attribute transformer method is not visible to the descriptor class for this mapping.

Action: You must either select a different method that is visible, or change the method in the class to make it visible.

525: The attribute transformer method *[method name]* is not a valid transformer method.

Cause: The selected attribute transformer method either has the wrong return type or accepts the wrong parameters to be a valid transformer method for this transformation mapping.

Action: You must either select a method with the correct return type and parameters, or change the selected method so that it meets these criteria.

526: No field transformer associations are specified.

Cause: No field transformer association has been specified for this transformation mapping.

Action: Specify at least one field transformer association.

527: No transformer is specified for the field *[field name]*.

Cause: No transformer specified for the given field.

Action: Specify a transformer for this field.

528: There is a missing field in the field transformer association.

Cause: There is no field specified for a field transformer association for this transformation mapping.

Action: Specify a field for all the field transformer associations for this transformation mapping.

529: There is a missing transformer class for the field *[field name]*.

Cause: The `Transformer` class is specified for this field transformer association, but the `Transformer` class is unspecified.

Action: Specify a `Transformer` class for the field transformer association for this field.

530: The transformer class [*class name*] for the field [*field name*] is not a valid transformer class.

Cause: The specified `Transformer` class for the field of this field transformer association is invalid.

Action: Specify a valid `Transformer` class for the field transformer association for this transformation mapping.

531: There is a missing transformer method for the field [*field name*].

Cause: A transformer method is specified for this field transformer association, but the transformer method is unspecified.

Action: Specify a transformer method for the field transformer association for this field.

532: The transformer method [*method name*] for the field [*field name*] is not visible to the parent descriptor's class.

Cause: The specified transformer method for the field transformer association for this field is not visible to the descriptor or the class of this mapping.

Action: You must either choose a method that is visible to the class, or change the method so that it is visible.

533: The field transformer method [*method name*] for the field [*field name*] is not a valid transformer method.

Cause: The specified method for the field transformer association for this field either has the incorrect return type, or accepts the wrong parameters.

Action: You must either select a method that has the correct return type and parameters, or change the currently selected method so that is has the correct return type and parameters.

540: No object type is selected.

Cause: You created an object type mapping, but did not select the type.

Action: You must select the **Object Type** and **Database Type** on the **General** tab of the mapping.

542: No object-type mappings have been specified.

Cause: You created an object type mapping, but did not create n object-to-database mapping.

Action: You must specify at least one mapping (**Database Value** and **Object Value**) on the **General** tab of the mapping.

545: NCharacter, NString, and NClob database types are currently supported only on the Oracle9*i* platform.

Cause: You attempted to map a database type that is not supported by your database.

Action: The database type for a type conversion mapping or direct-to-field mapping can be `NCharacter`, `NString`, or `NCLOB` only if you are using an Oracle9*i* database.

550: Attribute is typed as a ValueHolderInterface, but the mapping does not use Value Holder Indirection.

Cause: You did not specify indirection or transparent indirection for the mapping.

Action: If the class attribute is of type `ValueHolderInterface`, you must use `ValueHolder` indirection for the mapping.

551: Mapping uses ValueHolder Indirection, but its associated attribute is not a ValueHolderInterface.

Cause: You selected indirection without a `ValueHolderInterface`.

Action: If you select the **Use Indirection** (**ValueHolder**) option for a one-to-many, many-to-many, or direct collection mapping, the associated class attribute must be `ValueHolderInterface`.

560: The container class for this mapping must implement oracle.toplink.indirection.IndirectContainer.

Cause: This mapping uses transparent indirection, but the `Container` class selected for its container policy is not an `IndirectContainer`.

Action: You must either select a `Container` class that is an `IndirectContainer`, or remove transparent indirection from the mapping.

570: The chosen reference descriptor is not an aggregate descriptor.

Cause: This is an aggregate mapping, but the selected reference descriptor is not an aggregate descriptor.

Action: You must either select a reference descriptor for this mapping that is an aggregate descriptor, or change this mapping to no longer be an aggregate mapping.

571: Aggregate fields are not specified.

Cause: You created an aggregate mapping without specifying specific fields.

Action: Every **Field Description** on the **Fields** tab must contain a unique **Field** for aggregate mappings.

572: Aggregate mapping fields must be unique.

Cause: You created an aggregate mapping without specifying unique fields.

Action: Every **Field Description** on the **Fields** tab must contain a unique **Field** for aggregate mappings.

573: The selected field does not exist on this descriptor's associated tables.

Cause: The field selected for one of the aggregate-path-to-fields for this aggregate mapping does not exist on any of the descriptor's associated tables.

Action: You must either select a different field for the path-to-field, or add the field to the appropriate table.

580: No XML field specified.

Cause: You mapped an attribute in an XML or EIS descriptor, but did not select an XML field.

Action: You must complete the **XML Field** field on the **General** tab of the mapping.

581: The specified XPath is not valid within the current schema.

Cause: The XPath specified for this mapping does not resolve in the schema.

Action: You must either select a different XPath, or alter the schema so that this XPath will resolve.

582: The specified XPath does not represent text data.

Cause: The XPath specified for this direct mapping does not resolve to a direct field in the schema.

Action: You must either select a different XPath, alter the schema so that this XPath will resolve to a direct field, or change the mapping type.

583: The specified XPath does not represent a single xml field.

Cause: The XPath specified for this mapping resolves to a field which is a collection, but this is not a collection mapping.

Action: You must either select a different XPath, alter the schema so that this XPath will resolve to a singular field, or change the mapping type.

590: The chosen reference descriptor is not a root eis descriptor.

Cause: The reference descriptor selected for this EIS reference mapping is not a root descriptor. Reference mappings in EIS descriptors must be root descriptors.

Action: You must either select a different reference descriptor for this mapping which is a `root` descriptor, or change the mapping type.

591: No relationship partner is specified.

Cause: This mapping has **Maintains Bidirectional Relationship** selected, but no relationship partner is specified.

Action: You must either deselect **Maintains Bidirectional Relationship**, or select a relationship partner.

592: The relationship partner must be an EIS One-to-One or EIS One-to-Many mapping.

Cause: The relationship partner selected for this mapping is not of the type `EIS` one-to-one or EIS one-to-many.

Action: You must select an `EIS` one-to-one or EIS one-to-many mapping as the relationship partner for this mapping, or deselect **Maintains Bidirectional Relationship**.

593: The specified relationship partner mapping does not specify this mapping as its own relationship partner.

Cause: The mapping selected as the relationship partner for this mapping does not have this mapping selected as its relationship partner. For these relationships to be bidirectional, you must select the relationship partner for both mappings.

Action: You must either go to the mapping selected as the relationship partner for this mapping and select this mapping as its relationship partner, or select a different relationship partner mapping for this mapping to maintain this mapping as its relationship partner.

594: There is a missing source XML field.

Cause: No field has been specified as the source XML field for this mapping.

Action: You must specify a source XML field.

595: There is a missing target XML field.

Cause: No field has been specified as the target XML field for this mapping.

Action: You must specify a target XML field.

600: A foreign key grouping element is required if there are multiple field pairs.

Cause: No foreign key grouping element is specified for this mapping and multiple field pairs.

Action: You must specify a foreign key grouping element.

601: The foreign key grouping element does not contain all foreign keys fields.

Cause: The specified foreign key grouping element does not contain all the foreign key fields.

Action: You must either remove the foreign key fields not contained in this foreign key grouping element, or pick a foreign key grouping element that contains all the foreign key fields.

602: A delete all interaction is specified, but the mapping is not private owned.

Cause: A `deleteall` interaction is specified for this mapping, but the mapping is not private owned.

Action: You must either make the mapping private owned, or remove the `deleteall` interaction.

610: At least one field pair must be specified, unless the mapping has no selection interaction and is read-only.

Cause: No field pairs are specified, and this mapping has a `selection` interaction specified and/or is not read-only.

Action: You must either specify a field pair for the mapping, or make the mapping read-only and remove the `selection` interaction.

XML Schema Errors (700 – 706)

This section lists TopLink Workbench XML schema errors.

701: A database table can only have one IDENTITY column defined.

Cause: You defined more than one identity column for this table.

Action: On the database table's **Columns** tab, leave only one identity (**Identity**) column. See "Working With Column Properties" on page 4-25.

702: A size is required for the column *[column]*.

Cause: You did not specify any size for this column. The default size is 0.

Action: On the database table's **Columns** tab, specify the size (**Size**) for the column (field). See "Working With Column Properties" on page 4-25.

703: The reference *[table reference]* does not have any field pairs.

Cause: You added a reference for a table, but the reference does not include a key pair.

Action: On the database table's **References** tab, specify source and target field pairs for the table reference. See "Creating Table References" on page 4-26.

704: A key pair has not been completely specified for a reference.

Cause: A reference table is missing a complete key pair (source and target fields).

Action: You must specify a foreign key reference for the database table. On the database table's **References** tab, add a complete key pair. "Creating Table References" on page 4-26.

705: A development login has not been specified.

Cause: You created a relational TopLink Workbench project, but did not specify a development login.

Action: On the Database property sheet, select a **Development Login** from the available defined logins, or add a new login. See "Configuring Development and Deployment Logins" on page 20-6.

706: A deployment login has not been specified.

Cause: You created a relational TopLink Workbench project, but did not specify a deployment login.

Action: On the Database property sheet, select a **Deployment Login** from the available defined logins, or add a new login. See "Configuring Development and Deployment Logins" on page 20-6.

Session Errors (800 – 812)

This section lists the TopLink sessions XML errors.

801: *[session name]* Login - The connection URL has to be specified.

Cause: You have not specified a connection URL for the session (when using a database driver manager). Each session must have at least one login connection.

Action: On the session's **Login – Connection** tab, complete the **Driver URL** field. See "Configuring a Session Login" on page 74-4.

802: *[session name]* Login - The driver class has to be specified.

Cause: You have not specified a driver class for the session (when using a data source database driver).

Action: On the session's **Login – Connection** tab, complete the **Driver Class** field. See "Configuring a Session Login" on page 74-4.

803: [session or connection pool name]Login - Login - The data source name has to be specified.

Cause: You have not specified a driver class for the session login (when using a J2EE data source database driver).

Action: On the session's or connection pool's **Login – Connection** tab, complete the **Data Source** field. See "Configuring a Session Login" on page 74-4.

804: Login - Session Broker - It has to have at least one session, either a server or a database session.

Cause: You created a session broker but did not add any sessions. Each session broker must contain a session.

Action: On the session broker's **General – Sessions** tab, select a session to add to this broker. See Chapter 79, "Configuring Session Broker and Client Sessions".

805: *[session name]* Database Session - It has to have at least one XML file or a class specified.

Cause: Your database session does not have a primary project (an associated deployment XML file or Java class file).

Action: On the session's **Project – General** tab, complete the **Primary Project** field. See "Configuring a Primary Mapping Project" on page 74-2.

806: Login - The transport class has to be specified.

Cause: You selected a custom (user-defined) cache coordination type, but did not specify the transport class for cache coordination.

Action: On the session's **Cache Coordination** tab, complete the **Transport Class** field, or select a different cache coordination type. See Chapter 88, "Configuring a Coordinated Cache".

807: *[session name]* Login - The location of the log file has to be specified.

Cause: You are using standard logging and selected to have the log saved to a file, but did not select a file name and location.

Action: On the session's **Logging** tab, complete the **Log Location** field. See "Configuring Logging" on page 74-4.

811: *[session or broker name]* - An external transaction controller (JTA) has to be specified.

Cause: You selected a custom server platform, but did not specify the JTA for the platform.

Action: On the session or session broker's **General – Server Platform** tab, complete the **External Transaction Controller (JTA)** field. See "Configuring the Server Platform" on page 74-14.

812: *[session or broker name]* - A server class has to be specified.

Cause: You selected a custom server platform, but did not specify the server class for the platform.

Action: On the session or session broker's **General – Server** Platform tab, complete the **Server Class** field. See "Configuring the Server Platform" on page 74-14.

Common Classpath Problems

The following are some common TopLink Workbench error messages that may result from invalid classpath information. See "Configuring Project Classpath" on page 19-3 for more information.

The TopLink Workbench does not display the class(es) to import.

Cause: Your classes are not available for import on the **Select Classes** dialog box.

Action: Ensure that the class is in your project's classpath (on the project's **General** properties tab). Ensure that the class is in the `.zip` or `.jar` file. You cannot import compressed classes.

The TopLink Workbench generates an exception error when importing classes.

Cause: TopLink class import utility did not start correctly. One of the classes includes a static initialization method, which may cause the import utility to fail.

Action: Ensure that your project's classpath points to the root directory of your package hierarchy. For example, to import the `com.company.class` package in the `C:\classes\com\company` directory, your project classpath should be `C:\classes\`.

The TopLink Workbench fails to import the class, but does not generate an exception error.

Cause: The classpath containing your JDBC drivers should still be on your system CLASSPATH. TopLink Workbench classpath is for domain classes only.

Action: Ensure that you have properly indicated the directories that contain your domain class(es) to map on the project's **General** tab.

Data Source Problems

This section includes common problems that you may encounter when connecting TopLink to your data source. This section includes the following topics:

- Database Connection Problems

Database Connection Problems

This section describes common errors and problems you may encounter when communicating with or logging in to the database.

The class *[class]* was not found.

Cause: You attempted to log in to the database, but TopLink could not find the JDBC driver for the database.

Action: Ensure that the `JDBC_CLASSPATH` in the `setenv.cmd` file points to your JDBC driver JAR files. Verify that your PATH includes all files (for example, native `.dll` files) required by the driver. If the path to your JDBC driver JAR files contains spaces, then the path must be enclosed in double-quotes in the `setenv.cmd` file. For example:

```
set JDBC_CLASSPATH="C:\Program Files\some directory\driver.jar\"
```

For more information, see "Configuring the TopLink Workbench Environment" on page 4-2.

Username or password could be invalid.

Cause: TopLink was unable to log in to the database.

Action: Ensure that the **Username** and **Password** for the database are correct.Verify with your DBA that the database is set up and operating correctly.

You must define a development login.

Cause: You attempted to log in to the database from TopLink Workbench, but you did not define a development login.

Action: On the database property sheet, select a **Development Login**, or create a new **Defined Login**. See "Configuring Login Information" on page 20-5.

No database driver has been specified.

Cause: You attempted to log in to the database from TopLink Workbench, but you did not complete the login information.

Action: Complete all the required fields on the database property sheet for the selected development login. See "Configuring Login Information" on page 20-5.

Invalid URL specified.

Cause: You attempted to log in to the database from TopLink Workbench, but the URL is incorrect.

Action: Complete the **URL** field on the database property sheet for the selected development login. See "Configuring Login Information" on page 20-5.

15

Troubleshooting Application Deployment

This chapter discusses some of the general troubleshooting issues surrounding entity bean configuration and deployment. It lists many of the common exceptions and exception messages that you may encounter when you try to deploy persistent entity beans, using TopLink.

If you have any problems installing TopLink, using TopLink Workbench, or require more information on any run-time exceptions that are generated by TopLink, consult the appropriate documentation.

This chapter contains information on:

- Generating Deployment JAR Files
- Common J2SE Deployment Exceptions
- Common BEA WebLogic Server Deployment Exceptions
- Common BEA WebLogic Server 6.1 Exceptions
- Common BEA WebLogic 7.0 Exceptions
- Common BEA WebLogic 8.1 Exceptions
- Common IBM WebSphere Application Server Exceptions

Generating Deployment JAR Files

If you experience trouble generating the JAR files for deployment:

- Ensure all environment entries (such as classpath) are configured properly.
- Identify which step of the build is failing (copying, compiling, running EJB compiler, and so on.)

Running the 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 execution of the EJB compile utility, try to determine which stage may be causing the failure.

For more information about the EJB compile utility, see the server documentation.

Common J2SE Deployment Exceptions

The following are some of the most common exceptions that may be encountered when you try to deploy a J2SE (non-J2EE) application using TopLink.

Classpath Exceptions

An exception may occur while setting up the connection pool. You must check the nested SQL exception to determine the cause of the exception. Typical reasons for this include the following:

- The location of JDBC driver is not specified on the classpath.
- The user name or password provided by the user is incorrect.
- The server URL or driver name is not properly specified.

You should consult the application server documentation and the JDBC driver documentation for help with this exception.

If the required TopLink JAR files have not been copied into the application extensions classpath, a classpath exception will be raised. You must ensure that the `toplink.jar` and `antlr.jar` files are copied into the `<Application Server install>\lib\app` directory.

If TopLink encounters problems finding the deployment `project.xml` or `sessions.xml` files, a classpath exception will be raised. Refer to Chapter 10, "Deploying a TopLink Application".

JDeveloper places the `sessions.xml` file in the `META-INF` directory. To load the `sessions.xml` file in a non-J2EE application, you must explicitly provide the location of the `sessions.xml` file as shown in Example 15–1.

Example 15–1 Location of the sessions.xml file

```
XMLSessionConfigLoader loader = new
          XMLSessionConfigLoader("META-INF/sessions.xml");
session = (DatabaseSession)SessionManager.getManager().getSession(
          loader, "MySession", Thread.currentThread().getContextClassLoader());
```

If the J2SE application is a single-user application, a `DatabaseSession` can be used instead of a `ServerSession`. This provides improved performance and reduces the number of database connections and login time.

Communication Exceptions

In situations where cache coordination is used, a communication exception may occur. A communication exception is a run-time exception that wraps all RMI, CORBA, or input and output exceptions that occur.

Refer to the Communication Exceptions (12000 - 12003) section on page 13-65 in the TopLink Exception Reference chapter for detailed information on communication exceptions that may occur.

Descriptor Validation Exceptions

A descriptor exception is a development exception that is raised when insufficient information is 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.

The internal exception, mapping and descriptor appear only if TopLink has enough information about the source of the problem to provide this information.

Refer to the Descriptor Exceptions (1 – 201) section on page 13-2 in the TopLink Exception Reference chapter for detailed information on descriptor validation exceptions that may occur.

Common BEA WebLogic Server Deployment Exceptions

The following are some of the most common exceptions that are encountered when you deploy entity beans to a BEA WebLogic Server.

For more information about specific versions, see the following:

- Common BEA WebLogic Server 6.1 Exceptions, on page 15-5
- Common BEA WebLogic 7.0 Exceptions, on page 15-8
- Common BEA WebLogic 8.1 Exceptions, on page 15-10

Assertion Error

Cause: This exception occurs if the `toplink.jar` file is not properly set on your classpath. The following exception message is returned:

```
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.WlsCMPDeployerERROR: ejbc
found errors
```

Action: Ensure the *<ORACLE_HOME>*`/toplink/jlib/toplink.jar` file is specified on your system classpath.

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 refer to errors in the specification of the project location reading in the properties file or validation errors due to improper mappings.

Exception 8001

Cause: This error occurs if the TopLink project file is not specified in the `toplink-ejb-jar.xml` file. The following exception message is returned:

```
<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.DeploymentExceptionEXCEPTION DESCRIPTION:
No TopLink project was specified for this bean.
atoracle.toplink.ejb.DeploymentException.noPro
jectSpecified(DeploymentException.java:132) at
oracle.toplink.internal.ejb.cmp.ProjectDeployment.readProject
(ProjectDeployment.java:378)
```

Action: Ensure there is an entry in the `toplink-ejb-jar.xml` file for either the `project-xml` or `project-class`.

Exception 8016

Cause: This exception can occur if the location of the TopLink project file for the bean is not properly specified. The following exception message is returned:

```
<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.DeploymentExceptionEXCEPTION DESCRIPTION:
An error occurred while setting up the project:
[java.io.FileNotFoundException: Account.xml]INTERNAL
EXCEPTION: java.io.FileNotFoundException:
Account.xmlatoracle.toplink.ejb.DeploymentException.errorCrea
tingProject(Unknown Source)
```

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 Start Up Connection Pool

Cause: An exception has occurred while setting up the connection pool. The following exception message is returned:

```
<Error> <JDBC> <Cannot startup connection pool "ejbPool"
weblogic.common.ResourceException: Cannot load driver class:
org.hsqldb.jdbcDriver>...
```

Action: Check the nested SQL exception to determine the cause of the exception. Typical reasons for this include the following:

- The JDBC driver is not on the classpath.
- The user name or password provided by the user is incorrect.
- The server URL or driver name is not properly specified.

You should consult the BEA WebLogic Server documentation and your JDBC driver documentation for help on the specific exception raised by BEA WebLogic.

Error Message

Cause: This problem occurs if you are using the GA version of BEA WebLogic Server 6.0. The following exception message is returned:

```
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)
```

Action: Upgrade to WebLogic Server 6.0 (Service Pack 1) or a higher release.

EJBC Found Errors

Cause: This error occurs if the `toplink.jar` file is not properly set on your classpath. The following exception message is returned:

```
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 classpath.
```

Action: Ensure the <*ORACLE_HOME*>`/toplink/jlib/toplink.jar` file is specified on your system classpath.

EJB Deployment Exception

Cause: This exception occurs if the location of the TopLink project file for the bean is not properly specified. The following exception message is returned:

```
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...
```

Action: Check the file name as it is specified in the `toplink-ejb-jar.xml` file, and the location of the TopLink project file on the file system.

Deploying EJB Component

Cause: A typical cause of this exception is that the `toplink-ejb-jar.xml` file is referring to a local DTD file using a file name or location that is incorrect. The following exception message is returned:

```
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)
```

Action: Ensure that all XML files refer to valid DTD files and locations.

Cannot Start Up Connection Pool ejbPool

Cause: This exception is raised to indicate that an error has occurred while setting up the connection pool. The following exception message is returned:

```
Cannot startup connection pool "ejbPool"
weblogic.common.ResourceException: Could not create pool
connection. The DBMS driver exception was:...
```

Action: Check the nested SQL exception to determine the cause of the error. Typical problems include the following:

- The driver is not on the classpath.
- The user name or password is incorrect.
- The server URL or driver name is not properly specified.

You should consult the BEA WebLogic Server documentation and your JDBC driver documentation for help on the specific exception raised by BEA WebLogic.

Other Exceptions

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 temporary directories created by BEA WebLogic Server. You can find them under the `wlserver6.1` directory, at the same level as the `config` directory.

Common BEA WebLogic Server 6.1 Exceptions

The following are a few of the most common exceptions you may encounter when deploying JAR files with TopLink and BEA WebLogic Server 6.1:

- Development Exceptions
- Deployment and Run-Time Exceptions

Development Exceptions

Missing Persistence Type

Cause: There is no entry in the `persistence.install` file for TopLink CMP. This may occur if the TopLink installation was interrupted or a BEA WebLogic Server Service Pack was applied. The following exception message is returned:

```
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
```

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

Cause: The `toplink.jar` file is not properly set in your classpath. The following exception message is returned:

```
Error while loading persistence resource TopLink_CMP_
Descriptor.xml Make sure that the persistence type is in your
classpath.
```

Action: Ensure that the classpath includes the <*ORACLE_HOME*>/toplink/jlib/toplink.jar file.

Wrong BEA WebLogic Version

Cause: You are trying to compile your code using BEA WebLogic Server 6.0. The following exception message is returned:

```
C:\<ORACLE_HOME>\toplink\examples\weblogic\wls61\
examples\ejb\cmp20\singlebean\Account.java:10: cannot resolve
symbolsymbol  : class EJBLocalObjectlocation: interface
examples.ejb.cmp20.singlebean.Accountpublic interface Account
extends EJBLocalObject {
```

Action: Compile using BEA WebLogic Server 6.1.

Deployment and Run-Time Exceptions

Missing Persistence Type

Cause: There is no entry in the `persistence.install` file for TopLink CMP. This may occur if the TopLink installation was interrupted, or a BEA WebLogic Server Service Pack was applied. The following exception message is returned:

```
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).
```

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

Cause: The `toplink.jar` file is not properly set in your classpath. The following exception message is returned:

```
<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
classpath.atweblogic.ejb20.persistence.InstalledPersistence.i
nitialize(InstalledPersistence.java:214)atweblogic.ejb20.pers
istence.InstalledPersistence.getInstalledType(InstalledPersis
tence.java:113)
```

Action: Ensure that the classpath includes the *<ORACLE_HOME>*`/toplink/jlib/toplink.jar` file.

Wrong Persistence Version

Cause: You may be using a `persistence-version` meant for BEA WebLogic Server 7.0. The following exception message is returned:

```
DATE and TIME> <Error> <J2EE> <Error deploying application
ejb20_cmp_account:Unable to deploy EJB: Account from ejb20_
cmp_
account.jar:java.lang.AbstractMethodErroratweblogic.ejb20.dep
loyer.ClientDrivenBeanInfoImpl.deploy(ClientDrivenBeanInfoImp
l.java:807)atweblogic.ejb20.deployer.Deployer.deployDescripto
r(Deployer.java:1234)atweblogic.ejb20.deployer.Deployer.deplo
y(Deployer.java:947)atweblogic.j2ee.EJBComponent.deploy(EJBCo
mponent.java:30)
```

Action: Use a `persistence-version` of 4.0.

Cannot Start Up Data Source

Cause: An exception has occurred while setting up the data source. The following exception message is returned:

```
EXCEPTION [TOPLINK-7060] (TopLink (WLS
CMP)-X.X):oracle.toplink.exceptions.ValidationExceptionEXCEPT
ION DESCRIPTION: Cannot acquire datasource
[jdbc/ejbNonJTSDataSource].INTERNAL EXCEPTION:
javax.naming.NameNotFoundException: Unable to resolve
jdbc.ejbNonJTSDataSource Resolved: '' Unresolved:'jdbc' ;
remaining name 'ejbNonJTSDataSource'
```

Action: Check the nested SQL exception to determine the cause of the exception. For more information, see "7060: CANNOT_ACQUIRE_DATA_SOURCE" on page 13-48. For more information on a specific exception raised by WebLogic, see the BEA WebLogic Server documentation and your JDBC driver documentation.

Wrong WebLogic Version

Cause: You are trying to compile your code using BEA WebLogic Server 6.0. The following exception message is returned:

```
<DATE and TIME> <Error> <Management> <Error parsing XML
descriptor for application TopLink_Domain:Name=ejb20_cmp_
account,
Type=Applicationweblogic.xml.process.ProcessorFactoryExceptio
```

```
n: Could not locate processor for public id = "-//Sun
Microsystems, Inc.//DTD J2EE Application
1.3//EN"atweblogic.xml.process.ProcessorFactory.getProcessor(
ProcessorFactory.java:181)atweblogic.xml.process.ProcessorFac
tory.getProcessor(ProcessorFactory.java:164)
```

Action: Compile using BEA WebLogic Server 6.1.

Common BEA WebLogic 7.0 Exceptions

The following are a few of the most common exceptions you may encounter when deploying JAR files with TopLink and BEA WebLogic Server 7.0:

- Development Exceptions
- Deployment Exceptions

Development Exceptions

Missing Persistence Type

Cause: There is no entry in the `persistence.install` file for TopLink CMP. This may occur if the TopLink installation was interrupted, or a BEA WebLogic Server Service Pack was applied. The following exception message is returned:

```
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
```

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.

Missing Persistence Type

Cause: There is no entry in the `persistence.install` file for TopLink CMP. This may occur if the TopLink installation was interrupted, or a BEA WebLogic Server Service Pack was applied. The following exception message is returned:

```
ERROR:
atweblogic.ejb20.persistence.InstalledPersistence.initialize(
InstalledPersistence.java:214)atweblogic.ejb20.persistence.In
stalledPersistence.getInstalledType(InstalledPersistence.java
:113)atweblogic.ejb20.deployer.MBeanDeploymentInfoImpl.getPer
sistenceType(MBeanDeploymentInfoImpl.java:584
```

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.

Wrong WebLogic Version

Cause: You are trying to compile your JAR file using BEA WebLogic Server 6.1. The following exception message is returned:

```
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
```

Action: Compile using BEA WebLogic Server 7.0.

Deployment Exceptions

Missing Persistence Type

Cause: There is no entry in the `persistence.install` file for TopLink CMP. This may occur if the TopLink installation was interrupted or a BEA WebLogic Server Service Pack was applied. The following exception message is returned:

```
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)
```

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

Cause: The `toplink.jar` file is not properly set in your classpath. The following exception message is returned:

```
java.lang.NullPointerExceptionatweblogic.ejb20.deployer.EJBDe
ployer.deactivate(EJBDeployer.java:1513)atweblogic.ejb20.depl
oyer.EJBDeployer.undeploy(EJBDeployer.java:301)atweblogic.ejb
20.deployer.Deployer.deploy(Deployer.java:875)atweblogic.j2ee
.EJBComponent.deploy(EJBComponent.java:70)
```

Action: Ensure that the classpath includes the *<ORACLE_HOME>*`/toplink/jlib/toplink.jar` file.

Cannot Start Up Data Source

Cause: An exception has occurred while setting up the data source. The following exception message is returned:

```
EXCEPTION [TOPLINK-7060] (TopLink (WLS CMP) -
X.X.X): oracle.toplink.exceptions.ValidationExceptionEXCEPTIO
N DESCRIPTION: Cannot acquire datasource
[jdbc/ejbNonJTSDataSource].INTERNAL EXCEPTION:
javax.naming.NameNotFoundException: Unable to resolve
jdbc.ejbNonJTSDataSource Resolved: '' Unresolved:'jdbc' ;
remaining name 'ejbNonJTSDataSource'
```

Action: Check the nested SQL exception to determine the cause of the exception. For more information, see "7060: CANNOT_ACQUIRE_DATA_SOURCE" on

page 13-48. For more information on a specific exception raised by WebLogic, see the BEA WebLogic Server documentation and your JDBC driver documentation.

Common BEA WebLogic 8.1 Exceptions

The following are a few of the most common exceptions you may encounter when deploying JAR files with TopLink and BEA WebLogic Server 8.1:

- Development Exceptions
- Deployment Exceptions

Development Exceptions

Missing Persistence Type

Cause: There is no entry in the `persistence.install` file for TopLink CMP. This may occur if the TopLink installation was interrupted, or a BEA WebLogic Server Service Pack was applied. The following exception message is returned:

```
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).
```

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.

Missing CMP entry in Persistence File

Cause: There is no entry in the `persistence.install` file for TopLink CMP. This may occur if the TopLink installation was interrupted, or a BEA WebLogic Server Service Pack was applied. The following exception message is returned:

```
ERROR: ejbc couldn't invoke compiler
```

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

Cause: The `toplink.jar` file is not properly set in your classpath. The following exception message is returned:

```
Error occurred while loading persistence resource TopLink_
CMP_Descriptor.xml. Make sure that the persistence type is in
your classpath.

ERROR: ejbc couldn't invoke compiler
```

Action: Ensure that the classpath includes the `<ORACLE_HOME>/toplink/jlib/toplink.jar` file.

Wrong WebLogic Version

Cause: You are trying to compile your EJB JAR file using BEA WebLogic Server 7.0. The following exception message is returned:

```
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"ERORejbc found errors
```

Action: Compile using BEA WebLogic Server 8.1.

Deployment Exceptions

Missing Persistence Type

Cause: There is no entry in the `persistence.install` file for TopLink CMP. This may occur if the TopLink installation was interrupted or a BEA WebLogic Server Service Pack was applied. The following exception message is returned:

```
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.jarModule: 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)
```

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

Cause: The `toplink.jar` file is not properly set in your classpath. The following exception message is returned:

```
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.jarModule:
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(In
stalledPersistence.java:212)at
```

```
weblogic.ejb20.persistence.InstalledPersistence.getInstalledT
ype(InstalledPersistence.java:114)
```

Action: Ensure that the classpath includes the <*ORACLE_HOME*>/toplink/jlib/toplink.jar file.

Common IBM WebSphere Application Server Exceptions

When the IBM WebSphere application server is started, it attempts to deploy the JAR files that are specified for deployment within the application server.

Exceptions that occur when the server is started are usually configuration problems that involve classpath issues, environment variable configuration, and database login configuration. Review the IBM WebSphere application server documentation after starting the server.

This section contains some of the exceptions that can be encountered when running the IBM WebSphere application server, along with their possible causes and recommended solutions.

Class Not Found Exception

Cause: The class is not included on the WebSphere application extensions classpath or in the EJB or WAR module.

Action: Ensure that all required classes are included in the correct location. For more information about classpath locations, see the *IBM WebSphere InfoCenter*.

Class Not Found Exception

Cause: The required TopLink JAR files have not been copied into the application extensions classpath.

Action: Ensure that the `toplink.jar` and `antlr.jar` files are copied into the <*WebSphere install*>\lib\app directory.

oracle.toplink.exceptions.DatabaseException

Cause: A TopLink exception has occurred.

Action: Refer to the specific exception code. The exception code appears in the square brackets in the exception message, such as `[TopLink-1016]`). Exceptions observed here may be exceptions in reading in the properties file, or validation errors due to improper mappings.

Refer to the Database Exceptions (4002 – 4018) section on page 13-27 in the TopLink Exception Reference chapter for detailed information on database exceptions that may occur; refer to the Communication Exceptions (12000 - 12003) section on page 13-65 in the TopLink Exception Reference chapter for detailed information on communication exceptions that may occur.

Exception [6066]

Cause: A bean was created outside of a transaction and then a second bean was created either in or out of a transaction. The following exception message is returned:

```
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.
```

Action: Ensure that all bean creation is performed within the context of a transaction.

Cause: The bean was not removed during `ejbPassivate` method.

Action: Ensure that the `ejbPassivate` method removes 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]

Cause: An incorrect primary key object is being used with a bean. The following exception message is returned:

```
oracle.toplink.exceptions.ValidationException: Exception
occurred in reflective EJB bean primary key extraction,
please ensure your primary key object is defined correctly:
key = 301, bean = <beanName>
```

Action: Ensure that you are using the correct primary key object for a bean.

Exception [7066]

Cause: An attempt was made to create or remove a bean outside of a transaction. The following exception message is returned:

```
oracle.toplink.exceptions.ValidationException: Cannot create
or remove beans unless a JTS transaction is present,
bean=<bean>
```

Action: Ensure that all removal and creation of beans is performed within a transaction.

Exception [7068]

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. The following exception message is returned:

```
oracle.toplink.exceptions.ValidationException: The project
class <projectclass> was not found for the <toplink_session_
name> using default class loader.
```

Action: Ensure that the project class provided in the exception is on the IBM WebSphere application server dependent classpath.

Exception [7069]

Cause: An amendment method was called, but cannot be found. The following exception message is returned:

```
oracle.toplink.exceptions.ValidationException: An exception
occurred looking up or invoking the project amendment method,
<amendmentMethod> on the class <amendmentClass>;
```

Action: Ensure that the required amendment method exists on the class that is specified.

Exception [7070]

Cause: The `toplink.properties` file cannot be found. The following exception message is returned:

```
oracle.toplink.exceptions.ValidationException: A
toplink.properties resource bundle must be located on the
classpath in a TopLink directory.
```

Action: Ensure that the location of the `toplink.properties` file is on the classpath.

Exception [7079]

Cause: The descriptor that is listed was not found in the session that is specified on the deployment descriptor. The following exception message is returned:

```
EXCEPTION DESCRIPTION: The descriptor for [<bean class>] was
not found in the session [<session name>]. Check the project
being used for this session.
```

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]

Cause: The `toplink-ejb-jar.xml` file was not found. The following exception message is returned:

```
No "meta-inf/toplink-ejb-jar.xml" could be found in your
classpath. The CMP session could not be read in from file.
```

Action: Ensure that the `toplink-ejb-jar.xml` file is located in the deployed `ejb-jar` file under the `meta-inf` directory.

Exception [9002]

Cause: The project class that is specified for the session in the `toplink-ejb-jar.xml` file cannot be found. The following exception message is returned:

```
EXCEPTION [TOPLINK-9002] (TopLink - X.X.X):
oracle.toplink.exceptions.SessionLoaderExceptionEXCEPTION
DESCRIPTION: Unable to load Project class [<project class>].
```

Action: Ensure that the project class has been included in the deployed JAR file with the entity beans.

Problems at Run Time

This section lists some of the common exceptions that can occur at run time when using the TopLink CMP for IBM WebSphere application server.

Exception [6026]

Cause: A required named query does not exist. The following exception message is returned:

```
oracle.toplink.exceptions: Query is not defined
```

Action: Implement the named query. The stack trace of the exception contains the `finder` method that failed.

Common TopLink for IBM WebSphere Deploy Tool Exceptions

This section lists common exceptions that may occur when running the TopLink for IBM WebSphere application server Deploy Tool.

Class Not Found Exceptions

Cause: The class that is specified was not found; it is not included on the deploy tool classpath or the system classpath.

Action: Ensure that all required classes are included on the correct classpath. For more information about classpath setup, see the *IBM WebSphere Getting Started* document.

Note: The Deploy Tool calls external IBM classes to generate deployed code. Any exceptions that are thrown from these classes are written to `System.out`. Check **Tracing** to view the most detailed information.

Part VI

Mapping and Configuration Overview

This part describes how to use TopLink to map persistent objects to a data source and how to capture that information for use with the TopLink run-time component. It contains the following chapters.

- Chapter 16, "Understanding TopLink Mapping and Configuration Concepts"

  This chapter introduces the metadata, contained in the descriptor, used by TopLink to generate SQL statements that create, read, modify, and delete objects.

16

Understanding TopLink Mapping and Configuration Concepts

TopLink uses metadata (see "Understanding TopLink Metadata" on page 2-19) to describe how objects relate to a data source representation. Your mapping and configuration activities construct this metadata.

After creating the metadata, you can use it in any number of applications by referencing the metadata from a session (see Chapter 72, "Understanding TopLink Sessions"). The TopLink runtime uses this metadata in all persistence and data transformation operations.

This chapter includes information on the following:

- Mapping and Configuration Concepts

Mapping and Configuration Concepts

This section describes concepts unique to TopLink mapping and configuration, including the following:

- Projects
- Descriptors
- Mappings

Projects

The `Project` class is the primary container in which TopLink stores its mapping and configuration metadata. A project relates a set of object classes to a data source at the data model level.

A project contains a descriptor (see "Descriptors" on page 16-2) for each class and each descriptor contains a mapping (see "Mappings" on page 16-2) for each data member that TopLink should persist or transform.

Using TopLink Workbench, you can export mapping and configuration metadata into a deployment XML file called `project`. For more information, see "Exporting Project Information" on page 18-13.

After creating the project XML file, you must associate it with a session so that TopLink can use it at run time. For more information, see "Configuring a Primary Mapping Project" on page 74-2.

For Enterprise JavaBeans (EJB) applications where there is no session, deploy the project XML file to the target application server. In this context, the project XML file is also known as the deployment XML file.

For more information, see the following:

- "sessions.xml File" on page 8-4
- "project.xml File" on page 8-2
- "Understanding Projects" on page 17-1.

Descriptors

Descriptors describe how a Java class relates to a data source representation. They relate object classes to the data source at the data model level. For example, persistent class attributes may map to database columns.

TopLink uses descriptors to store the information that describes how an instance of a particular class can be represented in a data source (see "Mappings" on page 16-2). Most descriptor information can be defined by TopLink Workbench, then read from the project XML file at run time.

See "Understanding Descriptors" on page 23-1 for more information.

Mappings

Mappings describe how individual object attributes relate to a data source representation. Mappings can involve a complex transformation or a direct entry.

TopLink uses mappings to determine how to transform data between object and data source representation. Most mapping information can be defined by TopLink Workbench, then read from the project XML file at run time. Mappings are owned by descriptors (see "Descriptors" on page 16-2).

See "Understanding Mappings" on page 30-1 for more information.

Part VII

Projects

This part describes the TopLink artifact used to contain mapping and data source-specific information. It contains the following chapters.

- Chapter 17, "Understanding Projects"

  This chapter describes each of the different TopLink project types and important project concepts.

- Chapter 18, "Creating a Project"

  This chapter contains procedures for creating TopLink projects.

- Chapter 19, "Configuring a Project"

  This chapter explains how to configure TopLink project options common to two or more project types.

- Chapter 20, "Configuring a Relational Project"

  This chapter explains how to configure project options specific to a relational project.

- Chapter 21, "Configuring an EIS Project"

  This chapter explains how to configure project options specific to an enterprise information system (EIS) project.

- Chapter 22, "Configuring an XML Project"

  This chapter explains how to configure project options specific to an XML project.

17

Understanding Projects

A TopLink project encapsulates both mapping metadata and, where relevant, data source access information. The project is the primary object used by TopLink at run time. Each session (excluding the session broker) in a deployed application references a single project.

This chapter explains the following:

- TopLink Project Types
- Project Concepts
- Understanding the Project API
- Understanding Sequencing in Relational Projects
- Understanding XML Namespaces

TopLink Project Types

Table 17–1 lists the project types available in TopLink, classifies each as basic or advanced, and indicates how to create each.

Table 17–1 TopLink Project Types

Project Type	Description	Type	TopLink Workbench	Java
"Relational Projects" on page 17-6	A project for transactional persistence of Java objects to a relational database or an object-relational database accessed using Java Database Connectivity (JDBC). Supports TopLink queries and expressions.	Basic	✔	✔
"EIS Projects" on page 17-7	A project for transactional persistence of Java objects to a nonrelational data source accessed using a J2EE Connector Architecture (J2C) adapter and any supported EIS record type, including indexed, mapped, or XML. Supports TopLink queries and expressions.	Advanced	✔	✔
"XML Projects" on page 17-9	A project for nontransactional, nonpersistent (in-memory) conversion between Java objects and XML schema (XSD)-based documents using Java Architecture for XML Binding (JAXB). Does not support TopLink queries and expressions.	Advanced	✔	✔

For more information, see the following:

- Chapter 18, "Creating a Project"
- Chapter 19, "Configuring a Project"

- Chapter 72, "Understanding TopLink Sessions"

Project Concepts

This section describes concepts unique to TopLink projects, including the following:

- Project Architecture
- Relational and Nonrelational Projects
- Persistent and Nonpersistent Projects
- Projects and Login
- Projects and Platforms
- Projects and Sequencing
- XML Namespaces

Project Architecture

The project type you choose determines the type of descriptors and mappings you can use. There is a project type for each data source type that TopLink supports.

Table 17–2 summarizes the relationship between project, descriptor, and mappings.

Table 17–2 Project, Descriptor, and Mapping Support

Project	Descriptor	Mapping
Relational Projects	Relational Descriptors Object-Relational Descriptors	Relational Mappings Object-Relational Mappings
EIS Projects	EIS Descriptors	EIS Mappings
XML Projects	XML Descriptors	XML Mappings

Relational and Nonrelational Projects

TopLink supports both relational and nonrelational projects.

Relational projects persist Java objects to a relational database.

Nonrelational projects persist Java objects to another (nonrelational) type of data source, or perform nonpersistent (see "Persistent and Nonpersistent Projects" on page 17-2) data conversion. For example, to persist Java objects to an EIS data source by using a J2C adapter, use an EIS project. To perform nonpersistent (in-memory) conversions between Java objects and XML elements, use an XML project.

Persistent and Nonpersistent Projects

TopLink supports projects you use for applications that require persistent storage of Java objects. For example, use a relational project to persist Java objects to a relational database, or an EIS project to persist Java objects to an EIS data source by way of a J2C adapter.

TopLink also supports projects you use for applications that require only nonpersistent (in-memory) data conversion. For example, use an XML project to perform nonpersistent (in-memory) conversion between Java objects and XML elements.

Projects and Login

The login (if any) associated with a project determines how the TopLink runtime connects to the project's data source.

A login includes details of data source access, such as authentication, use of connection pools, and use of external transaction controllers. A login owns a platform.

A platform includes options specific to a particular data source, such as binding, use of native SQL, use of batch writing, and sequencing. For more information about platforms, see "Projects and Platforms" on page 17-4.

For projects that do not persist to a data source, a login is not required. For projects that do persist to a data source, a login is always required.

In TopLink Workbench, the project type determines the type of login that the project uses, if applicable.

You can use a login in a variety of roles. A login's role determines where and how you create it. The login role you choose depends on the type of project you are creating and how you intend to use the login:

- Non-CMP Session Role: Session Login
- CMP Deployment Role: Deployment Login
- Development Role: Development Login

Non-CMP Session Role: Session Login

You create a session login in the `sessions.xml` file for TopLink applications that do not use container-managed persistence (CMP).

Typically, the TopLink runtime instantiates a project when you load a session from the `sessions.xml` file (see Chapter 75, "Acquiring and Using Sessions at Run Time"). The runtime also instantiates a login and its platform based on the information in the `sessions.xml` file.

The TopLink runtime uses the information in the session login whenever you perform a persistence operation using the session in your non-CMP TopLink application.

In this case, you do not configure a deployment login (see "CMP Deployment Role: Deployment Login" on page 17-3).

If you are using TopLink Workbench and your login is based on a relational database platform, you must also configure a development login (see "Development Role: Development Login" on page 17-4).

If a `sessions.xml` file contains a login, this login overrides any other login definition.

There is a session login type for each project type that persists to a data source. For a complete list of login types available, see "Data Source Login Types" on page 81-2.

For information on configuring a session login, see "Configuring a Session Login" on page 74-4.

CMP Deployment Role: Deployment Login

You create a deployment login in the `project.xml` file (also known as the `toplink-ejb-jar.xml` file) for a TopLink-enabled CMP application.

When you deploy your TopLink-enabled CMP application with its `toplink-ejb-jar.xml` file, the application server or EJB container uses the

information in the deployment login whenever your business logic performs a persistence operation from within an entity bean with container-managed persistence.

In this case, you do not configure a session login (see "Non-CMP Session Role: Session Login" on page 17-3). In fact, there is no `session.xml` file at all (see "CMP Applications and Session Metadata" on page 8-4).

If you are using TopLink Workbench and your login is based on a relational database platform, you must also configure a development login (see "Development Role: Development Login" on page 17-4).

For information on creating a deployment login, see "Configuring Development and Deployment Logins" on page 20-6.

Development Role: Development Login

Using TopLink Workbench, you create a development login in the TopLink Workbench project file when your project is based on a relational database platform.

TopLink Workbench uses the information in the development login whenever you perform a data source operation from within TopLink Workbench, for example, whenever you read or write schema information from or to a data store during application development. The development login information is never written to a `sessions.xml` or `project.xml` file.

The development login is never used when you deploy your application: it is overridden by either the `sessions.xml` file (see "Non-CMP Session Role: Session Login" on page 17-3) or the `project.xml` file (see "CMP Deployment Role: Deployment Login" on page 17-3).

For more information on creating a development login, see "Configuring Development and Deployment Logins" on page 20-6.

Projects and Platforms

The platform (if any) associated with a project tells the TopLink runtime what specific type of data source a project uses.

A platform includes options specific to a particular data source, such as binding, use of native SQL, use of batch writing, and sequencing.

A login includes details of data source access, such as authentication, use of connection pools, and use of external transaction controllers. A login owns a platform. For more information about logins, see "Projects and Login" on page 17-3.

For projects that do not persist to a data source, a platform is not required. For projects that do persist to a data source, a platform is always required.

In TopLink Workbench, the project type determines the type of platform that the project uses, if applicable.

There is a platform type for each project type that persists to a data source. For a complete list of platform types available, see "Data Source Platform Types" on page 81-3.

Projects and Sequencing

An essential part of maintaining object identity (see "Cache Type and Object Identity" on page 87-3) is sequencing: managing the assignment of unique values to distinguish one instance from another.

Projects have different sequencing requirements, depending on their types:

- For relational projects, you typically obtain object identifier values from a separate sequence table (or database object) dedicated to managing object identifier values (see "Understanding Sequencing in Relational Projects" on page 17-14).
- For EIS projects, you typically use a returning policy (see "Configuring Returning Policy" on page 25-67) to obtain object identifier values managed by the EIS data source.
- For XML projects, because you are simply performing transformations between objects and XML documents, sequencing is not an issue.

To configure sequencing, you must configure:

- How to obtain sequence values (see "Configuring how to Obtain Sequence Values" on page 17-5), and
- Where to write sequence values when an instance of a descriptor's reference class is created (see "Configuring Where to Write Sequence Values" on page 17-5)

Depending on the type of sequencing you use and the architecture of your application, you may consider using a sequence connection pool. For more information, see "Sequence Connection Pools" on page 81-8.

Configuring how to Obtain Sequence Values

To determine how TopLink obtains sequence values, you configure TopLink sequencing at the project or session level, depending on the type of project you are building:

- In a CMP project, you do not configure a session directly: in this case, you must configure sequences at the project level (see "Configuring Sequencing at the Project Level" on page 20-3).
- In a non-CMP project, you can configure a session directly: in this case, you can use session-level sequence configuration instead of project-level sequence configuration or to override project level sequence configuration on a session-by-session basis, if required (see "Configuring Sequencing at the Session Level" on page 83-4).

Configuring Where to Write Sequence Values

To tell TopLink into which table and column to write the sequence value when an instance of a descriptor's reference class is created, you configure TopLink sequencing at the descriptor level (see "Configuring Sequencing at the Descriptor Level" on page 26-3).

XML Namespaces

As defined in http://www.w3.org/TR/REC-xml-names/, an XML namespace is a collection of names, identified by a URI reference, which are used in XML documents as element types and attribute names. To promote reusability and modularity, XML document constructs should have universal names, whose scope extends beyond their containing document. XML namespaces are the mechanism which accomplishes this.

XML namespaces are applicable in projects that reference an XML schema: EIS projects that use XML records (see "EIS Projects" on page 17-7) and XML projects (see "XML Projects" on page 17-9).

For more information, see "Understanding XML Namespaces" on page 17-22.

Relational Projects

Use a relational project for transactional persistence of Java objects to a conventional *relational* database (see "Building Relational Projects for a Relational Database" on page 17-6) or to an *object-relational* database that supports data types specialized for object storage (see "Building Relational Projects for an Object-Relational Database" on page 17-6), both accessed using JDBC.

Note: If you are using TopLink Workbench, you must add your JDBC driver to the TopLink Workbench classpath. If you are using TopLink Workbench and direct to XML type mappings (see "Direct-to-XMLType Mapping" on page 33-4), you must add the Oracle Database `xdb.jar` file to the TopLink Workbench classpath.

For more information, see "Configuring the TopLink Workbench Environment" on page 4-2.

In a relational project, you can make full use of TopLink queries and expressions (see Chapter 93, "Understanding TopLink Queries").

Building Relational Projects for a Relational Database

TopLink Workbench provides complete support for creating relational projects that map Java objects to a conventional relational database accessed using JDBC.

Table 17–3 describes the components of a relational project for a relational database.

Table 17–3 Components of a Relational Project for a Relational Database

Component	Supported Types
Data Source	For more information, see the following: ■ "DatabaseLogin" on page 81-2 ■ "Database Platforms" on page 81-3
Descriptors	For more information, see "Relational Descriptors" on page 23-11.
Mappings	For more information, see the following: ■ Part IX, "Mappings" ■ Part X, "Relational Mappings"

Building Relational Projects for an Object-Relational Database

TopLink Workbench does not currently support relational projects for an object-relational database. You must create such a relational project in Java.

Using Java, you can create a relational project for transactional persistence of Java objects to an object-relational database that supports data types specialized for object storage (such as Oracle Database) accessed using JDBC.

When using TopLink to build a relational project for an object-relational database, consider the following:

- You must create a Java class and a TopLink `ObjectRelationalDescriptor` for each structured type (`Struct/object-type`)
- TopLink supports only arrays (`Varrays`) of basic types or arrays on structured types (`Struct/object-type`).

TopLink does not support arrays of `Refs` or arrays of nested tables.

- TopLink supports only nested tables of `Refs`.

  TopLink does not support nested tables of basic types, structured types, or array types.

The general development process for building a relational project for an object-relational database is as follows:

1. Define structured object-types in the database.
2. Define tables of the structured object-types in the database.
3. Define the Java classes that will map to the structured object-types.
4. Create a relational project (see "Creating a Project" on page 18-1).
5. Create an object-relational descriptor for each Java class (see "Creating an Object-Relational Descriptor" on page 24-3).
6. Create object-relational mappings from each persistent field of each Java class to the corresponding object-types and object-type tables (see Chapter 47, "Configuring an Object-Relational Mapping").

Table 17–4 describes the components of a relational project for an object-relational database.

Table 17–4 Components of a Relational Project for an Object-Relational Database

Component	Supported Types
Data Source	For more information, see the following: ■ "DatabaseLogin" on page 81-2 ■ "Database Platforms" on page 81-3
Descriptors	For more information, see "Object-Relational Descriptors" on page 23-11.
Mappings	For more information, see the following: ■ Part IX, "Mappings" ■ Part XI, "Object-Relational Mappings"

EIS Projects

Use an EIS project for transactional persistence of Java objects to a *nonrelational* data source accessed using a J2EE Connector Architecture (J2C) adapter and EIS records.

J2C provides a Common Client Interface (CCI) API to access nonrelational EIS. This provides a similar interface to nonrelational data sources as JDBC provides for relational data sources. This API defines several types of nonrelational record types including mapped and indexed. XML has emerged as the standard format to exchange data, and most leading J2C adapter providers have extended the CCI API to define XML data records.

To use a JCA adapter with TopLink EIS, the JCA adapter must support the JCA CCI interface. At run time, your JCA adapter and the Java `connector.jar` file (that contains the `javax.resource.cci` and `javax.resource.spi` interfaces that TopLink EIS uses) must be on your application or application server classpath.

If you are using TopLink Workbench, you must add your JCA adapter to the TopLink Workbench classpath. By default, TopLink Workbench updates its classpath to include the Java 1.5 `connector.jar` file from `<TOPLINK_HOME>/j2ee/home/lib`. If this

version of the `connector.jar` file is incompatible with your environment, edit the `workbench.cmd` or `workbench.sh` file in `<TOPLINK_HOME>/bin` to change the path to this file. For more information, see "Configuring the TopLink Workbench Environment" on page 4-2.

EIS includes legacy data sources, enterprise applications, legacy applications, and other information systems. These systems include such sources as Custormer Information Control System (CICS), Virtual Storage Access Method (VSAM), Information Management System (IMS), ADABASE database, and flat files.

Oracle recommends using EIS projects to integrate TopLink with a legacy or nonrelational data source. Other methods of accessing EIS data sources include:

- Using a specialized JDBC driver that allows connecting to an EIS system as if it were a relational database. You could use a TopLink relational project with these drivers (see "Relational Projects" on page 17-6).
- Linking to or integrating with the EIS data from a relational database, such as Oracle Database.
- Using a proprietary API to access the EIS system. In this case it may be possible to wrap the API with a J2C CCI interface to allow usage with a TopLink EIS project.

TopLink provides support for mapping Java objects to EIS mapped, indexed, and XML records, through J2C, using the TopLink mappings described in Part XII, "EIS Mappings".

You configure a TopLink EIS descriptor to use a particular EIS record format (see "Configuring Record Format" on page 28-5). TopLink EIS mappings use their EIS descriptor's record format configuration to determine how to map their Java objects to EIS records.

If you use XML records, the TopLink runtime performs XML data conversion based on one or more XML schemas. In an EIS project that uses XML records, TopLink Workbench directly references schemas in the deployment XML, and exports mappings configured with respect to the schemas you specify. For information on how to use TopLink Workbench with XML schemas, see "Working With XML Schemas" on page 4-33. For information on how TopLink supports XML namespaces, see "XML Namespaces" on page 17-5.

Table 17–5 describes the components of an EIS project.

Table 17–5 EIS Project Components

Component	Supported Types
Data Source	For more information, see the following: ■ "EISLogin" on page 81-3 ■ "EIS Platforms" on page 81-4
Descriptors	For more information, see "EIS Descriptors" on page 23-12.
Mappings	For more information, see the following: ■ Part IX, "Mappings" ■ Part XII, "EIS Mappings"

You can create an EIS project with TopLink Workbench for use with EIS XML records (see "Building EIS Projects With XML Records" on page 17-9) or you can build an EIS project in Java for use with any supported EIS record type (see "Building EIS Projects With Indexed or Mapped Records" on page 17-9).

In an EIS project, your EIS interactions (see "Enterprise Information System (EIS) Interactions" on page 93-19) can make full use of TopLink queries (see Chapter 93, "Understanding TopLink Queries"). However, you cannot use TopLink expressions with EIS: in an EIS project, interactions replace expressions.

Building EIS Projects With XML Records

TopLink Workbench provides complete support for creating EIS projects that map Java objects to EIS XML records.

Using TopLink Workbench, you can create an EIS project for transactional persistence of Java objects to a non-relational data source accessed using a J2C adapter and EIS XML records.

The TopLink runtime performs XML data conversions based on one or more XML schemas. In an EIS project, TopLink Workbench does not directly reference schemas in the deployment XML, but instead exports mappings configured in accordance to specific schemas.

EIS queries use `XMLInteraction`. For more information, see "Using EIS Interactions" on page 94-24.

Building EIS Projects With Indexed or Mapped Records

TopLink Workbench does not currently support non-XML EIS projects. You must create such an EIS project in Java.

Using Java, you can create an EIS project for transactional persistence of Java objects to a nonrelational data source accessed using a J2C adapter and any supported EIS record type including indexed, mapped, or XML records.

If you use XML records, the TopLink runtime performs XML data conversion based on one or more XML schemas. When you create an EIS project in Java, you configure mappings with respect to these schemas, but the TopLink runtime does not directly reference them.

You can base queries on any supported EIS interaction: `IndexedInteraction`, `MappedInteraction` (including `QueryStringInteraction`), or `XMLInteraction` (including `XQueryInteraction`). For more information, see "Using EIS Interactions" on page 94-24.

XML Projects

Use an XML project for nontransactional, nonpersistent (in-memory) conversions between Java objects and XML documents using JAXB (see "TopLink Support for Java Architecture for XML Binding (JAXB)" on page 17-10 and "JAXB Validation" on page 17-14). TopLink Workbench provides complete support for creating XML projects.

The TopLink runtime performs XML data conversion based on one or more XML schemas. In an XML project, TopLink Workbench directly references schemas in the deployment XML, and exports mappings configured with respect to the schemas you specify. For information on how to use TopLink Workbench with XML schemas, see "Working With XML Schemas" on page 4-33. For information on how TopLink supports XML namespaces, see "XML Namespaces" on page 17-5.

Table 17–6 describes the components of an XML project.

Table 17–6 XML Project Components

Component	Supported Types
Data Source	None
Descriptors	For more information, see "XML Descriptors" on page 23-12.
Mappings	For more information, see the following: ■ Part IX, "Mappings" ■ Part XIII, "XML Mappings"

In an XML project, you do not use TopLink queries and expressions.

TopLink Support for Java Architecture for XML Binding (JAXB)

JAXB provides a standard Java object-to-XML API. For more information, see `http://java.sun.com/xml/jaxb/index.html`.

TopLink provides an extra layer of functions on top of JAXB. It allows for the creation and subsequent manipulation of mappings (in the form of a TopLink Workbench project) from an existing object model, without requiring the recompilation of the JAXB object model.

An essential component of this function is the TopLink JAXB compiler. Using the TopLink JAXB compiler, you can generate both a TopLink XML project and JAXB-compliant object model classes from your XML schema.

The TopLink JAXB compiler simplifies JAXB application development with TopLink by automatically generating (see "Creating an XML Project From an XML Schema" on page 18-6) both the required JAXB files (see "Understanding JAXB-Specific Generated Files" on page 17-10) and the TopLink files ("Understanding TopLink-Specific Generated Files" on page 17-11) from your XML schema (XSD) document.

For more information on using the JAXB and TopLink-specific files that the TopLink JAXB compiler generates, see "Using TopLink JAXB Compiler Generated Files at Run Time" on page 17-13.

Understanding JAXB-Specific Generated Files

The TopLink JAXB compiler generates the following JAXB-specific files from your XSD:

- Content and Element Interfaces
- Implementation Classes
- Object Factory Class
- JAXB Properties File

The JAXB runtime uses these files as specified by the JAXB specification.

All JAXB-specific files are generated in the output directory you define, and in the subdirectories implied by the target package name you define. For more information about TopLink JAXB binding compiler options, see "Creating an XML Project From an XML Schema" on page 18-6.

Before you compile your generated classes, be sure to configure your integrated development environment (IDE) classpath to include `<ORACLE_HOME>\lib\xml.jar`. For example, see Chapter 6, "Using an Integrated Development Environment".

Content and Element Interfaces All interfaces are named according to the content, element, or implementation `name` attribute defined in the XSD.

Implementation Classes All implementation classes are named according to the content, element, or implementation `name` attribute defined in the XSD and a suffix of `Impl`.

The generated implementation classes are simple domain classes, with private attributes for each JAXB property, and public `get` and `set` methods that return or set attribute values.

Object Factory Class The `ObjectFactory` class provides an instance factory method for each content and element interface. For example, given an element interface `ItemsImpl`, there would be an instance factory method in the `ObjectFactory` class with the following signature:

```
public ItemsImpl createItemsImpl() throws javax.xml.bind.JAXBException
```

The `ObjectFactory` class also provides a dynamic instance factory allocator with the following signature:

```
public static Object newInstance(Class javaContentInterface)
    throws javax.xml.bind.JAXBException
```

JAXB Properties File The JAXB properties file is named `jaxb.properties` and it contains a single entry that defines the `javax.xml.bind.context.factory` property with a value equal to the fully qualified class name of the TopLink implementation of `JAXBContext`:

```
javax.xml.bind.context.factory=oracle.toplink.ox.jaxb.JAXBContextFactory
```

Understanding TopLink-Specific Generated Files

The TopLink JAXB compiler generates the following TopLink-specific files from your XSD:

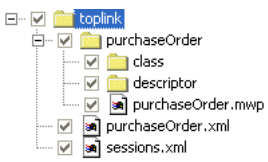
- TopLink Sessions XML File
- TopLink Project XML File
- TopLink Workbench Project
- Typesafe Enumeration Converter Amendment Method DescriptorAfterLoads Class

You use these files to customize the TopLink metadata that corresponds to the generated JAXB-specific objects.

All TopLink-specific files are generated in a subdirectory, named `toplink`, of the output directory you specify (see "Creating an XML Project From an XML Schema" on page 18-6).

The `toplink` subdirectory is organized as Figure 17–1 illustrates.

Figure 17–1 JAXB Binding Compiler Generated Files and Directories



TopLink Sessions XML File In the generated `sessions.xml` file, the `name` element is the name of the context path and the `project-xml` element is the name of the generated project XML file.

Example 17–1 shows a typical `sessions.xml` file where the context path is `examples.ox.model`.

Example 17–1 Typical Generated sessions.xml File

```
<?xml version="1.0" encoding="US-ASCII"?>
<toplink-sessions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file://xsd/sessions_10_0_3.xsd" version="0">
    <session xsi:type="database-session">
        <name>customer_example</name>
         <primary-project xsi:type="xml">purchaseOrder.xml</primary-project>
        <login xsi:type="xml-login">
            <platform-class>oracle.toplink.ox.platform.SAXPlatform</platform-class>
        </login>
    </session>
</toplink-sessions>
```

As in any TopLink project, the TopLink session is your primary facade to the TopLink XML API (see "Using TopLink JAXB Compiler Generated Files at Run Time" on page 17-13).

TopLink Project XML File The project XML file is named the same as the XSD, but with a file extension of `xml`. In Figure 17–1, the name of the generated project XML file is `purchaseOrder.xml`.

The project XML file contains a descriptor and associated mappings for each content, element, and implementation class.

TopLink Workbench Project The TopLink Workbench project is written to a subdirectory as Figure 17–1 illustrates. This subdirectory contains a TopLink Workbench project file named with the same name as the XSD, but with a file extension of `mwp` and the required `descriptor` and `class` subdirectories. In Figure 17–1, the TopLink Workbench project is named `purchaseOrder.mwp`.

Optionally, you can use this TopLink Workbench project to customize the generated descriptors and mappings and reexport the project XML file.

Typesafe Enumeration Converter Amendment Method DescriptorAfterLoads Class The TopLink JAXB compiler will generate a single class called `DescriptorAfterLoads` if any implementation class contains a mapping to a type safe enumeration (see "Mappings and JAXB Typesafe Enumerations" on page 30-24).

The TopLink JAXB compiler will update this `DescriptorAfterLoads` class with a descriptor amendment method called `amend<`*`ImplementationClassName`*`>` for each implementation class that contains a mapping to a typesafe enumeration. The amendment method sets an instance of `JAXBTypesafeEnumConverter` on each mapping that maps to a typesafe enumeration in that implementation class.

The TopLink Workbench project that the TopLink JAXB compiler creates (see "TopLink Workbench Project" on page 17-12) configures the implementation class descriptor with this amendment method. You can open the generated TopLink Workbench project and regenerate deployment XML without losing support for this feature.

Using TopLink JAXB Compiler Generated Files at Run Time

At run time, you can access the TopLink JAXB compiler-generated files:

- Using the TopLink XML Context
- Using the JAXB Context

Using the TopLink XML Context TopLink provides an `XMLContext` class with which you can create instances of TopLink `XMLMarshaller`, `XMLUnmarshaller`, and `XMLValidator`.

The `XMLContext` is thread-safe. For example, if multiple threads accessing the same `XMLContext` object request an `XMLMarshaller`, each will receive their own instance of `XMLMarshaller`, so any state that the `XMLMarshaller` maintains will be unique to that process. The same is true of instances of `XMLUnmarshaller` and `XMLValidator`.

By using the `XMLContext`, you can use TopLink XML in multithreaded architectures, such as the binding layer for Web services.

To use the TopLink `XMLContext`, do the following:

1. Configure your application classpath to include your domain object class files (see "Understanding JAXB-Specific Generated Files" on page 17-10) and the TopLink runtime.
2. Acquire the session manager (see "Acquiring the Session Manager" on page 75-2).
3. Use the session manager to acquire your XML session using the `sessions.xml` file you generated with the TopLink JAXB compiler (see "Acquiring a Session From the Session Manager" on page 75-3).
4. Get the XML project instance from the session:

   ```
   Project myProject = session.getProject();
   ```

5. Create a TopLink XML context instance with the project:

   ```
   XMLContext context = new XMLContext(myProject);
   ```

6. Use the `XMLContext` to create a TopLink `XMLMarshaller`, `XMLUnmarshaller`, and `XMLValidator`:

   ```
   XMLMarshaller marshaller = context.createMarshaller();
   marshaller.marshal(myObject, outputStream);
   marshaller.setFormattedOutput(true);

   XMLUnmarshaller unmarshaller = context.createUnmarshaller();
   Employee emp = (Employee)unmarshaller.unmarshal(new File("employee.xml"));

   XMLValidator validator = context.createValidator();
   boolean isValid = validator.validate(emp);
   ```

Using the JAXB Context You can create an instance of `JAXBContext` using the target package name you selected for your generated files (see "Understanding JAXB-Specific Generated Files" on page 17-10) as the context path, as Example 17–2 shows. This example assumes that you configure your application classpath to include your domain object class files.

The `JAXBContext` is thread-safe.

Example 17–2 Using the Context Path

```
JAXBContext jaxbContext = JAXBContext.newInstance("examples.ox.model");
Unmarshaller unmarshaller = jaxbContext.createUnmarshaller();
PurchaseOrder purchaseOrder = (PurchaseOrder)
    unmarshaller.unmarshal(new File("purchaseOrder.xml"));
```

JAXB Validation

TopLink can validate both complete object trees and subtrees against the XML schema that was used to generate the implementation classes. In addition, TopLink will validate both root objects (objects that correspond to the root element of the XML document) and nonroot objects against the schema used to generate the object's implementation class.

When validating an object tree, TopLink performs the following checks (in order):

1. Check that element appears in the document at the specified location.
2. If **maxOccurs** or **minOccurs** is specified, check number of elements.
3. If **type** is specified, check that element value satisfies the type constraints.
4. If a **fixed value** is specified, check that the element value matches it.
5. If **restrictions** (length, patterns, enumerations, and so on) are specified, check that the element value satisfies it.
6. If an **ID** type is specified during a `validateRoot` operation, check that the ID value is unique in the document.
7. If an **IDREF** type is specified during a `validateRoot` operation, check that the ID referenced exists in the document.

If validation errors are encountered, TopLink stops validating the object tree and creates a `ValidationEvent` object, according to the JAXB specification. If an error occurs in a subobject, TopLink will not validate further down that object's subtree.

For more information on using TopLink XML to perform validation, see "Using TopLink JAXB Compiler Generated Files at Run Time" on page 17-13.

For additional information on JAXB and validation, refer to the JAXB specification at `http://java.sun.com/xml/jaxb/`.

Understanding the Project API

This section describes the following:

- Project Inheritance Hierarchy

Project Inheritance Hierarchy

There is only one type of project: `oracle.toplink.sessions.Project`.

Understanding Sequencing in Relational Projects

In an relational project, you store persistent objects for your application in database tables that represent the class of instantiated object. As Figure 17–2 shows, each row of the VEHICLE_POOL table represents an instantiated object from that class, and the VEH_ID column holds the primary key for each object.

Figure 17–2 Sequencing Elements in a Class Database Table



You configure TopLink sequencing at the project or session level (see "Configuring Sequencing at the Project Level" on page 20-3 or "Configuring Sequencing at the Session Level" on page 83-4) to tell TopLink how to obtain values for the primary key column: that is, what type of sequencing to use (see "Sequencing Types" on page 17-16).

You configure TopLink sequencing at the descriptor level (see "Configuring Sequencing at the Descriptor Level" on page 26-3) to tell TopLink into which table and column to write the sequence value when an instance of a descriptor's reference class is created.

Note: When choosing a column type for a primary key value, ensure that the type provides a suitable precision. For example, if you use a `TIMESTAMP` type but your database platform's `TIMESTAMP` is defined only to the second, then identical values may be returned for objects created within the same second.

This section describes the following:

- Sequencing Configuration Options
- Sequencing Types
- Sequencing and Preallocation Size
- Sequencing With Entity Beans WIth Container-Managed Persistence

Sequencing Configuration Options

You can configure sequencing using either TopLink Workbench or Java (but not both).

Using TopLink Workbench, create one sequence with a single preallocation size that applies to all descriptors that require sequencing. You can configure table sequencing (see "Table Sequencing" on page 17-16) or native sequencing (see "Native Sequencing With an Oracle Database Platform" on page 17-18). If you choose table sequencing, you can either use default table and column names or specify your own (see "Default Versus Custom Sequence Table" on page 17-17). Oracle recommends using TopLink Workbench to configure sequencing. Using TopLink Workbench, you can easily configure the sequencing options applicable to most applications. For more information, see "Configuring Sequencing at the Project Level" on page 20-3 or "Configuring Sequencing at the Session Level" on page 83-4.

Using Java, you can configure any sequence type that TopLink supports (see "Sequencing Types" on page 17-16). You can create any number and combination of sequences per project. You can create a sequence object explicitly or use the platform default sequence (see "Default Sequencing" on page 17-18). You can associate the same sequence with more than one descriptor or associate different sequences (and different sequence types) to various descriptors. You can configure a separate preallocation size for each descriptor's sequence. For more information, see "Using Java" on page 83-5.

Sequencing Types

TopLink supports the following sequence types:

- Table Sequencing
- Unary Table Sequencing
- Query Sequencing
- Default Sequencing
- Native Sequencing With an Oracle Database Platform
- Native Sequencing With a Non-Oracle Database Platform

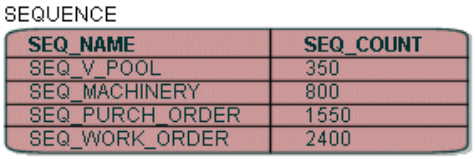
Table Sequencing

With table sequencing, you create a single database table that includes sequencing information for one or more sequenced objects in the project. TopLink maintains this table to track sequence numbers for these object types.

As Figure 17–3 shows, the table may contain sequencing information for more than one 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 17–3 TopLink Table Sequence Table



The rows of the `SEQUENCE` table represent each sequence object: one for each class that participates in sequencing or a single sequence object across several classes so that they can benefit from the same preallocation pool. When you configure sequencing at the descriptor level (see "Configuring Sequencing at the Descriptor Level" on page 26-3), you specify the `SEQ_NAME` for the class. Add a row with that name to the `SEQUENCE` table and initialize the `SEQ_COUNT` column to the value `0`.

Each time a new instance of a class is created, TopLink obtains the required sequence value. For efficiency, TopLink uses preallocation to reduce the number of table accesses required to obtain sequence values (see "Sequencing and Preallocation Size" on page 17-20).

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

- Use TopLink Workbench to create the table. See "Generating Tables on the Database" on page 4-32 for more information.
- Use the TopLink table creator to create and update the table manually. See "Generating SQL Creation Scripts" on page 4-29 for more information.

You can configure table sequencing using TopLink Workbench or Java. Oracle recommends that you use TopLink Workbench. For more information about

configuring table sequencing, see "Configuring Sequencing at the Project Level" on page 20-3 or "Configuring Sequencing at the Session Level" on page 83-4.

Default Versus Custom Sequence Table In most cases, you implement table sequencing using the default table and column names. However, you may want to specify your own table and column names if:

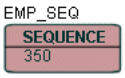
- You want to use an existing sequence table for sequencing.
- You do not want to use the default naming convention for the table and its columns.

Unary Table Sequencing

Although similar to table sequencing (see "Table Sequencing" on page 17-16), with unary table sequencing, you create a separate sequence table for each sequenced object in the project.

As Figure 17–4 shows, sequencing information appears in the table for a single class that uses sequencing. You can name the table anything you want but it must contain only one column named (by default) `SEQUENCE`.

Figure 17–4 TopLink Unary Table Sequence Table



When you configure sequencing at the descriptor level, you specify the sequence name for the class: this is the name of the unary table sequence table. Figure 17–4 shows a unary table sequence for the `Employee` class. The `Employee` class descriptor is configured (see "Configuring Sequencing at the Descriptor Level" on page 26-3) with a sequence name of `EMP_SEQ` to match the unary table sequence table name. TopLink adds a row to this table and initializes the `SEQUENCE` column to the value `1`.

Each time a new class is created, TopLink obtains the required sequence value from the single row of the unary sequence table corresponding to the class. For efficiency, TopLink uses preallocation to reduce the number of table accesses required to obtain sequence values (see "Sequencing and Preallocation Size" on page 17-20).

You can create the unary table sequence table on the database in one of two ways:

- Use TopLink Workbench to create the table. See "Generating Tables on the Database" on page 4-32 for more information.
- Use the TopLink table creator to create and update the table manually. See "Generating SQL Creation Scripts" on page 4-29 for more information.

If you are migrating a BEA WebLogic CMP application to OC4J and TopLink persistence (see "Migrating BEA WebLogic Persistence to OC4J TopLink Persistence" on page 7-16), the TopLink migration tool does not migrate BEA WebLogic single column sequence tables to TopLink unary sequence tables (see "Unary Table Sequencing" on page 17-17). After migration, you must manually configure your project to use TopLink unary sequence tables if your application originally used single column sequence tables in BEA WebLogic.

Currently, you can only configure unary table sequencing in Java using the `UnaryTableSequence` class (for more information, see "Using Java" on page 83-5).

Query Sequencing

With query sequencing, you can access a sequence resource using custom read (`ValueReadQuery`) and update (`DataModifyQuery`) queries and a preallocation size that you specify. This allows you to perform sequencing using stored procedures and allows you to access sequence resources that are not supported by the other sequencing types that TopLink provides.

Currently, you can only configure query sequencing in Java using the `QuerySequence` class (for more information, see "Configuring Query Sequencing" on page 83-7).

Default Sequencing

The platform owned by a login is responsible for providing a default sequence instance appropriate for the platform type. For example, by default, a `DatabasePlatform` provides a table sequence using the default table and column names (see "Table Sequencing" on page 17-16).

You can access this default sequence directly using `DatasourceLogin` method `getDefaultSequence`, or indirectly by using the `DefaultSequence` class, a wrapper for the platform default sequence.

If you associate a descriptor with a nonexistent sequence, the TopLink runtime will create an instance of `DefaultSequence` to provide sequencing for that descriptor. For more information, see "Configuring the Platform Default Sequence" on page 26-6.

The main purpose of the `DefaultSequence` is to allow a sequence to use a different pre-allocation size than the project default.

Currently, you can only make use of default sequencing in Java (for more information, see "Using the Platform Default Sequence" on page 83-6).

Native Sequencing With an Oracle Database Platform

TopLink support for native sequencing with Oracle databases is similar to table sequencing (see "Table Sequencing" on page 17-16), except that TopLink does not maintain a table in the database. Instead, the database contains a sequence object that stores the current maximum number and preallocation size for sequenced objects. The sequence name configured at the descriptor level identifies the sequence object responsible for providing sequencing values for the descriptor's reference class.

You can configure native sequencing using TopLink Workbenchor Java. Oracle recommends that you use TopLink Workbench. For more information about configuring table sequencing, see "Configuring Sequencing at the Project Level" on page 20-3 or "Configuring Sequencing at the Session Level" on page 83-4.

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

As with table sequencing, 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 TopLink uses the result to build the sequencing pool.

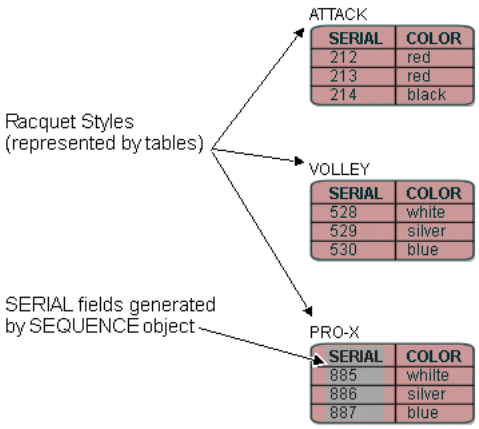
The key difference between this process and the process involved in table sequencing is that TopLink is unaware of the `INCREMENT` construct on the `SEQUENCE` object. TopLink sequencing and the Oracle `SEQUENCE` object operate in isolation. To avoid sequencing errors in the application, set the TopLink preallocation size and the Oracle `SEQUENCE` object `INCREMENT` to the same value. Note that the Oracle sequence object must have a starting value equal to the preallocation size because when TopLink gets the next sequence value, it assume it has the previous preallocation size of values.

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 create only one `SEQUENCE` object for those classes.

For example, in Figure 17–5, 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 17–5 Example of Database Tables–Racquet Information



The manufacturer can do either of the following:

- *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 racquets.* 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.

Native Sequencing With a Non-Oracle Database Platform

Several databases support a type of native sequencing in which the database management system generates the sequence numbers.

When you create a database table for a class that uses native sequencing, include a 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:** TopLink does not support native sequencing in IBM DB2 databases.

When you insert a new object into the table, 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, TopLink reads back the primary key for the new object so that the object has an identity in the TopLink cache.

> **Note:** This type of sequencing does not support preallocation, so the preallocation size must be set to 1. To take advantage of sequence preallocation, Oracle recommends that you use table sequencing on these databases instead of native sequencing.

If your database provides native sequencing, but TopLink does not directly support it, you may be able to access the native sequence object using a query sequence and stored procedures. For more information, see "Query Sequencing" on page 17-18.

You can configure native sequencing using TopLink Workbenchor Java. Oracle recommends that you use TopLink Workbench. For more information about configuring table sequencing, see "Configuring Sequencing at the Project Level" on page 20-3 or "Configuring Sequencing at the Session Level" on page 83-4.

Sequencing and Preallocation Size

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

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, TopLink sets preallocation size to `50`. You can specify preallocation size either in TopLink Workbench or as part of the session login.

Preallocation size configuration applies to table sequencing and Oracle native sequencing. In Oracle native sequencing, the sequence preallocation size must match the Oracle sequence object increment size. Preallocation is not available for native sequencing in other databases as they use an auto-assigned sequence column. Oracle recommends that you use table sequencing in non-Oracle databases to allow preallocation.

For table sequencing, TopLink maintains a pool of preallocated values for each sequenced class. When TopLink exhausts this pool of values, it acquires a new pool of values, as follows:

1. 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 17–3. If you create a new purchase order and TopLink has exhausted its pool of sequence numbers, then TopLink executes a SQL statement to increment `SEQ_COUNT` for `SEQ_PURCH_ORDER` by the preallocation size (in this case, the TopLink default of `50`). The database increments `SEQ_COUNT` for `SEQ_PURCH_ORDER` to `1600` and returns this number to TopLink.

2. TopLink calculates a maximum and a minimum value for the new sequence number pool, and creates the pool of values.

3. TopLink populates the object sequence attribute with the first number in the pool and writes the object to the class table.

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

Using TopLink Workbench, you specify a preallocation size when you choose a sequencing type at the project or session level. That preallocation size applies to all descriptors.

Using Java, you can specify a different preallocation size for each sequence that you create.

For more information about configuring preallocation size, see "Configuring Sequencing at the Project Level" on page 20-3 or "Configuring Sequencing at the Session Level" on page 83-4.

Sequencing With Entity Beans WIth Container-Managed Persistence

To implement sequencing for entity beans with container-managed persistence, use a sequencing strategy that implements preallocation, such as table sequencing or Oracle native sequencing. Preallocation ensures that the entity bean primary key is available at the `ejbPostCreate` method. If you use non-Oracle native sequencing (for example, Sybase, Microsoft SQL Server, or Informix database native sequencing), be aware that:

- Non-Oracle 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 prior to 3.0 expect that the primary key is available at `ejbPostCreate` method.
- TopLink CMP integration for IBM WebSphere application server does not support native sequencing other than Oracle native sequencing.
- OC4J and BEA WebLogic Server 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 TopLink cache and the container.

TopLink CMP Integration With IBM WebSphere Application Server

The TopLink CMP integration with IBM WebSphere application server does not automatically provide the primary key after calling the `ejbCreate` method. If you

deploy to a IBM WebSphere application server, explicitly set the primary key in the `ejbCreate` method. Example 17–3 illustrates this call in a WebSphere integration.

Example 17–3 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
}
```

This example uses the TopLink `SessionLookupHelper` to look up the correct session and uses the session to assign a sequence number to the bean.

TopLink CMP Integration With OC4J and BEA WebLogic Server

In the TopLink CMP integration with OC4J and BEA WebLogic Server, 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 17–4 illustrates this call in a WebLogic integration.

Example 17–4 Setting Primary Key in OC4J and BEA WebLogic

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

If you are migrating a BEA WebLogic CMP application to OC4J and TopLink persistence (see "Migrating BEA WebLogic Persistence to OC4J TopLink Persistence" on page 7-16), the TopLink migration tool automatically configures your project to use unary table sequencing (see "Unary Table Sequencing" on page 17-17) if your application originally used single-column sequence tables in BEA WebLogic.

Understanding XML Namespaces

As defined in http://www.w3.org/TR/REC-xml-names/, an XML namespace is a collection of names, identified by a URI reference, which are used in XML documents as element types and attribute names. To promote reusability and modularity, XML document constructs should have universal names, whose scope extends beyond their containing document. XML namespaces are the mechanism which accomplishes this.

XML namespaces are applicable in projects that reference an XML schema: EIS projects that use XML records (see "EIS Projects" on page 17-7) and XML projects (see "XML Projects" on page 17-9).

This section describes the following:

- TopLink Workbench Namespace Resolution
- Element and Attribute Form Options
- TopLink Runtime Namespace Resolution

TopLink Workbench Namespace Resolution

Using TopLink Workbench, you can configure the XML schema namespace for your project. For more information, see "Configuring XML Schema Namespace" on page 4-37.

Element and Attribute Form Options

The `xsd:schema` element provides attributes that you can use to specify how elements and attributes should be qualified by namespace.

This section describes the consequences of the following combinations of element and attribute form configuration:

- Element Form Default Qualified and Attribute Form Default Unqualified
- Element and Attribute Form Default Unqualified
- Element and Attribute Form Default Qualified

Element Form Default Qualified and Attribute Form Default Unqualified

Example 17–5 shows an XML schema in which a target namespace is set. It is coded with `elementFormDefault` set to `qualified` and `attributeFormDefault` set to `unqualified`. This means all elements must be namespace qualified and globally declared attributes must be namespace qualified and locally defined attributes must not be namespace qualified.

Example 17–5 XML Schema with Element Form Default Qualified and Attribute Form Default Unqualified

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified"
    xmlns="urn:namespace-example"
    targetNamespace="urn:namespace-example">
    <xsd:element name="customer" type="customer-type"/>
    <xsd:complexType name="customer-type">
        <xsd:sequence>
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element ref="date-of-birth"/>
        </xsd:sequence>
        <xsd:attribute name="id" type="xsd:integer"/>
    </xsd:complexType>
    <xsd:element name="date-of-birth" type="xsd:date"/>
</xsd:schema>
```

Example 17–6 shows an XML document that conforms to this XML schema.

Example 17–6 XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
<ns:customer xmlns:ns="urn:namespace-example" id="1">
    <ns:name>Jane Doe</ns:name>
    <ns:date-of-birth>1975-02-21</ns:date-of-birth>
</ns:customer>
```

Example 17–7 shows the Java code for a `Customer` class `XMLDescriptor` and XML mappings for its attributes to illustrate how this schema configuration affects the XPaths you specify for default root element and mappings (for more information, see "Configuring an XML Descriptor" on page 29-1 and Chapter 63, "Configuring an XML Mapping").

Example 17–7 XML Descriptors and Mappings

```
NamespaceResolver namespaceResolver = new NamespaceResolver();
namespaceResolver.put("ns", "urn:namespace-example");

XMLDescriptor customerDescriptor = new XMLDescriptor();
```

```
customerDescriptor.setJavaClass(Customer.class);
customerDescriptor.setDefaultRootElement("ns:customer");
customerDescriptor.setNamespaceResolver(namespaceResolver);

XMLDirectMapping idMapping = new XMLDirectMapping();
idMapping.setAttributeName("id");
idMapping.setXPath("@id");
customerDescriptor.addMapping(idMapping);

XMLDirectMapping nameMapping = new XMLDirectMapping();
nameMapping.setAttributeName("name");
nameMapping.setXPath("ns:name/text()");
customerDescriptor.addMapping(nameMapping);

XMLDirectMapping birthDateMapping = new XMLDirectMapping();
birthDateMapping.setAttributeName("birthDate");
birthDateMapping.setXPath("ns:date-of-birth/text()");
customerDescriptor.addMapping(birthDateMapping);
```

Element and Attribute Form Default Unqualified

Example 17–8 shows an XML schema in which a target namespace is set. It is coded with `elementFormDefault` and `attributeFormDefault` set to `unqualified`. This means that globally defined nodes must be namespace qualified and locally defined nodes must not be namespace qualified.

Example 17–8 XML Schema with Element and Attribute Form Default Unqualified

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="unqualified"
    attributeFormDefault="unqualified"
    xmlns="urn:namespace-example"
    targetNamespace="urn:namespace-example">
    <xsd:element name="customer" type="customer-type"/>
    <xsd:complexType name="customer-type">
        <xsd:sequence>
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element ref="date-of-birth"/>
        </xsd:sequence>
        <xsd:attribute name="id" type="xsd:integer"/>
    </xsd:complexType>
    <xsd:element name="date-of-birth" type="xsd:date"/>
</xsd:schema>
```

Example 17–9 shows an XML document that conforms to this XML schema.

Example 17–9 XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
<ns:customer xmlns:ns="urn:namespace-example" id="1">
    <name>Jane Doe</name>
    <ns:date-of-birth>1975-02-21</ns:date-of-birth>
</ns:customer>
```

Example 17–10 shows the Java code for a `Customer` class `XMLDescriptor` and XML mappings for its attributes to illustrate how this schema configuration affects the XPaths you specify for default root element and mappings (for more information, see "Configuring an XML Descriptor" on page 29-1 and Chapter 63, "Configuring an XML Mapping").

Example 17–10 XML Descriptors and Mappings

```
NamespaceResolver namespaceResolver = new NamespaceResolver();
namespaceResolver.put("ns", "urn:namespace-example");

XMLDescriptor customerDescriptor = new XMLDescriptor();
customerDescriptor.setJavaClass(Customer.class);
customerDescriptor.setDefaultRootElement("ns:customer");
customerDescriptor.setNamespaceResolver(namespaceResolver);

XMLDirectMapping idMapping = new XMLDirectMapping();
idMapping.setAttributeName("id");
idMapping.setXPath("@id");
customerDescriptor.addMapping(idMapping);

XMLDirectMapping nameMapping = new XMLDirectMapping();
nameMapping.setAttributeName("name");
nameMapping.setXPath("name/text()");
customerDescriptor.addMapping(nameMapping);

XMLDirectMapping birthDateMapping = new XMLDirectMapping();
birthDateMapping.setAttributeName("birthDate");
birthDateMapping.setXPath("ns:date-of-birth/text()");
customerDescriptor.addMapping(birthDateMapping);
```

Element and Attribute Form Default Qualified

Example 17–11 shows an XML schema in which a target namespace is set. It is coded with `elementFormDefault` and `attributeFormDefault` set to qualified. This means that all nodes must be namespace qualified.

Example 17–11 XML Schema with Element and Attribute Form Default Qualified

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    attributeFormDefault="qualified"
    xmlns="urn:namespace-example"
    targetNamespace="urn:namespace-example">
    <xsd:element name="customer" type="customer-type"/>
    <xsd:complexType name="customer-type">
        <xsd:sequence>
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element ref="date-of-birth"/>
        </xsd:sequence>
        <xsd:attribute name="id" type="xsd:integer"/>
    </xsd:complexType>
    <xsd:element name="date-of-birth" type="xsd:date"/>
</xsd:schema>
```

Example 17–12 shows an XML document that conforms to this XML schema.

Example 17–12 XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
<ns:customer xmlns:ns="urn:namespace-example" ns:id="1">
    <ns:name>Jane Doe</ns:name>
    <ns:date-of-birth>1975-02-21</ns:date-of-birth>
</ns:customer>
```

Example 17–13 shows the Java code for a `Customer` class `XMLDescriptor` and XML mappings for its attributes to illustrate how this schema configuration affects the XPaths you specify for default root element and mappings (for more information, see

"Configuring an XML Descriptor" on page 29-1 and Chapter 63, "Configuring an XML Mapping").

Example 17–13 XML Descriptors and Mappings

```
NamespaceResolver namespaceResolver = new NamespaceResolver();
namespaceResolver.put("ns", "urn:namespace-example");

XMLDescriptor customerDescriptor = new XMLDescriptor();
customerDescriptor.setJavaClass(Customer.class);
customerDescriptor.setDefaultRootElement("ns:customer");
customerDescriptor.setNamespaceResolver(namespaceResolver);

XMLDirectMapping idMapping = new XMLDirectMapping();
idMapping.setAttributeName("id");
idMapping.setXPath("@ns:id");
customerDescriptor.addMapping(idMapping);

XMLDirectMapping nameMapping = new XMLDirectMapping();
nameMapping.setAttributeName("name");
nameMapping.setXPath("ns:name/text()");
customerDescriptor.addMapping(nameMapping);

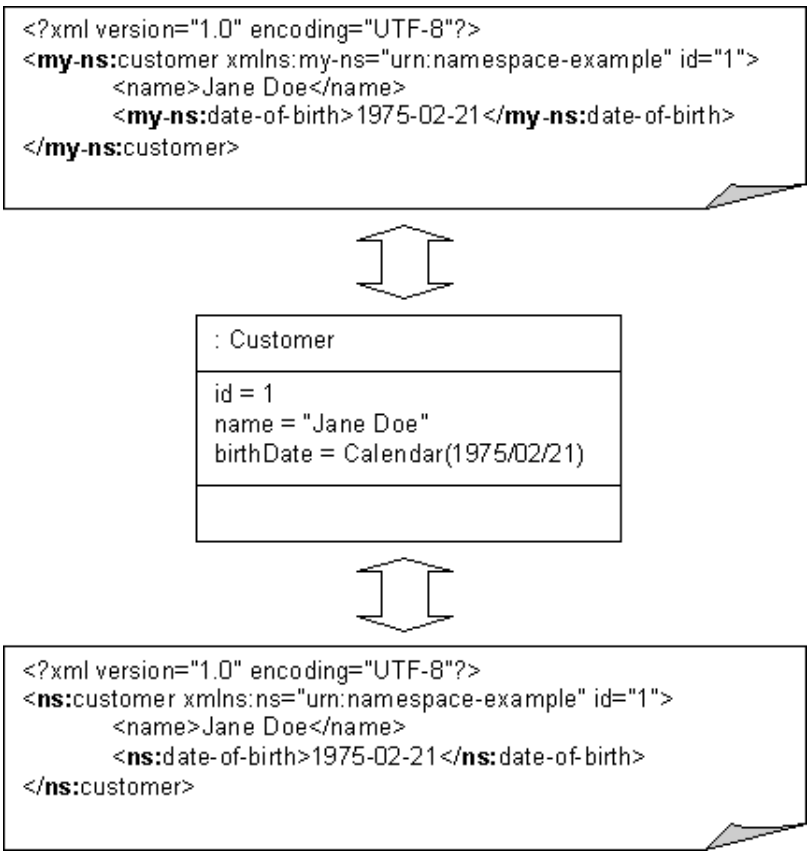
XMLDirectMapping birthDateMapping = new XMLDirectMapping();
birthDateMapping.setAttributeName("birthDate");
birthDateMapping.setXPath("ns:date-of-birth/text()");
customerDescriptor.addMapping(birthDateMapping);
```

TopLink Runtime Namespace Resolution

It is common for an XML document to include one or more namespaces. TopLink supports this using its `NamespaceResolver`. The namespace resolver maintains pairs of namespace prefixes and Uniform Resource Identifiers (URIs). TopLink uses these prefixes in conjunction with the XPath statements you specify on EIS mappings to XML records and XML mappings.

Although TopLink captures namespace prefixes in the XPath statements for mappings (if applicable), the input document is not required to use the same namespace prefixes. As Example 17–6 shows, TopLink will use the namespace prefixes specified in the mapping when creating new documents.

Figure 17–6 Namespaces in TopLink



18

Creating a Project

This chapter contains the following information:

- Project Creation Overview
- Working With Projects
- Exporting Project Information
- Working With the ejb-jar.xml File

For information on the various types of projects available, see "TopLink Project Types" on page 17-1.

Project Creation Overview

You can create a project using TopLink Workbench or Java code.

Oracle recommends using TopLink Workbench to create projects and generate deployment XML or Java source versions of the project for use at run time. For more information, see "Using TopLink Workbench" on page 18-2.

Alternatively, you can create projects in Java code. For an EIS project that uses a record type other than XML, you must use Java code. For more information, see "Using Java" on page 18-3 and Oracle TopLink API Reference.

You can use TopLink to create a project, if any of the following conditions are met:

- You have both an object and data model: see "Creating a Project for an Existing Object and Data Model" on page 18-5.
- You have an object model, but no data model yet: see "Creating a Project From an Existing Object Model" on page 18-5.
- You have a data model, but no object model yet: "Creating a Project From an Existing Data Model" on page 18-5.
- You have an XML schema (XSD) document, but no object model yet: see "Creating an XML Project From an XML Schema" on page 18-6.

  If you have both XSD and object model classes, you can create an XML project using the procedure described in "Using TopLink Workbench" on page 18-2.
- You have a non-OC4J CMP project, you can create a project by migrating your application to OC4J and TopLink persistence: see "Creating a Project by Migrating an EAR to OC4J" on page 18-9.
- You are deploying an EJB CMP application to OC4J, you can create a project (including all mappings and data model) automatically at deployment time: see

"Creating a Project From an OC4J EJB CMP EAR at Deployment Time" on page 18-10.

Using TopLink Workbench

When you create a project using TopLink Workbench, all project information is stored in the project file (`.mwp` file). This file references additional XML data files that contain the information about how the Java classes map to database tables or XML elements.

Using TopLink Workbench, you can export this information as a TopLink project XML file (that is, the deployment XML file) that is read in by the TopLink runtime. You can also export this information as a Java class. For more information, see "Exporting Project Information" on page 18-13.

TopLink Workbench displays projects and their contents in the Navigator window. When you select a project, its attributes are displayed in the Editor window. See "Using the Navigator" on page 4-9 for more information. TopLink Workbench supports the following project types:



- Relational project



- XML project



- EIS project

Creating New TopLink Workbench Projects

This section includes information on creating a new TopLink Workbench project. To create a new project from an existing persistence application (such as OC4J), see Chapter 7, "Integrating TopLink With an Application Server". To create a new project from JAXB, see "TopLink Support for Java Architecture for XML Binding (JAXB)" on page 17-10.

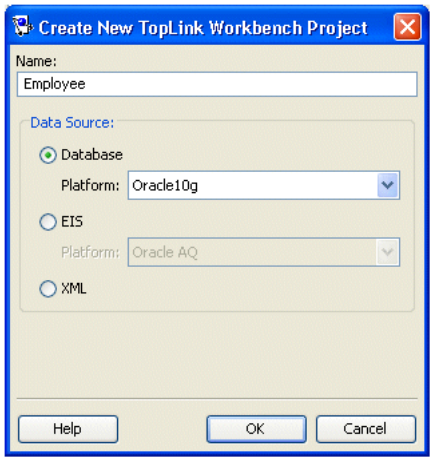
To create a new TopLink Workbench project, use this procedure:



1. Click **New** on the toolbar and select **Project**. The Create New TopLink Workbench Project dialog box appears.

   You can also create a new project by choosing **File** > **New** > **Project** from the menu.

Figure 18–1 Create New TopLink Workbench Project Dialog Box



Use the following information to enter data in each field of this dialog box:

Field	Description
Name	Enter the name of the TopLink Workbench project. This project name will also become the name of the `.mwp` file.
Data Source	Use these options to specify the type of project to create, and its data source.
Database	Select **Database** to create an relational project to a relational database. Use the **Platform** list to select the specific database platform. See "Relational Projects" on page 17-6 for more information.
EIS	Select **EIS** to create an EIS project to a nonrelational data source using XML records. Use the **Platform** list to specify the J2C adapter to use. See "EIS Projects" on page 17-7 for more information.
XML	Select **XML** to create a nontransactional, nonpersistent XML project to an XML schema. Alternatively, you can generate both an XML project and object model classes (see "Creating an XML Project From an XML Schema" on page 18-6). See "XML Projects" on page 17-9 for more information.

For more project information, continue with the following:

- Configure the project (see Chapter 19, "Configuring a Project").
- Add mappings and descriptors (see Chapter 23, "Understanding Descriptors" and Chapter 30, "Understanding Mappings").
- Export the project for use with the TopLink runtime (see "Exporting Project Information" on page 18-13).

Using Java

To create a project using Java code, use this procedure:

1. Implement a project class that extends the `oracle.toplink.sessions.Project` class (see Example 18–1).
2. Compile the project class.

Example 18–1 Specifying a TopLink Project in Code

```
/**
* The class EmployeeProject is an example of an Oracle 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. Project-level properties, such as the name of the project, should be
specified here
*/
public EmployeeProject() {
```

```
  setName("EmployeeProject");
  applyLogin();

  addDescriptor(buildAddressDescriptor());
  addDescriptor(buildEmployeeDescriptor());
  addDescriptor(buildPhoneNumberDescriptor());
}

// Data source information
public void applyLogin() {
  DatabaseLogin login = new DatabaseLogin();

  // use platform appropriate for underlying database
  login.usePlatform(
    new oracle.toplink.platform.database.oracle.Oracle9Platform());
  login.setDriverClassName("oracle.jdbc.OracleDriver");
  login.setConnectionString("jdbc:oracle:thin:@HOST:PORT:SID");
  login.setUserName("USER NAME");
  login.setEncryptedPassword("PASSWORD, ENCRYPTED");

  // Configuration Properties
    setDatasourceLogin(login);
}

/**
* Descriptors are built by defining table info, setting properties (caching, etc.)
and by adding mappings to the descriptor
*/

// SECTION: DESCRIPTOR
public ClassDescriptor buildAddressDescriptor() {
  RelationalDescriptor descriptor = new RelationalDescriptor();

  // specify the class to be made persistent
  descriptor.setJavaClass(examples.servletjsp.model.Address.class);

  // specify the tables to be used and primary key
  descriptor.addTableName("ADDRESS");
  descriptor.addPrimaryKeyFieldName("ADDRESS.ADDRESS_ID");

  // Descriptor Properties
  descriptor.useSoftCacheWeakIdentityMap();
  descriptor.setIdentityMapSize(100)
  descriptor.useRemoteSoftCacheWeakIdentityMap()
  descriptor.setRemoteIdentityMapSize(100)
  descriptor.setSequenceNumberFieldName("ADDRESS.ADDRESS_ID")
  descriptor.setSequenceNumberName("ADD_SEQ");
  descriptor.setAlias("Address");

  // Mappings
  DirectToFieldMapping cityMapping = new DirectToFieldMapping();
  cityMapping.setAttributeName("city");
  cityMapping.setFieldName("ADDRESS.CITY");
  descriptor.addMapping(cityMapping);

  // … Additional mappings are added to the descriptor using the addMapping()
method

return descriptor;
}
```

Note: Use TopLink 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 "Exporting Project Java Source" on page 18-14.

Creating a Project for an Existing Object and Data Model

If you have both an existing object model (Java classes for your domain objects) and data model (such as an existing database schema), use this procedure to create your TopLink project.

This procedure applies to relational project types.

Using TopLink Workbench

1. Create the project (see "Using TopLink Workbench" on page 18-2).
2. Configure the project classpath (see "Configuring Project Classpath" on page 19-3).
3. Import classes (see "Importing and Updating Classes" on page 4-50).
4. Import database tables (see "Importing Tables From a Database" on page 4-22).
5. Configure project options (see Chapter 19, "Configuring a Project").

Creating a Project From an Existing Object Model

If you have an existing object model (Java classes for your domain objects), but you do not have a corresponding data model, use this procedure to create your TopLink project and automatically generate the corresponding data model.

This procedure applies to relational projects.

Using TopLink Workbench

1. Create the project (see "Using TopLink Workbench" on page 18-2).
2. Configure the project classpath (see "Configuring Project Classpath" on page 19-3).
3. Import classes (see "Importing and Updating Classes" on page 4-50).
4. Generate database tables. For more information, see the following:
   - "Creating New Tables" on page 4-22
   - "Generating Tables on the Database" on page 4-32
5. Configure project options (see Chapter 19, "Configuring a Project").

Creating a Project From an Existing Data Model

If you have an existing data model (such as a database schema), but you do not have a corresponding data model (Java classes for domain objects), use this procedure to create your TopLink project and automatically generate the corresponding object model.

This procedure applies to relational projects.

Using TopLink Workbench

1. Create the project (see "Using TopLink Workbench" on page 18-2).
2. Import database tables (see "Importing Tables From a Database" on page 4-22).
3. Generate classes. For more information, see either of the following:
   - "Generating Classes and Descriptors From Database Tables" on page 4-30
   - "Generating EJB Entity Beans and Descriptors From Database Tables" on page 4-31
4. Configure project options (see Chapter 19, "Configuring a Project").

Creating an XML Project From an XML Schema

If you have an existing data model (XML schema document), but you do not have a corresponding object model (Java classes for domain objects), use this procedure to create your TopLink project and automatically generate the corresponding object model.

> **Note:** If you have both XSD and object model classes, you can create an XML project using the procedure described in "Using TopLink Workbench" on page 18-2.

Using the TopLink JAXB compiler simplifies JAXB application development with TopLink by automatically generating both the required JAXB files and the TopLink files from your XML schema (XSD) document. Once generated, you can open the TopLink Workbench project to fine-tune XML mappings without having to recompile your JAXB object model.

You can use the TopLink JAXB compiler from TopLink Workbench (see "Using TopLink Workbench" on page 18-6) or from the command line (see "Using the Command Line" on page 18-8). Oracle recommends that you use TopLink Workbench.

> **Note:** Before you compile your generated classes, be sure to configure your IDE classpath to include `<ORACLE_HOME>\lib\xml.jar`. For example, see Chapter 6, "Using an Integrated Development Environment".

For more information, see the following:

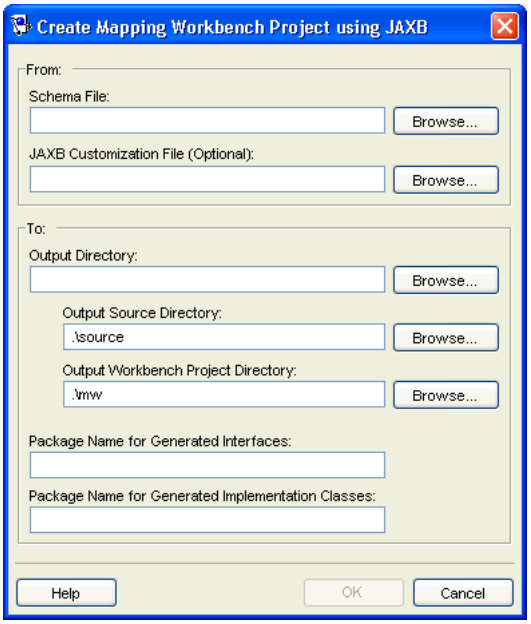
- "TopLink Support for Java Architecture for XML Binding (JAXB)" on page 17-10
- "Using TopLink JAXB Compiler Generated Files at Run Time" on page 17-13

Using TopLink Workbench

To create a new, mapped TopLink Workbench project from an XML schema using JAXB, use this procedure:

1. From TopLink Workbench, select **File > New > Project > From XML Schema (JAXB)**.

Figure 18–2 Create TopLink Workbench Project using JAXB Dialog Box



Use the following information to enter data in each field of this dialog box:

Field	Description
From	Use these fields to specify your existing JAXB information.
Schema File	Click **Browse** and select the fully qualified path to your XSD file.
JAXB Customization File	This in an optional setting. It can be used if you have a standard JAXB configuration file that you wish to use to override the default JAXB compiler behavior. The JAXB customization file contains binding declarations for customizing the default binding between an XSD component and its Java representation.
To	Use these fields to specify the location and options of the TopLink Workbench project.
Output Directory	Click **Browse** and select the path to the directory into which generated files are written. All paths used in the project are relative to this directory.
Output Source Directory	Click **Browse** and select the path to the directory (relative to the **Output Directory**) into which generated interfaces, implementation classes, and deployment files are written. Default: directory named `source` in the specified output directory.
Output Workbench Project Directory	Click Browse and select the path to the directory (relative to the **Output Directory**) into which the TopLink Workbench project files are written. Default: directory named `mw` in the specified output directory.
Package Name for Generated Interfaces	The optional name of the package to which generated interfaces belong. This defines your context path. If it is not specified, a package name of `jaxbderived.<schema name>` is used where `<schema name>` is the name of the schema specified by the **Schema File** field.

Field	Description
Package Name for Generated Implementation Classes	The optional name of the package to which generated implementation classes belong. This defines your context path. If it is not specified, a package name of `jaxbderived.<schema name>` is used where `<schema name>` is the name of the schema specified by the **Schema File** field.

The TopLink JAXB compiler generates JAXB-specific files (see "Understanding JAXB-Specific Generated Files" on page 17-10) and TopLink-specific files (see "Understanding TopLink-Specific Generated Files" on page 17-11).

Optionally, open the generated TopLink Workbench project (see "TopLink Workbench Project" on page 17-12), customize the generated mappings and descriptors, and reexport the TopLink project XML.

Note: Before you compile your generated classes, be sure to configure your IDE classpath to include `<ORACLE_HOME>\lib\xml.jar`. For example, see Chapter 6, "Using an Integrated Development Environment".

Using the Command Line

To create a new, mapped Oracle TopLink Workbench project from an XML schema using JAXB from the command line, use the `tljaxb.cmd` or `tljaxb.sh` file (located in the `<ORACLE_HOME>/toplink/bin` directory) as follows:

1. Using a text editor, edit the `tljaxb.cmd` or `tljaxb.sh` file to set proxy settings (if required).

   If you are using a schema that imports another schema by URL and you are operating behind a proxy, then you must uncomment the lines shown in Example 18–2 or Example 18–3 and edit them to set your proxy host (name or IP address) and port:

Example 18–2 Proxy Settings in tljaxb.cmd

```
@REM set JVM_ARGS=%JVM_ARGS% -DproxySet=true -Dhttp.proxyHost= -Dhttp.proxyPort=
```

Example 18–3 Proxy Settings in tljaxb.sh

```
# JVM_ARGS="${JVM_ARGS} -DproxySet=true -Dhttp.proxyHost= -Dhttp.proxyPort="
```

2. Execute the `tljaxb.cmd` or `tljaxb.sh` file (located in the `<ORACLE_HOME>/toplink/bin` directory).

   The TopLink JAXB compiler generates JAXB-specific files (see "Understanding JAXB-Specific Generated Files" on page 17-10) and TopLink-specific files (see "Understanding TopLink-Specific Generated Files" on page 17-11).

   Example 18–4 illustrates how to generate an object model from a schema using the Toplink JAXB compiler. Table 18–1 lists the compiler arguments.

Example 18–4 Generating an Object Model from a Schema with tljaxb.cmd

```
tljaxb.cmd -sourceDir ./app/src -generateWorkbench -workbenchDir ./app/mw  -schema
purchaseOrder.xsd -targetPkg examples.ox.model.if -implClassPkg
examples.ox.model.impl
```

Table 18–1 TopLink JAXB Binding Compiler Arguments

Argument	Description	Optional?
`-help`	Prints this usage information.	Yes
`-version`	Prints the release version of the TopLink JAXB compiler.	Yes
`-sourceDir`	The path to the directory into which generated interfaces, implementation classes, and deployment files are written. Default: directory named `source` in the specified output directory.	Yes
`-generateWorkbench`	Generate a TopLink Workbench project and necessary project files. If omitted, only runtime information is generated.	Yes
`-workbenchDir`	The path to the directory into which the TopLink Workbench project files are written. This argument requires the `-generateWorkbench` argument. Default: directory named `mw` in the specified output directory.	Yes
`-schema`	The fully qualified path to your XSD file.	No
`-targetPkg`	The name of the package to which both generated interfaces and classes belong. This defines your context path. To specify a different package for implementation classes, set the `-implClassPkg` argument. Default: a package name of `jaxbderived.<schema name>` where `<schema name>` is the name of the schema specified by the `-schema` argument.	Yes
`-implClassPkg`	The name of the package to which generated implementation classes belong. If this option is set, interfaces belong to the package specified by the `-targetPkg` argument. This defines your context path.	Yes
`-interface`	Generate only interfaces. This argument is optional. Default: generate both interfaces and implementation classes.	Yes
`-verbose`	The interfaces and classes generated. This argument is optional. Default: not verbose.	Yes
`-customize`	The fully qualified path and file name of a standard JAXB customization file that you can use to override default JAXB compiler behavior.	Yes

3. Optionally, open the generated TopLink Workbench project (see "TopLink Workbench Project" on page 17-12) in TopLink Workbench, customize the generated mappings and descriptors, and reexport the TopLink project XML.

> **Note:** Before you compile your generated classes, be sure to configure your IDE classpath to include `<ORACLE_HOME>\lib\xml.jar`. For example, see Chapter 6, "Using an Integrated Development Environment".

Creating a Project by Migrating an EAR to OC4J

If you configure TopLink to be the default persistence for your application server, TopLink provides automated support for migrating your existing J2EE CMP application server-specific `ejb-jar.xml` file to the required `toplink-ejb-jar.xml` file, including providing an appropriate TopLink Workbench project.

This procedure applies to CMP relational projects only.

For more information, see "Persistence Manager Migration" on page 7-4.

Creating a Project From an OC4J EJB CMP EAR at Deployment Time

For a CMP application deployed to OC4J configured to use TopLink as the persistence manager, you can use the TopLink default mapping feature to automatically generate a TopLink project, including descriptors and mappings for all persistent objects, at deployment time.

This procedure applies only to CMP relational projects deployed to OC4J configured to use TopLink as the default persistence manager.

For more information, see "Default Mapping in CMP Projects Using OC4J at Run Time" on page 30-4.

Working With Projects

Using TopLink Workbench, you can perform the following project functions:

- Opening Existing Projects
- Saving Projects
- Generating the Project Status Report

See Chapter 19, "Configuring a Project" for additional information on working with TopLink Workbench projects.

Opening Existing Projects

Use this procedure to open an existing project:

Caution: For most prior release projects, simply opening the project in TopLink Workbench will upgrade your project. However, to upgrade release 9.0.3 (and earlier) projects, you must follow specific upgrade procedures and use the Package Rename tool.

Refer to *Oracle TopLink Release Notes* and *Oracle TopLink Getting Started Guide* for more information.



1. Click **Open Project** on the toolbar. The Choose a File dialog box appears. You can also open a project by choosing **File** > **Open** from the menu.

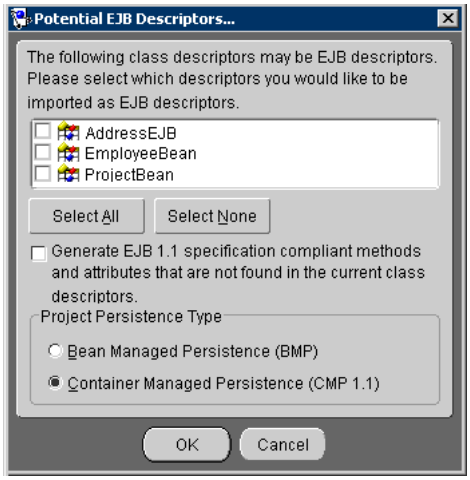
   Note: The **File** menu option contains a list of recently opened projects. You can select one of these projects to open. See "General Preferences" on page 4-13 for information on customizing this list.

   ---

2. Select the TopLink Workbench project file (`.mwp`) to open, and click **Open**. TopLink Workbench displays the project information.

   If you open a TopLink Workbench version 3.*n* project that contains EJB information, the Potential EJB Descriptors dialog box appears.

Figure 18–3 Potential EJB Descriptors Dialog Box

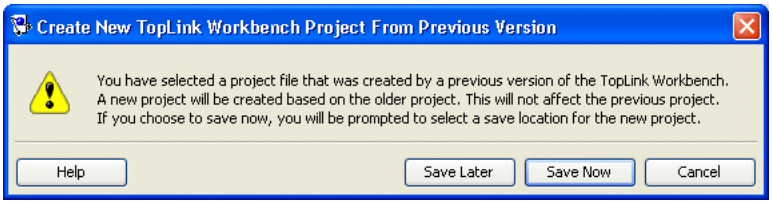


3. Select which of the descriptors should be imported as EJB descriptors, the project persistence type, and click **OK**.

You can also specify whether or not TopLink Workbench generates methods and attributes that comply with the EJB specification, if they are not found within the current class descriptor(s).

If you open a TopLink Workbench version 9.0.3 (or later) project, the Create New TopLink Workbench Project from Previous Version dialog box appears.

Figure 18–4 Create New TopLink Workbench Project From Previous Version Dialog Box



To convert the old project to the current format and view the project immediately, click **Save Later**.

To convert the old project to the current format and save it to a new location and then view the project, click **Save Now**.

Saving Projects

TopLink Workbench does not automatically save your project. Be sure to save your project often to avoid losing data.

To save your project(s), use this procedure:

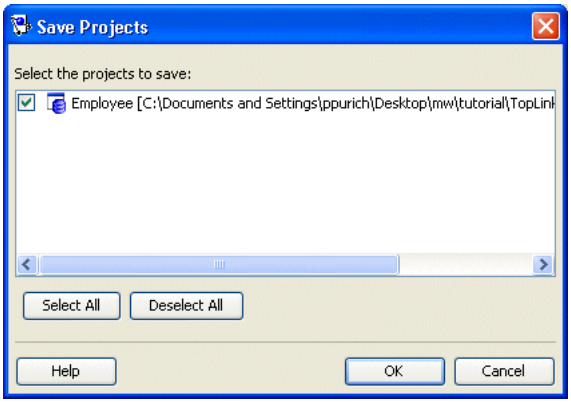


1. Click **Save** or **Save All** to save your project(s).

   You can also save a project by choosing **File** > **Save** or **File** > **Save All** from the menu.

2. If you close TopLink Workbench while there are currently unsaved changes, the Save Project dialog box appears.

Figure 18–5 Save Projects Dialog Box



3. Select the project(s) to save and click **OK**.

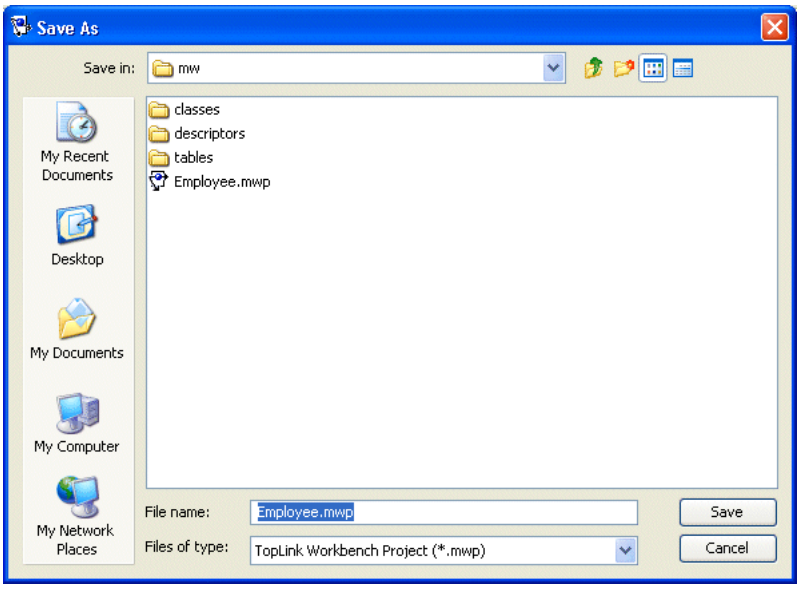
   Click **Select All** to select all the available projects.

Saving Projects With a New Name or Location

To save your project with a different name or location, use this procedure:

1. Choose **File** > **Save As**. The Save As dialog box appears.

Figure 18–6 Save As Dialog Box



2. Select a name and location, then click **Save**.

Caution: Do not rename the `.mwp` file outside of TopLink Workbench. To rename a project, use the **Save As** option.

Generating the Project Status Report

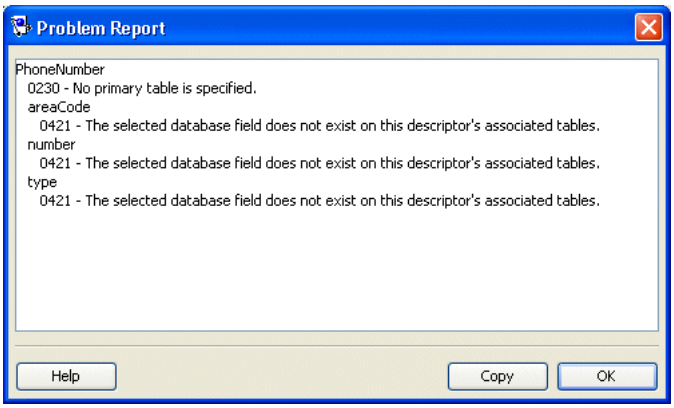
Use the project status report to display a list of all warnings and errors in the TopLink Workbench project. This report is similar to the Problems window (see "Working With

TopLink Workbench" on page 4-3), but lets you easily copy and paste the errors into documents or messages. To generate the project status report, use this procedure:

1. Right-click the **Problems** label above the **Problems** window and select **Problem Report**. The Project Status Report dialog box appears, displaying the status of each TopLink Workbench project.

   You can also generate the project status report by selecting **Tools** > **Problem Report** from the menu.

Figure 18–7 Problem Report Dialog Box



See Chapter 14, "TopLink Workbench Error Reference" for information on each reported error.

To copy the report to another application, click **Copy**.

Exporting Project Information

To use your project with the TopLink Foundation Library at run time, you must either generate deployment XML or export the project to Java source code.

For all project types, TopLink Workbench can generate and export the following project information:

- Exporting Deployment XML Information (`project.xml` file)
- Exporting Model Java Source

> **Note:** When exporting Java source and deployment XML, TopLink Workbench writes the database password (if applicable) using Java Cryptography Extension (JCE) encryption (when using JDK 1.4). For information on how to specify password encryption options, see "Configuring Password Encryption" on page 82-2.
>
> Refer to *Oracle TopLink Getting Started Guide* for information on using password encryption with JDK 1.3 and earlier.

For relational projects only, TopLink Workbench can also generate and export the following project information:

- Exporting Project Java Source
- Exporting Table Creator Files

Exporting Deployment XML Information



To export your deployment XML file (`project.xml`), use this procedure (see Chapter 8, "Creating TopLink Files for Deployment" for detailed information):

1. Select the project and click **Export Deployment XML**.

   You can also right-click the project in the **Navigator** and choose **Export** > **Project Deployment XML** from the context menu or choose **Selected** > **Export** > **Project Deployment XML** from the menu.

   If you have not defined deployment and source code generation defaults (see Chapter 19, "Configuring a Project") TopLink Workbench prompts for a file name and directory.

   Note: If your project contains errors, the `project.xml` may not be valid. See Chapter 14, "TopLink Workbench Error Reference" for information on each reported error.

To generate the deployment XML file that is compatible with projects prior to 10*g* (10.1.3.1.0), see "Configuring Deprecated Direct Mappings" on page 19-12.

Exporting Model Java Source

To generate the project model's Java source code, use this procedure:

1. Right-click the project, package, or specific descriptor in the **Navigator** and choose **Export** > **Export Model Java Source** from the context menu. TopLink Workbench creates a `.java` file for each selected descriptor.

   You can also choose **Workbench** > **Export** > **Export Model Java Source** or **Selected** > **Export** > **Model Java Source** from the menu or click **Generate Source Code** on the **Class** tab. See "Configuring Class Information" on page 4-41 for more information.

If you have not defined deployment and source code generation defaults (see "Configuring a Project" on page 19-1) TopLink Workbench prompts for a root directory.

Note: If your TopLink Workbench project uses UTF-8 character set, you must use a compatible JDK when compiling the exported Java source.

Exporting Project Java Source

For relational projects only, you can convert the project to Java source code. Generally, the generated code executes faster and deploys easier than XML files. See "Generating Java Code for Descriptors" on page 24-6 to export the model source for a *specific descriptor* in a project. To convert your relational project to Java source, use this procedure:

1. Right-click the project in the **Navigator** and choose **Export** > **Project Java Source** from the context menu.



   You can also choose **Workbench** > **Export** > **Export Java Source** or **Selected** > **Export** > **Project Java Source** from the menu.

If you have not defined deployment and source code generation defaults (see "Configuring a Project" on page 19-1) TopLink Workbench prompts for a project class name and directory.

Note: If your TopLink Workbench project uses the UTF-8 character set, you must use a compatible JDK when compiling the exported Java source.

If your project contains errors, the `project.xml` file may not be valid. See Chapter 14, "TopLink Workbench Error Reference" for information on each reported error.

To generate Java source that is compatible with projects prior to 10*g* (10.1.3.1.0), see "Configuring Deprecated Direct Mappings" on page 19-12.

Exporting Table Creator Files

For relational projects only, you can create Java source code to generate database tables defined in the project using this procedure:

1. Right-click the project in the **Navigator** and choose **Export** > **Table Creator Java Source** from the context menu.

   You can also choose **Workbench** > **Export** > **Table Creator Java Source** or **Selected** > **Export** > **Table Creator Java Source** from the menu.

If you have not defined deployment and source code generation defaults (see Chapter 19, "Configuring a Project") TopLink Workbench prompts for a class name and root directory.

Working With the ejb-jar.xml File

For TopLink Workbench relational projects that use EJB 2.0 CMP, use the `ejb-jar.xml` file to store persistence information for the application server. With TopLink Workbench, you can import information from an existing `ejb-jar.xml` file into your project, or you can create and update the `ejb-jar.xml` file from your project.

Each TopLink Workbench project uses a single `ejb-jar.xml` file. For each entity bean in the file, you should have an EJB descriptor in the project. All entity beans must use the same persistence type.

As you make changes in your project, you can update the `ejb-jar.xml` file to reflect your project. Additionally, if you edit the `ejb-jar.xml` file outside TopLink Workbench, you can update your project to reflect the current file.

Table 18–2 describes how fields in the `ejb-jar.xml` file correspond to specific functions in TopLink Workbench.

Table 18–2 ejb-jar.xml Fields and TopLink Workbench

ejb-jar.xml Fields	TopLink Workbench
`primkey`	Bean attribute mapped to the primary key in the database table (see "Configuring Primary Keys" on page 25-3).

Table 18–2 (Cont.) ejb-jar.xml Fields and TopLink Workbench

ejb-jar.xml Fields	TopLink Workbench
`ejb-name, prim-key-class, local, local-home, remote, home, and ejb-class`	EJB descriptor information on the EJB Info tab (see "Configuring a Descriptor With EJB Information" on page 25-45).
`abstract-schema-name`	Descriptor Alias field (see "Configuring Descriptor Alias" on page 25-7).
`cmp-field`	Direct (non-relationship) attributes on the Descriptor Info tab (see "Configuring Common Descriptor Options" on page 25-1).
`cmp-version`	Persistence Type field on the General tab (see "Configuring Persistence Type" on page 19-5). The `persistence-type` is set to `container`.
`query`	Queries listed in Queries tab (see "Configuring Named Queries at the Descriptor Level" on page 25-10). Note: The `findByPrimaryKey` query is not in the `ejb-jar.xml` file as per the EJB 2.0 specification.
`relationships`	One-to-one, one-to-many, and many-to-many mappings (see Part X, "Relational Mappings").

Writing to the ejb-jar.xml File

To update the `ejb-jar.xml` file based on the current TopLink Workbench information, use this procedure:

Note: Use the EJB preferences to specify whether or not TopLink Workbench automatically updates the `ejb-jar.xml` file when you save the project.

Note: You can also write the information to a `.jar` file. TopLink Workbench automatically places the `ejb-jar.xml` file in the proper location (`META-INF/ejb-jar.xml`).

1. Choose **Selected** > **Write Project to ejb-jar.xml** from the menu.

   You can also right-click the project in the **Navigator** and choose **Write Project to ejb-jar.xml** from the context menu.

   - If the project does not currently contain an `ejb-jar.xml` file, the system prompts you to create a new file.
   - If the system detects that changes were made to the `ejb-jar.xml` file but not yet read into TopLink Workbench (for example, you changed the file outside TopLink Workbench), then the system prompts you to read the file before writing the changes.

Reading From the ejb-jar.xml File

To read the `ejb-jar.xml` information and update your TopLink Workbench project, use this procedure.

Tip: To automatically create EJB descriptors in TopLink Workbench for all entities, read the `ejb-jar.xml` file before adding any classes in TopLink Workbench.

1. Choose **Selected** > **Update Project from ejb-jar.xml** from the menu.

   You can also right-click the project in the **Navigator** window and choose **Update Project from ejb-jar.xml** from the context menu.

Note: If you are using TopLink Workbench behind a firewall, before reading from the `ejb-jar.xml` file, you may need to configure TopLink Workbench with a proxy (see "Help Preferences" on page 4-14). If TopLink Workbench fails to read the `ejb-jar.xml` file due to connection timeout or no route to host, proxy configuration is required.

19

Configuring a Project

This chapter describes how to configure TopLink projects.

Table 19–1 lists the types of TopLink projects that you can configure and provides a cross-reference to the type-specific chapter that lists the configurable options supported by that type.

Table 19–1 Configuring TopLink Projects

If you are creating...	See also...
Relational Projects	Chapter 20, "Configuring a Relational Project"
EIS Projects	Chapter 21, "Configuring an EIS Project"
XML Projects	Chapter 22, "Configuring an XML Project"

Table 19–2 lists the configurable options shared by two or more TopLink project types.

For more information, see the following:

- "Project Creation Overview" on page 18-1
- "Understanding Projects" on page 17-1

Configuring Common Project Options

Table 19–2 lists the configurable options shared by two or more TopLink project types. In addition to the configurable options described here, you must also configure the options described for the specific TopLink Project Types, as shown in Table 19–1.

Table 19–2 Common Project Options

Option	Type	TopLink Workbench	Java
"Configuring Project Save Location" on page 19-2	Basic	✔	
"Configuring Project Classpath" on page 19-3	Basic	✔	
"Configuring Mapped Field Access at the Project Level" on page 19-4	Basic	✔	
"Configuring Persistence Type" on page 19-5	Basic	✔	
"Configuring Default Descriptor Advanced Properties" on page 19-7	Advanced	✔	
"Configuring Existence Checking at the Project Level" on page 19-8	Advanced	✔	✔
"Configuring Project Deployment XML Options" on page 19-10	Advanced	✔	
"Configuring Model Java Source Code Options" on page 19-11	Advanced	✔	

Table 19–2 (Cont.) Common Project Options

Option	Type	TopLink Workbench	Java
"Configuring Deprecated Direct Mappings" on page 19-12	Advanced	✔	
"Configuring Cache Type and Size at the Project Level" on page 19-13	Advanced	✔	✔
"Configuring Cache Isolation at the Project Level" on page 19-16	Advanced	✔	✔
"Configuring Cache Coordination Change Propagation at the Project Level" on page 19-17	Advanced	✔	✔
"Configuring Cache Expiration at the Project Level" on page 19-19	Advanced	✔	✔
"Configuring Project Comments" on page 19-21	Advanced	✔	

Configuring Project Save Location

You can configure a project save location only when using TopLink Workbench.

Table 19–3 summarizes which projects support a project save location.

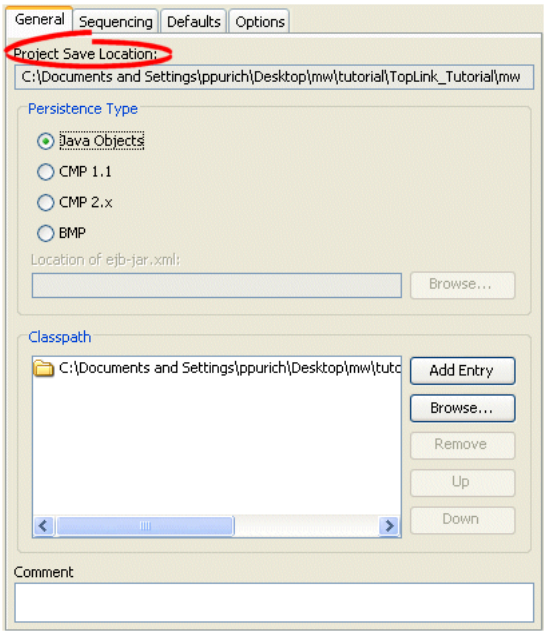
Table 19–3 Project Support for Project Save Location

Descriptor	Using TopLink Workbench	Using Java
Relational Projects	✔	
EIS Projects	✔	
XML Projects	✔	

Using TopLink Workbench

The Project Save Location field on the project's General tab is for display only. This field shows the full directory path for the project. All relative locations used in the project are based on this location.

Figure 19–1 General Tab, Project Save Location



To select a new location, right-click on the project in the **Navigator** and select **Save As** from the context menu. See "Saving Projects" on page 18-11 for more information.

Configuring Project Classpath

The TopLink project uses a classpath–a set of directories, JAR files, and ZIP files–when importing Java classes and defining object types.

Table 19–4 summarizes which projects support project classpath configuration.

Table 19–4 Project Support for Project Classpath

Descriptor	Using TopLink Workbench	Using Java
Relational Projects	✔	
EIS Projects	✔	
XML Projects	✔	

Do not include JDBC drivers or other elements required to access the data source in the project classpath. Use the `setenv` file to specify these application-level settings (see "Configuring the TopLink Workbench Environment" on page 4-2).

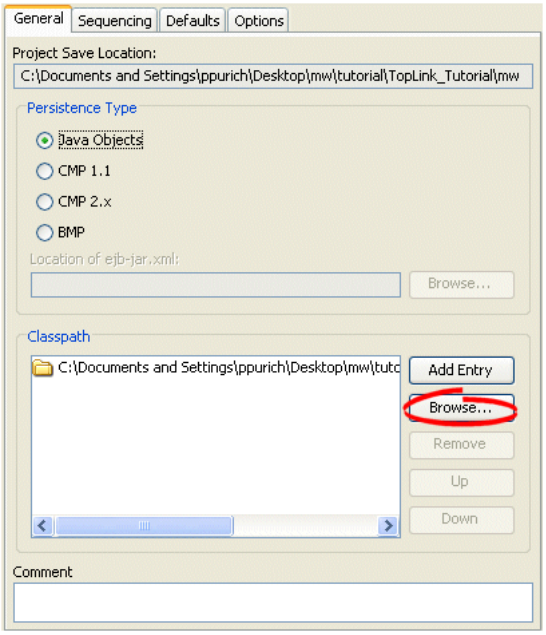
After you configure the project classpath, you can use TopLink Workbench to import classes into your project (see "Importing and Updating Classes" on page 4-50).

Using TopLink Workbench

To specify the project classpath information, use this procedure:

1. Select the project object in the **Navigator**.
2. Click the **General** tab in the **Editor**. The General tab appears.

Figure 19–2 General Tab, Classpath Options



To add a new classpath entry, click **Add Entry** or **Browse** and select the directory, `.jar` file, or `.zip` file for this project. To create a relative classpath, select an entry and edit the path, as necessary. The path will be relative to the **Project Save Location**.

To remove a classpath entry, select the entry and click **Remove**.

To change the order of the entries, select the entry and click **Up** or **Down**.

Configuring Mapped Field Access at the Project Level

By default, when TopLink performs a persistence operation, it accesses the persistent attributes of an object directly: this is known as direct field access. Alternatively, you can configure TopLink to access persistent attributes using accessor methods of the object: this is known as method access.

Oracle recommends using field access for mappings. Not only is it more efficient, but using method access may cause issues if the method produces unexpected side-effects.

Table 19–5 summarizes which projects support mapped field access configuration.

Table 19–5 Project Support for Mapped Field Access

Descriptor	Using TopLink Workbench	Using Java
Relational Projects	✔	
EIS Projects	✔	
XML Projects	✔	

This section describes configuring mapped field access at the project level: by default, this configuration applies to all descriptors and their mappings.

> **Note:** If you change the access default, existing mappings retain their current access settings, but new mappings will be created with the new default.

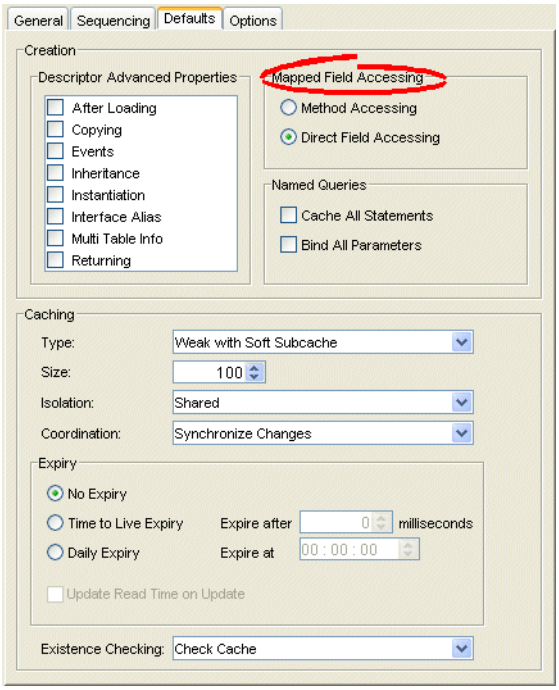
You can also configure mapped field access at the mapping level to override this project-level configuration on a mapping-by-mapping basis. For more information, see "Configuring Method Accessing" on page 32-14.

Using TopLink Workbench

To specify the field access method information, use this procedure:

1. Select the project object in the **Navigator**.
2. Select the **Defaults** tab in the **Editor**. The Defaults tab appears.

Figure 19–3 Defaults Tab, Field Accessing Options



Use this table to enter data in the following fields of the tab to specify the default field access method for newly created descriptors:

Field	Description
Method Accessing	Use Method Accessing as the default field access method.
Direct Field Accessing	Use Direct Field Accessing as the default field access method.

Configuring Persistence Type

You can configure your project persistence type only when using TopLink Workbench.

Using TopLink Workbench, you can specify the persistence type to use with the project. For example, your TopLink project may use plain Java objects, entity beans with container-managed persistence, or entity beans with bean-managed persistence (BMP).

Table 19–6 summarizes which projects support a persistence type configuration.

Table 19–6 Project Support for Persistence Type

Descriptor	Using TopLink Workbench	Using Java
Relational Projects	✔	
EIS Projects	✔	
XML Projects		

To create a TopLink descriptor for a persistent class, TopLink Workbench reads a compiled Java `.class` file to read its attributes and relationships. See "Descriptors" on page 16-2 for more information on TopLink descriptors.

For EJB projects, you can specify an `ejb-jar.xml` file from which TopLink will read and to which it will write the necessary persistence information. You use the `ejb-jar.xml` file to map the virtual fields of the entity beans with container-managed persistence (called container-managed fields, defined by `<cmp-field>` tag) or relationships (called container-managed relationship, defined by `<cmr-field>` tag) to a data source.



TopLink Workbench defines all descriptors for entity classes (as defined in the `ejb-jar.xml` file) as EJB descriptors. TopLink Workbench does not create (or remove) descriptors for the interfaces and primary key class for the entity when refreshing from the `ejb-jar.xml` file.

> **Note:** TopLink Workbench creates class descriptors for entity classes not defined in the `ejb-jar.xml` file. You must manually change the descriptor type (see "Configuring a Descriptor With EJB Information" on page 25-45).

To update your project from the XML file, right-click an EJB descriptor and select **Update Descriptors from ejb-jar.xml**. You can also update the project by choosing **Selected > Update Descriptors from ebj-jar.xml** from the menu.

For more information on creating and using deployment files such as the `ejb-jar.xml` file, see the following:

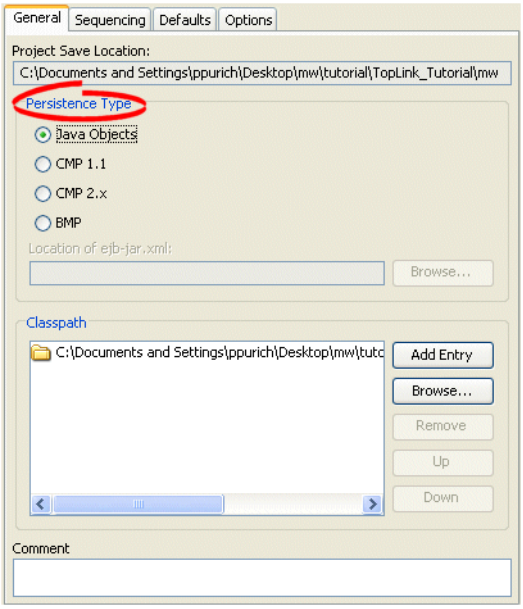
- "Understanding TopLink Deployment File Creation" on page 8-1
- Chapter 7, "Integrating TopLink With an Application Server"
- Chapter 9, "Packaging a TopLink Application"
- Chapter 10, "Deploying a TopLink Application"

Using TopLink Workbench

To specify the persistence information, use this procedure:

1. Select the project object in the **Navigator**.
2. Select the **General** tab in the **Editor**. The General tab appears.

Figure 19–4 General Tab, Persistence Type Options



Use this table to enter data in the following fields on the project's **General** tab to configure the persistence options:

Field	Description
Persistence Type	Specify the persistence type of the project: **Java Objects**, **CMP 1.1**, **CMP 2.x**, or **BMP**. For EJB projects, specify the location of the `ejb-jar.xml` file. Note: This field does not apply to XML projects.
Location of ejb-jar.xml	Specify the location of the `ejb-jar.xml` file for this project. "Working With the ejb-jar.xml File" on page 18-15 for more information. Note: This field applies to EJB projects only.

Configuring Default Descriptor Advanced Properties

You can configure default descriptor advanced properties only when using TopLink Workbench.

By default, TopLink Workbench displays a subset of features for each descriptor type. You can modify this subset so that descriptors include additional advanced properties by default.

You can also select specific advanced properties for individual descriptors (see Chapter 25, "Configuring a Descriptor").

Table 19–7 summarizes which projects support default descriptor advanced property configuration.

Table 19–7 Project Support for Default Descriptor Advanced Properties

Descriptor	Using TopLink Workbench	Using Java
Relational Projects	✔	
EIS Projects	✔	

Table 19–7 (Cont.) Project Support for Default Descriptor Advanced Properties

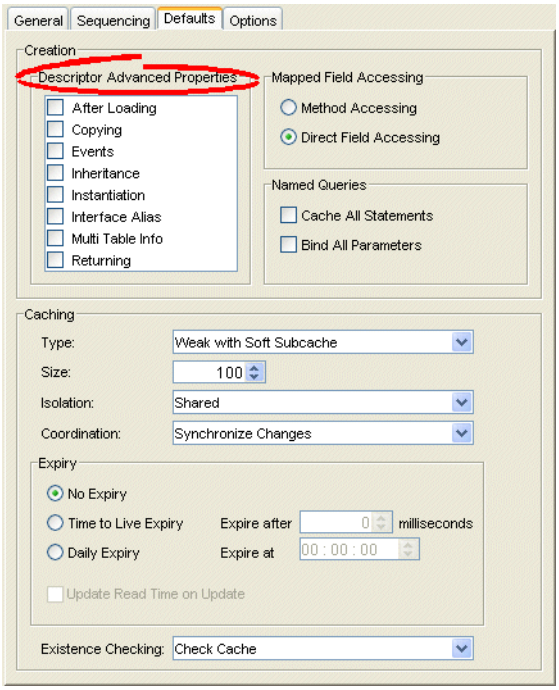
Descriptor	Using TopLink Workbench	Using Java
XML Projects	✔	

Using TopLink Workbench

To specify the default advanced properties for newly created descriptors in your project, use this procedure:

1. Select the project object in the **Navigator**.
2. Select the **Defaults** tab in the **Editor**. The Defaults tab appears.

Figure 19–5 Defaults Tab, Descriptor Advanced Properties



Select which **Descriptor Advanced Properties** to add to newly created descriptors. The list of advanced properties will vary, depending on the project type.

See Chapter 25, "Configuring a Descriptor" for detailed information on each advanced property.

Configuring Existence Checking at the Project Level

When TopLink writes an object to the database, it runs an existence check to determine whether to perform an insert or an update operation.

By default, TopLink checks against the cache. Oracle recommends that you use this default existence check option for most applications. Checking the database for existence can cause a performance bottleneck in your application.

Table 19–8 summarizes which projects support existence checking configuration.

Table 19–8 Project Support for Existence Checking

Descriptor	Using TopLink Workbench	Using Java
Relational Projects	✔	✔
EIS Projects	✔	✔
XML Projects		

By default, this configuration applies to all descriptors in a project. You can also configure existence checking at the descriptor level to override this project-level configuration on a descriptor-by-descriptor basis. For more information, see "Configuring Cache Existence Checking at the Descriptor Level" on page 25-43.

For more information see the following:

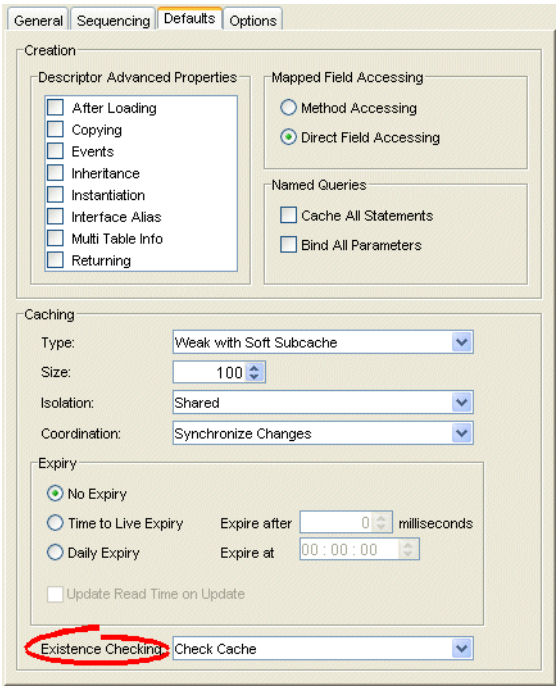
- "Cache Type and Object Identity" on page 87-3
- "Queries and the Cache" on page 93-29
- "Using Registration and Existence Checking" on page 99-5

Using TopLink Workbench

To specify the existence checking information, use this procedure:

1. Select the project object in the **Navigator**.
2. Select the **Defaults** tab in the **Editor**. The Defaults tab appears.

Figure 19–6 Defaults Tab, Existence Checking Options



Use this table to enter data in following fields to specify the existence checking options for newly created descriptors:

Field	Description
Check Cache	Check the session cache. If the object is not in the cache, assume that the object does not exist (do an insert). If the object is in the cache, assume that the object exists (do an update). Oracle recommends using this option for most applications.
Check Database	If an object is not in the cache, query the database to determine if the object exists. If the object exists, do an update. Otherwise, do an insert. Selecting this option may negatively impact performance. For more information, see "Check Database" on page 99-5.
Assume Existence	Always assume objects exist: always do an update (never do an insert). For more information, see "Assume Existence" on page 99-5.
Assume Nonexistence	Always assume objects do not exist: always do an insert (never do an update). For more information, see "Assume Nonexistence" on page 99-5.

Configuring Project Deployment XML Options

You can configure project deployment XML options only when using TopLink Workbench.

Using TopLink Workbench, you can specify the default file names, class names, and directories, when exporting or generating deployment XML. Directories are relative to the project save location (see "Configuring Project Save Location" on page 19-2), and will contain folders for each generated package.

Table 19–9 summarizes which projects support deployment XML options.

Table 19–9 Project Support for Project Deployment XML Options

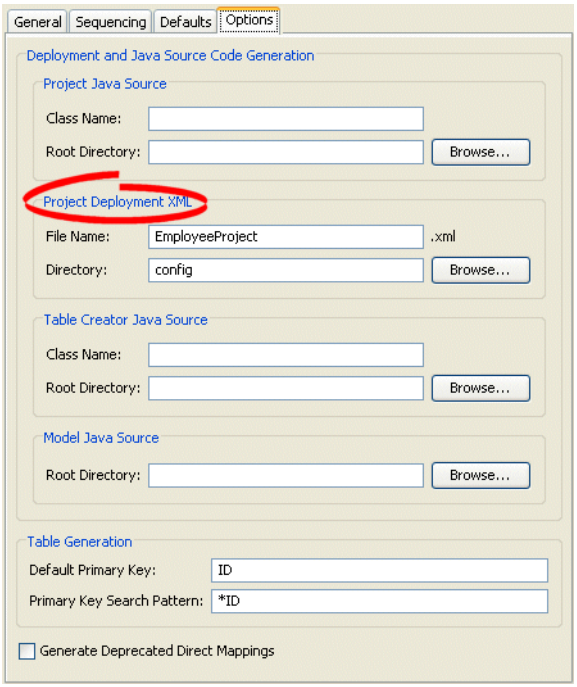
Descriptor	Using TopLink Workbench	Using Java
Relational Projects	✔	
EIS Projects	✔	
XML Projects	✔	

Using TopLink Workbench

To specify the default export options, use this procedure:

1. Select the project object in the **Navigator**.
2. Select the **Options** tab in the **Editor**. The Options tab appears.

Figure 19–7 Options Tab, Project Deployment XML Options



Use this table to enter data in following fields to specify the default Project Deployment XML options:

Field	Description
File Name	File name (such as `project.xml`) to use when generating project deployment XML.
Directory	Directory in which to save the generated deployment XML file.

Configuring Model Java Source Code Options

You can configure model java source code options only when using TopLink Workbench.

Using TopLink Workbench, you can specify the default file names, class names, and directories, when exporting or generating Java source code for your domain objects. Directories are relative to the project save location (see "Configuring Project Save Location" on page 19-2), and will contain folders for each generated package.

Table 19–10 summarizes which projects support model Java source code options.

Table 19–10 Project Support for Model Java Source Options

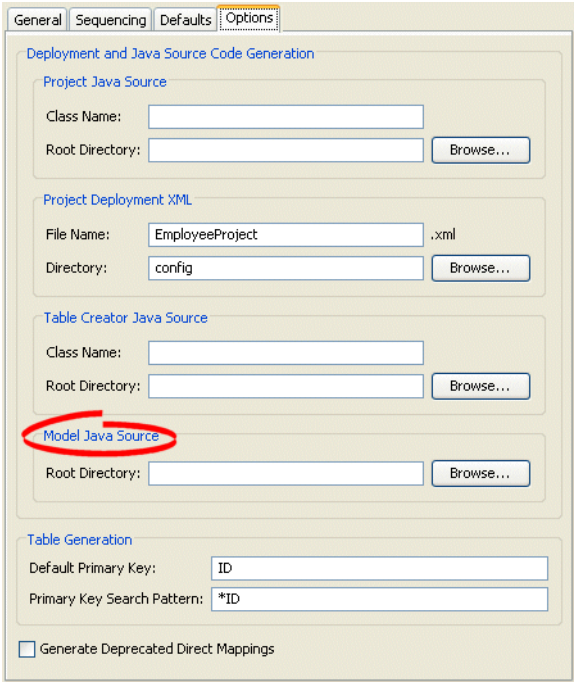
Descriptor	Using TopLink Workbench	Using Java
Relational Projects	✔	
EIS Projects	✔	
XML Projects	✔	

Using TopLink Workbench

To specify the default export options, use this procedure:

1. Select the project object in the **Navigator**.
2. Select the **Options** tab in the **Editor**. The Options tab appears.

Figure 19–8 Options Tab, Model Java Source options



Specify the project root directory to which TopLink Workbench generates model Java source files. For more information, see "Generating Java Code for Descriptors" on page 24-6.

Configuring Deprecated Direct Mappings

You can configure deprecated direct mapping options only when using TopLink Workbench.

Starting with this release, TopLink no longer uses the following direct mapping types:

- Type conversion
- Object type
- Serialized object

Instead, TopLink uses a direct-to-field mapping with a specialized converter.

To generate backward-compatible deployment XML and Java source code files, use the **Generate Deprecated Direct Mappings** option.

Table 19–11 summarizes which projects support deprecated direct mapping options.

Table 19–11 Project Support for Deprecated Direct Mapping Options

Descriptor	Using TopLink Workbench	Using Java
Relational Projects	✔	
EIS Projects		

Table 19–11 (Cont.) Project Support for Deprecated Direct Mapping Options

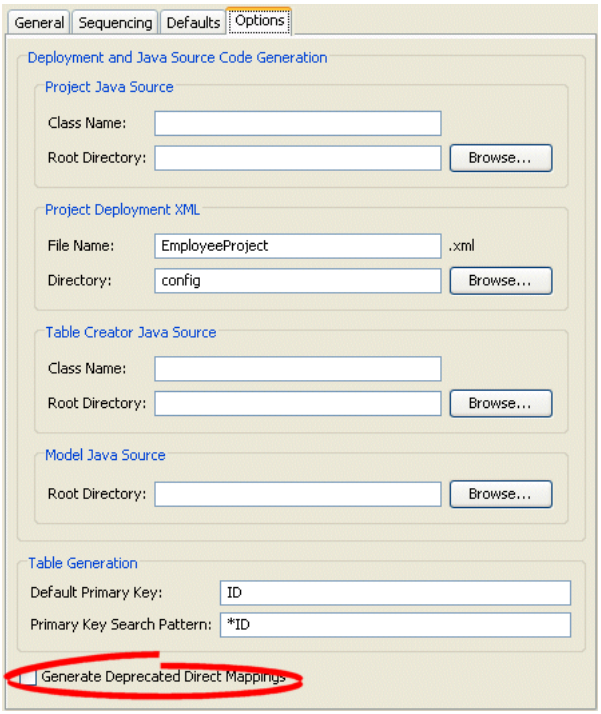
Descriptor	Using TopLink Workbench	Using Java
XML Projects		

Using TopLink Workbench

To specify if TopLink Workbench should generate the deprecated direct mappings (instead of using the converter) when exporting projects, use this procedure:

1. Select the project object in the **Navigator**.
2. Select the **Options** tab in the **Editor**. The Options tab appears.

Figure 19–9 Options Tab, Generate Deprecated Direct Mappings Option



Select the **Generate Deprecated Direct Mappings** option on the tab to specify that TopLink Workbench should generate backward-compatible code (using the deprecated direct mappings, instead of the converter).

Configuring Cache Type and Size at the Project Level

The TopLink cache is an in-memory repository that stores recently read or written objects based on class and primary key values. TopLink uses the cache to:

- improve performance by holding recently read or written objects and accessing them in-memory to minimize database access
- manage locking and isolation level
- manage object identity

Table 19–12 summarizes which projects support identity map configuration.

Table 19–12 Project Support for Identity Map Configuration

Descriptor	Using TopLink Workbench	Using Java
Relational Projects	✔	✔
EIS Projects	✔	✔
XML Projects		

The cache options you configure at the project level apply globally to all descriptors. Use this section to define global cache options for a TopLink project.

You can override the project-level identity map configuration by defining identity map configuration at the descriptor level. For information on caching and defining identity map configuration for a specific descriptor, see "Configuring Cache Type and Size at the Descriptor Level" on page 25-35.

> **Note:** When using TopLink Workbench, changing the project's default identity map does not affect descriptors that already exist in the project; only newly added descriptors ar affected.

For detailed information on caching and object identity, and the recommended settings to maximize TopLink performance, see to "Cache Type and Object Identity" on page 87-3.

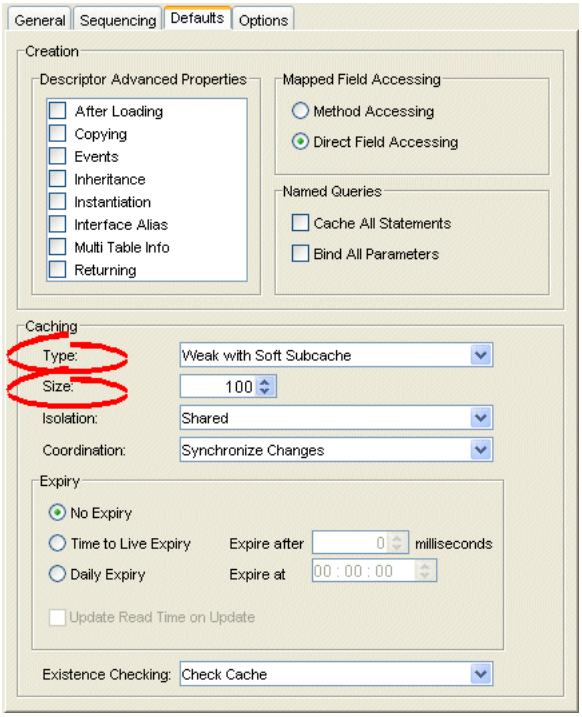
For more information about the cache, see Chapter 87, "Understanding the Cache".

Using TopLink Workbench

To specify the cache identity map, use this procedure:

1. Select the project object in the **Navigator**.
2. Select the **Defaults** tab in the **Editor**. The Defaults tab appears.

Figure 19–10 Defaults Tab, Cache Identity Map Options



Use this table to enter data in each of the following fields to specify the caching options:

Field	Description
Type	Use the **Type** list to choose the identity map: ▪ **Weak with Soft Subcache** (`SoftCacheWeakIdentityMap`)–cache first *n* elements in soft space, anything after that in weak space (see "Soft and Hard Cache Weak Identity Maps" on page 87-4). ▪ **Weak with Hard Subcache** (`HardCacheWeakIdentityMap`)–cache first *n* elements in soft space, anything after that in hard space (see "Soft and Hard Cache Weak Identity Maps" on page 87-4). ▪ **Weak** (`WeakIdentityMap`)–cache everything in weak space (see "Weak Identity Map" on page 87-3). ▪ **Full** (`FullIdentityMap`)–cache everything permanently (see "Full Identity Map" on page 87-3). ▪ **None** (`NoIdentityMap`)–cache nothing (see "No Identity Map" on page 87-4). For more information, see "Cache Type and Object Identity" on page 87-3. Changing the project's default identity map does not affect descriptors that already exist in the project.
Size	Specify the size of the cache: ▪ When using **Weak with Soft Subcache** or **Weak with Hard Subcache**, the size is the *maximum* number of objects stored in the identity map. ▪ When using **Full** or **Weak**, the size indicates the *starting size* of the identity map.

Configuring Cache Isolation at the Project Level

If you plan to use isolated sessions (see "Cache Isolation" on page 87-9), you must configure descriptors as isolated for any object that you want confined to an isolated session cache.

Configuring a descriptor to be isolated means that TopLink will not store the object in the shared session cache and the object will not be shared across client sessions. This means that each client will have their own object read directly from the database. Objects in an isolated client session cache can reference objects in their parent server session's shared session cache, but no objects in the shared session cache can reference objects in an isolated client session cache. Isolation is required when using Oracle Database Virtual Private Database (VPD) support or database user-based read security. Isolation can also be used if caching is not desired across client sessions.

Table 19–12 summarizes which projects support cache isolation configuration.

Table 19–13 Project Support for Cache Isolation Configuration

Descriptor	Using TopLink Workbench	Using Java
Relational Projects	✔	✔
EIS Projects	✔	✔
XML Projects		

The cache isolation options you configure at the project level apply globally to all descriptors. Use this section to define global options for a TopLink project.

You can override the project-level configuration by defining cache isolation options at the descriptor level. For information, see "Configuring Cache Isolation at the Descriptor Level" on page 25-37.

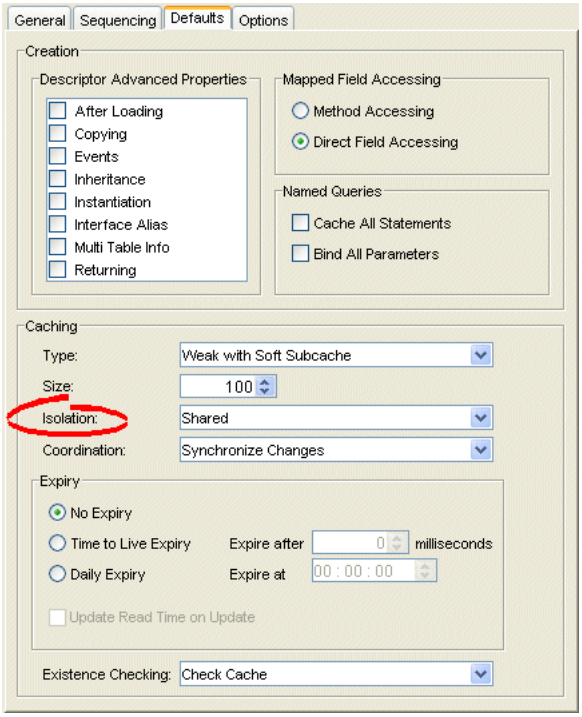
Note: When using TopLink Workbench, changing the project's default cache isolation option does not affect descriptors that already exist in the project; only newly added descriptors ar affected.

Using TopLink Workbench

To specify the cache isolation options, use this procedure:

1. Select the project object in the **Navigator**.
2. Select the **Defaults** tab in the **Editor**. The Defaults tab appears.

Figure 19–11 Defaults Tab, Cache Isolation Options



Use the **Isolation** list to choose one of the following:

- **Isolated**–if you want all objects confined to an isolated client session cache. For more information, see "Cache Isolation" on page 87-9.
- **Shared**–if you want all objects visible in the shared session cache (default).

Configuring Cache Coordination Change Propagation at the Project Level

If you plan to use a coordinated cache (see "Understanding Cache Coordination" on page 87-10), you can configure how and under what conditions a coordinated cache propagates changes.

Table 19–12 summarizes which projects support cache coordination change propagation configuration.

Table 19–14 Project Support for Cache Coordination Change Propagation Configuration

Descriptor	Using TopLink Workbench	Using Java
Relational Projects	✔	✔
EIS Projects	✔	✔
XML Projects		

The cache coordination change propagation options you configure at the project level apply globally to all descriptors. Use this section to define global options for a TopLink project.

You can override the project-level configuration by defining cache coordination change propagation options at the descriptor level. For information, see "Configuring Cache Coordination Change Propagation at the Descriptor Level" on page 25-40.

To complete your coordinated cache configuration, see Chapter 88, "Configuring a Coordinated Cache".

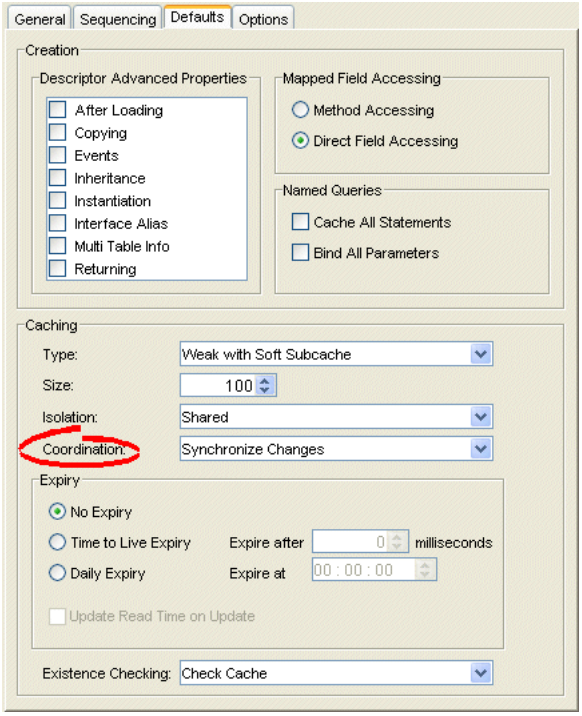
Note: When using TopLink Workbench, changing the project's default cache coordination change propagation option does not affect descriptors that already exist in the project; only newly added descriptors ar affected.

Using TopLink Workbench

To specify the coordinated cache change propagation options, use this procedure:

1. Select the project object in the **Navigator**.
2. Select the **Defaults** tab in the **Editor**. The Defaults tab appears.

Figure 19–12 Defaults Tab, Coordination Options



Use the following information to enter data in the Coordination field:

Coordination Option	Description	When to Use
None	For both existing and new instances, do not propagate a change notification.	Infrequently read or changed objects.
Synchronize Changes	For an existing instance, propagate a change notification that contains each changed attribute. For a new instance, propagate an object creation (along with all the new instance's attributes) only if the new instance is related to other existing objects that are also configured with this change propagation option.	Frequently read or changed objects that contain few attributes or in cases where only a few attributes are frequently changed. Objects that have many or complex relationships.

Coordination Option	Description	When to Use
Synchronize Changes and New Objects	For an existing instance, propagate a change notification that contains each changed attribute. For a new instance, propagate an object creation (along with all the new instance's attributes).	Frequently read or changed objects that contain few attributes or in cases where only a few attributes are frequently changed. Objects that have few or simple relationships.
Invalidate Changed Objects	For an existing instance, propagate an object invalidation that marks the object as invalid in all other sessions. This tells other sessions that they must update their cache from the data source the next time this object is read. For a new instance, no change notification is propagated.	Frequently read or changed objects that contain many attributes in cases where many of the attributes are frequently changed.

Configuring Cache Expiration at the Project Level

By default, objects remain in the cache until they are explicitly deleted (see "Deleting Objects" on page 98-7) or garbage-collected when using a weak identity map (see "Configuring Cache Type and Size at the Project Level" on page 19-13). Alternatively, you can configure an object with a `CacheInvalidationPolicy` that lets you specify, either automatically or manually, that an object is invalid: when any query attempts to read an invalid object, TopLink will go to the data source for the most up-to-date version of that object and update the cache with this information.

Using cache invalidation ensures that your application does not use stale data. It provides a better performing alternative to refreshing (see "Configuring Cache Refreshing" on page 25-27).

Table 19–15 summarizes which projects support cache invalidation configuration.

Table 19–15 Project Support for Cache Invalidation Configuration

Descriptor	Using TopLink Workbench	Using Java
Relational Projects	✔	✔
EIS Projects	✔	✔
XML Projects		

The cache invalidation options you configure at the project level apply globally to all descriptors. Use this section to define global cache invalidation options for a TopLink project.

You can override the project-level cache invalidation configuration by defining cache invalidation at the descriptor (see "Configuring Cache Expiration at the Descriptor Level" on page 25-42) or query level (see "Configuring Cache Expiration at the Query Level" on page 96-21).

You can customize how TopLink communicates the fact that an object has been declared invalid to improve efficiency if you are using a coordinated cache. For more information, see "Configuring Cache Coordination Change Propagation at the Descriptor Level" on page 25-40.

Note: When using TopLink Workbench, changing the project's default cache invalidation does not affect descriptors that already exist in the project; only newly added descriptors are affected.

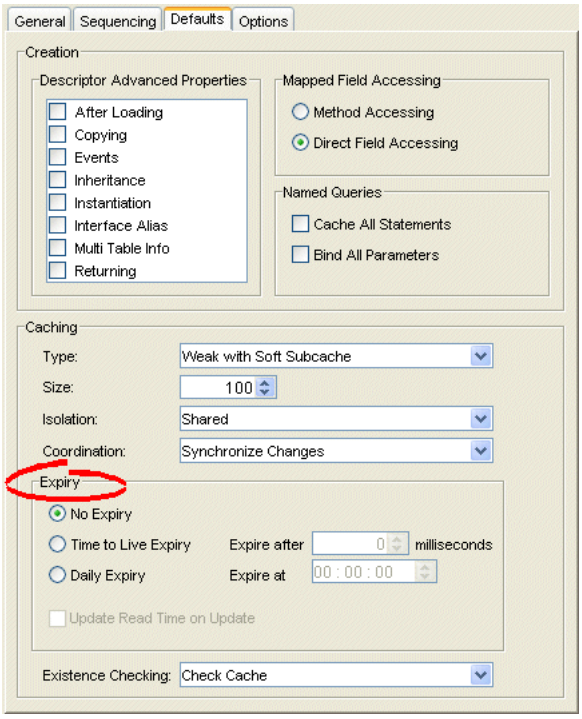
For more information, see "Cache Invalidation" on page 87-8.

Using TopLink Workbench

To specify the cache invalidation options for the project, use this procedure:

1. Select the project object in the **Navigator**.
2. Select the **Defaults** tab in the **Editor**. The Defaults tab appears.

Figure 19–13 Defaults Tab, Cache Expiry Options



Use this table to enter data in the following fields on this tab:

Field	Description
No Expiry	Specify that objects in the cache do not expire.
Time to Live Expiry	Specify that objects in the cache will expire after a specified amount of time. Use the **Expire After** field to indicate the time (in milliseconds) after which the objects will expire.
Daily Expiry	Specify that objects in the cache will expire at a specific time each day. Use the **Expire At** field to indicate the exact time to the second (using a 24-hour clock) at which the objects will expire.
Update Read Time on Update	Specify if the expiry time should be reset after updating an object.

Configuring Project Comments

You can define a free-form textual comment for each project. You can use these comments however you whish: for example, to record important project implementation details such as the purpose or importance of a project.

Comments are stored in the TopLink Workbench project, in the TopLink deployment XML file. There is no Java API for this feature.

Table 19–16 summarizes which projects support this option.

Table 19–16 Project Support for Project Comments

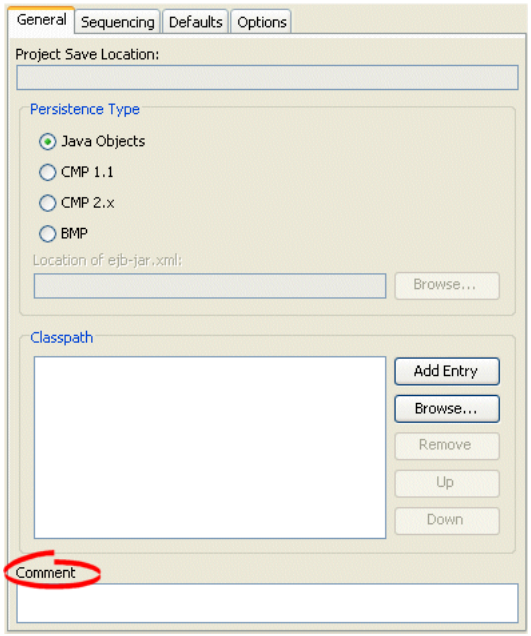
Project Type	Using TopLink Workbench	Using Java
Relational Projects	✔	
EIS Projects	✔	
XML Projects	✔	

Using TopLink Workbench

To specify a comment for the project, use this procedure:

1. Select the project object in the **Navigator**.
2. Select the **General** tab in the **Editor**. The General tab appears.

Figure 19–14 General Tab, Comments Options



3. Enter descriptive text information in the **Comment** field.

20

Configuring a Relational Project

This chapter describes the various components that you must configure in order to use a relational project.

For more information, see the following:

- "Project Creation Overview" on page 18-1
- "Relational Projects" on page 17-6

Relational Project Configuration Overview

In addition to the configurable options described here, you must also configure the base class options described in Table 19–2 on page 19-1.

Table 20–1 lists the configurable options for relational projects.

Table 20–1 Configurable Options for Relational Projects

Option	Type	TopLink Workbench	Java
"Configuring Project Save Location" on page 19-2	Basic	✔	
"Configuring Persistence Type" on page 19-5	Basic	✔	
"Configuring Project Classpath" on page 19-3	Basic	✔	
"Configuring Project Comments" on page 19-21	Basic	✔	
"Configuring Mapped Field Access at the Project Level" on page 19-4	Basic	✔	✔
"Configuring Default Descriptor Advanced Properties" on page 19-7	Basic	✔	✔
"Configuring Existence Checking at the Project Level" on page 19-8	Basic	✔	✔
"Configuring Project Deployment XML Options" on page 19-10	Basic	✔	✔
"Configuring Model Java Source Code Options" on page 19-11	Basic	✔	✔
"Configuring Deprecated Direct Mappings" on page 19-12	Basic	✔	✔
"Configuring Relational Database Platform at the Project Level" on page 20-2	Basic	✔	✔
"Configuring Sequencing at the Project Level" on page 20-3	Basic	✔	✔
"Configuring Login Information" on page 20-5	Basic	✔	✔
"Configuring Development and Deployment Logins" on page 20-6	Basic	✔	
"Configuring Cache Type and Size at the Project Level" on page 19-13	Advanced	✔	✔
"Configuring Cache Isolation at the Project Level" on page 19-16	Advanced	✔	✔

Table 20–1 (Cont.) Configurable Options for Relational Projects

Option	Type	TopLink Workbench	Java
"Configuring Cache Coordination Change Propagation at the Project Level" on page 19-17	Advanced	✔	✔
"Configuring Cache Expiration at the Project Level" on page 19-19	Advanced	✔	
"Configuring Named Query Parameterized SQL and Statement Caching at the Project Level" on page 20-7	Advanced	✔	
"Configuring Table Generation Options" on page 20-9	Advanced	✔	
"Configuring Table Creator Java Source Options" on page 20-10	Advanced	✔	
"Configuring Project Java Source Code Options" on page 20-11	Advanced	✔	

This chapter also describes logging into a database during development when using TopLink Workbench. For more information, see "Logging in to the Database" on page 20-7.

Configuring Relational Database Platform at the Project Level

For each relational project, you must specify the database platform (such as Oracle Database 10*g*). This platform configuration is overridden by the session login, if configured.

For more information, see the following:

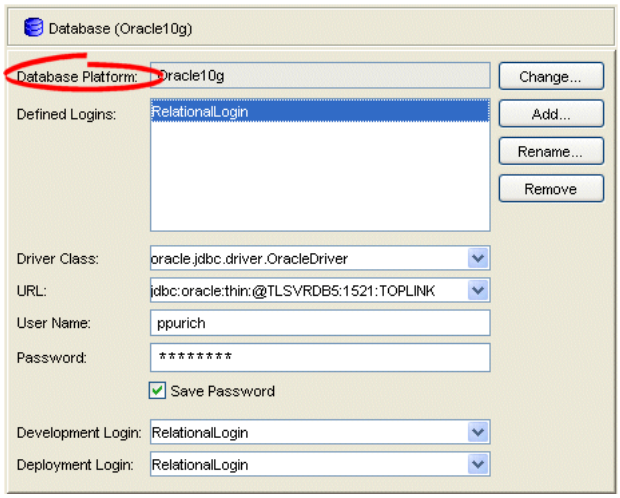
- "Configuring a Relational Database Platform at the Session Level" on page 83-1
- "Data Source Platform Types" on page 81-3

Using TopLink Workbench

To specify the database platform of a relational project, use this procedure:

1. Select the database object in the **Navigator**. The Database property sheet appears.

Figure 20–1 Database Property Sheet, Database Platform Options



Click **Change** to select a new database platform for the project. For more information, see "Data Source Platform Types" on page 81-3.

Configuring Sequencing at the Project Level

Sequencing allows TopLink to automatically assign the primary key or ID of an object when the object is inserted.

You configure TopLink sequencing at the project or session level to tell TopLink how to obtain sequence values: that is, what type of sequences to use.

In a CMP project, you do not configure a session directly: in this case, you must configure sequences at the project level. In a non-CMP project, you can configure a session directly: in this case, you can use a session-level sequence configuration to override project-level sequence configuration, on a session-by-session basis, if required (see "Configuring Sequencing at the Session Level" on page 83-4).

Using TopLink Workbench (see "Using TopLink Workbench" on page 20-3), you can configure table sequencing (see "Table Sequencing" on page 17-16) and native sequencing ("Native Sequencing With an Oracle Database Platform" on page 17-18 and "Native Sequencing With a Non-Oracle Database Platform" on page 17-19) and you can configure a preallocation size that applies to all sequences (see "Sequencing and Preallocation Size" on page 17-20).

Using Java (see "Using Java" on page 20-4), you can configure any sequence type that TopLink supports ("Sequencing Types" on page 17-16). You can create any number and combination of sequences. You can create a sequence object explicitly or use the default sequence that the platform creates. You can associate the same sequence with more than one descriptor and you can configure a separate preallocation size for each descriptor's sequence.

If you are migrating a BEA WebLogic CMP application to OC4J and TopLink persistence (see "Migrating BEA WebLogic Persistence to OC4J TopLink Persistence" on page 7-16), the TopLink migration tool does not migrate BEA WebLogic single column sequence tables to TopLink unary sequence tables (see "Unary Table Sequencing" on page 17-17). After migration, you must manually configure your project to use TopLink unary sequence tables if your application originally used single column sequence tables in BEA WebLogic.

After configuring the sequence type at the project (or session) level, to enable sequencing, you must configure a descriptor with a sequence field and a sequence name (see "Configuring Sequencing at the Descriptor Level" on page 26-3).

For more information about sequencing, see "Understanding Sequencing in Relational Projects" on page 17-14.

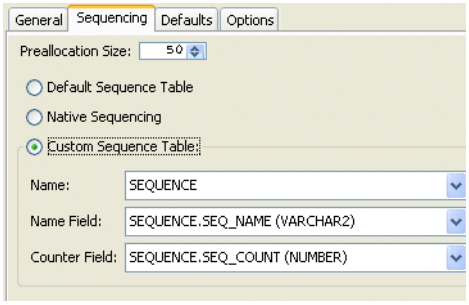
Note: Changing the project's default sequencing options does not affect descriptors that already exist in the project; only newly added descriptors are affected.

Using TopLink Workbench

To specify the sequencing information for the project, use this procedure:

1. Select the project object in the **Navigator**.
2. Select the **Sequencing** tab in the **Editor**. The Sequencing tab appears.

Figure 20–2 Sequencing Tab



Use this table to enter data in the following fields to configure the sequencing information:

Field	Description
Preallocation Size	Specify the default preallocation size (see "Sequencing and Preallocation Size" on page 17-20). Default is **50**. The preallocation size you configure applies to all sequences.
Default Sequence Table	Select this option to use table sequencing (see "Table Sequencing" on page 17-16) with default sequence table name `SEQUENCE`, default sequence name field `SEQ_NAME`, and default sequence counter field `SEQ_COUNT`.
Native Sequencing	Select this option to use a sequencing object (see "Native Sequencing With an Oracle Database Platform" on page 17-18 or "Native Sequencing With a Non-Oracle Database Platform" on page 17-19) created by the database platform. This option applies only to Oracle, Sybase, Microsoft SQL, and IBM Informix database platforms.
Custom Sequence Table	Select this option to use table sequencing (see "Table Sequencing" on page 17-16) with a sequence table name, sequence name field, and sequence counter field name that you specify.
Name	Specify the name of the sequence table.
Name Field	Specify the name of the column used to store the sequence name.
Counter Field	Specify the name of the column used to store the sequence count.

Using Java

Using Java, you can configure a project to use multiple, different sequences as Example 20–1 shows.

Example 20–1 Configuring Sequencing at the Project Level in Java

```
// Enable native sequencing for the project as the default. Configured the default
preallocation size
project.getLogin().useNativeSequencing();
project.getLogin().setSequencePreallocationSize(50);

// Configure the EMP_SEQ to not use preallocation
DefaultSequence empSequence = new DefaultSequence("EMP_SEQ", 1);
project.getLogin().addSequence(empSequence);

// Configure the PROJ_SEQ to use a seperate sequence table
UnarySequence projSequence = new UnarySequence("PROJ_SEQ_TAB", "COUNTER");
project.getLogin().addSequence(projSequence);
```

Configuring Login Information

This section describes how to define a login to a relational database. After you define a login, you must designate its role (see "Configuring Development and Deployment Logins" on page 20-2).

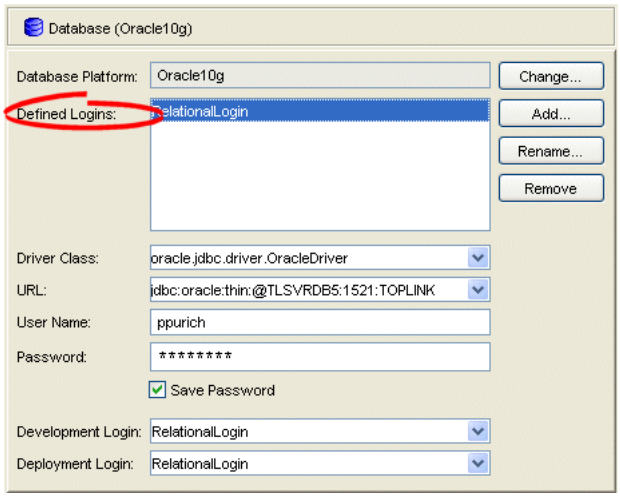
After you create a login (see "Configuring Login Information" on page 20-5) and specify it as a development login (see "Configuring Development and Deployment Logins" on page 20-6), you can log in to a database instance (see "Logging in to the Database" on page 20-7).

Using TopLink Workbench

To create or edit a database login, use this procedure:

1. Select the database object in the **Navigator**. The Database property sheet appears.

Figure 20–3 Database Property Sheet, Database Login Fields



2. Click **Add** to create a new Defined Login.

Use this table to enter data in the following fields on the Database property sheet to configure the database login:

Field	Description
Defined Logins	Login used to access the database. Click **Add** to add a new login, or **Remove** to delete an existing login.
Driver Class	The JDBC driver to use to connect to the database.
URL	The URL used to connect to the appropriate database.
User Name	The name required to log in to the database.
Password	The password required to log in to the database. **Note:** When exporting Java source and deployment XML (see "Exporting Project Information" on page 18-13), TopLink Workbench writes the database password (if applicable) using JCE encryption (when using JDK 1.4). Refer to *Oracle TopLink Getting Started Guide* for information on using password encryption with JDK 1.3 and earlier. For information on how to specify password encryption options, see "Configuring Password Encryption" on page 82-2.

Field	Description
Save Password	Whether or not to save the **Password** for this **Defined Login**.

Configuring Development and Deployment Logins

This section describes how to designate a defined login's role. For information on how to define a login, see "Configuring Login Information" on page 20-5. TopLink recognizes the following login roles:

- Development Role
- CMP Deployment Role
- Non-CMP Session Role

Development Role

While using TopLink Workbench to develop a project (see "Development Role: Development Login" on page 17-4), you must define a login (see "Configuring Login Information" on page 20-5) and designate it as the development login. The development login is stored in the TopLink project file. TopLink Workbench uses the information in the development login whenever you perform a data source operation from within TopLink Workbench. For example, when you read or write schema information from or to a data source during application development, the development login information is never written to a `sessions.xml` or `project.xml` file and is overridden by the deployment login (or the session login) at run time.

For more information on how to use a development login to connect to a database, see "Logging in to the Database" on page 20-7.

CMP Deployment Role

If you are creating a CMP project (see "CMP Deployment Role: Deployment Login" on page 17-3), you may define a run-time login (see "Configuring Login Information" on page 20-5) and designate it as the deployment login. This is the login that the application will use at run time, unless overridden in the `sessions.xml` file, CMP deployment file, or through Java code.

Non-CMP Session Role

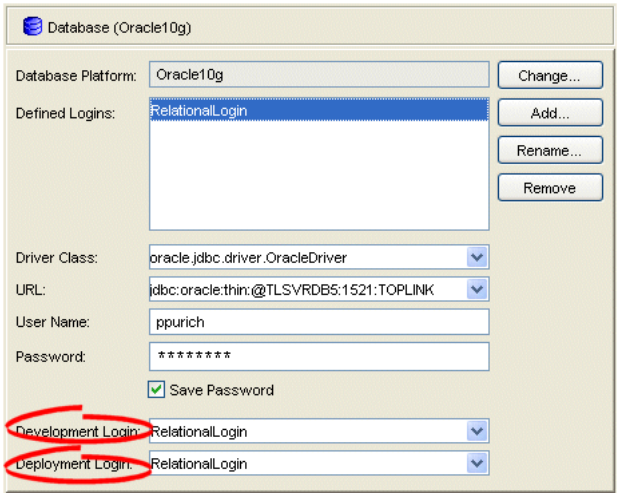
If you are creating a non-CMP project (see "Non-CMP Session Role: Session Login" on page 17-3), Oracle recommends that you use the `sessions.xml` file to store the sessions your project uses at run time (see "Data Source Login Types" on page 81-2).

Using TopLink Workbench

To specify different development and deployment database logins, use this procedure:

1. Select the database object in the **Navigator**. The Database property sheet appears.

Figure 20–4 Database Property Sheet, Development and Deployment Login Options



Use this table to enter data in the following fields on the Database property sheet to configure the login:

Field	Description
Development Login	The **Defined Login** to be used by TopLink Workbench during development to connect with the database, and to read or write table information. For more information on how to use a development login to connect to a database, see "Logging in to the Database" on page 20-7.
Deployment Login	The **Defined Login** to be used by your TopLink-enabled application during deployment.

Logging in to the Database

Using TopLink Workbench, after you create a login (see "Configuring Login Information" on page 20-5) and specify it as a development login (see "Configuring Development and Deployment Logins" on page 20-6), you can log in to a database instance.

You must log in to the database before importing or exporting table information.

To log in to the database, use one of the following procedures:



- Select the database object in the **Navigator** and click **Login**. TopLink Workbench logs in to the database.
- Right-click on the database object in the **Navigator** and choose **Log In to Database** from the context menu, or choose **Selected > Log In to Database** from the menu.



The database icon in the Navigator window changes to indicate you are now logged in to the database.

Configuring Named Query Parameterized SQL and Statement Caching at the Project Level

You can configure TopLink to use parameterized SQL and prepared statement caching for all named queries and finders.

These setting apply only to named queries (see "Named Queries" on page 93-16)–not to all queries in general or write operations.

Generally, to use parameterized SQL, you should configure it on the session's login (see "Configuring JDBC Options" on page 83-9). Configuring at the session level ensures that *all* queries will use parameterized SQL.

Note: For applications using a J2EE data source or external connection pool, you must configure statement caching in the J2EE server's data source–not in TopLink.

Table 20–2 summarizes which projects support query statement caching and binding configuration.

Table 20–2 Project Support for Default Named Query Caching and Binding

Descriptor	Using TopLink Workbench	Using Java
Relational Projects	✔	✔
EIS Projects		
XML Projects		

This section describes configuring statement caching and binding options at the project level. This configuration applies to *all* named queries you create on the descriptors in this project.

You can also configure named query statement caching and binding options at the query level to override this project-level configuration on a query-by-query basis (see "Configuring Named Query Options" on page 25-22).

Using parameterized SQL, you can create and store queries that are complete except for one or more bound parameters. The TopLink runtime binds the current parameter values when executing the query. This approach avoids the preparation of SQL execution and, thus, improves the performance of frequently executed SQL statements.

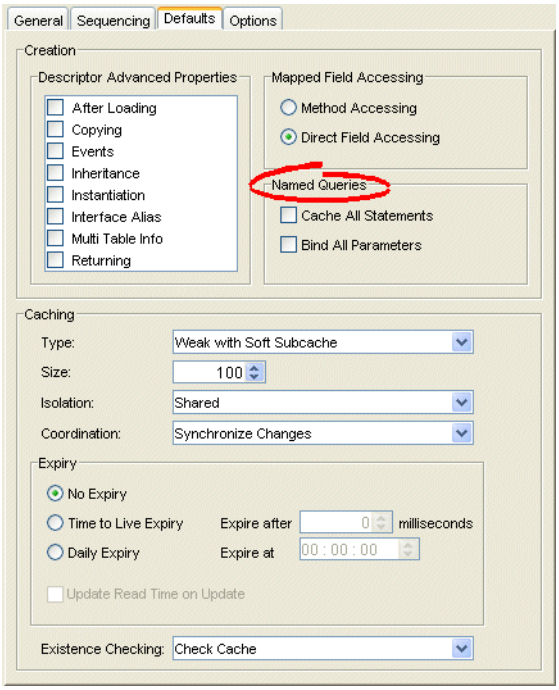
By default, Oracle TopLink writes data directly into its generated SQL, and does not use parameterized SQL. This is due to the fact that many JDBC drivers do not fully support parameter binding and impose restrictive size or type limits. For more information, see "Parameterized SQL (Binding) and Prepared Statement Caching" on page 11-15.

Using TopLink Workbench

To specify the named query options, use this procedure:

1. Select the project object in the **Navigator**.
2. Select the **Defaults** tab in the **Editor**. The Defaults tab appears.

Figure 20–5 Defaults Tab, Named Queries Options



Use this table to enter data in following fields on the Defaults tab to specify the named query options for newly created descriptors.:

Field	Description
Cache All Statements	Caches the query's prepared statement in the TopLink statement cache.
Bind All Parameters	Binds all of the query's parameters.

Configuring Table Generation Options

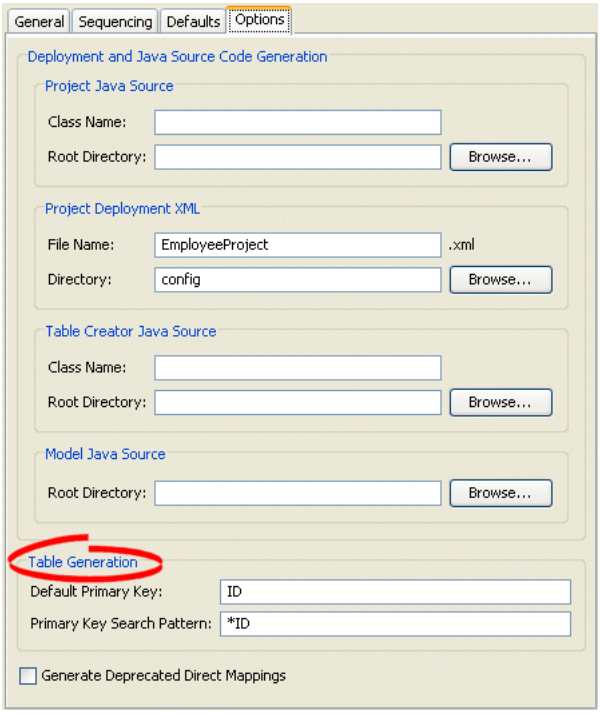
Using TopLink Workbench, you can configure options that apply when you generate database tables from the descriptors you define in your TopLink Workbench project. The resulting tables and columns will conform to the naming restrictions of the project's target database.

Using TopLink Workbench

To specify the default table generation options, use this procedure:

1. Select the project object in the **Navigator**.
2. Select the **Options** tab in the **Editor**. The Options tab appears.

Figure 20–6 Options Tab, Table Generation Options



Use this table to enter data in the following fields to specify the default export and generation options.

Field	Description
Default Primary Key	Enter the default name to use when generating primary keys.
Primary Key Search Pattern	Enter the default search pattern to use when generating primary keys.

Configuring Table Creator Java Source Options

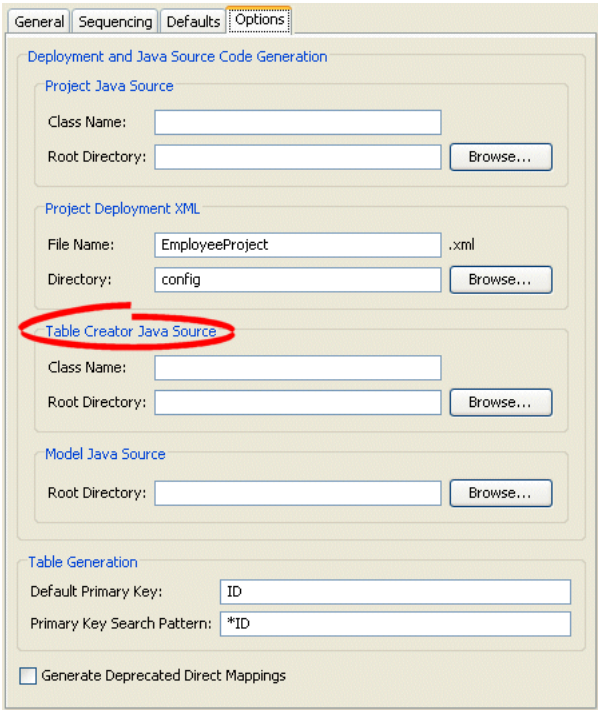
Using TopLink Workbench, you can configure options that apply when you export Java source code that you can use to create database tables.

Using TopLink Workbench

To specify the default Java code generation options, use this procedure:

1. Select the project object in the **Navigator**.
2. Select the **Options** tab in the **Editor**. The Options tab appears.

Figure 20–7 Options Tab, Table Creator Java Source Options



Use this table to enter data in the following fields to specify the default table creator options.

Field	Description
Class Name	Base class name to use when generating table's Java source code from the project.
Root Directory	Directory for storing the generated source code.

Configuring Project Java Source Code Options

Using TopLink Workbench, you can export a project as Java source. You can configure the class name and root directory that TopLink Workbench uses when exporting the project to Java source code.

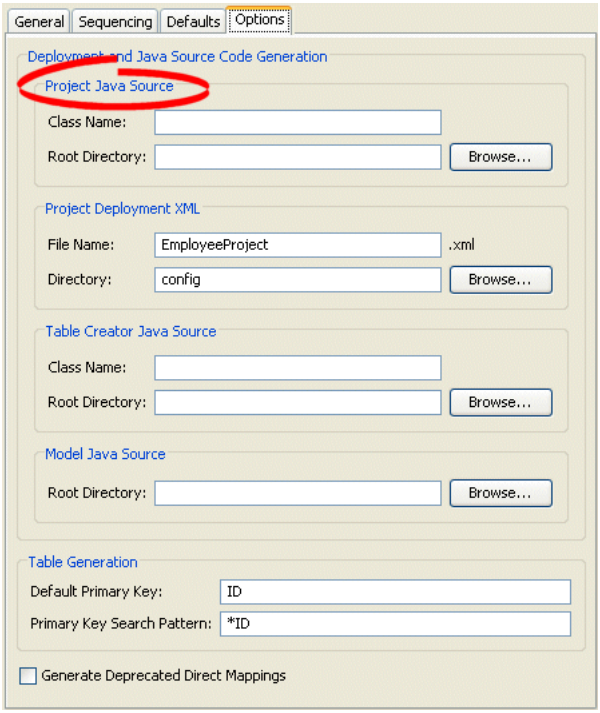
For more information on exporting a project as Java source, see "Exporting Project Java Source" on page 18-14.

Using TopLink Workbench

To specify the default Java code generation options, use this procedure:

1. Select the project object in the **Navigator**.
2. Select the **Options** tab in the **Editor**. The Options tab appears.

Figure 20–8 Options Tab, Project Java Source Options



Use this table to enter data in the following fields to specify the default export and generation options:

Field	Description
Class Name	Base class name to use when generating Java source code from the project.
Root Directory	Directory for storing the generated source code.

21

Configuring an EIS Project

This chapter describes the various components that you must configure in order to use an enterprise information system (EIS) project.

For more information, see the following:

- "Project Creation Overview" on page 18-1
- "EIS Projects" on page 17-7

EIS Project Configuration Overview

lists the configurable options for EIS projects.

Table 21–1 Configurable Options for EIS Projects

Option	Type	TopLink Workbench	Java
"Configuring Project Save Location" on page 19-2	Basic	✔	
"Configuring Persistence Type" on page 19-5	Basic	✔	
"Configuring Project Classpath" on page 19-3	Basic	✔	
"Configuring Project Comments" on page 19-21	Advanced	✔	
"Configuring Mapped Field Access at the Project Level" on page 19-4	Basic	✔	
"Configuring Default Descriptor Advanced Properties" on page 19-7	Advanced	✔	
"Configuring Existence Checking at the Project Level" on page 19-8	Advanced	✔	✔
"Configuring Project Deployment XML Options" on page 19-10	Advanced	✔	
"Configuring Model Java Source Code Options" on page 19-11	Advanced	✔	
"Configuring EIS Data Source Platform at the Project Level" on page 21-2	Basic	✔	✔
"Configuring EIS Connection Specification Options at the Project Level" on page 21-2	Basic	✔	✔
"Configuring Model Java Source Code Options" on page 19-11	Basic	✔	
"Configuring XML Parser Platform" on page 7-3	Advanced		✔
"Importing an XML Schema" on page 4-34	Basic	✔	
"Configuring XML Schema Namespace" on page 4-37	Advanced	✔	✔
"Configuring Cache Type and Size at the Project Level" on page 19-13	Advanced	✔	✔

Table 21–1 (Cont.) Configurable Options for EIS Projects

Option	Type	TopLink Workbench	Java
"Configuring Cache Isolation at the Project Level" on page 19-16	Advanced	✔	✔
"Configuring Cache Coordination Change Propagation at the Project Level" on page 19-17	Advanced	✔	✔
"Configuring Cache Expiration at the Project Level" on page 19-19	Advanced	✔	

Configuring EIS Data Source Platform at the Project Level

For each EIS project, you must specify one of the following J2C data source platforms that you will be using:

- Oracle AQ
- Attunity Connect
- IBM MQSeries

This platform configuration is overridden by the session login, if configured.

For more information, see the following:

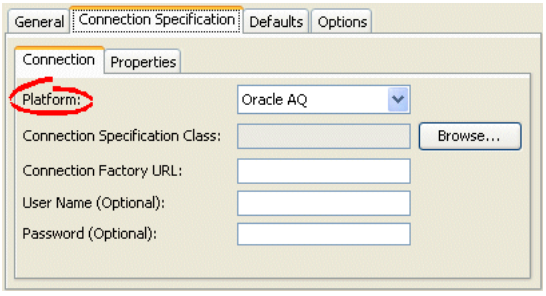
- "Configuring an EIS Data Source Platform at the Session Level" on page 84-1
- "Data Source Platform Types" on page 81-3

Using TopLink Workbench

To specify the data source platform of an EIS project, use this procedure:

1. Select an EIS project object in the **Navigator**.
2. Select the **Connection Specifications** tab in the **Editor**. The Connection Specifications tab appears.
3. Select the **Connection** tab. The Connection tab appears.

Figure 21–1 Connection Tab, Platform Option



Select the EIS platform for this project from the list of options. For more information, see "Data Source Platform Types" on page 81-3.

Configuring EIS Connection Specification Options at the Project Level

You can configure connection information at the project level for an EIS application. This information is stored in the `project.xml` file. The Oracle TopLink runtime uses

this information as its deployment login: whenever your EIS application performs a persistence operation when deployed in a J2EE application server.

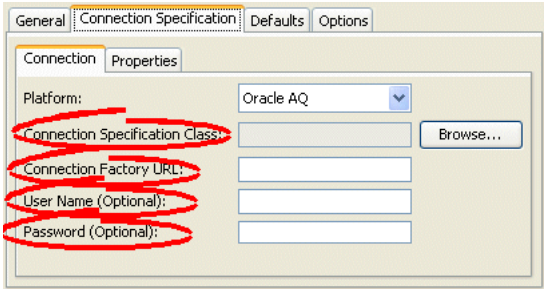
This connection configuration is overridden by the connection information at the session level, if configured. For more information about session level configuration, see "Configuring EIS Connection Specification Options at the Session Level" on page 84-2.

Using TopLink Workbench

To specify the connection information for an EIS project, use this procedure.

1. Select an EIS project object in the **Navigator**.
2. Select the **Connection Specifications** tab in the **Editor**. The Connection Specifications tab appears.
3. Select the **Connection** tab. The Connection tab appears.

Figure 21–2 Connection Tab, Connection Specification Options



Use this table to enter data in the following fields to configure the connection specification options:

Field	Description
Connection Specification Class	Specify the appropriate connection specification class for the selected **Platform**. Click **Browse** to choose from all the classes in the TopLink class path. (example: if **Platform** is `oracle.toplink.eis.aq.AQPlatform`, use `oracle.toplink.eis.aq.AQEISConnectionSpec`). For more information on platform configuration, see "Configuring an EIS Data Source Platform at the Session Level" on page 84-1.
Connection Factory URL	Specify the appropriate connection factory URL (as a J2EE JNDI name) for the selected **Connection Specification Class** (example: `java:comp/env/eis/attuntiy`).
Username	Specify the name required to log in to the data source.
Password	Specify the password required to log in to the data source. **Note:** When exporting Java source and deployment XML (see "Exporting Project Information" on page 18-13), TopLink Workbench writes the database password (if applicable) using JCE encryption (when using JDK 1.4). Refer to *Oracle TopLink Getting Started Guide* for information on using password encryption with JDK 1.3 and earlier. For information on how to specify password encryption options, see "Configuring Password Encryption" on page 82-2.

22

Configuring an XML Project

This chapter describes the various components that you must configure to use an XML project.

For more information, see the following:

- "Project Creation Overview" on page 18-1
- "XML Projects" on page 17-9

XML Project Configuration Overview

Table 22–1 lists the configurable options for XML projects.

Table 22–1 Configurable Options for XML Projects

Option	Type	TopLink Workbench	Java
"Configuring Project Save Location" on page 19-2	Basic	✔	
"Configuring Project Classpath" on page 19-3	Basic	✔	
"Configuring Project Comments" on page 19-21	Advanced	✔	
"Configuring Mapped Field Access at the Project Level" on page 19-4			
"Configuring Project Deployment XML Options" on page 19-10	Basic	✔	
"Configuring XML Parser Platform" on page 7-3	Advanced		✔
"Importing an XML Schema" on page 4-34	Basic	✔	
"Configuring XML Schema Namespace" on page 4-37	Advanced	✔	✔
"Configuring Model Java Source Code Options" on page 19-11	Advanced	✔	
"Configuring Default Descriptor Advanced Properties" on page 19-7	Advanced	✔	
"Configuring Mapped Field Access at the Project Level" on page 19-4	Advanced	✔	✔

Part VIII

Descriptors

This part describes the TopLink artifact used to describe persistent objects. It contains the following chapters.

- Chapter 23, "Understanding Descriptors"

  This chapter describes each of the different TopLink descriptor types and important descriptor concepts.

- Chapter 24, "Creating a Descriptor"

  This chapter contains procedures for creating TopLink descriptors.

- Chapter 25, "Configuring a Descriptor"

  This chapter explains how to configure TopLink descriptor options common to two or more descriptor types.

- Chapter 26, "Configuring a Relational Descriptor"

  This chapter explains how to configure descriptor options specific to a relational descriptor.

- Chapter 27, "Configuring an Object-Relational Descriptor"

  This chapter explains how to configure descriptor options specific to an object-relational descriptor.

- Chapter 28, "Configuring an EIS Descriptor"

  This chapter explains how to configure descriptor options specific to an EIS descriptor.

- Chapter 29, "Configuring an XML Descriptor"

  This chapter explains how to configure descriptor options specific to an XML descriptor.

23

Understanding Descriptors

TopLink uses descriptors to store the information that describes how an instance of a particular class can be represented by a data source. Descriptors own mappings that associate class instance variables with a data source and transformation routines that are used to store and retrieve values. As such, the descriptor acts as the connection between a Java object and its data source representation.

This chapter includes information on the following:

- Descriptor Types
- Descriptor Concepts
- Understanding the Descriptor API

Descriptor Types

Table 23–1 lists the descriptor types you use to describe the classes in your object model and classifies them as basic or advanced.

Table 23–1 TopLink Descriptor Types

Descriptor Type	Description	Type	TopLink Workbench	Java
"Relational Descriptors" on page 23-11	Describes Java objects that you map to tables in a relational database. Applicable to all relational databases that TopLink supports.	Basic	✔	✔
"Object-Relational Descriptors" on page 23-11	Describes Java objects that you map to tables in a relational database that provides special database data types that correspond more closely to object types. Applicable only to the relational databases that TopLink supports that provide these special data types.	Advanced		✔
"EIS Descriptors" on page 23-12	Describes Java objects that you map to an EIS data source by way of a J2C adapter.	Basic	✔	✔
"XML Descriptors" on page 23-12	Describes Java objects that you map, in memory, to complex types in XML documents defined by an XML schema document (XSD).	Basic	✔	✔

For more information, see the following:

- Chapter 24, "Creating a Descriptor"
- Chapter 25, "Configuring a Descriptor"

Descriptor Concepts

This section introduces descriptor concepts unique to TopLink, including the following:

- Descriptor Architecture
- Descriptors and Inheritance
- Descriptors and EJB
- Fetch Groups
- Amendment and After-Load Methods
- Descriptors and Aggregation
- Descriptor Event Manager
- Descriptor Query Manager
- Descriptors and Sequencing
- Descriptors and Locking
- Default Root Element

Descriptor Architecture

A **descriptor** stores all the information describing how an instance of a particular object class can be represented in a data source.

TopLink descriptors contain the following information:

- The persistent Java class it describes and the corresponding data source (database tables, XML complex type, or EIS interaction)
- A collection of mappings, which describe how the attributes and relationships for that class are stored in the database
- The primary key information (or equivalent) of the data source
- A list of query keys (or aliases) for field names
- Information for sequence numbers
- A set of optional properties for tailoring the behavior of the descriptor, including support for caching refresh options, identity maps, optimistic locking, the event manager, and the query manager

There is a descriptor type for each data source type that TopLink supports. In some cases, multiple descriptor types are valid for the same data source type. The type of descriptor you use determines the type of mappings that you can define.

Table 23–2 summarizes the relationship between project, descriptor, and mappings.

Table 23–2 Project, Descriptor, and Mapping Support

Project	Descriptor	Mapping
Relational Projects	Relational Descriptors	Relational Mappings
	Object-Relational Descriptors	Object-Relational Mappings
EIS Projects	EIS Descriptors	EIS Mappings
XML Projects	XML Descriptors	XML Mappings

Descriptors and Inheritance

Inheritance describes how a derived (child) class inherits the characteristics of its superclass (parent). You can use descriptors to describe the inheritance relationships between classes in relational, EIS, and XML projects.

In the descriptor for a child class, you can override mappings that have been specified in the descriptor for a parent class, or map attributes that have not been mapped at all in the parent class descriptor.

For more information, see "Understanding Descriptors and Inheritance" on page 23-12.

Descriptors and EJB

You can use descriptors to describe the characteristics of entity beans with container-managed or bean-managed persistence.

When mapping enterprise beans, you create a descriptor for the bean class: you do not create a descriptor for the local interface, remote interface, home class, or primary key class.

When using TopLink Workbench, you must define the project with the correct entity bean type (such as entity beans with container-managed or bean-managed persistence) and import the `ejb-jar.xml` file for the beans into the TopLink Workbench project.

For CMP projects, you use the `ejb-jar.xml` file to define the bean's mapped attributes. A descriptor of a bean with container-managed persistence contains a CMP policy used to configure CMP-specific options.

> **Note:** For EJB 3.0 projects, you can use annotations to define the bean's mapped attributes.

This section describes the following:

- Nondeferred Changes
- Creating a New Entity Bean and ejbCreate / ejbPostCreate Methods
- Inheritance

Nondeferred Changes

By default, TopLink defers all changes until commit time: this is the most efficient approach that produces the least number of data source interactions.

Alternatively, you can configure an entity bean's descriptor for nondeferred changes. This means that as you change the persistent fields of the entity bean, TopLink CMP modifies the relational schema immediately.

Using nondeferred changes, you can achieve backward compatibility with the native behavior of some EJB containers (such as OC4J). You can also accommodate advanced applications that rely on the database and entity changes being synchronized for such things as triggers or stored procedures based on transient state within the transaction, deletion and creation of rows with the same primary key, or other complex queries that depend on transient transaction state.

Nondeferred changes have the disadvantage of being the least efficient approach: they produce the greatest number of data source interactions.

When you configure TopLink CMP to support nondeferred changes, TopLink will continue to handle constraints for mapped relationships among entity beans with the

same deferral setting. However, you are responsible for handling any errors that result from making changes to a class that is not deferred, but related to a class that is deferred when a constraint exists between these two classes.

> **Note:** When you configure a descriptor for nondeferred changes, TopLink CMP does not apply nondeferred changes to dependent objects. Dependent objects are subject to default deferred changes: the relational schema is not modified until commit.

For more information, see "Configuring a Descriptor With EJB Information" on page 25-45.

Creating a New Entity Bean and ejbCreate / ejbPostCreate Methods

When you create a new entity bean, by default, the bean's life cycle can be thought of as follows:

1. `ejbCreate` method:

   After the insert, the EJB container retrieves the primary key allocated by the database for the created instance.

   For a relational project:

   a. `INSERT INTO ...`

   b. `SELECT FROM ...`

   For an EIS project:

   a. `Write object ...`

   b. `Find object ...`

2. `ejbPostCreate` method:

   The EJB container updates container-managed relationship (CMR) fields. The EJB container needs the primary key obtained in the `ejbCreate` method.

   For a relational project:

   a. `UPDATE SET ...`

   For an EIS project:

   a. `Write object ...`

However, if you have non-null foreign key constraints in your database, doing a data source modification after the `ejbCreate` method executes can cause problems. To get around this, some application servers (such as OC4J) allow you to create new objects after the `ejbPostCreate` method executes, and rely on the container to resolve the foreign key constraint.

For more information, see "Configuring a Descriptor With EJB Information" on page 25-45.

Inheritance

TopLink allows you to configure inheritance for CMP descriptors, with some reservations.

For more information, see "Inheritance and EJB" on page 23-17.

Fetch Groups

By default, when you execute an object-level read query for a particular object class, TopLink returns all the persistent attributes mapped in the object's descriptor. With this single query, all the object's persistent attributes are defined, and calling their `get` methods returns the value directly from the object.

When you are interested in only some of the attributes of an object, it may be more efficient to return only a subset of the object's attributes using a fetch group with which you can define a subset of an object's attributes and associate the fetch group with either a `ReadObjectQuery` or `ReadAllQuery` query.

For more information, see the following:

- "Configuring Fetch Groups" on page 25-79
- "Fetch Groups and Object-Level Read Queries" on page 93-13

Amendment and After-Load Methods

Using TopLink Workbench, you can associate a static Java method that is called when a descriptor is loaded at run time. This method can amend the run-time descriptor instance through the descriptor Java code API. Use this method to make some advanced configuration options that may not be currently supported by TopLink Workbench.

Descriptors can only be modified before the session has been connected; descriptors should not be modified after the session has been connected.

For more information, see "Configuring Amendment Methods" on page 25-81.

Descriptors and Aggregation

Two objects–a source (parent or owning) object and a target (child or owned) object–are related by aggregation if there is a strict one-to-one relationship between them, and all the attributes of the target object can be retrieved from the same data source representation as the source object. This means that if the source object exists, then the target object must also exist, and if the source object is destroyed, then the target object is also destroyed.

In this case, the descriptors for the source and target objects must be designated to reflect this relationship as follows:

- Aggregate and Composite Descriptors in Relational Projects
- Root and Composite Descriptors in EIS Projects
- Composite Descriptors in XML Projects

Aggregate and Composite Descriptors in Relational Projects

In a relational project, you can designate the descriptor as an aggregate (see "Relational Aggregate Descriptors" on page 24-2).

This lets you configure an aggregate mapping (see Chapter 43, "Configuring a Relational Aggregate Object Mapping") to associate data members in the target object with fields in the source object's underlying database tables.

When you designate a relational descriptor as an aggregate, TopLink lets you specify a mapping type for each field in the target class, but defers associating the field with a database table until you configure the aggregate object mapping in the source descriptor. In other words, the target class descriptor defines *how* each target class

field is mapped, but the source class descriptor defines *where* each target class field is mapped. This lets you share an aggregate object among many parent descriptors mapped to different tables.

When you designate a relational descriptor as an aggregate, you tell TopLink that the class will be a target of an aggregate object mapping, and this ensures that the TopLink runtime handles the target class as follows:

- It inserts, updates, and deletes the target class in parallel with its source class.
- It does not cache the target class on its own; instead, it caches the target class as part of its source class.
- It does not allow the target class to be read, written, deleted, or registered in a unit of work.

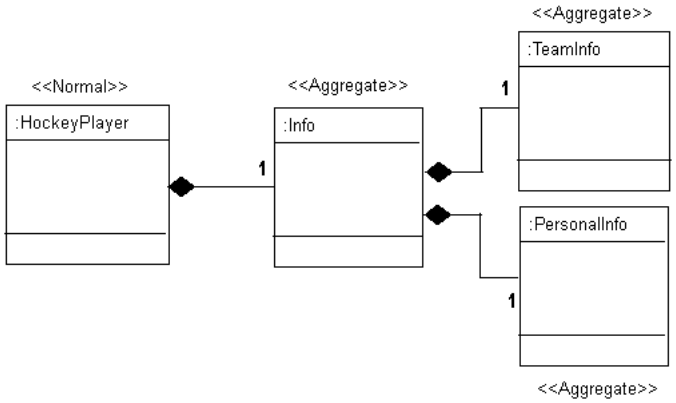
When working with aggregate relational descriptors, consider the following:

- Relational Aggregates and Nesting
- Relational Aggregates and Inheritance
- Relational Aggregates and EJB

For more information, see "Configuring a Relational Descriptor as a Class or Aggregate Type" on page 26-11.

Relational Aggregates and Nesting TopLink supports nested aggregates. In Figure 23–1 source class `HockeyPlayer` is a normal nonaggregate class descriptor. It owns target class `Info` which is designated as an aggregate. The `Info` class itself owns target classes `PersonalInfo` and `TeamInfo` which are each designated as aggregates.

Figure 23–1 Nested Aggregates



In EJB 3.0, an aggregate is known as an embeddable. In the EJB 3.0 specification, an embeddable may not contain another embeddable (that is, the EJB 3.0 specification does not support nested aggregates).

However, if you deploy a TopLink-enabled EJB 3.0 application with persistence to OC4J, you can take advantage of a TopLink extension of the EJB 3.0 specification to configure nested embeddables. Note that if you do so, your application will not be strictly EJB 3.0-compliant. Example 23–1 shows the classes from Figure 23–1 using EJB 3.0 annotations to take advantage of the TopLink extension of the EJB 3.0 specification to allow `Info` (an embeddable) to own embeddables `TeamInfo` and `PersonalInfo`.

Example 23–1 Nested Embeddables

```
public class HockeyPlayer implements Serializable {
    private int playerId;
    private Info Info;
    private String lastName;
    private String firstName;
    ...
    @Embedded
    public Info getInfo() {
        return Info;
    }
}

@Embeddable
public class Info implements Serializable {
    TeamInfo teamInfo; // TopLink extension of EJB 3.0 allows Embeddable with Embeddable
    PersonalInfo personalInfo;

    public Info() {}

    @Embedded
    public PersonalInfo getPersonalInfo() {
        return personalInfo;
    }

    public void setPersonalInfo(PersonalInfo personalInfo) {
        this.personalInfo = personalInfo;
    }

    @Embedded
    public TeamInfo getTeamInfo() {
        return teamInfo;
    }

    public void setTeamInfo(TeamInfo teamInfo) {
        this.teamInfo = teamInfo;
    }
}

@Embeddable
public class PersonalInfo implements Serializable {
    private int age;
    private double weight;
    private double height;
    ...
}

@Embeddable
public class TeamInfo implements Serializable {
    private String position;
    private int jerseyNumber;
    private HockeyTeam hockeyTeam;
    ...
}
```

Relational Aggregates and Inheritance You can configure inheritance for a relational descriptor designated as an aggregate (see "Descriptors and Inheritance" on page 23-3), however, in this case, *all* the descriptors in the inheritance tree must be aggregates. Aggregate and class descriptors cannot exist in the same inheritance tree.

Relational Aggregates and EJB You can use relational aggregate descriptors in an EJB project, but you cannot configure EJB information for a relational descriptor designated as an aggregate (see "Descriptors and EJB" on page 23-3).

For more information on using relational aggregates and EJB 3.0, see "Relational Aggregates and Nesting" on page 23-6.

Root and Composite Descriptors in EIS Projects

In an EIS project, you can designate the descriptor as a composites (see "EIS Composite Descriptors" on page 24-5).

The type of EIS mapping you whish to create will determine whether you configure an EIS descriptor as a composite or root (see "Composite and Reference EIS Mappings" on page 53-4).

For more information, see "Configuring an EIS Descriptor as a Root or Composite Type" on page 28-8.

You cannot configure EJB information for an EIS descriptor designated as an composite (see "Descriptors and EJB" on page 23-3).

You can configure inheritance for an EIS descriptor designated as a composite (see "Descriptors and Inheritance" on page 23-3), however, in this case, *all* the descriptors in the inheritance tree must be composites. Composite and root descriptors cannot exist in the same inheritance tree.

Composite Descriptors in XML Projects

In an XML project, descriptors are always composites.

Because XML descriptors are always composites, you can configure inheritance for an XML descriptor without considering its type (see "Descriptors and Inheritance" on page 23-3).

Descriptor Event Manager

In relational and EIS projects, TopLink raises various instances of `DescriptorEvent` (see Table 25–26 and Table 25–28) during the persistence life cycle. Each descriptor owns an instance of `DescriptorEventManager` that is responsible for receiving these events and dispatching them to the descriptor event handlers registered with it.

Using a descriptor event handler, you can execute your own application specific logic whenever descriptor events occur, allowing you to take customized action at various points in the persistence life-cycle. For example, using a descriptor event handler, you can do the following:

- Synchronize persistent objects with other systems, services, and frameworks.
- Maintain nonpersistent attributes of which 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 TopLink mappings.

For more information, see the following:

- "Configuring a Domain Object Method as an Event Handler" on page 25-59
- "Configuring a Descriptor Event Listener as an Event Handler" on page 25-62

Descriptor Query Manager

Each relational and EIS descriptor provides an instance of `DescriptorQueryManager` that you can use to configure the following:

- named queries (see "Configuring Named Queries at the Descriptor Level" on page 25-10)
- custom default queries for basic persistence operations (see "Configuring Default Query Implementations" on page 93-23)
- additional join expressions (see "Configuring Additional Join Expressions" on page 93-24)

For more information on using the query manager, see "Descriptor Query Manager Queries" on page 93-23.

Descriptors and Sequencing

An essential part of maintaining object identity is managing the assignment of unique values (that is, a specific sequence) to distinguish one object instance from another. For more information, see "Projects and Sequencing" on page 17-4.

Sequencing options you configure at the project (or session) level determine the type of sequencing that TopLink uses. In a CMP project, you typically configure the sequence type at the project level (see "Configuring Sequencing at the Project Level" on page 20-3). In a non-CMP project, you can use session-level sequence configuration to override project-level sequence configuration, on a session-by-session basis, if required (see "Configuring Sequencing at the Session Level" on page 83-4).

After configuring the sequence type, for each descriptor's reference class, you must associate one attribute, typically the attribute used as the primary key (see "Configuring Primary Keys" on page 25-3), with its own sequence (see "Configuring Sequencing at the Descriptor Level" on page 26-3).

Descriptors and Locking

You can configure a descriptor with any of the following locking policies to control concurrent access to a domain object:

- Optimistic–All users have read access to the data. When a user attempts to make a change, the application checks to ensure the data has not changed since the user read the data (see "Optimistic Version Locking Policies" on page 23-18 and "Optimistic Field Locking Policies" on page 23-20).
- Pessimistic–The first user who accesses the data with the purpose of updating it locks the data until completing the update (see "Pessimistic Locking Policy" on page 23-22).
- No locking–The application does not prevent users overwriting each other's changes.

Oracle recommends using optimistic locking for most types of applications to ensure that users do not overwrite each other's changes.

For more information, see the following:

- "Understanding Descriptors and Locking" on page 23-18
- "Configuring Locking Policy" on page 25-64

Default Root Element

You configure EIS root descriptors ("Configuring Default Root Element" on page 28-3) and XML descriptors ("Configuring Default Root Element" on page 29-5) with a

default root element so that the TopLink runtime knows the data source data type associated with the class the descriptor describes.

> **Note:** The undefined document root element of a referenced object is ignored during marshalling with an any collection mapping and object mapping.

This section describes what a default root element is and how TopLink uses it.

Consider the `Customer` and `Address` classes and their mappings shown in Example 23–2.

Example 23–2 Customer and Address Classes

```
Class: Customer
Default Root: customer
Attributes and Mappings:
    name:String                Drect Mapping to                name/text()
    billingAddress:Address     Composite Object Mapping to     billing-address
    shippingAddress:Address    Composite Object Mapping to     shipping-address

Class: Address
Default Root: address
Attributes and Mappings:
    street:String              Direct Mapping to               street/text()
    city:String                Direct Mapping to               city/text()
```

These classes correspond to the XML schema shown in Example 23–3.

Example 23–3 Customer and Address Schema

```
<xsd:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xsd:complexType name="address-type">
        <xsd:sequence>
            <element name="street" type="xsd:string"/>
            <element name="city" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="customer" type="customer-type"/>
    <xsd:complexType name="customer-type">
        <xsd:sequence>
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="billing-address" type="address-type"/>
            <xsd:element name="shipping-address" type="address-type"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

When an instance of the `Customer` class is persisted to XML, the TopLink runtime performs the following:

1. Gets the default root element.

   The `Customer` class instance corresponds to the root of the XML document. The TopLink runtime uses the default root element specified on the descriptor (`customer`) to start the XML document. TopLink then uses the mappings on the descriptor to marshal the object's attributes:

   ```
   <customer>
       <name>…</name>
   </customer>
   ```

2. When the TopLink runtime encounters an object attribute such as `billingAddress`, it checks the mapping associated with it to determine with what element (`billing-address`) to continue:

```
<customer>
    <name>…</name>
    <billing-address/>
</customer>
```

The TopLink runtime checks the mapping's reference descriptor (`Address`) to determine what attributes to persist:

```
<customer>
    <name>…</name>
    <billing-address>
        <street>…</street>
        <city>…</city>
    </billing-address>
</customer>
```

Relational Descriptors

Relational descriptors describe Java objects that you map to tables in a relational database. You use them in relational projects (see "Relational Projects" on page 17-6).

Using relational descriptors in a relational project, you can configure relational mappings (see "Relational Mapping Types" on page 33-1).

For more information, see the following:

- "Creating a Relational Descriptor" on page 24-1
- "Configuring a Relational Descriptor" on page 26-1

Object-Relational Descriptors

The object-relational paradigm extends traditional relational databases to include object-oriented functions. Oracle, IBM DB2, Informix, and other DBMS databases allow users to store, access, and use complex data in more sophisticated ways.

The object-relational standard is an evolving standard concerned mainly with extending the database data structures and SQL (SQL 3).

Object-relational descriptors describe Java objects that you map to special relational database types that correspond more closely to object types. Using these special object-relational database types can simplify mapping objects to relational database tables. Not all relational databases support these special object-relational database types.

Using object-relational descriptors in a relational project, you can configure object-relational mappings to these special object-relational database data types (see "Object-Relational Mapping Types" on page 46-1).

For more information, see the following:

- "Creating an Object-Relational Descriptor" on page 24-3
- Chapter 27, "Configuring an Object-Relational Descriptor"

EIS Descriptors

EIS descriptors describe Java objects that you map to an EIS data source by way of a J2C adapter.

Using EIS descriptors in an EIS project created with TopLink Workbench, you can configure EIS mappings (see "EIS Mapping Types" on page 53-1) to XML records.

Using EIS descriptors in an EIS project that you create in Java, you can configure EIS mappings to any supported EIS record type: XML, mapped, or indexed.

For more information, see the following:

- "Creating an EIS Descriptor" on page 24-4
- Chapter 28, "Configuring an EIS Descriptor"

XML Descriptors

XML descriptors describe Java objects that you map to simple and complex types defined by an XML schema document (XSD).

Using XML descriptors in an XML project, you can configure XML mappings (see "XML Mapping Types" on page 62-1), in memory, to XML elements defined by an XSD.

For more information, see the following:

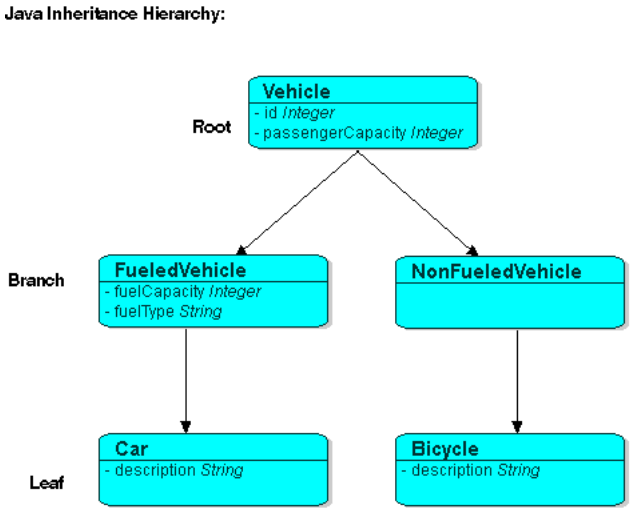
- "Creating an XML Descriptor" on page 24-5
- Chapter 29, "Configuring an XML Descriptor"

Understanding Descriptors and Inheritance

Inheritance describes how a derived class inherits the characteristics of its superclass. You can use descriptors to describe the inheritance relationships between classes in relational, EIS, and XML projects.

Figure 23–2 illustrates the `Vehicle` object model–a typical Java inheritance hierarchy. The root class `Vehicle` contains two branch classes: `FueledVehicle` and `NonFueledVehicle`. Each branch class contains a leaf class: `Car` and `Bicycle`, respectively.

Figure 23–2 Example Inheritance Hierarchy



TopLink recognizes the following three types of classes in an inheritance hierarchy:

1. The root class stores information about *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, the root class can be configured so queries on it return only instances of itself, without instances of its subclasses.

   For example, the `Vehicle` class in Figure 23–2 is a root class.

2. Branch classes have a persistent superclass and also have subclasses. By default, queries performed on the branch class return instances of the branch class and any of its subclasses. However, as with the root class, the branch class can be configured so queries on it return only instances of itself without instances of its subclasses.

   For example, the `FueledVehicle` class in Figure 23–2 is a branch class.

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

   For example, the `Car` class in Figure 23–2 is a leaf class.

In the descriptor for a child class, you can override mappings that have been specified in the descriptor for a parent class, or map attributes that have not been mapped at all in the parent class descriptor.

This section includes information on the following topics:

- Specifying a Class Indicator
- Inheritance and Primary Keys (Relational and EIS Only)
- Single and Multi-Table Inheritance (Relational Only)
- Aggregate and Composite Descriptors and Inheritance
- Inheritance and EJB

For more information about configuring inheritance for a parent (root) class descriptor, see "Configuring Inheritance for a Parent (Root) Descriptor" on page 25-52.

For more information about configuring inheritance for a child (branch or leaf) class descriptor, see "Configuring Inheritance for a Child (Branch or Leaf) Class Descriptor" on page 25-51.

Specifying a Class Indicator

When configuring inheritance, you configure the root class descriptor with the means of determining which subclasses it should instantiate.

You can do this in one of the following ways:

- Using Class Indicator Fields
- Using Class Extraction Methods

> **Note:** All leaf classes in the hierarchy must have a class indicator and they must have the same type of class indicator (field or class extraction method).

Using Class Indicator Fields

You can use a persistent attribute of a class to indicate which subclass should be instantiated. For example, in a relational descriptor, you can use a class indicator field in the root class table. The indicator field should not have an associated direct mapping unless it is set to read-only.

> **Note:** If the indicator field is part of the primary key, define a write-only transformation mapping for the indicator field (see Chapter 45, "Configuring a Relational Transformation Mapping").

You can use strings or numbers as values in the class indicator field.

The root class descriptor must specify how the value in the class indicator field translates into the class to be instantiated.

One approach is to configure the root class descriptor with a class indicator dictionary: a collection of key-values that associates a simple key, stored in the class indicator field, with a class to instantiate. Table 23–3 illustrates the class indicator dictionary for the `Vehicle` class's subclasses as shown in Figure 23–2.

Table 23–3 Class Indicator Dictionary for the Vehicle Class

Key	Value
F	FueledVehicle
N	NonFueledVehicle
C	Car
B	Bicycle

Another approach is to simply use the class name itself as the value stored in the class indicator field. This avoids having to define unique indicators for each class at the expense of a slightly larger key value (depending on the length of your class names).

Using Class Extraction Methods

You can define a Java method to compute the class indicator based on any available information in the object's data source record. Such a method is called a class extraction method.

Using a class extraction method, you do not need to include an explicit class indicator field in your data model and you can handle relationships that are too complex to describe using class indicator fields.

A class extraction method must have the following characteristics:

- it must be defined on the root descriptor's class
- it must be static
- it must take a `Record` as an argument
- it must return the `java.lang.Class` object to use for the `Record` passed in

You may also need to define only-instances and with-all-subclasses expressions (see "Specifying Expressions for Only-Instances and With-All-Subclasses" on page 23-16).

For example, Table 23–4 lists the rows in the `EMPLOYEE` table. The `Employee` class is the base class. `Director`, `Manager`, `Programmer`, and `TechWriter` classes each derive from the `Employee` class. However, in your application, instances of `Manager`, `Programmer`, and `TechWriter` classes must be represented as `Employee` instances and instances of `Director` must be represented as `Director` instances. Because there is no a one-to-one correspondence between class and `JOB_TYPE` field value, the `JOB_TYPE` field alone cannot serve as a class indicator field (see "Using Class Indicator Fields" on page 23-14). To resolve this issue, you could use the class extraction method shown in Example 23–4.

Table 23–4 EMPLOYEE Table

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

Example 23–4 Class Extraction Method

```
...
// If the JOB_TYPE field value in record equals 2, return the Director class.
// Return the Employee class for all other JOB_TYPE field values

public static Class getClassFromRecord(Record record)
{
    if (record.get("JOB_TYPE").equals(new Integer(2))
    {
         return Director.class;
    }
    else
    {
        return Employee.class;
    }
}
```

When configuring inheritance using a class extraction method, Oracle TopLink does not generate SQL for queries on the root class.

Specifying Expressions for Only-Instances and With-All-Subclasses If you use a class extraction method (see "Using Class Extraction Methods" on page 23-15), you must provide TopLink with expressions to correctly filter sibling instances for all classes that share a common table (see "Configuring Inheritance Expressions for a Parent (Root) Class Descriptor" on page 25-55).

Inheritance and Primary Keys (Relational and EIS Only)

For relational and EIS projects, TopLink assumes that all of the classes in an inheritance hierarchy have the same primary key, as set in the root descriptor. Child descriptors associated with data source representations that have different primary keys must define the mapping between the root primary key and the local one.

Single and Multi-Table Inheritance (Relational Only)

In a relational project, you can map your inheritance hierarchy to a single table ("Single Table Inheritance" on page 23-16) or to multiple tables ("Multitable Inheritance" on page 23-16). Use these options to achieve the balance between storage efficiency and access efficiency that is appropriate for your application.

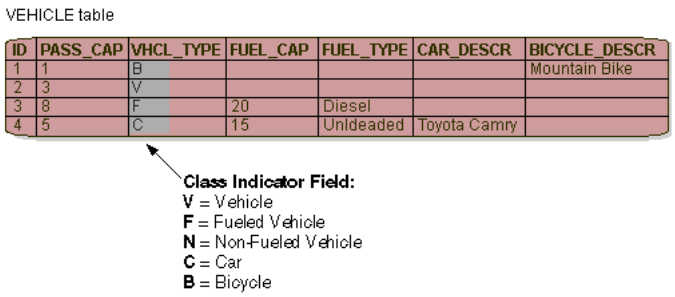
Single Table Inheritance

In this example, you store classes with multiple levels of inheritance in a single table to optimize database access speeds.

The entire inheritance hierarchy shown in Figure 23–2 can share the same table, as in Figure 23–3. The `FueledVehicle` and `NonFueledVehicle` subclasses can share the same table even though `FueledVehicle` has some attributes that `NonFueledVehicle` does not. The `NonFueledVehicle` instances waste database resources because the database must still allocate space for the unused portion of its row. However, this approach saves on accessing time because there is no need to join to another table to get the additional `FueledVehicle` information.

As Figure 23–3 shows, this approach uses a class indicator field. For more information, see "Specifying a Class Indicator" on page 23-14.

Figure 23–3 Inheritance Using a Superclass Table with Optional Fields



Multitable Inheritance

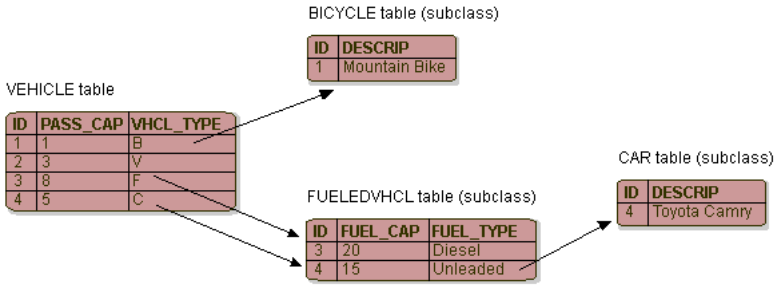
In this example, you store classes with multiple levels of inheritance in multiple tables to optimize database storage space.

In the inheritance hierarchy shown in Figure 23–2, for subclasses that require additional attributes, you use multiple tables instead of a single superclass table. This optimizes storage space because there are no unused fields in the database. However, this may affect performance because TopLink must read from more than one table

before it can instantiate the object. TopLink first looks at the class indicator field (see "Specifying a Class Indicator" on page 23-14) to determine the class of object to create, then uses the descriptor for that class to read from the subclass tables.

Figure 23–4 illustrates the TopLink implementation of the `FUELEDVHCL`, `CAR`, and `BICYCLE` tables. All objects are stored in the `VEHICLE` table. `FueledVehicle`, `Car`, and `Bicycle` information are also stored in secondary tables. Note that because the `NonFueledVehicle` class does not hold any attributes or relationships, it does not need a secondary table.

Figure 23–4 Inheritance Using Separate Tables for Each Subclass



Note: . In general, using multitable inheritance is inefficient because it can require excessive joins and multiple table fetching.

Inheritance View If a root or branch inheritance descriptor has subclasses that span multiple tables, you can configure a database view to optimize the performance of queries against the parent descriptor by outer-joining all of the subclass tables. This allows TopLink to fetch all of the subclass instances in one query, instead of multiple queries. It also allows queries for the parent class that use cursors or ordering.

You must define the view on the database as a database view that outer-joins all of the subclass tables. For more information, see "Configuring Reading Subclasses on Queries" on page 25-49.

Aggregate and Composite Descriptors and Inheritance

You can designate relational descriptors as aggregates, and EIS descriptors as composites. XML descriptors are always composites (see "Descriptors and Aggregation" on page 23-5).

When configuring inheritance for a relational aggregate descriptor, all the descriptors in the inheritance tree must be aggregates. The descriptors for aggregate and non-aggregate classes cannot exist in the same inheritance tree.

Similarly, when configuring inheritance for an EIS composite descriptor, all the descriptors in the inheritance tree must be composites. The descriptors for composite and noncomposite classes cannot exist in the same inheritance tree.

When configuring inheritance for an XML descriptor, because all XML descriptors are composites, descriptor type does not restrict inheritance.

Inheritance and EJB

Although inheritance is a standard tool in object-oriented modeling, the EJB specifications prior to 3.0 contain only general information regarding inheritance. You

should fully understand this information before implementing EJB inheritance. Be aware of the fact that future EJB specifications may dictate inheritance guidelines not supported by all application servers.

Understanding Descriptors and Locking

This section describes the various types of locking policy that TopLink supports, including the following:

- Optimistic Version Locking Policies
- Optimistic Version Locking Policies and Cascading
- Optimistic Field Locking Policies
- Pessimistic Locking Policy
- Locking in a Three-Tier Application

For more information, see "Configuring Locking Policy" on page 25-64.

Optimistic Version Locking Policies

With optimistic locking, all users have read access to the data. When a user attempts to make a change, the application checks to ensure the data has not changed since the user read the data.

Optimistic version locking policies enforce optimistic locking by using a version field (also known as a write-lock field) that you provide in the reference class that TopLink updates each time an object change is committed.

TopLink caches the value of this version field as it reads an object from the data source. When the client attempts to write the object, TopLink compares the cached version value with the current version value in the data source in the following way:

- If the values are the same, TopLink updates the version field in the object and commits the changes to the data source.
- If the values are different, the write operation is disallowed because another client must have updated the object since this client initially read it.

TopLink provides the following version-based optimistic locking policies:

- `VersionLockingPolicy`: requires a *numeric* version field; TopLink updates the version field by incrementing its value by one.
- `TimestampLockingPolicy`: requires a *timestamp* version field; TopLink updates the version field by inserting a new timestamp (this policy can be configured to get the time from the data source or locally; by default, the policy gets the time from the data source).

Note: In general, Oracle recommends numeric version locking, because:

- accessing the timestamp from the data source can cause a performance issue
- time stamp locking is limited to the precision that the database stores for timestamps

---

Whenever any update fails because optimistic locking has been violated, TopLink throws an `OptimisticLockException`. This should be handled by the application when performing any database modification. The application must notify the client of the locking contention, refresh the object, and have the client reapply its changes.

You can choose to store the version value in the object as a mapped attribute, or in the cache. In three-tier applications, you typically store the version value in the object to ensure it is passed to the client when updated (see "Locking in a Three-Tier Application" on page 23-22).

If you store the version value in the cache, you do not need to map it. If you do map the version field, you must configure the mapping as read-only (see "Configuring Read-Only Mappings" on page 32-2).

To ensure that the parent object's version field is updated whenever a privately owned child object is modified, consider "Optimistic Version Locking Policies and Cascading" on page 23-19.

When using optimistic version locking with the unit of work, consider "Using Optimistic Read Locking With forceUpdateToVersionField" on page 99-18.

Optimistic Version Locking Policies and Cascading

If your database schema is such that both a parent object and its privately owned child object are stored in the same table, then if you update the child object, the parent object's version field will be updated.

However, if the parent and its privately owned child are stored in separate tables, then changing the child will not, by default, update the parent's version field.

To ensure that the parent object's version field is updated in this case, you can either manually update the parent object's version field (see "Using Optimistic Read Locking With forceUpdateToVersionField" on page 99-18) or, if you are using a `TimestampLockingPolicy`, you can configure TopLink to automatically cascade the child object's version field update to the parent (see "Configuring Optimistic Locking Policy Cascading" on page 25-67).

After you enable optimistic version locking cascading, when a privately owned child object is modfied, TopLink will traverse the privately owned foreign reference mappings, updating all the parent objects back to the root.

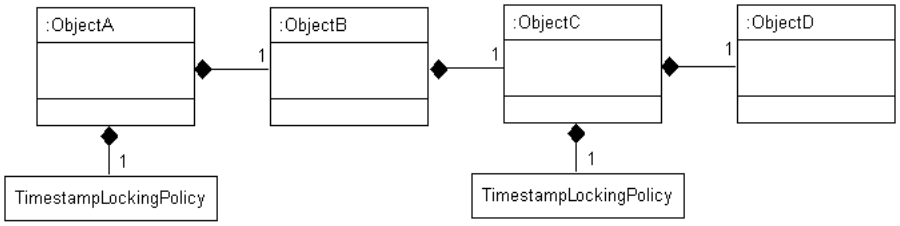
Optimistic version locking cascading is only applied if the child object is registered in a unit of work.

TopLink supports optimistic version locking cascading for:

- object changes in privately owned one-to-one and one-to-many mappings
- relationship changes (adding or removing) in the following collection mappings (privately owned or not):
  - direct collection
  - one-to-many
  - many-to-many
  - aggregate collection

Consider the example object graph shown in Figure 23–5

Figure 23–5 Optimistic Version Locking Policies and Cascading Example



In this example, `ObjectA` privately owns `ObjectB`, and `ObjectB` privately owns `ObjectC`, and `ObjectC` privately owns `ObjectD`.

Suppose you register `ObjectB` in a unit of work, modify an `ObjectB` field, and commit the unit of work. In this case, `ObjectB` checks the cache for `ObjectA` and, if not present, queries the database for `ObjectA`. `ObjectB` then notifies `ObjectA` of its change. `ObjectA` forces an update on its version optimistic locking field even though it has no changes to its corresponding table.

Suppose you register `ObjectA` in a unit of work, access its `ObjectB` to access its `ObjectC` to access its `ObjectD`, modify an `ObjectD` field, and commit the unit of work. In this case, `ObjectD` notifies `ObjectC` of its changes. `ObjectC` forces an update on its version optimistic locking field even though it has no changes to its corresponding table. `ObjectC` then notifies `ObjectB` of the `ObjectD` change. `ObjectB` then notifies `ObjectA` of the `ObjectD` change. `ObjectA` forces an update on its version optimistic locking field even though it has no changes to its corresponding table.

Optimistic Locking and Rollbacks

With optimistic locking, use the `UnitOfWork` method `commitAndResumeOnFailure` (see "Resuming a Unit of Work After Commit" on page 99-14) to rollback a locked object's value, if you store the optimistic lock versions in the cache.

If you store the locked versions in an object, you must refresh the objects (or their versions) on a failure. Alternatively, you can acquire a new unit of work on the failure and reapply any changes into the new unit of work.

Optimistic Field Locking Policies

Optimistic field locking policies enforce optimistic locking by using one or more of the fields that currently exist in the table to determine if the object has changed since the client read the object.

The unit of work caches the original state of the object when you first read the object or register it with the unit of work. At commit time, the unit of work compares the original values of the lock fields with their current values on the data source during the update. If any of the lock field's values have changed, an optimistic lock exception is thrown.

TopLink provides the following optimistic field locking policies:

- `AllFieldsLockingPolicy`: For update and delete operations, TopLink compares all the fields of the object with all the fields in the data source. If the original value of any fields differ from that in the data source, the write operation is disallowed.

For example, if you changed a customer's last name, TopLink might produce SQL like:

```
UPDATE CUSTOMER SET LNAME='new last name' WHERE ID=7 AND LNAME='old last name'
AND FNAME='Donald' AND B_DAY='1972' AND CREDIT_RATING='A+' AND EYE_COLOR='Blue'
```

The main disadvantage of this field locking policy is that it is not the most efficient, especially if the changed object has many attributes.

Note: This comparison is only on a per table basis. If an update operation is performed on an object that is mapped to multiple tables (multiple table inheritance), then only the changed fields for each table changed appear in the `where` clause.

- `ChangedFieldsLockingPolicy`: For update operations, TopLink compares only the fields of the object that have changed with the corresponding fields in the data source. If the original value of any such field differs from that in the data source, the write operation is disallowed. TopLink does not make any field comparisons for deletes.

  The main advantage of this field locking policy is that it allows concurrent updates of different fields. For example, if one thread updates a customer's last name and another thread updates the same customer's credit rating, and you configure the `Customer` descriptor with `ChangedFieldsLockingPolicy`, then TopLink might produce SQL like:

  ```
  // Unit of Work 1
  UPDATE CUSTOMER SET LNAME='new name' WHERE ID=7 AND LNAME='old name'
  // Unit of Work 2
  UPDATE CUSTOMER SET CREDIT_RATING='B' WHERE ID=7 AND CREDIT_RATING='A+'
  ```

- `SelectedFieldsLockingPolicy`: For update and delete operations, TopLink compares only the selected fields of the object with the corresponding fields in the data source. If the cached value of any such field differs from that in the data source, the write operation is disallowed.

  For example, if you select `Customer` attributes `LNAME` and `CREDIT_RATING`, then at run time, TopLink might produce SQL like:

  ```
  UPDATE CUSTOMER SET LNAME='new name' WHERE ID=7 AND LNAME='old name' AND
  CREDIT_RATING='A+'
  ```

Whenever any update fails because optimistic locking has been violated, TopLink throws an `OptimisticLockException`. This should be handled by the application when performing any database modification. The application must notify the client of the locking contention, refresh the object, and have the client reapply its changes.

When using field locking policies, a unit of work must be employed for updating the data source.

Note: You cannot use an instance of `FieldsLockingPolicy` if you are using `AttributeChangeTrackingPolicy` (see "Attribute Change Tracking Policy" on page 97-8).

Pessimistic Locking Policy

With pessimistic locking, the first user who accesses the data with the purpose of updating it locks the data until completing the update.

When using a pessimistic locking policy, you can configure the policy to either fail immediately or to wait until the read lock is acquired.

You can use a pessimistic locking policy only in a project with a container-managed persistence type (see "Configuring Persistence Type" on page 19-5) and with descriptors that have EJB information (see "Configuring a Descriptor With EJB Information" on page 25-45).

You can also use pessimistic locking (but not a pessimistic locking policy) at the query level (see "Configuring Named Query Options" on page 25-22).

TopLink provides an optimization for pessimistic locking when this locking is used with entity beans with container-managed persistence: if you set your query to pessimistic locking and run the query in its own new transaction (which will end after the execution of the finder), then TopLink overrides the locking setting and does not append `FOR UPDATE` to the SQL. However, the use of this optimization may produce an undesirable result if the pessimistic lock query has been customized by the user with a SQL string that includes FOR UPDATE. In this case, if the conditions for the optimization are present, the query will be reset to nonpessimistic locking, but the SQL will remain the same resulting in the locking setting of the query conflicting with the query's SQL string. To avoid this problem, you can take one of the following two approaches:

- Use EJB QL or a TopLink's expression (see Chapter 95, "Understanding TopLink Expressions") for the selection criteria. This will give TopLink control over the SQL generation.
- Place the finder in a transaction to eliminate conditions for the optimization.

Locking in a Three-Tier Application

If you are building a three-tier application, in order to correctly lock an object, you must obtain the lock before the object is sent to client for editing.

Optimistic Locking in a Three-Tier Application

If you are using optimistic locking, you have the following two choices for locking objects correctly:

1. Map the optimistic lock field in your object as not read-only and pass the version to the client on the read and back to the server on the update.

   You must define a non-read-only mapping for the version field and make the optimistic locking policy store the version value in the object, not the cache (in TopLink Workbench, this is done on the Locking tab by unchecking **Store Version in Cache**: see "Using TopLink Workbench" on page 25-65).

   Ensure that the original version value is sent to the client when it reads the object for the update. The client must then pass the original version value back with the update information, and this version must be set into the object to be updated after it is registered/read in the new unit of work on the server.

2. Hold the unit of work for the duration of the interaction with the client.

   Either through a stateful session bean, or in an HTTP session, store the unit of work used to read the object for the update for the duration of the client interaction.

Your must read the object through this unit of work before passing it to the client for the update. This ensures that the version value stored in the unit of work cache or in the unit of work clone will be the original value.

This same unit of work must be used for the update.

The first option is more commonly used, and is required if developing a stateless application.

Pessimistic Locking in a Three-Tier Application

If you are using pessimistic locking, you must use the unit of work to start a database transaction before the object is read. You must hold this unit of work and database transaction while the client is editing the object and until the client updates the object. You must use this same unit of work to update the object. If you are building a three-tier Web application (where it is not normally desirable to hold a database transaction open across client interactions), optimistic locking is normally more desirable than pessimistic locking (see "Optimistic Locking in a Three-Tier Application" on page 23-22).

Understanding the Descriptor API

The descriptor API can be used to define, or amend TopLink descriptors through Java code. The descriptor API classes are mainly in the `oracle.toplink.descriptors` package. These include the following classes:

- `ClassDescriptor` (abstract generic descriptor API)
- `RelationalDescriptor` (relational project-specific API)
- `DescriptorEventManager` (event API)
- `DescriptorQueryManager` (query API)
- `InheritancePolicy`
- `InterfacePolicy`
- `ReturningPolicy`
- Locking policies (various optimistic locking policies)

For object-relational, EIS, and XML projects, descriptor classes are in the `oracle.toplink.objectrelational`, `oracle.toplink.eis`, and `oracle.toplink.ox` packages, respectively.

This section describes the important descriptor classes in the Oracle TopLink Foundation Library, including:

- Descriptor Inheritance Hierarchy

Descriptor Inheritance Hierarchy

Example 23–5 illustrates the descriptor types that derive from class `oracle.toplink.descriptors.ClassDescriptor`.

Example 23–5 Descriptor Inheritance Hierarchy

```
class oracle.toplink.descriptors.ClassDescriptor
    class oracle.toplink.descriptors.RelationalDescriptor
        class oracle.toplink.objectrelational.ObjectRelationalDescriptor
    class oracle.toplink.eis.EISDescriptor
    class oracle.toplink.ox.XMLDescriptor
```

24

Creating a Descriptor

This chapter includes the following information:

- Descriptor Creation Overview
- Creating a Relational Descriptor
- Creating an Object-Relational Descriptor
- Creating an EIS Descriptor
- Creating an XML Descriptor
- Validating Descriptors
- Generating Java Code for Descriptors

Descriptor Creation Overview

For information on creating descriptors, see the following:

- "Creating a Relational Descriptor" on page 24-1
- "Creating an Object-Relational Descriptor" on page 24-3
- "Creating an EIS Descriptor" on page 24-4
- "Creating an XML Descriptor" on page 24-5

After you create a descriptor, you must configure its various options (see Chapter 25, "Configuring a Descriptor") and use it to define mappings.

For complete information on the various types of mapping that TopLink supports, see Chapter 30, "Understanding Mappings" and Chapter 31, "Creating a Mapping".

For complete information on the various types of descriptor that TopLink supports, see "Descriptor Types" on page 23-23.

Creating a Relational Descriptor

You can create a relational descriptor using TopLink Workbench (see "Using TopLink Workbench" on page 24-2) or Java code (see "Using Java" on page 24-2). Oracle recommends that you use TopLink Workbench to create and manage your relational descriptors.

For more information, see "Relational Descriptors" on page 23-11.

Using TopLink Workbench

Using TopLink Workbench, you can create the following types of descriptor in a relational project:

- Relational Class Descriptors
- Relational Aggregate Descriptors
- Relational Interface Descriptors

Relational Class Descriptors



By default, when you add a Java class to a relational project (see "Configuring Project Classpath" on page 19-3), TopLink Workbench creates a relational class descriptor for it. A class descriptor is applicable to any persistent object except an object that is owned by another in an aggregate relationship. In this case, you must describe the owned object with an aggregate descriptor (see "Relational Aggregate Descriptors" on page 24-2). Using a class descriptor, you can configure any relational mapping except aggregate collection and aggregate object mappings.

Relational Aggregate Descriptors



An aggregate object is an object that is strictly dependent on its owning object. Aggregate descriptors do not define a table, primary key, or many of the standard descriptor options as they obtain these from their owning descriptor. If you want to configure an aggregate mapping to associate data members in a target object with fields in a source object's underlying database tables (see Chapter 41, "Configuring a Relational Aggregate Collection Mapping" and Chapter 43, "Configuring a Relational Aggregate Object Mapping"), you must designate the target object's descriptor as an aggregate (see "Configuring a Relational Descriptor as a Class or Aggregate Type" on page 26-11).

Relational Interface Descriptors



If you add an interface to a relational project (see "Configuring Project Classpath" on page 19-3), TopLink Workbench creates an interface descriptor for it.

An interface is a collection of abstract behavior that other classes can use. It is a purely Java concept and has no representation on the relational database. Therefore, a descriptor defined for the interfaces does not map any relational entities on the database.

The interface descriptor includes the following elements:

- The Java interface it describes.
- The parent interface(s) it implements.
- A list of abstract query keys.

An interface descriptor does not define any mappings, because there is no concrete data or table associated with it. A list of abstract query keys is defined so that you can issue queries on the interfaces (see "Configuring Interface Query Keys" on page 25-33). A read query on the interface results in reading one or more of its implementors.

Using Java

Example 24–1 shows how to create a relational descriptor using Java code.

Example 24–1 Creating a Relational Descriptor in Java

```
RelationalDescriptor descriptor = new RelationalDescriptor();
descriptor.setJavaClass(YourClass.class);
```

To designate a relational descriptor as an aggregate, use `ClassDescriptor` method `descriptorIsAggregate`. For a `RelationalDescriptor` configured as an aggregate, you do not define a primary key, but when using Java, you must configure the associated table (see "Configuring Associated Tables" on page 26-2) and field mappings (see "Understanding Mappings" on page 30-1).

To allow a relational descriptor to participate in an aggregate collection mapping (see "Aggregate Collection Mapping" on page 33-10), use `ClassDescriptor` method `descriptorIsAggregateCollection`. For a `RelationalDescriptor` configured for use with an aggregate collection mapping, you do define primary keys (see "Configuring Primary Keys" on page 25-3) and an associated table (see "Configuring Associated Tables" on page 26-2), but you do not have to map the primary keys if they are shared from their parent.

To configure a relational descriptor for an interface, use `ClassDescriptor` method `setJavaInterface`, passing in the `java.lang.Class` of the interface. You should only use an interface descriptor for an interface that has multiple implementors. If an interface has only a single implementor, then rather than creating an interface descriptor, just set the implementor descriptor's interface alias (see "Configuring Interface Alias" on page 26-10).

Creating an Object-Relational Descriptor

You cannot create an object-relational descriptor using TopLink Workbench: you must use Java code. For more information on creating descriptors in Java code, see the Oracle TopLink API Reference.

For more information, see "Object-Relational Descriptors" on page 23-11.

Using Java

Use the `ObjectRelationalDescriptor` class to define an object-relational descriptor. This class extends `RelationalDescriptor` to add the following methods:

- `setStructureName`: call this method to set the name of the object-relational structure that represents the object class in the data source.
- `addFieldOrdering`: call this method repeatedly to define the order in which object attributes are persisted to the data source. This defines a field index that TopLink uses if your object-relational data source driver uses JDBC indexed arrays.

Example 24–2 shows an `Employee` object that is mapped to an Oracle Database using its object-relational features.

Example 24–2 Employee Class

```
public class Employee {
    Long id;
    String firstName;
    String lastName;

    ...
}
```

Example 24–3 shows the object-relational database type (`Employee_t`) created to model the `Employee` object within the database. Such an object-relational database type is also known as a structure. This example also shows how to create and populate a database table (called `department`) that stores instances of the `Employee_t` audio tape.

Example 24–3 Employee Object-Relational Data Model

```
CREATE TYPE EMPLOYEE_T AS OBJECT(
    ID NUMBER(10),
    F_NAME VARCHAR2(100),
    L_NAME VARCHAR2(100),
) NOT FINAL;

CREATE TABLE EMPLOYEES OF TYPE EMPLOYEE_T;
```

Example 24–4 shows how to code an object-relational descriptor in Java to describe the object-relational database type `Employee_t`.

Example 24–4 Creating an Object-Relational Descriptor in Java

```
import oracle.toplink.objectrelational.*;

ObjectRelationalDescriptor descriptor = new ObjectRelationalDescriptor();
descriptor.setJavaClass(Employee.class);
descriptor.setTableName("EMPLOYEES");
descriptor.setStructureName("EMPLOYEE_T");
descriptor.setPrimaryKeyFieldName("ID");
descriptor.addFieldOrdering("ID");
descriptor.addFieldOrdering("F_NAME");
descriptor.addFieldOrdering("L_NAME");
descriptor.addDirectMapping("id", "OBJECT_ID");
descriptor.addDirectMapping("firstName", "F_NAME");
descriptor.addDirectMapping("lastName", "L_NAME");
```

For more information on configuring object-relational descriptors, see "Configuring an Object-Relational Descriptor" on page 27-1.

For more information on the object-relational mappings that TopLink supports, see Chapter 46, "Understanding Object-Relational Mappings".

Creating an EIS Descriptor

You can create an EIS descriptor using TopLink Workbench (see "Using TopLink Workbench" on page 24-4) or Java code (see "Using Java" on page 24-5). Oracle recommends that you use TopLink Workbench to create and manage your EIS descriptors.

For more information, see "EIS Descriptors" on page 23-12.

Using TopLink Workbench

Using TopLink Workbench, you can create the following types of EIS descriptor in an EIS project:

- EIS Root Descriptors
- EIS Composite Descriptors



EIS Root Descriptors

You can modify an EIS descriptor's behavior by configuring it as a root EIS descriptor (see "Configuring an EIS Descriptor as a Root or Composite Type" on page 28-8). When you designate an EIS descriptor as a root, you tell the TopLink runtime that the EIS descriptor's reference class is a parent class: no other class will reference it by way of a composite object mapping or composite collection mapping. Using an EIS root descriptor, you can configure all supported mappings. You can also configure an EIS root descriptor with EIS interactions (see "Using EIS Interactions" on page 94-24). However, if you configure the EIS root descriptor with a composite object mapping or composite collection mapping, the reference descriptor you define must be an EIS composite descriptor; it cannot be another EIS root descriptor.



EIS Composite Descriptors

By default, when you add a class to an EIS project (see "Configuring Project Classpath" on page 19-3), TopLink Workbench creates an EIS descriptor for the class, and designates the EIS descriptor as a composite. When you designate an EIS descriptor as a composite, you tell the TopLink runtime that the EIS descriptor's reference class may be referenced by a composite object mapping or composite collection mapping. Using an EIS composite descriptor, you can configure all supported mappings. However, you cannot configure an EIS composite descriptor with EIS interactions: for this, you need an EIS root descriptor (see "EIS Root Descriptors" on page 24-5).

Using Java

Example 24–5 shows how to create a relational descriptor using Java code.

Example 24–5 Creating an EIS Descriptor in Java

```
EISDescriptor descriptor = new EISDescriptor();
descriptor.setJavaClass(YourClass.class);
```

To designate an EIS descriptor as a composite, use `ClassDescriptor` method `descriptorIsAggregate`.

Creating an XML Descriptor

You can create an XML descriptor using TopLink Workbench (see "Using TopLink Workbench" on page 24-5) or Java code (see "Using Java" on page 24-5). Oracle recommends that you use TopLink Workbench to create and manage your XML descriptors.

For more information, see "XML Descriptors" on page 23-12.

Using TopLink Workbench



When you add a class to an XML project (see "Configuring Project Classpath" on page 19-3), TopLink Workbench creates an XML descriptor for the class.

An XML descriptor is always a composite type.

Using Java

Example 24–6 shows how to create a relational descriptor using Java code.

Example 24–6 Creating an XML Descriptor in Java

```
XMLDescriptor descriptor = new XMLDescriptor();
descriptor.setJavaClass(YourClass.class);
```

Note: Use the `oracle.toplink.ox.XMLDescriptor` class. Do not use the deprecated `oracle.toplink.xml.XMLDescriptor` class.

Validating Descriptors

You can validate descriptors in the following ways:

- Run the project in a test environment and watch for and interpret any exceptions that occur. For more information, see Chapter 13, "TopLink Exception Reference".
- Run the TopLink integrity checker. For more information, see "Integrity Checker" on page 72-12.
- Review the project status report. For more information, see "Generating the Project Status Report" on page 18-12.

Generating Java Code for Descriptors

Typically, you capture descriptor configuration in the `project.xml` file and the TopLink runtime reads this information, and then creates and configures all necessary descriptor objects.

Alternatively, for relational projects only, you can export a TopLink project as a Java class (`oracle.toplink.sessions.Project`) that contains all descriptor configuration in Java. This lets you use TopLink Workbench to quickly create and configure descriptors, and then, manually code features that TopLink Workbench does not support. This gives you the best of both TopLink Workbench and Java access to your descriptors. After configuring your Java project class, compile it and include it in your application's JAR file.

For more information, see "Exporting Project Java Source" on page 18-14.

25

Configuring a Descriptor

This chapter describes how to configure TopLink descriptors.

Table 25–1 lists the types of TopLink descriptors that you can configure and provides a cross-reference to the type-specific chapter that lists the configurable options supported by that type.

Table 25–1 Configuring TopLink Descriptor

If you are creating...	See...
Relational Descriptors	Chapter 26, "Configuring a Relational Descriptor"
Object-Relational Descriptors	Chapter 27, "Configuring an Object-Relational Descriptor"
EIS Descriptors	Chapter 28, "Configuring an EIS Descriptor"
XML Descriptors	Chapter 29, "Configuring an XML Descriptor"

Table 25–2 lists the configurable options shared by two or more TopLink descriptor types.

For more information, see the following:

- "Descriptor Creation Overview" on page 24-1
- "Understanding Descriptors" on page 23-1

Configuring Common Descriptor Options

Table 25–2 lists the configurable options shared by two or more TopLink descriptor types. In addition to the configurable options described here, you must also configure the options described for the specific Descriptor Types, as shown in Table 25–1.

Table 25–2 Common Descriptor Options

Option	Type	TopLink Workbench	Java
"Configuring Primary Keys" on page 25-3	Basic	✔	✔
"Configuring Read-Only Descriptors" on page 25-5	Basic	✔	✔
"Configuring Unit of Work Conforming at the Descriptor Level" on page 25-6	Advanced	✔	✔
"Configuring Descriptor Alias" on page 25-7	Advanced	✔	✔
"Configuring Descriptor Comments" on page 25-79	Advanced	✔	
"Configuring Classes" on page 4-41	Basic	✔	

Table 25–2 (Cont.) Common Descriptor Options

Option	Type	TopLink Workbench	Java
"Configuring Named Queries at the Descriptor Level" on page 25-10	Advanced	✔	✔
"Configuring Query Timeout at the Descriptor Level" on page 25-26	Advanced	✔	✔
"Configuring Cache Refreshing" on page 25-27	Advanced	✔	✔
"Configuring Query Keys" on page 25-30	Advanced	✔	✔
"Configuring Interface Query Keys" on page 25-33	Advanced	✔	✔
"Configuring Cache Type and Size at the Descriptor Level" on page 25-35	Advanced	✔	✔
"Configuring Cache Isolation at the Descriptor Level" on page 25-37	Advanced	✔	✔
"Configuring Unit of Work Cache Isolation at the Descriptor Level" on page 25-38	Advanced		✔
"Configuring Cache Coordination Change Propagation at the Descriptor Level" on page 25-40	Advanced	✔	✔
"Configuring Cache Expiration at the Descriptor Level" on page 25-42	Advanced	✔	✔
"Configuring Cache Existence Checking at the Descriptor Level" on page 25-43	Advanced	✔	✔
"Configuring a Descriptor With EJB Information" on page 25-45	Advanced	✔	✔
"Configuring Reading Subclasses on Queries" on page 25-49	Advanced	✔	✔
"Configuring Inheritance for a Child (Branch or Leaf) Class Descriptor" on page 25-51	Advanced	✔	✔
"Configuring Inheritance for a Parent (Root) Descriptor" on page 25-52	Advanced	✔	✔
"Configuring Inheritance Expressions for a Parent (Root) Class Descriptor" on page 25-55	Advanced	✔	✔
"Configuring Inherited Attribute Mapping in a Subclass" on page 25-58	Advanced	✔	✔
"Configuring a Domain Object Method as an Event Handler" on page 25-59	Advanced	✔	✔
"Configuring a Descriptor Event Listener as an Event Handler" on page 25-62	Advanced	✔	✔
"Configuring Locking Policy" on page 25-64	Advanced	✔	✔
"Configuring Returning Policy" on page 25-67	Advanced	✔	✔
"Configuring Instantiation Policy" on page 25-70	Advanced	✔	✔
"Configuring Copy Policy" on page 25-71	Advanced	✔	✔
"Configuring Change Policy" on page 25-73	Advanced		✔
"Configuring a History Policy" on page 25-76	Advanced		✔
"Configuring Wrapper Policy" on page 25-78	Advanced		✔
"Configuring Fetch Groups" on page 25-79	Advanced		✔
"Configuring Amendment Methods" on page 25-81	Advanced	✔	
"Configuring a Mapping" on page 32-1	Basic	✔	✔

Configuring Primary Keys

A **primary key** is a unique identifier (made up of one or more persistent attributes) that distinguishes one instance of a class from all other instances of the same type. You use primary keys to define relationships and to define queries.

For the descriptors shown in Table 25–3, you must configure a primary key and you must ensure that your class contains one or more persistent fields suitable for this purpose.

Table 25–3 summarizes which descriptors support primary keys.

Table 25–3 Descriptor Support for Primary Keys

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors[1]	✔	✔
Object-Relational Descriptors		✔
EIS Descriptors[2]	✔	✔
XML Descriptors		

[1] Relational class descriptors only (see "Relational Class Descriptors" on page 24-2).

[2] EIS root descriptors only (see "EIS Root Descriptors" on page 24-5).

For a relational class (non-aggregate) descriptor, choose any unique database field or set of unique database fields from the descriptor's associated table (see "Configuring Associated Tables" on page 26-2).

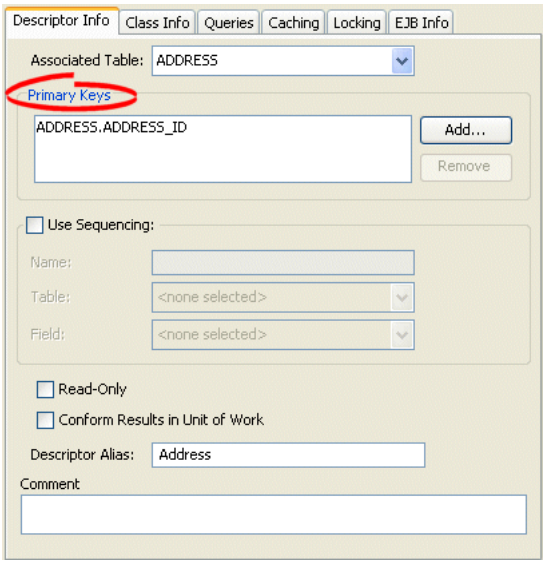
For an EIS root descriptor (see "Configuring an EIS Descriptor as a Root or Composite Type" on page 28-8), choose any unique attribute or text node or set of unique attributes or text nodes from the descriptor's schema context (see "Configuring Schema Context for an EIS Descriptor" on page 28-2).

Using TopLink Workbench

To associate a descriptor with one or more primary keys, use this procedure:

1. Select a descriptor in the **Navigator**. Its properties appear in the Editor.
2. Click the **Descriptor Info** tab. The Descriptor Info tab appears.

Figure 25–1 Descriptor Info Tab, Primary Key Options



Use this table to enter data in Primary Keys field on the descriptor's **Descriptor Info** tab to specify the primary key(s):

Field	Description
Primary Keys	To specify the primary keys for the table, click **Add** in order to do the following: ■ For a relational class descriptor, select a database field from the descriptor's associated table (see "Configuring Associated Tables" on page 26-2). ■ For an EIS root descriptor, select an attribute or text node from the descriptor's schema context (see "Configuring Schema Context for an EIS Descriptor" on page 28-2). For more information on choosing an element or attribute, see "Choosing a Schema Context" on page 28-3. To remove a primary key, select the key and click **Remove**.

Using Java

You can use Java to configure primary keys for:

- Relational Projects
- EIS Projects

Relational Projects

Use `ClassDescriptor` method `addPrimaryKeyFieldName` to specify the primary key field of the descriptor's table. This should be called for each field that makes up the primary key of the table.

If the descriptor has more than one table, and all other tables have the same primary key, use the `ClassDescriptor` method `addPrimaryKeyFieldName` to specify the the primary key in the first table.

If the descriptor has more than one table, and each table has a different primary key, use `ClassDescriptor` method `addMultipleTablePrimaryKeyFieldName` to map the primary key field names in each table.

EIS Projects

Use `EISDescriptor` method `addPrimaryKeyFieldName` to specify the primary key field of the descriptor's class. Call this method for each field that makes up the primary key.

Configuring Read-Only Descriptors

You can configure a relational class or EIS root descriptor as read-only. This indicates that instances of the reference class will never be modified.

Read-only descriptors are usually used within a unit of work as a performance gain, because there is no need to register, clone, and merge the read-only classes. For more information, see Chapter 97, "Understanding TopLink Transactions".

In a CMP project, you can declare an entity bean as read-only within the TopLink deployment XML file. For more information, see "Using Read-Only Entity Beans" on page 25-5.

Table 25–4 summarizes which descriptors support read-only configuration.

Table 25–4 Descriptor Support for Read Only

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors[1]	✔	✔
Object-Relational Descriptors		✔
EIS Descriptor[2]	✔	✔
XML Descriptors		

[1] Relational class descriptors only (see "Relational Class Descriptors" on page 24-2).

[2] EIS root descriptors only (see "EIS Root Descriptors" on page 24-5)

Note: Relational aggregate and EIS composite descriptors get their read-only setting from their owner.

Using Read-Only Entity Beans

TopLink can declare an entity bean with container-managed persistence as read-only. This ensures that the entity bean cannot be modified and allows TopLink to optimize unit of work performance.

If an attempt is made to modify a read-only entity bean (create, update, or remove), TopLink immediately throws a `javax.ejb.EJBException`: TopLink does not wait until the transaction commits.

If an attempt is made to change a CMR field on a read-only entity bean, TopLink throws a `javax.ejb.EJBException`.

When TopLink is configured as the OC4J persistence manager, the TopLink read-only bean configuration replaces the OC4J `READ-ONLY` CMP concurrency mode.

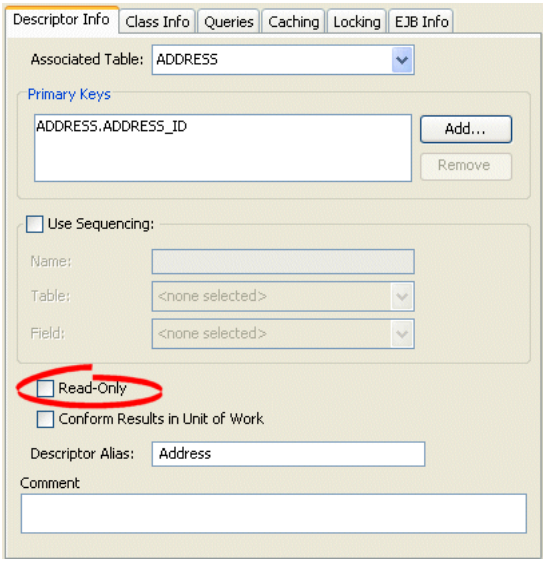
Using TopLink Workbench

To configure a descriptor as read-only use this procedure:

1. Select a descriptor in the **Navigator**. Its properties appear in the Editor.

2. Click the **Descriptor Info** tab. The Descriptor Info tab appears.

Figure 25–2 Descriptor Info Tab, Read Only Option



Specify whether this descriptor is read-only or not.

Using Java

Use `ClassDescriptor` method `setReadOnly`.

Configuring Unit of Work Conforming at the Descriptor Level

Conforming is a query feature that lets you include new, changed, or deleted objects in queries within a unit of work prior to committing the transaction. This feature enables you to query against your relative logical or transaction view of a data source.

Table 25–5 summarizes which descriptors support descriptor level unit of work conforming.

Table 25–5 Descriptor Support for Unit of Work Conforming

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors[1]	✔	✔
Object-Relational Descriptors		✔
EIS Descriptors[2]	✔	✔
XML Descriptors		

[1] Relational class descriptors only (see "Relational Class Descriptors" on page 24-2).

[2] EIS root descriptors only (see "EIS Root Descriptors" on page 24-5)

When you configure a descriptor to conform results in a unit of work, when you execute a query in the unit of work, TopLink filters the data source result set to the changes currently made in the unit of work. TopLink adds new or changed objects that correspond to the query's selection criteria and removes changed objects that no longer correspond to the query's selection criteria.

Note: For EIS root descriptors, only deleted objects would be filtered, not new or changed objects.

Conforming can reduce performance. Before you enable a descriptor for conforming, be aware of its limitations (see "Guidelines for Using Conforming" on page 99-8) and make sure that conforming is actually necessary.

For examples, see the following:

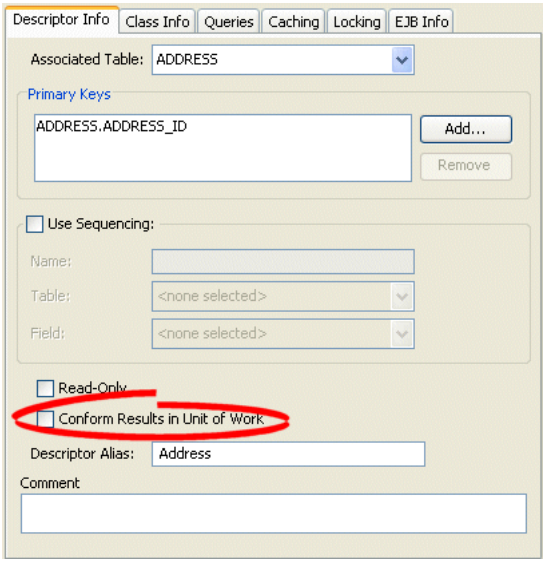
- "Using Conforming Queries and Descriptors" on page 99-8
- "Conforming Query Alternatives" on page 99-12

Using TopLink Workbench

To conform a descriptor's results in a unit of work, use this procedure:

1. Select a descriptor in the **Navigator**. Its properties appear in the Editor.
2. Click the **Descriptor Info** tab. The Descriptor Info tab appears.

Figure 25–3 Descriptor Info Tab, Conform Results in Unit of Work Option



Enable or disable conforming: when enabled, this feature ensures that any queries for this descriptor will conform the data source result with the current changes in the unit of work. For more information, see "Guidelines for Using Conforming" on page 99-8.

Using Java

Use `ClassDescriptor` method `setShouldAlwaysConformResultsInUnitOfWork(true)`.

Configuring Descriptor Alias

Use the descriptor alias to specify the value of the `ejb-jar.xml` attribute `abstract-schema-name`. This is the logical name that is referenced in EJB QL

queries. You should configure a descriptor alias for each entity bean with container-managed persistence. The descriptor alias defaults to the class name.

Descriptor alias only applies in projects where you configure the persistence type (see "Configuring Persistence Type" on page 19-5) to use EJB.

Table 25–6 summarizes which descriptors support descriptor alias configuration.

Table 25–6 Descriptor Support for Descriptor Alias Configuration

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors	✔	
Object-Relational Descriptors		
EIS Descriptors[1]	✔	
XML Descriptors		

[1] EIS root descriptors only (see "EIS Root Descriptors" on page 24-5).

For more information, see the following:

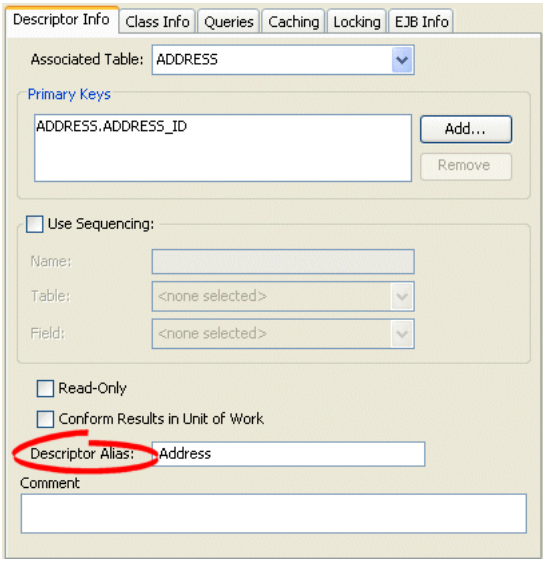
- "ejb-jar.xml File" on page 8-5
- "EJB Finders" on page 93-24

Using TopLink Workbench

To specify a descriptor alias, use this procedure:

1. In the **Navigator**, select a descriptor.
2. Click the **Descriptor Info** tab in the Property window.

Figure 25–4 Descriptor Info Tab, Descriptor Alias Field



In the **Descriptor Alias** field, enter an alias for this descriptor. This is the value of the `ejb-jar.xml` attribute `abstract-schema-name`. It is the logical name that is referenced in EJB QL queries. The default is the class name.

Using Java

Use `ClassDescriptor` method `setAlias` passing in the descriptor alias as a `String`.

Configuring Descriptor Comments

You can define a free-form textual comment for each descriptor. You can use these comments however you whish: for example, to record important project implementation details such as the purpose or importance of a descriptor.

Comments are stored in the TopLink Workbench project, in the TopLink deployment XML file. There is no Java API for this feature.

Table 25–7 summarizes which descriptors support descriptor comment configuration.

Table 25–7 Descriptor Support for Descriptor Comment Configuration

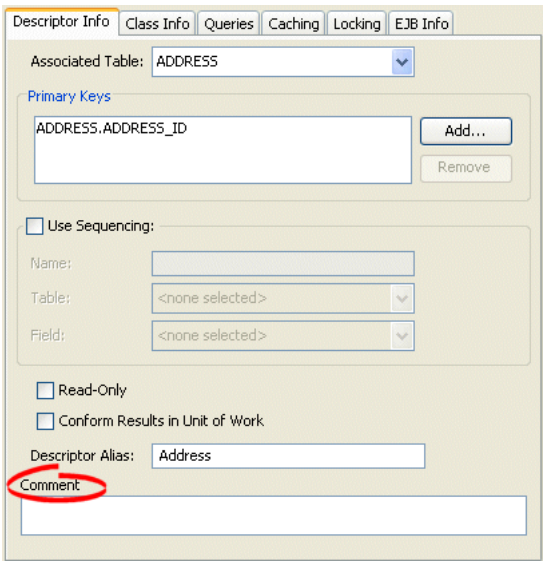
Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors	✔	
Object-Relational Descriptors		
EIS Descriptors	✔	
XML Descriptors	✔	

Using TopLink Workbench

To create a comment for a descriptor, use this procedure:

1. In the **Navigator**, select a descriptor.
2. Click the **Descriptor Info** tab in the Property window.

Figure 25–5 Descriptor Info Tab, Comment Field



In the **Comment** field, enter a description of this descriptor.

Configuring Named Queries at the Descriptor Level

A named query is a TopLink query that you create and store, by name, in a descriptor's `DescriptorQueryManager` for later retrieval and execution. Named queries improve application performance because they are prepared once and they (and all their associated supporting objects) can be efficiently reused thereafter making them well suited for frequently executed operations.

If a named query is global to a `Class`, configure it at the descriptor level. Alternatively, you can configure a named query at the session level (see "Configuring Named Queries at the Session Level" on page 74-21).

Use named queries to specify SQL, EJB QL, or TopLink `Expression` queries to access your data source.

For entity bean descriptors, you must define a named query for every finder defined for the corresponding entity bean.

Using TopLink Workbench, you can configure named queries for a subset of query types and store them in a descriptor's `DescriptorQueryManager` (see "Using TopLink Workbench" on page 25-10).

Using Java, you can create named queries for all query types and store them in a descriptor's `DescriptorQueryManager` (see "Using Java" on page 25-25).

Table 25–4 summarizes which descriptors support named query configuration.

Table 25–8 Descriptor Support for Named Queries

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors[1]	✔	✔
Object-Relational Descriptors		✔
EIS Descriptor[2]	✔	✔
XML Descriptors		

[1] Relational class descriptors only (see "Relational Class Descriptors" on page 24-2).

[2] EIS root descriptors only (see "EIS Root Descriptors" on page 24-5).

For CMP projects, the `ejb-jar.xml` file stores query lists. You can define the queries in the file, and then read them into TopLink Workbench, or define them on the **Queries** tab and write them to the file. See "Working With the ejb-jar.xml File" on page 18-15 for more information.

After you create a named query, you can execute it by name and class on the TopLink session (see "Using Named Queries" on page 94-18).

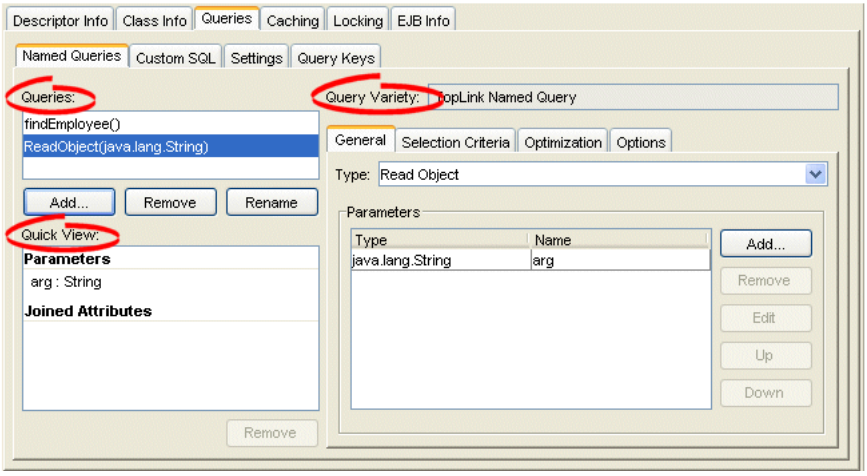
For more information about named queries, see "Named Queries" on page 93-16.

Using TopLink Workbench

To create a named query, use this procedure

1. In the **Navigator**, select a descriptor. Its properties appear in the Editor.
2. Click the **Queries** tab in the **Editor**. The Queries tab appear with three additional tabs.
3. Click the **Named Queries** tab in the Queries tab. The Named Queries tab appears.

Figure 25–6 Queries Tab–Named Queries Subtab



Use the following information to complete each field on this tab:

Field	Description
Queries	Lists the existing queries for this descriptor. ▪ To create a new query, click **Add** (see "Adding Named Queries" on page 25-12). ▪ To delete an existing query, select the query and click **Delete**. TopLink Workbench prompts for confirmation. ▪ To rename an existing query, select the query and click **Rename**. The Rename dialog box appears. Type a new name for the query and click **OK**.
Query Variety	Displays the variety of the currently selected query (see "Adding Named Queries" on page 25-12).
Quick View	Lists the parameters and joined attributes defined for the query. Clicking on a heading in the **Quick View** area selects the corresponding subtab. You can also remove parameters or attributes from the Quick View area by selecting the item and clicking **Remove**.

The Named Queries tab includes the following subtabs:

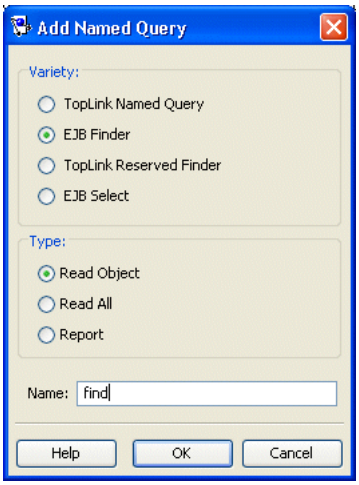
- **General**–See "Configuring Named Query Type and Parameters" on page 25-13.
- **Selection Criteria**–See "Configuring Named Query Selection Criteria" on page 25-14.
- **Order**–This tab appears for `ReadAllQuery` queries only. See "Configuring Read All Query Order" on page 25-15.
- **Optimization**–See "Configuring Named Query Optimization" on page 25-16.
- **Attributes**–This tab appears for `ReportQuery` queries only. See "Configuring Named Query Attributes" on page 25-17.
- **Group/Order**–This tab appears for `ReportQuery` queries only. "Configuring Named Query Group/Order Options" on page 25-19.
- **Calls**–This tab appears for EIS root descriptors only (for `ReadAllQuery` and `ReadObjectQuery` queries). See "Creating an EIS Interaction for a Named Query" on page 25-20.

- **Options**–See "Configuring Named Query Options" on page 25-22.

Adding Named Queries

Use this dialog box to create a new named query.

Figure 25–7 Add Named Query Dialog Box



Use the following information to complete the dialog box and create the named query:

Field	Description
Variety	For EJB descriptors only, select the variety of query:
TopLink Named Query	Select to create a general purpose TopLink query of the type given by the **Type** area. You can execute this query by name on the TopLink session passing in the class and arguments (see "Using Named Queries" on page 94-18).
EJB Finder	Select to create a TopLink query of the type given by the **Type** area for use as the implementation of the EJB finder method of the name you specify. The TopLink runtime executes this query when you call the EJB finder method of the given name.
TopLink Reserved Finder	Select to create a TopLink query of the type given by the **Type** area for use as the implementation of the TopLink reserved finder method name you select. The TopLink runtime executes this query when you call the EJB finder method of the given name. For more information, see "Predefined Finders" on page 93-24.
EJB Select	Select to create a TopLink query of the type given by the **Type** area for use as the implementation of the EJB life cycle method `ejbSelect`. The TopLink runtime executes this query whenever the `ejbSelect` method is called.

Field	Description
Type	Select the type of query: ▪ **ReadObject** (`ReadObjectQuery`) ▪ **ReadAll** (`ReadAllQuery`) ▪ **Report**[1] (`ReportQuery`) **Note**: If **Variety** is set to **TopLink Reserved Finder**, you cannot select a query **Type** .
Name	The name of this query. ▪ If **Variety** is set to **TopLink Named Query**, you can specify any name. ▪ If **Variety** is set to **EJB Finder**, the name must be prefixed by **find**. ▪ If **Variety** is set to **TopLink Reserved Finder**, select from the list of available names that TopLink reserves. For more information, see "Predefined Finders" on page 93-24. ▪ If **Variety** is set to **EJB Select**, the name must be **ejbSelect**.

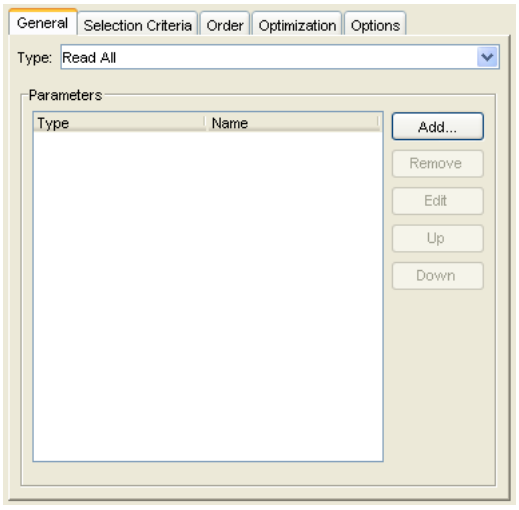
[1] Relational descriptors only.

Enter the necessary information and click **OK**. TopLink Workbench adds the query to the list of queries in the Named Query tab.

Configuring Named Query Type and Parameters

Use this tab to select the query type and add or remove parameters.

Figure 25–8 Named Queries, General Tab



Use the following information to complete each field on this tab:

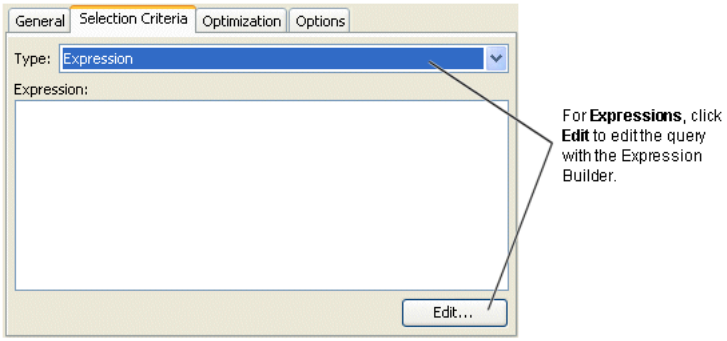
Field	Description
Type	Select the type of query from the list. You can create any of the following query types: ▪ **ReadAllQuery** ▪ **ReadObjectQuery** ▪ **ReportQuery**[1] To create other types of query, you must use Java (see "Using Java" on page 25-25). When you change the type of an existing query, TopLink Workbench preserves any configuration that is common between the old and new type and warns you if changing the type will result in the loss of configuration that is not shared by the new type.
Parameters	For queries that take parameters, specify the parameters: ▪ To add a new parameter, click **Add**. The Add Query Parameter dialog box appears. Click **Browse** to select the type, specify a name, and click **OK**. ▪ To delete an existing parameter, select the parameter and click **Remove**. TopLink Workbench prompts for confirmation. ▪ To modify an existing parameter, select the parameter and click **Edit**. The Edit Query Parameter dialog box appears. Modify the name and type of the parameter and click **OK**. ▪ To change the order of the parameters, select an existing parameter and click **Up** or **Down**.
Type	Select the class of the parameter's type.
Name	Enter the name of the parameter.

[1] Relational descriptors only.

Configuring Named Query Selection Criteria

Use this tab to specify the format of the named query and enter the query string.

Figure 25–9 Named Queries, Selection Criteria Tab



Use the following information to complete each field on this tab:

Field	Description
Type	Specify if query uses a TopLink **Expression**, **SQL**, or **EJB QL**.
Expression or **Query String**	If the **Type** is **SQL** or **EJB QL**, enter the query string (either SQL or EJB QL). TopLink Workbench does not validate the query string. See a note that follows this table for information on query syntax.

Note: Use a combination of an escape character and a double-quote (\") instead of just a double-quote (") when defining your query using SQL or EJB QL. For example:

```
SELECT OBJECT(employee) FROM Employee employee WHERE
employee.name = \"Bob\"
```

If you fail to do so, the generated Java code would look as follows:

```
query.setEJBQLString("SELECT OBJECT(employee) FROM Employee
employee WHERE employee.name = "Bob"");
```

The preceding code produces an error at compile time.

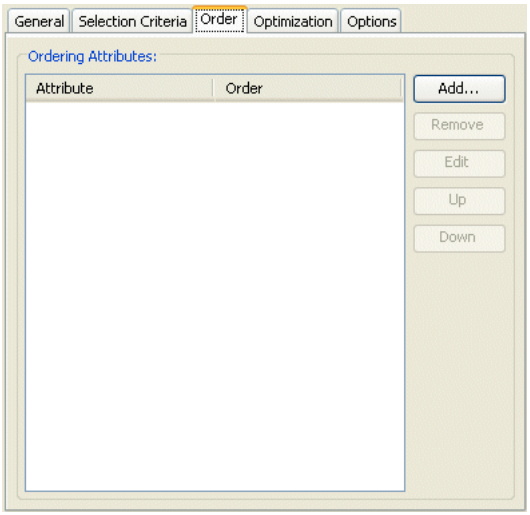
If you define your query using the correct syntax, the generated Java code will contain no errors and be similar to the following:

```
query.setEJBQLString("SELECT OBJECT(employee) FROM Employee
employee WHERE employee.name = \"Bob\"");
```

Configuring Read All Query Order

Use this tab to specify how the results of a read all query should be ordered by attribute name.

Figure 25–10 Named Queries, Order Tab



Select one of the following actions:

- To add a new attribute by which to order query results, click **Add**. The Add Ordering Attribute dialog box appears. Select the mapped attribute to order by, specify ascending or descending order, and then click **OK**.
- To change the order of the order attributes, select an existing attribute and click **Up** or **Down**.
- To modify an existing order attribute's ordering options, select an existing attribute and click **Edit**.
- To remove an order attribute, select an existing attribute and click **Remove**.

Configuring Named Query Optimization

You can optimize a named query by configuring batch (`ReadAllQuery` only) or joining (`ReadAllQuery` and `ReadObjectyQuery`) attributes.

For more information on using batch reading, see Query Optimization on page 11-16, "Reading Case 2: Batch Reading Objects" on page 11-23, and "Using Batch Reading" on page 94-10.

For more information on joining, see "Join Reading and Object-Level Read Queries" on page 93-12 and "Using Join Reading" on page 94-11.

Use this tab to specify batch reading and joining attributes.

Figure 25–11 Named Queries, Optimization Tab



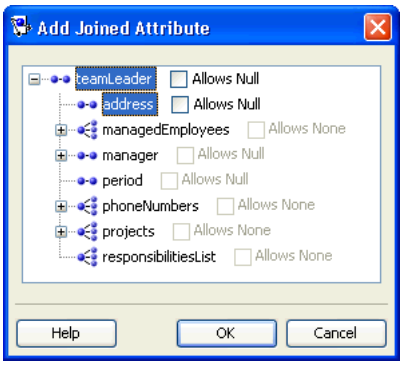
Select one of the following actions for **Batch Read Attributes** (`ReadAllQuery` only):

- To add a new batch read attribute, click **Add**. The Add Batch Read Attribute dialog box appears. Select the mapped attribute and click **OK**.
- To change the order of the batch read attributes, select an existing attribute and click **Up** or **Down**.
- To modify an existing batch read attribute's options, select an existing attribute and click **Edit**.
- To remove a batch read attribute, select an existing attribute and click **Remove**.

Select one of the following actions for **Joined Attributes** (`ReadAllQuery` and `ReadObjectQuery`):

- To add a new joined attribute, click **Add**. The Add Joined Attribute dialog box appears.

Figure 25–12 Add Joined Attribute Dialog Box



Select the mapped attribute. Optionally, enable or disable **Allows Null** or, for a `Collection` attribute, **Allows None**. Click **OK**.

- To change the order of the joined attributes, select an existing attribute and click **Up** or **Down**.
- To modify an existing joined attribute's options, select an existing attribute and click **Edit**.
- To remove a joined attribute, select an existing attribute and click **Remove**.

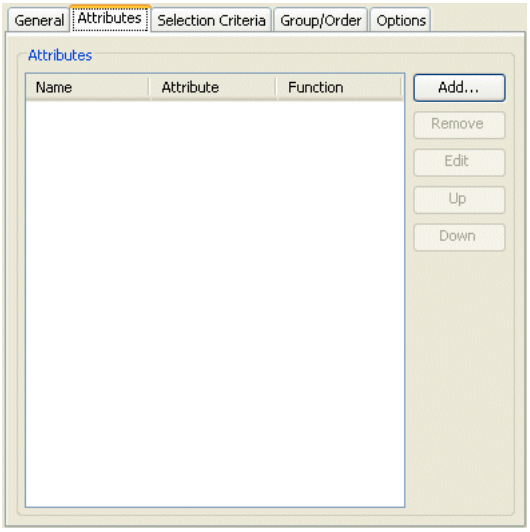
Configuring Named Query Attributes

For `ReportQuery` queries only, you can configure report query functions to apply to one or more attributes.

For more information about report queries, see "Report Query" on page 93-15.

Use this tab to configure report query attributes.

Figure 25–13 Named Queries, Attributes Tab



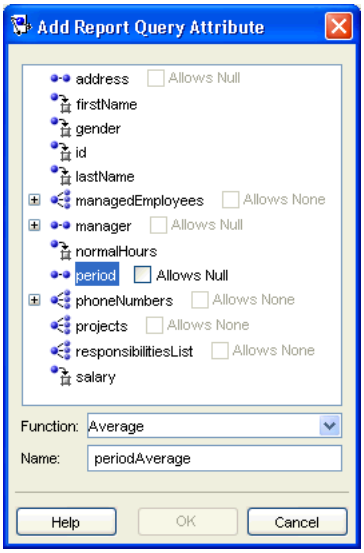
Select one of the following actions for **Attributes** (`ReportQuery` only):

- To add a new report query attribute, click **Add**. The Add Joined Attribute dialog box appears. Continue with "Adding Report Query Attributes" on page 25-18.
- To change the order of the report query attribute attributes, select an existing attribute and click **Up** or **Down**.
- To modify an existing report query attribute's options, select an existing attribute and click **Edit**.
- To remove a report query attribute, select an existing attribute and click **Remove**.

> **Note:** You can only choose attributes that are configured with a direct mapping (converters included) or a user-defined query key.

Adding Report Query Attributes Use this dialog box to add a report query attribute.

Figure 25–14 Add Report Query Attribute Dialog Box



Select the attribute you want in this report query and use the following table to complete the dialog box and add the report query attribute:

Option	'Description
Allows None or **Allows Null**	Use the **Allows Null** and **Allows None** options to define an expression with an outer join. Check the **Allows Null** option to use the `ExpressionBuilder` method `getAllowingNull`. Check the **Allows None** option for `Collection` attributes to use the `ExpressionBuilder` method `anyOfAllowingNone`. For more information, see "Using TopLink Expression API For Joins" on page 95-7.

Option	‘Description
Function	Select from the list of report query functions that TopLink provides. This function will be applied to the specified attribute. You must select an attribute for all functions, except **Count**. Alternatively, you can enter the name of a custom function that you implement in your database. For more information, see `Expression` method `getFunction` in the API reference.
Name	The name associated with the calculated value. By default, the name is `<AttributeName><FunctionName>`.

Enter the necessary information and click **OK**. TopLink Workbench adds the report query attribute to the list of attributes in the Attribute tab.

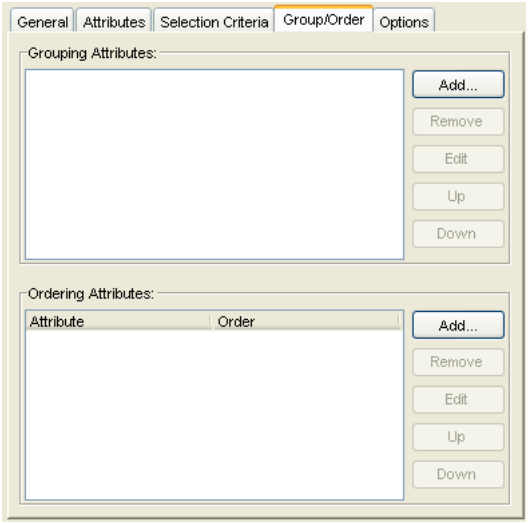
Configuring Named Query Group/Order Options

For `ReportQuery` queries only, you can configure grouping and ordering attributes.

For more information about report queries, see "Report Query" on page 93-15.

Use this tab to specify grouping and ordering attributes.

Figure 25–15 Named Queries, Group/Order Tab



Select one of the following actions for **Grouping Attributes** (`ReportQuery` only):

- To add a new grouping attribute, click **Add**. The Add Grouping Attribute dialog appears. Select the desired mapped attribute and click **OK**.
- To change the order of the grouping attributes, select an existing attribute and click **Up** or **Down**.
- To modify an existing grouping attribute’s options, select an existing attribute and click **Edit**.
- To remove a grouping attribute, select an existing attribute and click **Remove**.

Select one of the four following actions for **Ordering Attributes** (`ReportQuery` only):

- To add a new ordering attribute, click **Add**. The Add Ordering Attribute dialog box appears. Continue with "Adding Ordering Attributes" on page 25-20.
- To change the order of the ordering attributes, select an existing attribute and click **Up** or **Down**.
- To modify an existing ordering attribute's options, select an existing attribute and click **Edit**.
- To remove an ordering attribute, select an existing attribute and click **Remove**.

Adding Ordering Attributes Use this dialog box to add a report query ordering attribute.

Figure 25–16 Add Ordering Attribute Dialog Box



Use the following information to complete the fields on the dialog box and add an ordering attribute:

Option	'Description
Selected Attribute	Select this option to view a list of the report query attributes you added (see "Configuring Named Query Attributes" on page 25-17). Select an attribute and choose its ordering option in the **Order** field.
New Attribute	Select this option to view a list of all class attributes. Select an attribute and choose its ordering option in the **Order** field.
Order	Select ascending or descending.

Enter the necessary information and click **OK**. TopLink Workbench adds the report query attribute to the list of attributes in the Attribute tab.

Creating an EIS Interaction for a Named Query

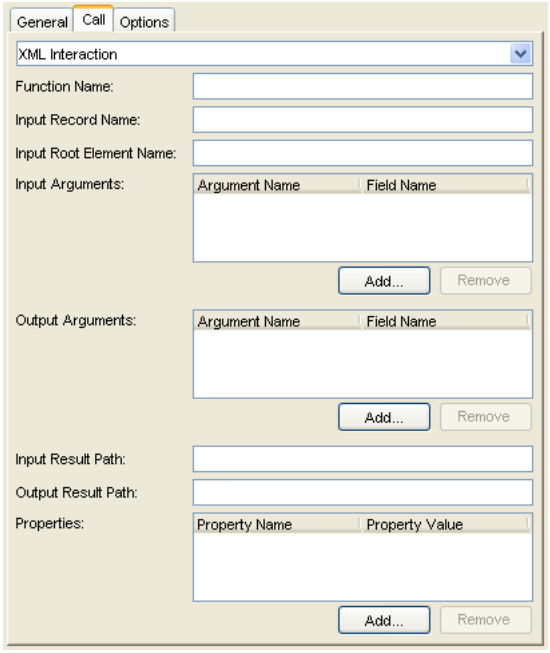
For an EIS root descriptor, you can define EIS interactions to invoke methods on an EIS.

You can use TopLink to define an interaction as a named query for read object and read all object queries, as described here. These queries are not called for basic persistence operations ("Configuring Custom EIS Interactions for Basic Persistence Operations" on page 28-6); you can call these additional queries by name in your application for special purposes.

Call Tab

Use this tab to define an interaction as a named query for read object and read all object queries.

Figure 25–17 Call Tab



Use the following information to complete each field on the tab:

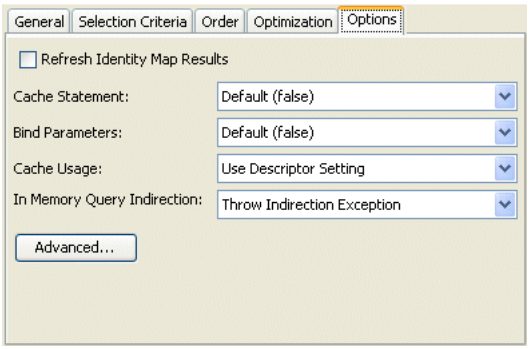
Field	Description
Interaction Type	Using TopLink Workbench, you can only use XML Interactions. You cannot change this field.
Function Name	Specify the name of the EIS function that this call type (Read Object or Read All) invokes on the EIS.
Input Record Name	Specify the name passed to the J2C adapter when creating the input record.
Input Root Element	Specify the root element name to use for the input DOM.
Input Arguments	Specify the query argument name to map to the interaction field or XPath nodes in the argument record. For example, if you are using XML records, use this option to map input argument `name` to the XPath `name/first-name`.
Output Arguments	Specify the result record field or XPath nodes to map the correct nodes in the record used by the descriptor's mappings. For example, if you are using XML records, use this option to map the output `fname` to `name/first-name`. Output arguments are not required if the interaction returns an XML result that matches the descriptor's mappings.
Input Result Path	Use this option if the EIS interaction expects the interaction arguments to be nested in the XML record. For example, specify `arguments`, if the arguments were to be nested under the root element `exec-find-order`, then under an `arguments` element.

Field	Description
Output Result Path	Use this option if the EIS interaction result record contains the XML data that maps to the objects in a nested structure. For example, specify `order`, if the results were return under a root element `results`, then under an `order` element.
Properties	Specify any properties required by your EIS platform. For example, property name `operation` (from `AQPlatform.QUEUE_OPERATION`) and property value `enqueue` (from `AQPlatform.ENQUEUE`).

Configuring Named Query Options

Use this tab to configure additional options for the query.

Figure 25–18 Named Queries, Options Tab



Use the following information to complete each field on the tab:

Field	Description
Refresh Identity Map Results[2]	Refreshes the attributes of the object(s) resulting from the query. If cascading is used, the private parts of the objects will also be refreshed.
Cache Statement[1]	Caches the prepared statements. This requires full parameter binding as well (see **Bind Parameters**).
Bind Parameters[1]	Binds all of the query's parameters.
Cache Usage[2]	Selects how TopLink should use the session cache when a query is executed: ▪ Use descriptor settings ▪ Do not check cache ▪ Check cache by exact primary key ▪ Check cache by primary key ▪ Check cache then database ▪ Check cache only ▪ Conform results in unit of work For more information, see the following: ▪ "Configuring Cache Usage for In-Memory Queries" on page 93-30. ▪ "Configuring Unit of Work Conforming at the Descriptor Level" on page 25-6

Field	Description
In Memory Query Indirection[2]	Selects how TopLink should handle indirection when an in-memory or conforming query is executed: ■ Throw indirection exception–if this object uses indirection and indirection has not been triggered, TopLink will throw an exception. ■ Trigger indirection–if this object uses indirection and indirection has not been triggered, TopLink will trigger indirection. ■ Ignore exception return conformed–returns conforming if an untriggered value holder is encountered. That is, you expect results from the database to conform, and an untriggered value holder is taken to mean that the underlying attribute has not changed. ■ Ignore exception return not conformed–returns not conforming if an untriggered value holder is encountered. For more information, see the following: ■ "Handling Exceptions Resulting From In-Memory Queries" on page 93-33. ■ "Indirection" on page 30-5.
Return Choice[3]	Selects how TopLink should handle `ReportQuery` results: ■ Result collection–return `ReportQuery` results as a `Collection` of `ReportQueryResult` objects. ■ Single result–return only the first `ReportQueryResult` object (not wrapped in a `Collection` or `Map`). Use this option if you know that the `ReportQuery` returns only one row. ■ Single value–return only a single value. Use this option if you know that the `ReportQuery` returns only one row that contains only one attribute. ■ Single attribute–return only a single `Collection` of values. If the query returns multiple rows, but each row only has a single attribute, this option will return a `Collection` of values, instead of a `Collection` of `ReportQueryResults`. For more information, see "Collection Query Results" on page 93-8.
Retrieve Primary Keys[3]	Selects whether or not TopLink retrieves the primary key values within each result. You can use the primary keys to retrieve the real objects. ■ None–do not retrieve primary keys ■ All–retrieve primary keys for each object read; ■ First–return only the first primary key value (in the case of a composite primary key). This can be used if you just want to know if something exists or not, but do not really care about the value.

[1] For more information, see "Parameterized SQL (Binding) and Prepared Statement Caching" on page 11-15.

[2] For `ReadObjectQuery` and `ReadAllQuery` queries only.

[3] For `ReportQuery` queries only.

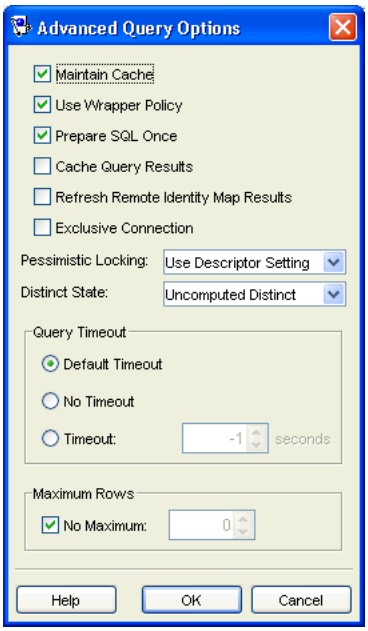
Click **Advanced** to configure additional options. See "Configuring Named Query Advanced Options" on page 25-24.

Configuring Named Query Advanced Options

To configure additional advanced query options, use this procedure.

1. From the Named Queries – Options tab, click **Advanced**. The Advanced Query Options dialog box appears.

Figure 25–19 Advanced Query Options Dialog Box



Use the following information to enter data in each field and click **OK**:

Field	Description
Maintain Cache	Specify whether to use the cache for the query or to build objects directly from the database result. You should only use this option if you are executing a partial object query (see "Partial Object Queries" on page 93-11), whose results cannot be cached. For more information, see "Disabling the Identity Map Cache Update During a Read Query" on page 93-34.
Use Wrapper Policy	Specify whether or not the named query will use the wrapper policy configured for this descriptor. For more information, see "Configuring Wrapper Policy" on page 25-78.
Prepare SQL Once	Specify the `setShouldPrepare()` for the named query. By default, TopLink optimizes queries to generate their SQL only once. You may need to disable this option for certain types of queries that require dynamic SQL based on their arguments, such as: ▪ Expressions that use `equal` where the argument value could be `null`. This may cause problems on databases that require `IS NULL`, instead of `= NULL`. ▪ Expressions that use `in` and use parameter binding. This will cause problems as the `in` values must be bound individually.

Field	Description
Cache Query Results	Specify the `cacheQueryResults` method for the query. The query will only access the database the first time it is executed. Subsequent execution will return exactly the original result. For more information, see "Caching Results in a ReadQuery" on page 96-20.
Refresh Remote Identity Map Results	Specify the `refreshRemoteIdentityMapResult` method for the query. TopLink can refresh the attributes of the object(s) resulting from the query. With cascading, TopLink will also refresh the private parts of the object(s).
Exclusive Connection	Specify whether or not the named query will use an exclusive connection. You can also configure exclusive connection acquisition at the session level (see "Configuring Connection Policy" on page 74-19.
Pessimistic Locking	Specify the specific pessimistic locking policy for the query or use the locking policy from the descriptor.
Distinct State	Specify if TopLink prints the `DISTINCT` clause, if a distinct has been set.
Query Timeout	Specify if the query will time out (or abort) after a specified number of seconds.
Maximum Rows	Specify if the query will limit the results to a specified number of rows. Use this to option for queries that could return an excessive number of objects.

Using Java

To configure named queries in Java, use a descriptor amendment method (see "Configuring Amendment Methods" on page 25-81). Example 25–1 illustrates an amendment method that creates a named query and adds it to the `DescriptorQueryManager`.

Example 25–1 Creating a Named Query with an Amendment Method

```
public class EmployeeAmmendmentMethodClass {
....
    // Create named query with Employee as its reference class
    public static void createEmployeeQuery(ClassDescriptor descriptor) {
        ReadObjectQuery query = new ReadObjectQuery(Employee.class);
        ExpressionBuilder emp = query.getExpressionBuilder();
        Expression firstNameExpression =
            emp.get("firstName").equal(emp.getParameter("firstName"));
        query.setSelectionCriteria(firstNameExpression);
        query.addArgument("firstName");

        descriptor.getQueryManager().addQuery(
            "employeeReadByFirstName", query);
    }
}
```

Configuring Query Timeout at the Descriptor Level

You can specify how the TopLink runtime handles the duration of queries on a descriptor's reference class. Specifying a query timeout at the descriptor level applies to all queries on the descriptor's reference class. A query timeout ensures that your application does not block forever over a hung or lengthy query that does not return in a timely fashion.

Table 25–4 summarizes which descriptors support query timeout configuration.

Table 25–9 Descriptor Support for Cache Refresh

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors[1]	✔	✔
Object-Relational Descriptors		✔
EIS Descriptor		
XML Descriptors		

[1] Relational class descriptors only (see "Relational Class Descriptors" on page 24-2).

You can also configure a timeout on a per-query basis. For more information, see the following:

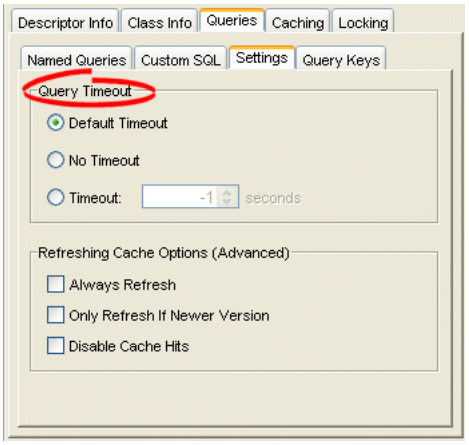
- "Configuring Named Query Advanced Options" on page 25-24
- "Configuring Query Timeout at the Query Level" on page 94-10

Using TopLink Workbench

To configure how TopLink handles the duration of queries to this descriptor, use this procedure:

1. Select a descriptor in the **Navigator**. Its properties appear in the Editor.
2. Click the **Queries** tab. The Queries tab appears.
3. Click the **Settings** tab. The Settings tab appears.

Figure 25–20 Descriptor Queries Settings Tab, Query Timeout Options



Use the following table to enter data in the fields on the descriptor's **Settings** tab to specify how TopLink handles query duration:

Field	Description
Default Timeout	TopLink throws a `DatabaseException` if a query on this descriptor does not return within the timeout period you configure on the parent descriptor. If there is no parent descriptor, the query timeout defaults to **No Timeout**.
No Timeout	TopLink blocks until a query on this descriptor returns.
Timeout	Enter the timeout period in seconds. TopLink throws a `DatabaseException` if a query on this descriptor does not return within this time.

Using Java

Use `DescriptorQueryManager` method `setQueryTimeout` passing in the timeout value as a number of milliseconds.

Configuring Cache Refreshing

By default, TopLink caches objects read from a data source (see Chapter 87, "Understanding the Cache"). Subsequent queries for these objects will access the cache and thus improve performance by reducing data source access and avoiding the cost of rebuilding object's and their relationships. Even if a query, such as a read-all query, accesses the data source, if the objects corresponding to the records returned are in the cache, TopLink will use the cache objects.

This can lead to stale data in the application. Although using an appropriate locking policy (see "Configuring Locking Policy" on page 25-64) is the only way to ensure that stale or conflicting data does not get committed to the data source, sometimes certain data in the application changes so frequently that it is desirable to always refresh the data, instead of only refreshing the data when a conflict is detected.

You can specify how the TopLink runtime handles cache refreshing for all queries on a descriptor's reference class.

Table 25–4 summarizes which descriptors support query cache refresh configuration.

Table 25–10 Descriptor Support for Query Cache Refresh

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors[1]	✔	✔
Object-Relational Descriptors		✔
EIS Descriptor[2]	✔	✔
XML Descriptors		

[1] Relational class descriptors only (see "Relational Class Descriptors" on page 24-2).

[2] EIS root descriptors only (see "EIS Root Descriptors" on page 24-5).

Configuring descriptor-level cache refresh may affect performance. As an alternative, consider configuring the following:

- cache refresh on a query-by-query basis (see "Refreshing the Cache" on page 93-35)
- cache expiration (see "Configuring Cache Expiration at the Descriptor Level" on page 25-42)
- isolated caching (see "Cache Isolation" on page 87-9)

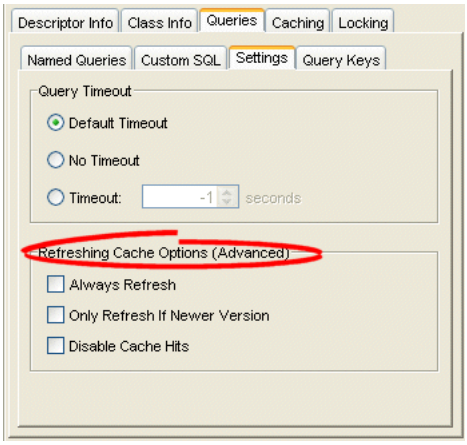
For more information, see "Cache Optimization" on page 11-13.

Using TopLink Workbench

To configure how TopLink refreshes the cache for queries to this descriptor, use this procedure:

1. Select a descriptor in the **Navigator**. Its properties appear in the Editor.
2. Click the **Queries** tab. The Queries tab appears.
3. Click the **Settings** tab. The Settings tab appears.

Figure 25–21 Descriptor Queries Settings Tab, Cache Refreshing Options



Use the following table to enter data in the fields on the descriptor's **Settings** tab to specify how TopLink will refresh the cache for queries:

Field	Description
Always Refresh	Refreshes the cache on all queries. Avoids stale data by ensuring that any query that accesses the data source will refresh the resulting objects in the cache. This has no effect on queries that get a cache hit and never access the data source, such as read-object primary key queries or in-memory queries. Configuring descriptor level cache refresh may affect performance. As an alternative, consider configuring: ▪ cache refresh on a query-by-query basis (see "Refreshing the Cache" on page 93-35) ▪ cache expiration (see "Configuring Cache Expiration at the Descriptor Level" on page 25-42) ▪ isolated caching (see "Cache Isolation" on page 87-9)

Field	Description
Only Refresh If Newer Version	Refreshes the cache only if the object in the database is newer than the object in the cache (as determined by the **Optimistic Locking** field). See "Configuring Locking Policy" on page 25-64 for more information. Improves performance by avoiding unnecessary refreshing of an object if its version matches the data source version. This option does not cause refreshing on its own: you must use it in combination with **Always Refresh**, query refreshing (see "Refreshing the Cache" on page 93-35), or cache expiration (see "Configuring Cache Expiration at the Descriptor Level" on page 25-42).
Disable Cache Hits	When selected, TopLink bypasses the cache and goes to the database for read object queries based on primary key. Using this option in conjunction with **Always Refresh** ensures that TopLink always goes to the database. This option ensures that all queries including read-object primary key queries will always access the data source. This option does not cause refreshing on its own: you must use it in combination with **Always Refresh**. This option can cause a serious performance issue: avoid whenever possible.

Caution: Use the **Always Refresh** and **Disable Cache Hits** properties with caution as they may lead to poor performance. As an alternative, consider configuring cache refresh on a query-by-query basis (see "Refreshing the Cache" on page 93-35) or configuring cache expiration (see "Configuring Cache Expiration at the Descriptor Level" on page 25-42). For more information about cache performance, see "Cache Optimization" on page 11-13.

Using Java

Configure cache refresh options using the following `ClassDescriptor` methods:

- `setShouldAlwaysRefreshCache`
- `setShouldAlwaysRefreshCacheOnRemote`
- `setShouldDisableCacheHits`
- `setShouldDisableCacheHitsOnRemote`
- `setShouldOnlyRefreshCacheIfNewerVersion`

Use these methods in a descriptor amendment method (see "Configuring Amendment Methods" on page 25-81) as Example 25–2 illustrates.

Example 25–2 Configuring Remote Refreshing

```
public void addToDescriptor(ClassDescriptor descriptor) {
    descriptor.setShouldRefreshCacheOnRemote(true);
    descriptor.setShouldDisableCacheHitsOnRemote(true);
}
```

Configuring Query Keys

A **query key** is a schema-independent alias for a database field name. For example, consider a class `Employee` with attribute `firstName` mapped directly to a database field `F_NAME` in database table `EMPLOYEE`. Without a query key, when you create a query or expression that involves `Employee` attribute `firstName`, you must use the database management system-specific field name `F_NAME`. This makes it more difficult to build a query and ties the query to the schema. With a query key, you can refer to this field using a schema-independent alias, such as `firstName`.

Table 25–11 summarizes which descriptors support query keys.

Table 25–11 Descriptor Support for Query Keys

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors	✔	✔
Object-Relational Descriptors		✔
EIS Descriptors		
XML Descriptors		

Using query keys offers the following advantages:

- Enhances code readability in TopLink expressions and simplifies expression development. You can compose expressions entirely within the context of your object model.
- Increases portability by making code independent of the database schema. If you rename a field in your schema, you can redefine the query key without changing any code that uses it.
- Query keys used with interface descriptors allow the implementor descriptor's tables to have different field names.

Query keys are automatically generated for all mapped attributes. The name of the query key is the name of the class attribute specified in your object model.

For information on how to use query keys in queries and expressions, see "Query Keys" on page 93-4.

When query keys are generated and how you can add or modify query keys depends on the type of mapping or descriptor involved:

- Direct Mappings
- Relationship Mappings
- Interface Descriptors

Direct Mappings

TopLink Workbench automatically generates query keys for all direct mappings at the time you create the mapping.

TopLink Workbench provides support for adding or modifying query keys for simple unmapped attributes that could be mapped by a direct mapping: for example, the `version` field used for optimistic locking or the `type` field used for inheritance. You cannot modify or remove automatically generated query keys.

Relationship Mappings

TopLink automatically generates query keys for all relationship mappings at run time.

For example, if you have a class `Customer` with attribute `orders` mapped in a one-to-many relationship to class `PurchaseOrders`, then the TopLink runtime will generate a query key named `orders` for this `Customer` attribute.

TopLink Workbench does not currently support adding or modifying the query keys for relationship mappings. If you must add or modify such a query key, you must do so in Java code, using a descriptor amendment method.

Interface Descriptors

Interface descriptors (see "Relational Interface Descriptors" on page 24-2) define only the query keys that are shared among their implementors. In the descriptor for an interface, only the name of the query key is specified.

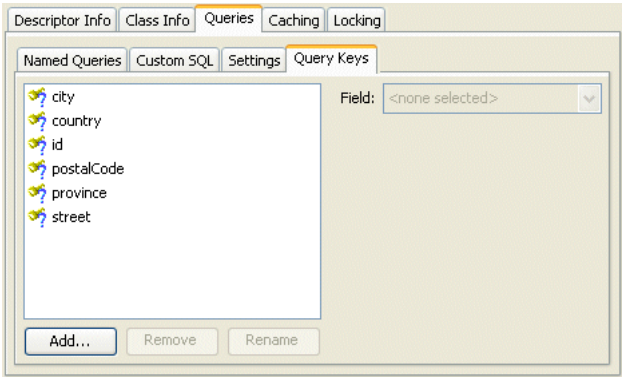
TopLink Workbench provides support for choosing the implementors of an interface that share at least one common automatically generated query key (see "Configuring Interface Query Keys" on page 25-33).

Using TopLink Workbench

To add query keys to simple unmapped fields and to view the query keys automatically generated for directly mapped attributes, use this procedure:

1. Select a descriptor in the **Navigator**. Its properties appear in the Editor.
2. Click the **Query Keys** tab in the **Editor**.

Figure 25–22 Queries, Query Keys Tab



To add a new query key, click **Add.**

To delete an existing query key, select the query key and click **Remove**.

To rename an existing query key, select the query key and click **Rename**.

Use the **Field** list to select the field in the table associated with the query key.

Using Java

To manually create a relationship query key, implement a descriptor amendment method (see "Configuring Amendment Methods" on page 25-81) that uses one of the following `ClassDescriptor` methods to register the query keys:

- `addQueryKey`–specify a query key using an instance of `QueryKey` such as `DirectQueryKey`, `DirectCollectionQueryKey`, `ManyToManyQueryKey`, `OneToManyQueryKey`, or `OneToOneQueryKey`.
- `addDirectQueryKey`–add a query key that maps directly to the given database field.
- `addAbstractQueryKey`–add an abstract query key to an interface descriptor. Any implementors of that interface must define the query key defined by this abstract query key.

Examples 25–3, 25–4, and 25–5 illustrate how to define a query key in Java code.

Example 25–3 Defining a 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 addQueryKey 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 of which the employee is a
leader (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 25–4 Defining a One-to-Many Query Key

```
/* Add a one-to-many query key 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);
```

Example 25–5 Defining a Many-to-Many Query Key

```
/* Add a many-to-many query key to an employee project that uses a join table*/
ManyToManyQueryKey projectsQueryKey = new ManyToManyQueryKey();
projectsQueryKey.setName("projects");
projectsQueryKey.setReferenceClass(Project.class);
ExpressionBuilder builder = new ExpressionBuilder();
projectsQueryKey.setJoinCriteria(
    builder.getTable("EMP_PROJ").getField("EMP_ID").equal(
    builder.getParameter("EMPLOYEE.EMP_ID").and(
    builder.getTable("EMP_PROJ").getField("PROJ_ID").equal(
    builder.getField("PROJECT.PROJ_ID")));
descriptor.addQueryKey(projectsQueryKey);
```

Example 25–6 illustrates how to implement a `Descriptor` amendment method to define a one-to-one query key. In this example, the object model for the `Address` class does not include a reference to its owner, an `Employee` object. You can amend the `Address` class descriptor to add a query key named `owner` to make up for this deficiency. At run time, you can compose expressions that select `Address` objects based on this `owner` query key.

Example 25–6 Defining a One-to-One Query Key with an Amendment Method

```
/* 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);
}
```

Configuring Interface Query Keys

A query key is a schema independent alias for a database field name. For more information about query keys, see "Configuring Query Keys" on page 25-30.

Interface descriptors (see "Relational Interface Descriptors" on page 24-2) are defined only with query keys that are shared among their implementors. In the descriptor for an interface, only the name of the query key is specified.

In each implementor descriptor, the key must be defined with the appropriate field from one of the implementor descriptor's tables.

This allows queries and relationship mappings to be defined on the interface using the query key names.

Interface query keys are supported in relational database projects only.

Table 25–11 summarizes which descriptors support interface query keys.

Table 25–12 Descriptor Support for Interface Query Keys

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors	✔	✔
Object-Relational Descriptors		✔
EIS Descriptors		
XML Descriptors		

Consider an `Employee` that contains a contact of type `Contact`. The `Contact` class is an interface with two implementors: `Phone` and `Email`. The `Phone` class has attributes `id` and `number`. The `Email` class has attributes `id` and `address`. Figure 25–23 illustrates the generated keys:

Figure 25–23 Automatically Generated Query Keys for Phone and Email



Both classes have an attribute, id, that is directly mapped to fields that have different names. However, a query key is generated for this attribute. For the `Contact` interface descriptor, you must indicate that the `id` query key must be defined for each of the implementors.

If either of the implementor classes did not have the `id` query key defined, TopLink Workbench flags that descriptor as deficient.

Now that a descriptor with a commonly shared query key has been defined for `Contact`, you can use it as the reference class for a variable one-to-one mapping (see "Using Queries on Variable One-to-One Mappings" on page 96-5).

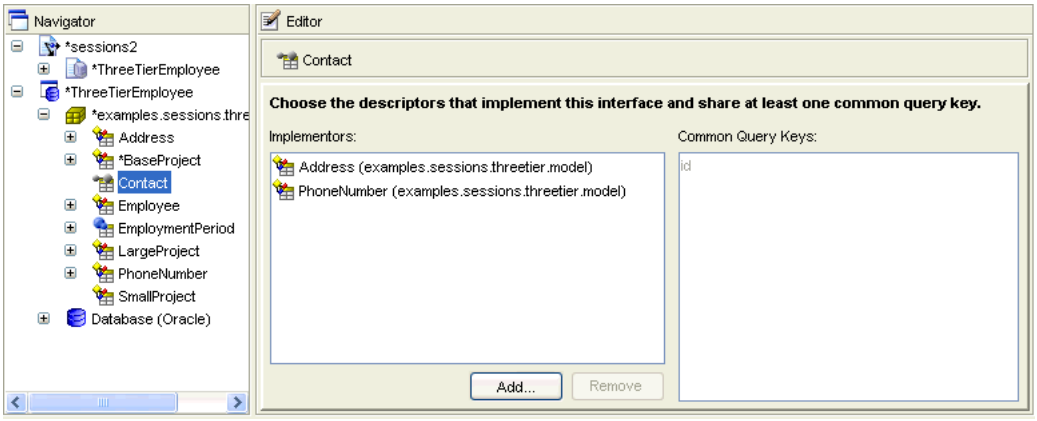
For example, you can now create a variable one-to-one mapping for the `contact` attribute of `Employee`. When you edit the foreign key field information for the mapping, you must match the `Employee` descriptor's tables to query keys from the `Contact` interface descriptor.

Using TopLink Workbench

To choose the implementors of an interface that share at least one common automatically generated query key, use this procedure.

1. Select an interface descriptor in the **Navigator**. Its properties appear in the Editor.

Figure 25–24 Interface Descriptor Editor Window



To choose an implementor of the selected interface that shares at least one common query key, click **Add.**

To remove an implementor of the selected interface, select the implementor and click **Remove**.

Using Java

Example 25–7 shows how to define the `Contact` interface and `Email` and `Phone` implementors in Java.

Example 25–7 Defining Interface Query Keys

```
Descriptor contactInterfaceDescriptor = new Descriptor();
    contactInterfaceDescriptor.setJavaInterface(Contact.class);
    contactInterfaceDescriptor.addAbstractQueryKey("id");
Descriptor emailClassDescriptor = new Descriptor();
    emailClassDescriptor.setJavaClass(Email.class);
    emailClassDescriptor.addDirectQueryKey("id", "E_ID");
    emailClassDescriptor.getInterfacePolicy().addParentInterface(Contact.class);
    emailClassDescriptor.setTableName("INT_EML");
    emailClassDescriptor.setPrimaryKeyFieldName("E_ID");
    emailClassDescriptor.setSequenceNumberName("SEQ");
    emailClassDescriptor.setSequenceNumberFieldName("E_ID");
    emailClassDescriptor.addDirectMapping("emailID", "E_ID");
    emailClassDescriptor.addDirectMapping("address", "ADDR");
Descriptor phoneClassDescriptor = new Descriptor();
    phoneClassDescriptor.setJavaClass(Phone.class);
    phoneClassDescriptor.getInterfacePolicy().addParentInterface(Contact.class);
    phoneClassDescriptor.addDirectQueryKey("id", "P_ID");
    phoneClassDescriptor.setTableName("INT_PHN");
    phoneClassDescriptor.setPrimaryKeyFieldName("P_ID");
    phoneClassDescriptor.setSequenceNumberName("SEQ");
    phoneClassDescriptor.setSequenceNumberFieldName("P_ID");
    phoneClassDescriptor.addDirectMapping("phoneID", "P_ID");
    phoneClassDescriptor.addDirectMapping("number", "P_NUM");
```

Configuring Cache Type and Size at the Descriptor Level

The TopLink cache is an in-memory repository that stores recently read or written objects based on class and primary key values. TopLink uses the cache to do the following:

- improve performance by holding recently read or written objects and accessing them in-memory to minimize database access
- manage locking and isolation level
- manage object identity

Table 25–13 summarizes which descriptors support identity map configuration.

Table 25–13 Descriptor Support for Identity Map

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors[1]	✔	✔
Object-Relational Descriptors		✔
EIS Descriptors[2]	✔	✔
XML Descriptors		

[1] Relational class descriptors only (see "Relational Class Descriptors" on page 24-2).

[2] EIS root descriptors only (see "EIS Root Descriptors" on page 24-5).

This configuration overrides the default identity map configuration defined at the project level (see "Configuring Cache Type and Size at the Project Level" on page 19-13).

For detailed information on caching and object identity, and the recommended settings to maximize TopLink performance, see to "Cache Type and Object Identity" on page 87-3.

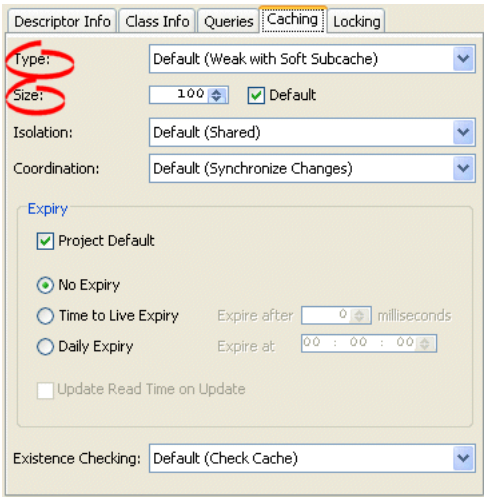
For more information about the cache, see Chapter 87, "Understanding the Cache".

Using TopLink Workbench

To specify the identity map information for a descriptor, use this procedure:

1. Select the descriptor in the **Navigator**.
2. Select the **Caching** tab in the **Editor**. The Caching tab appears.

Figure 25–25 Caching Tab, Identity Map Options



Use the following table to enter data in following fields on the **Caching** tab:

Field	Description
Type[1]	Use the **Type** list to choose the identity map as follows: ▪ **Weak with Soft Subcache** (`SoftCacheWeakIdentityMap`)–cache first *n* elements in soft space, anything after that in weak space (see "Soft and Hard Cache Weak Identity Maps" on page 87-4) ▪ **Weak with Hard Subcache** (`HardCacheWeakIdentityMap`)–cache first *n* elements in hard space, anything after that in weak space (see "Soft and Hard Cache Weak Identity Maps" on page 87-4) ▪ **Weak** (`WeakIdentityMap`)–cache everything in weak space (see "Weak Identity Map" on page 87-3) ▪ **Full** (`FullIdentityMap`)–cache everything permanently (see "Full Identity Map" on page 87-3) ▪ **None** (`NoIdentityMap`)–cache nothing (see "No Identity Map" on page 87-4) For more information, see "Cache Type and Object Identity" on page 87-3. Changing the project's default identity map does not affect descriptors that already exist in the project.

Field	Description
Size[1]	Specify the size of the cache as follows: ▪ When using **Weak with Soft Subcache** or **Weak with Hard Subcache**, the size is the size of the subcache. ▪ When using **Full** or **Weak**, the size indicates the *starting size* of the identity map.
Default	When you enter a cache size, the **Default** check box is cleared. To reset the size to the default for the selected cache type, check the **Default** check box.

[1] If a descriptor is a child in an inheritance hierarchy, TopLink makes this field read only and displays the options from the parent root descriptor. For more information, see "Inheritance" on page 23-4.

Using Java

Use one of the following `ClassDescriptor` methods to configure the descriptor to use the appropriate type of identity map:

- `useFullIdentitMap`
- `useWeakIdentityMap`
- `useSoftCacheWeakIdentityMap`
- `useHardCacheWeakIdentityMap`
- `useNoIdentityMap`

Use the `ClassDescriptor` method `setIdentityMapSize` to configure the size of the identity map.

Configuring Cache Isolation at the Descriptor Level

If you plan to use isolated sessions (see "Cache Isolation" on page 87-9), you must configure descriptors as isolated for any object that you want confined to an isolated session cache.

Configuring a descriptor to be isolated means that TopLink will not store the object in the shared session cache and the object will not be shared across client sessions. Each client will have their own object read directly from the database. Objects in an isolated client session cache can reference objects in their parent server session's shared session cache, but no objects in the shared session cache can reference objects in an isolated client session cache. Isolation is required when using Oracle Database Virtual Private Database (VPD) support or database user-based read security. Isolation can also be used if caching is not desired across client sessions.

Table 25–13 summarizes which descriptors support cache isolation configuration.

Table 25–14 Descriptor Support for Cache Isolation Map

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors[1]	✔	✔
Object-Relational Descriptors		✔
EIS Descriptors[2]	✔	✔
XML Descriptors		

[1] Relational class descriptors only (see "Relational Class Descriptors" on page 24-2).

[2] EIS root descriptors only (see "EIS Root Descriptors" on page 24-5).

This configuration overrides the default cache isolation configuration defined at the project level (see "Configuring Cache Isolation at the Project Level" on page 19-16).

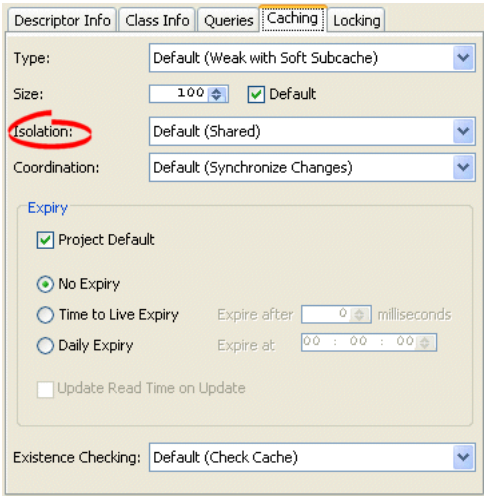
Note: If you configure a descriptor as isolated, it cannot participate in a coordinated cache (see "Configuring Cache Coordination Change Propagation at the Descriptor Level" on page 25-40).

Using TopLink Workbench

To specify the cache isolation options, use this procedure:

1. Select the descriptor in the **Navigator**.
2. Select the **Caching** tab in the **Editor**. The Caching tab appears.

Figure 25–26 Caching Tab, Isolation Options



Use the **Isolation** list to choose one of the following:

- **Isolated**–if you want all objects confined to an isolated client session cache. For more information, see "Cache Isolation" on page 87-9.
- **Shared**–if you want all objects visible in the shared session cache (default).

Using Java

To specify that a class is isolated, use a descriptor amendment method (see "Configuring Amendment Methods" on page 25-81) to call `ClassDescriptor` method `setIsIsolated`, passing in a `boolean` of `true`.

Configuring Unit of Work Cache Isolation at the Descriptor Level

Use this policy to determine how a unit of work uses a session cache for a specific class. Table 25–15 lists the unit of work cache isolation options.

Table 25–15 Unit of Work Cache Isolation Options

Option	Description
Using the Session Cache After the Transaction	**USE_SESSION_CACHE_AFTER_TRANSACTION** Objects built from new data accessed after a unit of work early transaction are stored in the session cache. This options is the most efficient as it allows the cache to be used after an early transaction. **This option only applies to descriptors that are read-only. If the** descriptor is not read-only, this setting is reverted to ISOLATE_CACHE_ALWAYS (see Always Isolating the Cache option).
Isolating New Data After the Transaction	**ISOLATE_NEW_DATA_AFTER_TRANSACTION** (default) Objects built from new data accessed after a unit of work early transaction are only stored in the unit of work. This still allows previously cached objects to be accessed in the unit of work after an early transaction, while ensuring that uncommitted data will never be put in the session cache by storing any object built from new data only in the unit of work
Isolating the Cache after the Transaction	**ISOLATE_CACHE_AFTER_TRANSACTION** After a unit of work early transaction the session cache is no longer used for this class. Objects are directly built from the database data and only stored in the unit of work, *even if previously cached*. **Note**: This option my affect performance because you are bypassing the session cache after an early transaction.
Always Isolating the Cache	**ISOLATE_CACHE_ALWAYS** The session cache will *never* be used for the class. Objects are directly built from the database data and only stored in the unit of work. New objects and changes will never be merged into the session cache. **Note**: This option my affect performance because you are bypassing the session cache. However if this class is isolated or pessimistic locked and always accessed in a transaction, this can avoid having to build two copies of the object.

Most of these options apply only to a Unit of Work in an early transaction, such as:

- A Unit of Work that was flushed (`write changes`)
- Issued a modify query
- Acquired a pessimistic lock

Using Java

To specify that a class is isolated, use a descriptor amendment method (see "Configuring Amendment Methods" on page 25-81) to call `ClassDescriptor` method `setUnitOfWorkCacheIsolationLevel`:

```
public void setUnitOfWorkCacheIsolationLevel(int unitOfWorkCacheIsolationLevel) {
    this.unitOfWorkCacheIsolationLevel = unitOfWorkCacheIsolationLevel;
}
```

Configuring Cache Coordination Change Propagation at the Descriptor Level

If you plan to use a coordinated cache (see "Understanding Cache Coordination" on page 87-10), you can configure how, and under what conditions, a coordinated cache propagates changes for a given descriptor.

Table 25–13 summarizes which descriptors support cache isolation configuration.

Table 25–16 Descriptor Support for Cache Coordination Change Propagation Configuration

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors[1]	✔	✔
Object-Relational Descriptors		✔
EIS Descriptors[2]	✔	✔
XML Descriptors		

[1] Relational class descriptors only (see "Relational Class Descriptors" on page 24-2).

[2] EIS root descriptors only (see "EIS Root Descriptors" on page 24-5).

This configuration overrides the default cache coordination change propagation configuration defined at the project level (see "Configuring Cache Coordination Change Propagation at the Project Level" on page 19-17).

To complete your coordinated cache configuration, see Chapter 88, "Configuring a Coordinated Cache".

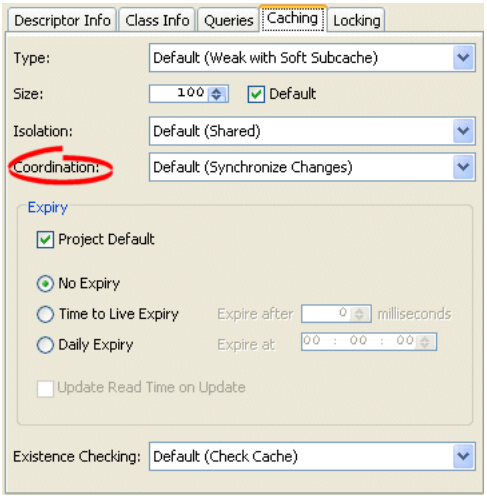
Note: If you configure a descriptor as isolated (see "Configuring Cache Isolation at the Descriptor Level" on page 25-37), it cannot participate in a coordinated cache.

Using TopLink Workbench

To specify the coordinated cache change propagation options, use this procedure:

1. Select the descriptor in the **Navigator**.
2. Select the **Caching** tab in the **Editor**. The Caching tab appears.

Figure 25–27 Caching Tab, Coordination Options



Use the following information to enter data in the **Coordination** field:

Coordination Option	Description	When to Use
None	For both existing and new instances, do not propagate a change notification.	Infrequently read or changed objects.
Synchronize Changes	For an existing instance, propagate a change notification that contains each changed attribute. For a new instance, propagate an object creation (along with all the new instance's attributes) only if the new instance is related to other existing objects that are also configured with this change propagation option.	Frequently read or changed objects that contain few attributes or in cases where only a few attributes are frequently changed. Objects that have many or complex relationships.
Synchronize Changes and New Objects	For an existing instance, propagate a change notification that contains each changed attribute. For a new instance, propagate an object creation (along with all the new instance's attributes).	Frequently read or changed objects that contain few attributes or in cases where only a few attributes are frequently changed. Objects that have few or simple relationships.
Invalidate Changed Objects	For an existing instance, propagate an object invalidation that marks the object as invalid in all other sessions. This tells other sessions that they must update their cache from the data source the next time this object is read. For a new instance, no change notification is propagated.	Frequently read or changed objects that contain many attributes in cases where many of the attributes are frequently changed.

Using Java

Use a descriptor amendment method (see "Configuring Amendment Methods" on page 25-81) to invoke `ClassDescriptor` method `setCacheSynchronizationType` passing in one of the following parameters:

- `ClassDescriptor.DO_NOT_SEND_CHANGES`
- `ClassDescriptor.SEND_OBJECT_CHANGES`
- `ClassDescriptor.SEND_NEW_OBJECTS_WITH_CHANGES`

- `ClassDescriptor.INVALIDATE_CHANGED_OBJECTS`

Configuring Cache Expiration at the Descriptor Level

By default, objects remain in the cache until they are explicitly deleted (see "Deleting Objects" on page 98-7) or garbage-collected when using a weak identity map (see "Configuring Cache Isolation at the Project Level" on page 19-16). Alternatively, you can configure an object with a `CacheInvalidationPolicy` that allows you to specify, either automatically or manually, that an object is invalid: when any query attempts to read an invalid object, TopLink will go to the data source for the most up-to-date version of that object and update the cache with this information.

Using cache invalidation ensures that your application does not use stale data. It provides a better performing alternative to always refreshing (see "Configuring Cache Refreshing" on page 25-27).

Table 25–17 summarizes which descriptors support a cache invalidation policy.

Table 25–17 Descriptor Support for Cache Invalidation Policy

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors[1]	✔	✔
Object-Relational Descriptors		✔
EIS Descriptors[2]	✔	✔
XML Descriptors		

[1] Relational class descriptors only (see "Relational Class Descriptors" on page 24-2).

[2] EIS root descriptors only (see "EIS Root Descriptors" on page 24-5).

You can override the project-level cache invalidation configuration (see "Configuring Cache Expiration at the Project Level" on page 19-19) by defining cache invalidation at the descriptor or query level (see "Configuring Cache Expiration at the Query Level" on page 96-21).

You can customize how TopLink communicates the fact that an object has been declared invalid to improve efficiency, if you are using a coordinated cache. For more information, see "Configuring Cache Coordination Change Propagation at the Descriptor Level" on page 25-40.

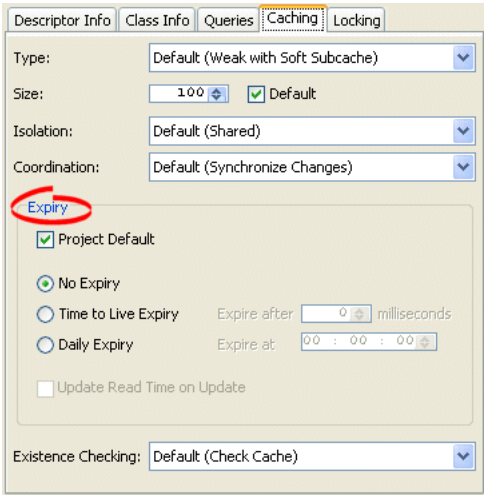
For more information, see "Cache Invalidation" on page 87-8.

Using TopLink Workbench

To specify the cache invalidation information for a descriptor, use this procedure:

1. Select the descriptor in the **Navigator**.
2. Select the **Caching** tab in the **Editor**. The **Caching** tab appears.

Figure 25–28 Caching Tab, Expiration Options



Use this table to enter data in the following fields on the **Caching** tab to specify the cache invalidation policy for the descriptors.

Field	Description
Project Default	Use the project's cache expiration options for this descriptor. See "Configuring Cache Expiration at the Project Level" on page 19-19 for more information.
No Expiry	Specify that objects in the cache do not expire.
Time to Live Expiry	Specify that objects in the cache will expire after a specified amount of time. Use the **Expire After** field to indicate the time (in milliseconds) after which the objects will expire.
Daily Expiry	Specify that objects in the cache will expire at a specific time each day. Use the **Expire At** field to indicate the exact time to the second (using a 24-hour clock) at which the objects will expire.
Update Read Time on Update	Specify if TopLink should reset the cached object's expiry time after the TopLink successfully updates the object.

Using Java

Use `ClassDescriptor` method `setCacheInvalidationPolicy` to set an appropriate instance of `CacheInvalidationPolicy`.

Configuring Cache Existence Checking at the Descriptor Level

When TopLink writes an object to the database, TopLink runs an existence check to determine whether to perform an insert or an update.

By default, TopLink checks against the cache. Oracle recommends that you use this default existence check option for most applications. Checking the database for existence can cause a performance bottleneck in your application.

Table 25–18 summarizes which descriptors support existence checking.

Table 25–18 Descriptor Support for Existence Checking

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors[1]	✔	✔
Object-Relational Descriptors		✔
EIS Descriptors[2]	✔	✔
XML Descriptors		

[1] Relational class descriptors only (see "Relational Class Descriptors" on page 24-2).
[2] EIS root descriptors only (see "EIS Root Descriptors" on page 24-5).

You can configure existence checking at the descriptor level to override the project level configuration (see "Configuring Existence Checking at the Project Level" on page 19-8).

For more information see:

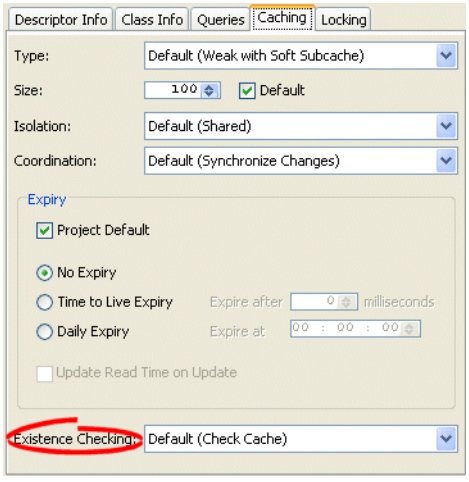
- "Cache Type and Object Identity" on page 87-3
- "Queries and the Cache" on page 93-29
- "Using Registration and Existence Checking" on page 99-5

Using TopLink Workbench

To specify the existence checking information for a descriptor, use this procedure:

1. Select the descriptor in the **Navigator**.
2. Select the **Caching** tab in the **Editor**. The Caching tab appears.

Figure 25–29 Caching Tab, Existence Checking Options



Use this table to enter data in the following fields of the tab to specify the existence checking options for newly created descriptors:

Field	Description
Check Cache	Check the session cache. If the object is not in the cache, assume that the object does not exist (do an insert). If the object is in the cache, assume that the object exists (do an update). Oracle recommends using this option for most applications.
Check Cache then Database	If an object is not in the cache, query the database to determine if the object exists. If the object exists, do an update. Otherwise, do an insert. Selecting this option may negatively impact performance. For more information, see "Check Database" on page 99-5.
Assume Existence	Always assume objects exist: always do an update (never do an insert). For more information, see "Assume Existence" on page 99-5.
Assume Non-Existence	Always assume objects do not exist: always do an insert (never do an update). For more information, see "Assume Nonexistence" on page 99-5.

Using Java

To configure existence checking at the descriptor level using Java, use `ClassDescriptor` method `getQueryManager` to acquire the `DescriptorQueryManager` from the descriptor and then use one of the following `DescriptorQueryManager` methods (see Example 25–8):

- `checkCacheForDoesExist`–check the session cache. If the object is not in the cache, assume that the object does not exist (do an insert). If the object is in the cache, assume that the object exists (do an update). Oracle recommends using this option for most applications.
- `checkDatabaseForDoesExist`–if an object is not in the cache, query the database to determine if the object exists. If the object exists, do an update. Otherwise, do an insert. Selecting this option may negatively impact performance. For more information, see "Check Database" on page 99-5.
- `assumeExistenceForDoesExist`–always assume objects exist: always do an update (never do an insert). For more information, see "Assume Existence" on page 99-5.
- `assumeNonExistenceForDoesExist`–always assume objects do not exist: always do an insert (never do an update). For more information, see "Assume Nonexistence" on page 99-5.

Example 25–8 Configuring Existence Checking Using Java

```
descriptor.getQueryManager().checkCacehForDoesExist();
```

Configuring a Descriptor With EJB Information

If your project uses EJB (see "Configuring Persistence Type" on page 19-5), you can use descriptors to describe the characteristics of entity beans with container-managed or bean-managed persistence.

Table 25–19 summarizes which descriptors support EJB information.

Table 25–19 Descriptor Support for EJB Information

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors[1]	✔	✔
Object-Relational Descriptors		
EIS Descriptors[2]	✔	✔
XML Descriptors		

[1] Relational class descriptors only (see "Relational Class Descriptors" on page 24-2).
[2] EIS root descriptors only (see "EIS Root Descriptors" on page 24-5).

When mapping enterprise beans, you create a descriptor for the bean class; you do not create a descriptor for the local interface, remote interface, home class, or primary key class.

When using TopLink Workbench, you must define the project with the correct entity bean type (such as container-managed or bean-managed persistence) and import the `ejb-jar.xml` file for the beans into the TopLink Workbench project.

For CMP projects, the `ejb-jar.xml` file defines the bean's attributes to be mapped. A descriptor for an entity bean container-managed persistence contains a CMP policy used to configure CMP-specific options.

For more information, see "Descriptors and EJB" on page 23-3.

Using TopLink Workbench

To configure a descriptor with EJB information, use this procedure:

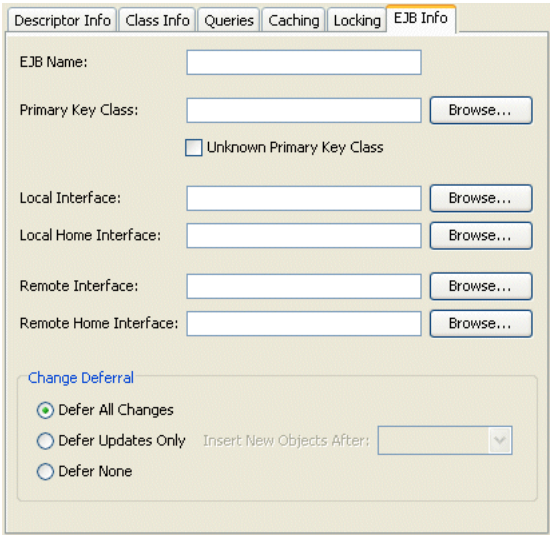
1. In the **Navigator**, select a relational descriptor.
2.  Click **EJB Descriptor** on the mapping toolbar.

   An EJB Info tab is added to the descriptor.

   To remove the EJB information for the selected descriptor, click **EJB Descriptor** again.

   The EJB Info tab is removed from the descriptor.
3. Click the **EJB Info** tab in the **Editor**

Figure 25–30 EJB Info Tab



Use the following information to enter data in each field on the tab:

Field	Description
EJB Name	Enter the bean's base name. This is specified in the **`<ejb-name>`** element of the `ejb-jar.xml` file and is for display only.
Primary Key Class	Enter the primary key. This is specified in the **`<prim-key-class>`** element of the `ejb-jar.xml` file and is for display only.
Unknown Primary Key Class	Check this option if you choose not to specify the primary key class or the primary key fields for an entity bean with container managed persistence. For example, select this field if the entity bean does not have a natural primary key or you want the deployer to select the primary key fields at deployment time. For more information, see "Unknown Primary Key Class Support" on page 7-25.
Local Interface	Enter the local interface. This is specified in the **`<local>`** element of the `ejb-jar.xml` file and is for display only.
Local Home Interface	Enter the local home interface. This is specified in the **`<local-home>`** element of the `ejb-jar.xml` file and is for display only.
Remote Interface	Enter the remote interface. This is specified in the **`<remote>`** element of the `ejb-jar.xml` file and is for display only.
Remote Home Interface	Enter the remote interface. This is specified in the **`<home>`** element of the `ejb-jar.xml` file and is for display only.
Change Deferral	Use these options to specify how TopLink updates the database for this EJB descriptor.
Defer All Changes	Specify not to send changes to the database until the JTA transaction is committed. This is the default TopLink behavior. This is the most efficient option that results in the least amount of data source interaction.

Field	Description
Defer Updates Only	Specify to send changes to the database immediately after any insert or delete operation, but do not send changes to the data source for update operations until the JTA transaction is committed. Select this option for backwards compatibility with some EJB containers, such as OC4J. For more information, see "Nondeferred Changes" on page 23-3). Use this option with caution as it will require the data source transaction and locks to be held longer and may cause referential integrity issues.
Defer None	Specify to send all changes to the database immediately. This is the least efficient option that generates the greatest amount of data source interaction. Select this option for backwards compatibility with some EJB containers, such as OC4J. For more information, see "Nondeferred Changes" on page 23-3).
Insert New Objects After	Specify to send new object insert changes to the database after bean life cycle method `ejbCreate` (default) or `ejbPostCreate`. This is only relevant when not deferring changes (see "Configuring Change Policy" on page 25-73). If non-null foreign key constraints cannot be satisfied when the insert is performed after `ejbCreate`, you may consider configuring TopLink CMP to do the insert after `ejbPostCreate`, if supported by your container. For more information, see "Creating a New Entity Bean and ejbCreate / ejbPostCreate Methods" on page 23-4.

Using Java

Using Java code, you can use descriptors to describe the characteristics of entity beans with container-managed (see "Configuring CMP Information" on page 25-48) or bean-managed (see "Configuring BMP Information" on page 25-49) persistence.

Configuring CMP Information

To configure CMP-specific information on a descriptor, define a `CMPPolicy`:

```
descriptor.setCMPPolicy(new CMPPolicy());
```

You can use the following `CMPPolicy` API to configure optional behavior of enterprise beans:

> **Note:** Most of these options are provided for compatibility with other CMP implementations. Use caution when using them as they will affect application performance.

- `setDeferModificationsUntilCommit`–By default TopLink defers all changes to the database until the transaction is committed. Use this method to configure TopLink to update the database after each EJB operation for the specified deferral level:
  - `CMPPolicy.NONE`–default behavior
  - `CMPPolicy.UPDATE_MODIFICATIONS`–update the database after each EJB operation for update modifications only

  - `CMPPolicy.ALL_MODIFICATIONS`–update the database after each EJB operation for all modifications

- `setNonDeferredCreateTime`–when using nondeferred writes (see setDeferModificationsUntilCommit), use this method to configure TopLink to insert a new enterprise bean before (`CMPPolicy.AFTER_EJBCREATE`) or after (`CMPPolicy.AFTER_EJBPOSTCREATE`) the `ejbPostCreate` method.
- `setForceUpdate`–use this method to make TopLink write all enterprise beans that have been accessed to the database regardless of whether they changed or not.
- `setUpdateAllFields`–use this method to configure TopLink to force all the fields of the bean to be updated instead of only the changed fields.
- `setPessimisticLockingPolicy`–use this method to configure EJB-level pessimistic locking.

Configuring BMP Information

BMP descriptors must be configured with a `BMPWrapperPolicy`. TopLink Workbench does not currently support defining the `BMPWrapperPolicy` so you must define this through Java code.

For more information, see "Configuring Wrapper Policy" on page 25-78.

Configuring Reading Subclasses on Queries

If you are mapping an inheritance hierarchy, by default, queries on root or branch classes return instances of the root class only.

Alternatively, you can configure a root or branch class descriptor to include subclasses when the root or branch class is queried.

You can also specify a database view to optimize the reading of subclasses. The view can be used to optimize queries for root or branch classes that have subclasses spanning multiple tables. The view must apply an outer-join or union all of the subclass tables.

Do not configure this option for leaf classes.

Table 25–20 summarizes which descriptors support inherited attribute mapping configuration.

Table 25–20 Descriptor Support for Inherited Attribute Mapping Configuration

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors	✔	✔
Object-Relational Descriptors		✔
EIS Descriptors		
XML Descriptors		

For more information, see "Descriptors and Inheritance" on page 23-3.

Using TopLink Workbench

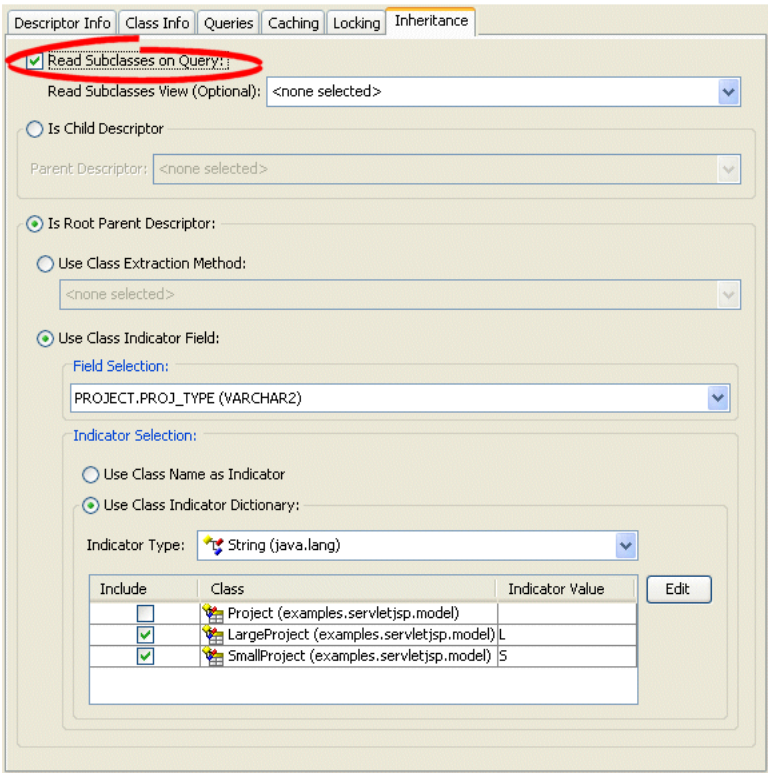
To configure reading classes on subqueries, use this procedure:

1. In the **Navigator**, select a root or branch descriptor.

If the **Inheritance** advanced property is not visible for the descriptor, right-click the descriptor and choose **Select Advanced Properties** > **Inheritance** from context menu or from the **Selected** menu.

2. Click the **Inheritance** tab.

Figure 25–31 Inheritance Tab, Read Subclasses on Query Option



Use the following information to enter data in Read Subclasses on Query and Read Subclasses View fields of the tab:

Field	Description
Read Subclasses on Query	Select this option to configure the root class descriptor to instantiate a subclass when the root class is queried.
Read Subclasses View	Optionally select a database view to use for reading subclasses.

Using Java

Create a descriptor amendment method ("Configuring Amendment Methods" on page 25-81) to customize the root or branch class descriptor's `InheritancePolicy` using `InheritancePolicy` method `dontReadSubclassesOnQueries` to configure a root or branch descriptor to not read subclasses. Optionally, you can use `InheritancePolicy` method `setReadAllSubclassesViewName` to optimize multiple table inheritance queries.

Example 25–9 shows an amendment method for the `Person` class. In this example, subclasses are not read on queries.

Example 25–9 Configuring Reading Subclasses on Queries

```
...
public static void addToPersonDescriptor(Descriptor descriptor) {
    descriptor.getInheritancePolicy().dontReadSubclassesOnQueries();
}
...
```

Configuring Inheritance for a Child (Branch or Leaf) Class Descriptor

Inheritance describes how a derived (child) class inherits the characteristics of its superclass (parent). When you designate a class as a child, you must also specify the descriptor that represents the child's parent in your inheritance hierarchy.

Table 25–39 summarizes which descriptors support child inheritance configuration.

Table 25–21 Descriptor Support for Child Inheritance Configuration

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors	✔	✔
Object-Relational Descriptors		✔
EIS Descriptors	✔	✔
XML Descriptors	✔	✔

For more information about inheritance, see "Descriptors and Inheritance" on page 23-3.

For more information about configuring inheritance for a parent (root) class descriptor, see "Configuring Inheritance for a Parent (Root) Descriptor" on page 25-52.

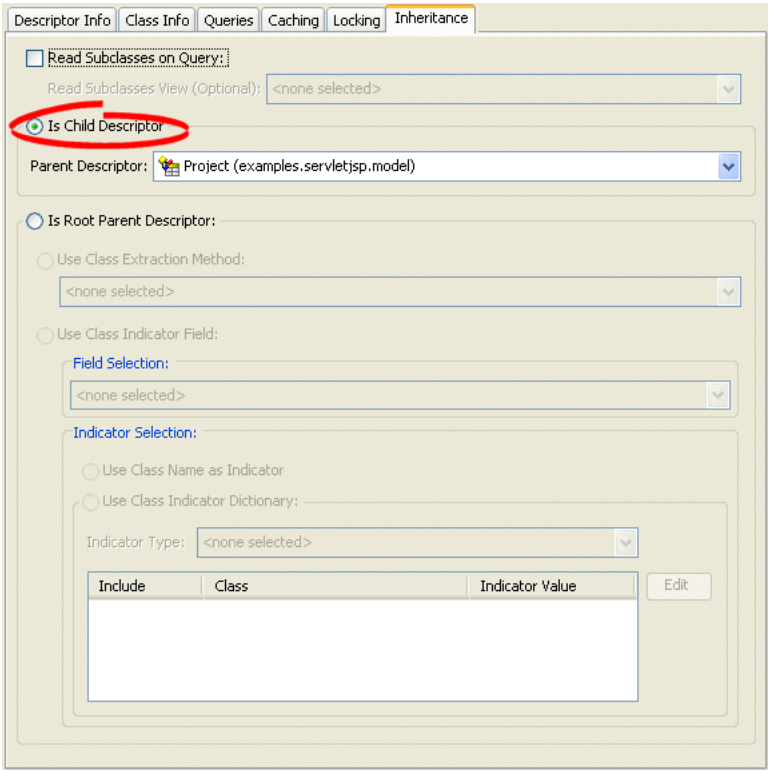
Using TopLink Workbench

To create a child (branch or leaf class) for an inheritance, use this procedure.

1. In the **Navigator**, select the descriptor you wish to specify as a child.
2. Choose the **Inheritance** tab in the **Property** window.

   If the **Inheritance** tab is not visible, right-click the descriptor and choose **Select Advanced Properties** > **Inheritance**.
3. Select the **Is Child Descriptor** option to specify this descriptor is a child class. The **Parent Descriptor** list is now enabled and the class indicator information is disabled.

Figure 25–32 Inheritance Tab, Child Descriptor Option



Use the following information to enter data in each child descriptor field on the tab:

Field	Description
Is Child Descriptor	Specify that this descriptor is a child class to be used in a branch or leaf.
Parent Descriptor	Use the list to select the parent of this descriptor. See "Descriptors and Inheritance" on page 23-3 for more information.

Using Java

Using Java, you can configure an inheritance child descriptor using `InheritancePolicy` method `setParentClass` as Example Example 25–10 shows.

Example 25–10 Configuring an Inheritance Child Descriptor

```
descriptor.getInheritancePolicy().setParentClass(ChildClass.class);
```

Configuring Inheritance for a Parent (Root) Descriptor

Inheritance describes how a derived (child) class inherits the characteristics of its superclass (parent). When you designate a class as a parent, you can configure how TopLink handles the class's inheritance hierarchy.

Table 25–24 summarizes which descriptors support parent inheritance configuration.

Table 25–22 Descriptor Support for Parent Inheritance Configuration

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors	✔	✔
Object-Relational Descriptors		✔
EIS Descriptors	✔	✔
XML Descriptors	✔	✔

For more information about configuring inheritance for a child (branch or leaf) class descriptor, see "Configuring Inheritance for a Child (Branch or Leaf) Class Descriptor" on page 25-51.

For more information, see "Descriptors and Inheritance" on page 23-3.

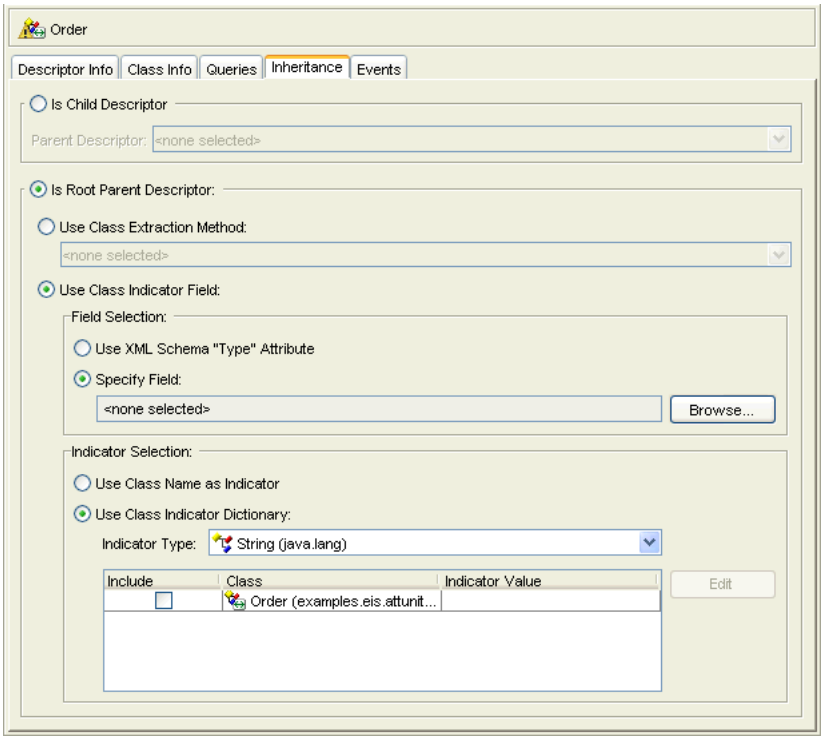
Using TopLink Workbench

To create a root class for an inheritance, use this procedure.

1. In the **Navigator**, select the descriptor you wish to specify as the root.
2. Choose the **Inheritance** tab in the **Property** window.

   If the **Inheritance** tab is not visible, right-click the descriptor and choose **Select Advanced Properties** > **Inheritance**.
3. Select the **Is Root Parent Descriptor** option to specify this descriptor is a root class.

Figure 25–33 Inheritance Tab, Configuring Inheritance for a Root Descriptor



Use this table to complete the following root descriptor field on the Inheritance tab:

Field	Description
Is Root Parent Descriptor	Select this option to specify this descriptor as the root (parent) of the inheritance hierarchy.
Use Class Extraction Method	Choose this option to specify a class indicator using a class extraction method, and select your static class extraction method from the list. For more information, see "Using Class Extraction Methods" on page 23-15.
Use Class Indicator Field	Choose this option to specify a class indicator using a class indicator field. For more information, see "Using Class Indicator Fields" on page 23-14.
Field Selection	Choose the field to use as the class indicator field.
Use XML Schema "Type" Attribute[1]	Select this option to use the type attribute specified in the XML schema for this descriptor's reference class.
Specify Field	For a relational descriptor, select the field of the database table associated with this descriptor (see "Configuring Associated Tables" on page 26-2). For an EIS root descriptor (using XML records) or an XML descriptor, click **Browse** to select an element attribute or text node.
Indicator Selection	Choose between using a class name as the class indicator field value or specifying specific class indicator field values for each (nonabstract) child class.
Use Class Name as Indicator	Choose this option to use class names as the class indicator field value.
Use Class Indicator Dictionary	Choose this option to specify specific class indicator field values for each (nonabstract) child class. When you choose this option, you must specify the data type of the class indicator field and the specific class indicator field values for each (nonabstract) child class.
Indicator Type	Select the data type from the list to specify the data type of the class indicator field. To specify the specific class indicator field values for each (nonabstract) child class, click **Edit** and enter the appropriate value for each child class.

[1] EIS root (see "EIS Root Descriptors" on page 24-5) or XML descriptors (see "XML Descriptors" on page 23-12) only.

Using Java

Create a descriptor amendment method ("Configuring Amendment Methods" on page 25-81) to customize the root class descriptor's `InheritancePolicy` using `InheritancePolicy` methods `setParentClass`, `setClassIndicatorFieldName`, `addClassIndicator`,

`useClassNameAsIndicator` and `setClassExtractionMethodName`, as required.

Example 25–13 shows amendment methods for the `Person` and `Student` classes where `Student` extends `Person` in a relational project. In this example, a class indicator field is used (see "Using Class Indicator Fields" on page 23-14).

Example 25–11 Configuring Inheritance for a Relational Root Class

```
...
public static void addToPersonDescriptor(Descriptor descriptor) {
    descriptor.getInheritancePolicy().setClassIndicatorFieldName("CLIENT_TYPE");
    descriptor.getInheritancePolicy().addClassIndicator("P");
}

public static void addToStudentDescriptor(Descriptor descriptor) {
    descriptor.getInheritancePolicy().setParentClass(Person.class);
    descriptor.getInheritancePolicy().setClassIndicatorFieldName("CLIENT_TYPE");
    descriptor.getInheritancePolicy().addClassIndicator("S");
}
...
```

If you are using a class-extraction method (see "Using Class Extraction Methods" on page 23-15), you may also need to use `InheritancePolicy` methods `setOnlyInstancesExpression` and `setWithAllSubclassesExpression` (see "Configuring Inheritance Expressions for a Parent (Root) Class Descriptor" on page 25-55).

Example 25–13 shows amendment methods for the `Person` and `Student` classes where `Student` extends `Person` in an EIS project using XML records. In this example, a class indicator field is used (see "Using Class Indicator Fields" on page 23-14).

Example 25–12 Configuring Inheritance for an EIS Root Class

```
...
public static void addToPersonDescriptor(Descriptor descriptor) {
    descriptor.getInheritancePolicy().setClassIndicatorField(
        new XMLField("@CLIENT_TYPE")
    );
    descriptor.getInheritancePolicy().addClassIndicator("P");
}

public static void addToStudentDescriptor(Descriptor descriptor) {
    descriptor.getInheritancePolicy().setParentClass(Person.class);
    descriptor.getInheritancePolicy().setClassIndicatorField(
        new XMLField("@CLIENT_TYPE")
    );
    descriptor.getInheritancePolicy().addClassIndicator("S");
}
...
```

Configuring Inheritance Expressions for a Parent (Root) Class Descriptor

If your class uses inheritance (see "Understanding Descriptors and Inheritance" on page 23-12) with a class extraction method (see "Using Class Extraction Methods" on page 23-15) you must provide TopLink with expressions to correctly filter sibling instances for all classes that share a common table.

Table 25–24 summarizes which descriptors support inheritance expression configuration.

Table 25–23 Descriptor Support for Inheritance Expression Configuration

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors		✔
Object-Relational Descriptors		✔
EIS Descriptors		
XML Descriptors		

Figure 25–34 shows a typical inheritance hierarchy. In this example, instances of both `Person` and `Student` are stored in the same `PERSON` table as Figure 25–35 shows: an instance of `Person` has a `null` value for `STUDENT_NUMBER`. Instances of `Company` are stored in a separate `COMPANY` table.

Figure 25–34 Example Inheritance Hierarchy

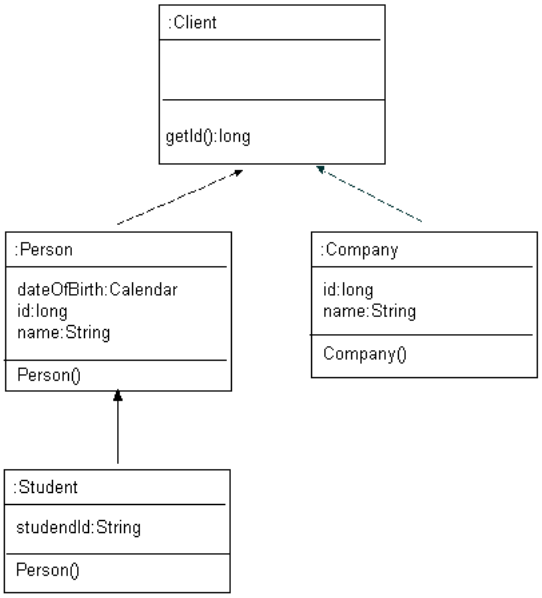


Figure 25–35 PERSON Table

DOB	ID	NAME	STUDENT_NUMBER
1964-08-24	123	Joan Smith	
1983-02-01	456	Jane Doe	4821

Queries on inheritance classes that share a common table, such as `Person` and `Student`, must filter out their sibling instances. TopLink performs this filtering using the `Expression` instances returned by the descriptor's `InheritancePolicy` methods `getOnlyInstancesExpression` and `getWithAllSubclassesExpression`.

Queries on a class that has its own table for its specific data, such as `Company`, and does not share this table with any sibling classes, do not require these expressions.

If you use a class indicator type field (see "Using Class Indicator Fields" on page 23-14), TopLink automatically generates the required expressions.

If you use a class extraction method (see "Using Class Extraction Methods" on page 23-15), you must provide TopLink with an expressions to correctly filter sibling instances for all classes that share a common table.

For concrete classes, you must define an only- instances expression.

For branch classes, you must define a with-all-subclasses expression.

When TopLink queries for a leaf class, it uses the only- instances expression to filter out any sibling classes.

When TopLink queries for a root or branch class whose subclasses do not define their own tables, it uses the with-all-subclasses expression. This is also the case when a subclass view is used (see "Configuring Reading Subclasses on Queries" on page 25-49).

When querying for a root or branch class that has subclasses that span multiple tables, a query is performed for each concrete class in the inheritance hierarchy using the only- instances expression to filter sibling classes.

When a class extraction method is used the only-instances expression is used to determine if a class is concrete. If a class does not require an only instances expression, do not enable reading subclasses on queries (see "Configuring Reading Subclasses on Queries" on page 25-49), otherwise TopLink will assume that the class has no instances and it will skip that class on queries.

For more information about inheritance expressions, see "Specifying Expressions for Only-Instances and With-All-Subclasses" on page 23-16.

Using Java

Create a descriptor amendment method ("Configuring Amendment Methods" on page 25-81) to customize the root class descriptor's `InheritancePolicy` using `InheritancePolicy` methods `setOnlyInstancesExpression` and `setWithAllSubclassesExpression`, as required.

Example 25–13 shows amendment methods for the `Person` and `Student` descriptors based on the class hierarchy shown in Figure 25–34 and the database table shown in Figure 25–35.

Example 25–13 Configuring Only-Instances Expressions

```
...
// Only-instances expression for Person
public static void addToPersonDescriptor(Descriptor descriptor) {
    ExpressionBuilder builder = new ExpressionBuilder();
    descriptor.getInheritancePolicy().setOnlyInstancesExpression(
        builder.getField("STUDENT_NUMBER").isNull()
    );
}
// Only-instances expression for Student
public static void addToStudentDescriptor(Descriptor descriptor) {
    ExpressionBuilder builder = new ExpressionBuilder();
    descriptor.getInheritancePolicy().setOnlyInstancesExpression(
        builder.getField("STUDENT_NUMBER").notNull()
    );
}
...
```

Example 25–14 shows amendment methods for the `Bicycle` and `NonFueledVehicle` descriptors based on the class hierarchy shown in Figure 23–2 if the vehicle hierarchy stored all of the classes in a single vehicle table, and there was not a class indicator, but a class extraction method instead.

Example 25–14 Configuring Only-Instances and With-All-Subclasses Expressions

```
// Bicycle amemndment
public static void addToBicycleDescriptor(Descriptor descriptor) {
    ExpressionBuilder builder = new ExpressionBuilder();
    descriptor.getInheritancePolicy().setOnlyInstancesExpression(
        builder.getField("BICYCLE_DESCR").notNull()
    );
}

// NonFueldVehicle ammendment
public static void addToNonFueledVehicleDescriptor(Descriptor descriptor) {
    ExpressionBuilder builder = new ExpressionBuilder();
    descriptor.getInheritancePolicy().setWithAllSubclassesExpression(
        builder.getField("FUEL_TYPE").isNull()
    );
}
```

Configuring Inherited Attribute Mapping in a Subclass

If you are defining the descriptor for a class that inherits attributes from another class, then you can create mappings for those attributes. If you remap an attribute that was already mapped in the superclass, then the new mapping applies to the subclass only. Any other siblings that inherit the attribute are unaffected.

If you leave inherited attributes unmapped, TopLink uses the mapping (if any) from the superclass if the superclass's descriptor has been designated as the parent descriptor.

Table 25–24 summarizes which descriptors support inherited attribute mapping configuration.

Table 25–24 Descriptor Support for Inherited Attribute Mapping Configuration

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors	✔	✔
Object-Relational Descriptors		✔
EIS Descriptors	✔	✔
XML Descriptors	✔	✔

For more information, see "Descriptors and Inheritance" on page 23-3.

Using TopLink Workbench

To map inherited attributes, use this procedure:

1. In the **Navigator**, right-click a descriptor and choose **Map Inherited Attributes** > *selected class* from the context menu or choose **Selected** > **Map Inherited Attributes** from the menu.

The mappings list now includes all the attributes from the superclass of this class.

2. Map any desired attributes. See Chapter 31, "Creating a Mapping" for more information.

Using Java

Using Java, attributes inherited by a subclass from a superclass will be visible and you can always create a mapping to these inherited attributes.

Configuring a Domain Object Method as an Event Handler

You can associate a domain object method with any of the descriptor events shown in Table 25–26. You can register any domain object method that:

- Is public.
- Returns void.
- Takes a single parameter of type `DescriptorEvent`

Table 25–25 summarizes which descriptors support domain object method event handler configuration.

Table 25–25 Descriptor Support for Domain Object Method Event Handler Configuration

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors	✔	✔
Object-Relational Descriptors		✔
EIS Descriptors	✔	✔
XML Descriptors		

For example, you can add a method `handlePostDelete` (that is public, returns void, and takes a single parameter of type `DescriptorEvent`) to your `Employee` object to handle `PostDeleteEvent` descriptor events. After you register that method with the `DescriptorEventManager` owned by the `Employee` object's descriptor as the handler for `PostDeleteEvent` descriptor events, whenever the Oracle TopLink runtime performs a post-delete operation on an instance of the `Employee` object, the runtime dispatches a `PostDeleteEvent` to the `handlePostDelete` method on the instance of the `Employee` object associated with that `PostDeleteEvent`.

The **Descriptor Event ID** column in Table 25–26 lists the `DescriptorEventManager` field name used to identify a particular event. The `DescriptorEvent` method `getEventCode` returns this value. For example:

```
if (descriptorEvent.getEventCode() == DescriptorEventManager.PreUpdateEvent) {
    // descriptorEvent represents a pre-update event
}
```

Table 25–26 Descriptor Events

Category	Descriptor Event ID	Description
Delete	`PreDeleteEvent`	Occurs before an object is deleted from the data source.
	`AboutToDeleteEvent`	Occurs when an object is deleted from the data source.
	`PostDeleteEvent`	Occurs after an object is deleted from the data source.

Table 25–26 (Cont.) Descriptor Events

Category	Descriptor Event ID	Description
Insert	`PreInsertEvent`	Occurs before an object is inserted in the data source.
	`AboutToInsertEvent`	Occurs when an object is inserted in the data source.
	`PostInsertEvent`	Occurs after an object is inserted into the data source.
Post-X	`PostBuildEvent`	Occurs after an object is built from the data source.
	`PostCloneEvent`	Occurs after an object has been cloned into a unit of work.
	`PostMergeEvent`	Occurs after an object has been merged from a unit of work.
	`PostRefreshEvent`	Occurs after an object is refreshed from the data source.
Update	`PreUpdateEvent`	Occurs before an object is updated in the data source. This may be called in a unit of work even if the object has no changes and does not require updating.
	`AboutToUpdateEvent`	Occurs when an object is updated in the data source. This method is called only if the object has changes in the unit of work.
	`PostUpdateEvent`	Occurs after an object is updated in the data source.
Write	`PreWriteEvent`	Occurs before an object is inserted or updated in the data source. This occurs before `PreInsertEvent` and `PreUpdateEvent`.
	`PostWriteEvent`	Occurs after an object is inserted or updated in the data source. This occurs after `PostInsertEvent` and `PostUpdateEvent`.

Alternatively, you can configure a descriptor event listener as an event handler (see "Configuring a Descriptor Event Listener as an Event Handler" on page 25-62).

Using TopLink Workbench

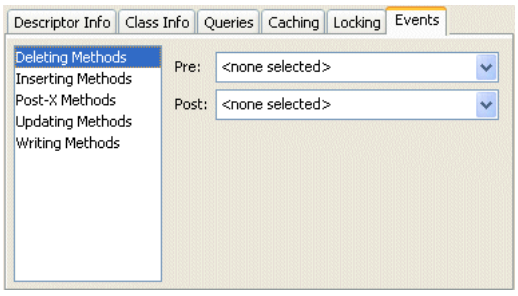
To select event methods, use this procedure:

1. Select a descriptor in the **Navigator**. Its properties appear in the Editor.

   If the **Events** advanced property is not visible for the descriptor, then right-click the descriptor and choose **Select Advanced Properties** > **Events** from context menu or from the **Selected** menu.

2. Click the **Event** tab in the **Editor**.

Figure 25–36 Events Tab



3. Select the appropriate method category from the list on the left.

Use this table to enter data in the following fields to select the appropriate domain object method:

Category	Option	Description
Deleting Methods	**Pre**	Select the domain object method that is invoked on an instance of its reference class *before* the instance is deleted from the data source.
	Post	Select the domain object method that is invoked on an instance of its reference class *after* the instance is deleted from the data source.
Inserting Methods	**Pre**	Select the domain object method that is invoked on an instance of its reference class *before* the instance is inserted in the data source.
	About To	Select the domain object method that is invoked on an instance of its reference class when the instance is inserted in the data source.
	Post	Select the domain object method that is invoked on an instance of its reference class *after* the instance is inserted into the data source.
Post-X Methods	**Build**	Select the domain object method that is invoked on an instance of its reference class *after* the instance is *built* from the data source.
	Clone	Select the domain object method that is invoked on an instance of its reference class *after* the instance is *cloned* into a unit of work.
	Merge	Select the domain object method that is invoked on an instance of its reference class *after* the instance is *merged* from a unit of work.
	Refresh	Select the domain object method that is invoked on an instance of its reference class *after* the instance is *refreshed* from the data source.
Updating Methods	**Pre**	Select the domain object method that is invoked on an instance of its reference class *before* the instance is updated in the data source. This may be called in a unit of work even if the object has no changes and does not require updating.
	About to	Select the domain object method that is invoked on an instance of its reference class when the instance is updated in the data source. This method is called *only* if the object has changes in the unit of work.
	Post	Select the domain object method that is invoked on an instance of its reference class *after* the instance is updated in the data source.
Writing Methods	**Pre**	Select the domain object method that is invoked on an instance of its reference class *before* the instance is inserted or updated in the data source. **Note:** This occurs before Pre-Insert and Pre-Update event methods are invoked.

Category	Option	Description
	Post	Select the domain object method that is invoked on an instance of its reference class *after* the instance is inserted or updated in the data source. **Note:** This occurs after Post-Insert or Post-Update event methods are invoked.

Using Java

Example 25–15 shows a domain object class with method `handlePostDelete` defined to handle `PostDeleteEvent` descriptor events. Example 25–16shows how to register this method as the `PostDeleteEvent` event handler. Whenever the TopLink runtime performs a post-delete operation on an instance of `Employee`, the runtime will dispatch a `PostDeleteEvent` to the `DescriptorEventManager` owned by the `Employee` object's descriptor. The `DescriptorEventManager` will then invoke the `handlePostDelete` method on the instance of `Employee` associated with that `PostDeleteEvent`.

Example 25–15 Domain Object Method as a Descriptor Event Handler

```
public class Employee {
    // domain object methods
    ...
    public void handlePostDelete(DescriptorEvent event) {
        // handler implementation
    }
}
```

Example 25–16 Registering a Domain Object Method as a Descriptor Event Handler

```
employeeDescriptor.getEventManager().setPostDeleteSelector("handlePostDelete");
```

Configuring a Descriptor Event Listener as an Event Handler

You can create your own `DescriptorEventListner` and register it with a `DescriptorEventManger` in a descriptor amendment method. You can also configure a `DescriptorEventListner` to be notified of events through the Java event model.

You can register any object that implements the `DescriptorEventListener` interface with the `DescriptorEventManager` owned by a domain object's descriptor to handle any descriptor event type (see Table 25–28). To quickly implement this interface, you can extend abstract class `DescriptorEventAdapter` and override only the methods for the events you are interested in.

Table 25–27 summarizes which descriptors support descriptor event listener configuration.

Table 25–27 Descriptor Support for Descriptor Event Listener Configuration

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors	✔	✔
Object-Relational Descriptors		✔

Table 25–27 (Cont.) Descriptor Support for Descriptor Event Listener Configuration

Descriptor	Using TopLink Workbench	Using Java
EIS Descriptors	✔	✔
XML Descriptors		

For example, you create a `DescriptorEventListener` to handle `PostBuildEvent` descriptor events for `Employee` objects. After you register this `DescriptorEventListener` with the `DescriptorEventManager` owned by the `Employee` object's descriptor, whenever the TopLink runtime performs a post-build operation on an instance of `Employee`, the runtime dispatches a `PostBuilEvent` to the event listener's `postBuild` method.

Table 25–28 lists the `DescriptorEventListener` methods associated with each descriptor event. The **Descriptor Event Listener Method** column lists the `DescriptorEventListener` methods associated with each `DescriptorEvent`.

Table 25–28 Descriptor Events

Category	Descriptor Event Listener Method	Description
Delete	`preDelete`	Occurs before an object is deleted from the data source.
	`aboutToDelete`	Occurs when an object is deleted from the data source.
	`postDelete`	Occurs after an object is deleted from the data source.
Insert	`preInsert`	Occurs before an object is inserted in the data source.
	`aboutToInsert`	Occurs when an object is inserted in the data source.
	`postInsert`	Occurs after an object is inserted into the data source.
Post-X	`postBuild`	Occurs after an object is built from the data source.
	`postClone`	Occurs after an object has been cloned into a unit of work.
	`postMerge`	Occurs after an object has been merged from a unit of work.
	`postRefresh`	Occurs after an object is refreshed from the data source.
Update	`preUpdate`	Occurs before an object is updated in the data source. This may be called in a unit of work even if the object has no changes and does not require updating.
	`aboutToUpdate`	Occurs when an object is updated in the data source. This method is called only if the object has changes in the unit of work.
	`postUpdate`	Occurs after an object is updated in the data source.
Write	`preWrite`	Occurs before an object is inserted or updated in the data source. This occurs before `PreInsertEvent` and `PreUpdateEvent`.
	`postWrite`	Occurs after an object is inserted or updated in the data source. This occurs after `PostInsertEvent` and `PostUpdateEvent`.

Alternatively, you can configure a domain object method as an event handler (see "Configuring a Domain Object Method as an Event Handler" on page 25-59).

Using Java

Example 25–17 shows a `DescriptorEventListener` that handles `PostBuildEvent` descriptor events. Example 25–18 shows how to register this `DescriptorEventListener` with the `Employee` object's descriptor. Whenever the TopLink runtime performs a post-build operation on an instance of `Employee`, the runtime will dispatch a post build event to the corresponding `DescriptorEventListener` method on each registered event listener (in this case, it calls the `postBuild` method).

Example 25–17 DescriptorEventListener

```
public class MyDescriptorEventListener extends DescriptorEventAdapter {
    public void postBuild(DescriptorEvent event) {
        // handler implementation
    }
}
```

Example 25–18 Registering a DescriptorEventListener with the DescriptorEventManager

```
descriptor.getEventManager().addListener(new MyDescriptorEventListener());
```

Configuring Locking Policy

You can configure a descriptor with a locking policy that prevents one user writing over another user's work.

Table 25–29 summarizes which descriptors support locking policies.

Table 25–29 Descriptor Support for Locking Policy

Descriptor	Optimistic Version Locking Policies	Optimistic Field Locking Policies	Pessimistic Locking Policy	Using TopLink Workbench	Using Java
Relational Descriptors[1]	✔	✔	✔	✔	✔
Object-Relational Descriptors	✔	✔	✔		✔
EIS Descriptors[2]	✔		✔	✔	✔
XML Descriptors					

[1] Relational class descriptors only (see "Relational Class Descriptors" on page 24-2).

[2] EIS root descriptors only (see "EIS Root Descriptors" on page 24-5).

Oracle recommends that you use a locking policy. You should use a locking policy in any multiuser environment to prevent one user writing over another user's changes. Although locking can be particularly important if multiple servers or multiple applications access the same data, even in a single server application, the same locking issue still exists. In a multiple-server environment, locking is still relevant even if your application uses cache refreshing or cache coordination.

If you are building a three-tier application, in order to correctly lock an object, you must obtain the lock before the object is sent to client to be edited. The type of locking you choose has an influence on how you can achieve this (see "Locking in a Three-Tier Application" on page 23-22).

Using TopLink Workbench

To specify a descriptor's locking policy, use this procedure:

1. In the **Navigator**, select a relational or EIS root descriptor.

   If the **Locking** advanced property is not visible for the descriptor, right-click the descriptor and choose **Select Advanced Properties > Locking** from the context menu or from the **Selected** menu.

2. Click the **Locking** tab.

Figure 25–37 Locking Tab for a Descriptor

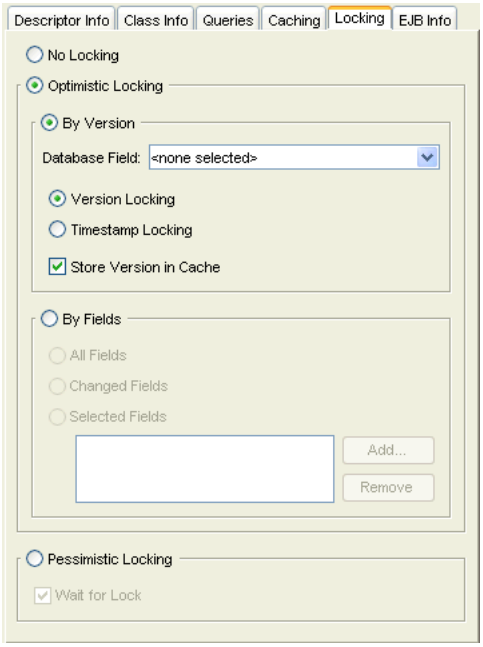
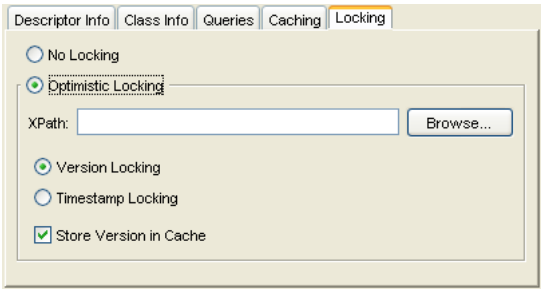


Figure 25–38 Locking Tab for an EIS Root Descriptor



Use this table to enter data in the following fields on the tab of the appropriate type:

Field	Description
Optimistic Locking	Specify that the descriptor uses optimistic locking.
By Version	Specify to use optimistic locking, based on versioning.
Database Field	Select the database field that contains the version value used for optimistic locking. This field appears for relational descriptors only.

Field	Description
XPath	Click **Browse** to define the path to the element or attribute that stores the version value. This field appears for EIS root descriptors only. Ensure that the attribute's type corresponds to the type of locking policy you choose (`numeric` for **Version Locking** and `timestamp` for **Timestamp Locking**).
Version Locking	Specify that the descriptor uses numeric version locking. The version field (defined by the **Database Field**, for relational descriptors, or the **XPath**, for EIS root descriptors) must be a `numeric` type
Timestamp Locking	Specify that the descriptor uses time stamp version locking, based on time stamp. The version field (defined by the **Database Field**, for relational descriptors, or the **XPath**, for EIS root descriptors) must be a `timestamp` type.
Store Version in Cache	Specify whether or not you want to store the version information in the cache. If you choose not to define a mapping for the version field, then you must enable this option to configure the descriptor to store the version value in the Oracle TopLink cache. If you choose to define a mapping for the version field, then you must disable this option in order to store the version value in the object. For more information, see "Optimistic Locking in a Three-Tier Application" on page 23-22.
By Fields[1]	Specify to use optimistic locking, based on database fields. These fields appear for relational descriptors only.
All Fields	Select *all* fields for optimistic locking.
Changed Fields	Select *only the changed* fields for optimistic locking.
Selected Fields	Click **Add** to select *specific* database fields for optimistic locking.
Pessimistic Locking	Specify to use pessimistic locking for this descriptor. This applies only to descriptors that have had EJB information configured for them (see "Configuring a Descriptor With EJB Information" on page 25-45).
Wait for Lock	Specify whether or not TopLink should wait for a data source lock. When not selected, the thread of execution will immediately throw a `DatabaseException` if it cannot acquire a read lock on the object. When selected, the thread of execution will wait indefinitely until the read lock is released, at which time, it will attempt to acquire it. Use this option with care as it can lead to application deadlocks.

[1] You cannot use field locking with the `AttributeChangeTrackingPolicy` (see "Attribute Change Tracking Policy" on page 97-8).

Using Java

This section describes the following:

- Configuring an Optimistic Locking Policy
- Configuring Optimistic Locking Policy Cascading
- Configuring a Pessimistic Locking Policy

Configuring an Optimistic Locking Policy

Use the `ClassDescriptor` method `setOptimisticLockingPolicy` to set an instance of the appropriate optimistic field locking policy:

- `FieldsLockingPolicy`
- `AllFieldsLockingPolicy`
- `ChangedFieldsLockingPolicy`
- `SelectedFieldsLockingPolicy`
- `VersionLockingPolicy`
- `TimestampLockingPolicy`

Use the `ClassDescriptor` method `getOptimisticLockingPolicy` to get the selected locking policy type and configure it.

Configuring Optimistic Locking Policy Cascading

If you are using a `VersionLockingPolicy`, you can enable cascading to configure TopLink to automatically force a version field update on a parent object when its privately owned child object's version field changes. Use `VersionLockingPolicy` method `setIsCascaded` passing in a `boolean` of `true` to enable cascading, or `false` to disable cascading.

For more information, see "Optimistic Version Locking Policies and Cascading" on page 23-19.

Configuring a Pessimistic Locking Policy

You can configure a descriptor with a `PessimisticLockingPolicy` only when using a `CMPPolicy`. That is, you only can configure a `PessimisticLockingPolicy` for descriptors that support EJB information (see "Configuring a Descriptor With EJB Information" on page 25-45) in a CMP project.

Instantiate a `CMPPolicy` and use `CMPPolicy` method `setPessimisticLockingPolicy` to set an instance of `PessimisticLockingPolicy`. Then use the `ClassDescriptor` method `setCMPPolicy` to set the `CMPPolicy`.

Configuring Returning Policy

Using a `ReturningPolicy`, you can obtain field values from the data source when inserting or updating an object. TopLink uses the values that the data source returns to update the object attributes that map to these fields. You can specify which fields to return for inserts and updates. For insert fields, you can also specify whether or not to include the field value in the insert operation.

A `ReturningPolicy` is useful when the data source provides default or initial field values through defaults, triggers, or stored procedures. You can also use a

`ReturningPolicy` to allow the data source to assign a sequence or primary key value.

Any object attribute that you do not configure in a descriptor's `ReturningPolicy` receives the default behavior: in the context of a unit of work, if the attribute has changed, its value is written to the database. If the SQL statement invokes a trigger or stored procedure that modifies the database field, the database generated value is not reflected by the object.

Use caution when deciding on whether or not to use a `ReturningPolicy`, as doing so may effect insert or update performance and is not compatible with batch writing (see "Batch Writing" on page 11-15).

By default, you can use a `ReturningPolicy` with an Oracle Database, in which case, TopLink uses the Oracle `RETURNING` clause (see "Using TopLink Workbench" on page 25-68).

You can use a `ReturningPolicy` with a non-Oracle database if you configure your descriptor's insert or update query to use a stored procedure that returns the desired returned values as output parameters (see "Using Java" on page 25-69).

Table 25–39 summarizes which descriptors support returning policy configuration.

Table 25–30 Descriptor Support for Fetch Group Configuration

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors	✔	✔
Object-Relational Descriptors		✔
EIS Descriptors[1]	✔	✔
XML Descriptors		

[1] EIS root descriptors only (see "EIS Root Descriptors" on page 24-5).

Using TopLink Workbench

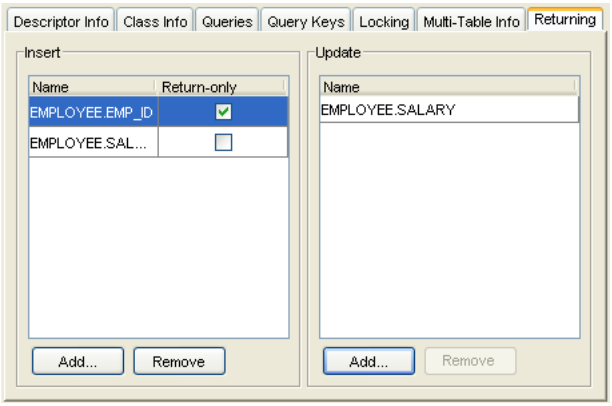
To specify the return policy for a descriptor, use this procedure:

1. Select a descriptor in the **Navigator**. Its properties appear in the Editor.

   If the **Returning** advanced property is not visible for the descriptor, right-click the descriptor and choose **Select Advanced Properties** > **Returning** from the context menu or from the **Selected** menu.

2. Click the **Returning** tab in the **Editor**.

Figure 25–39 Returning Tab



Use the following information to enter data in each field on the tab:

Field	Description
Insert	These options apply to insert operations:
Name	Click **Add** to add a database field to this `ReturningPolicy` for insert operations.
Return-only	When selected, TopLink only returns a value for this field; it will not include the field in the insert. When not selected, TopLink returns a value for this field and includes the value in the insert.
Update	These options apply to update operations:
Name	Click **Add** to add a database field to this `ReturningPolicy` for update operations

To remove a database field from the descriptor's `ReturningPolicy`, select the field in the **Insert** or **Update** window and click **Remove**.

> **Note:** If you are using TopLink Workbench, you cannot configure a returning policy for an attribute mapped with a transformation mapping (see "Transformation Mapping" on page 33-15).

Using Java

You use a `ReturningPolicy` to configure how TopLink handles returning with the attributes of an object on a field-by-field basis. Table 25–31 describes the `ReturnPolicy` methods you use to tell TopLink how to handle a particular database field. Each method takes a `String` or a `DatabaseField` type parameter as field name.

Table 25–31 Return Policy Methods

Method	Applies to SQL Statements of Type...	Writes Current Value of Field to Database?	Returns Database-Generated Result?
`addFieldForInsert`	INSERT	Yes	Yes
`addFieldForInsertReturnOnly`	INSERT	No	Yes

Table 25–31 (Cont.) Return Policy Methods

Method	Applies to SQL Statements of Type...	Writes Current Value of Field to Database?	Returns Database-Generated Result?
`addFieldForUpdate`	UPDATE	Yes	Yes

You configure a descriptor with a `ReturningPolicy` using `ClassDescriptor` method `setReturningPolicy`.

Configuring Instantiation Policy

The TopLink runtime instantiates new instances of a class according to the instantiation policy you configure on the class's descriptor.

Table 25–32 summarizes which descriptors support an instantiation policy.

Table 25–32 Descriptor Support for Instantiation Policy

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors	✔	✔
Object-Relational Descriptors		✔
EIS Descriptors	✔	✔
XML Descriptors	✔	✔

You can specify one of the following types of instantiation policy:

- Default: TopLink creates a new instance of a class by calling the class's default constructor.
- Method: TopLink creates a new instance of a class by calling a public static method that you define on the class descriptor.
- Factory: TopLink creates a new instance of a class by calling the appropriate methods on a separate class that you implement according to the Factory design pattern.

Using TopLink Workbench

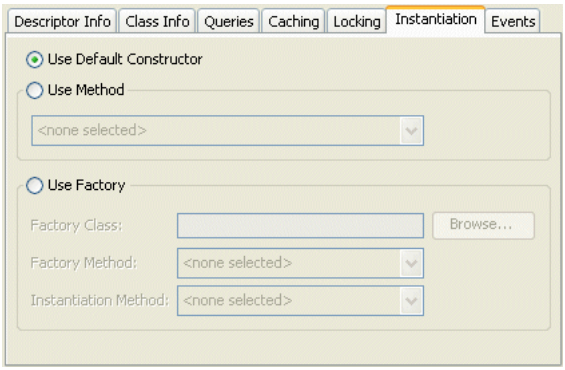
To set the instantiation policy for a descriptor, use this procedure:

1. In the **Navigator**, select a descriptor.

   If the Instantiation advanced property is not visible for the descriptor, right-click the descriptor and choose **Select Advanced Properties** > **Instantiation** from the context menu or from the **Selected** menu.

2. Click the **Instantiation** tab.

Figure 25–40 Instantiation Tab



Use the following information to enter data in each field on the tab:

Field	Description
Use Default Constructor	Specify if the default constructor of the class instantiates a new instance.
Use Method	Specify a method to execute to create objects from the database.
Method	Select the name of a method to be executed to create objects from the database. The method must be a public, static method on the descriptor's class and must return a new instance of the object.
Use Factory	Specify an object factory method.
Factory Class	Select the class of the factory object that creates the new instances.
Factory Method	Select the method to be used to obtain a factory object. Choose `<nothing>` to use the default constructor.
Instantiation Method	Select the method to be called on the factory object to obtain a new instance that will be populated with data from the data source.

Using Java

Use one of the following `ClassDescriptor` methods to set the appropriate type of instantiation policy:

- `useDefaultConstructorInstantiationPolicy`
- `useMethodInstantiationPolicy`
- `useFactoryInstantiationPolicy`

Configuring Copy Policy

The TopLink unit of work feature must be able to produce an exact copy (clone) persistent objects. Table 25–33 summarizes which descriptors support a copy policy.

Table 25–33 Configuring Descriptors with a Copy Policy

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors	✔	✔

Table 25–33 (Cont.) Configuring Descriptors with a Copy Policy

Descriptor	Using TopLink Workbench	Using Java
Object-Relational Descriptors		✔
EIS Descriptors	✔	✔
XML Descriptors	✔	✔

TopLink supports the following two ways of copying objects:

- Instantiation policy: By default, TopLink creates a new copy of an object by using the currently configured instantiation policy (see "Configuring Instantiation Policy" on page 25-70).
- Method: TopLink creates a new copy of an object by calling a method on the object that you specify. For example, you can specify the object's `clone` method (or any other appropriate method on the object).

Using TopLink Workbench

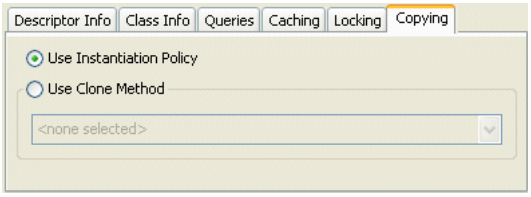
To specify the copy policy for a descriptor, use this procedure:

1. Select a descriptor in the **Navigator**. Its properties appear in the Editor.

   If the **Copying** advanced property is not visible for the descriptor, right-click the descriptor and choose **Select Advanced Properties** > **Copying** from the context menu or from the **Selected** menu.

2. Click the **Copying** tab in the **Editor**.

Figure 25–41 Copying Tab



Use the following information to enter data in each field on the tab:

Field	Description
Use Instantiation Policy	Creates a new instance of the object using the descriptor's instantiation policy (see "Configuring Instantiation Policy" on page 25-70).
Use Clone Method	Specifies whether or not to call the `clone` method of the object. Select a method from the list.

Using Java

Use one of the following `ClassDescriptor` methods to set the appropriate type of copy policy:

- `useCloneCopyPolicy()`: the object must provide a `clone` method
- `useCloneCopyPolicy(java.lang.String cloneMethodName)`
- `useInstantiationCopyPolicy()`

Configuring Change Policy

Use a change policy to specify how TopLink should track changes made to objects after you register them with a unit of work. Table 25–34 summarizes which descriptors support a change policy.

Table 25–34 Descriptor Support for Change Policy

Descriptor	Deferred Change Detection Policy	Object-Level Change Tracking Policy	Attribute Change Tracking Policy	Using TopLink Workbench	Using Java
Relational Descriptors[1]	✔	✔	✔		✔
Object-Relational Descriptors	✔	✔	✔		✔
EIS Descriptors[2]	✔	✔	✔		✔
XML Descriptors					

[1] Relational class descriptors only (see "Relational Class Descriptors" on page 24-2).
[2] EIS root descriptors only (see "EIS Root Descriptors" on page 24-5).

By default, TopLink uses the deferred change detection policy.

TopLink supports alternative change policies (policies other than `DeferredChangeDetectionPolicy`) for attributes that use a subset of the mappings that TopLink supports (see "Change Policy Mapping Support" on page 97-9).

For CMP applications and EJB 3.0 persistent applications deployed to OC4J TopLink automatically uses the attribute change tracking policy.

For more information, see "Unit of Work and Change Policy" on page 97-6.

Using Java

This section describes how to configure a descriptor with a change policy using Java, and how to implement persistent classes for those change policies that are intrusive. It includes information on configuring the following:

- Configuring Deferred Change Detection Policy
- Configuring Object Change Tracking Policy
- Configuring Attribute Change Tracking Policy

Configuring Deferred Change Detection Policy

The `DeferredChangeDetectionPolicy` provides good unit of work commit performance for a wide range of object change characteristics. It is the default change policy. For more information, see "Deferred Change Detection Policy" on page 97-7).

Because it is the default, you do not need to explicitly configure this policy.

To configure TopLink to use a `DeferredChangeDetectionPolicy`, create a descriptor amendment method (see "Configuring Amendment Methods" on page 25-81) that sets the change policy, as Example 25–19 illustrates.

Configuring Object Change Tracking Policy

The `ObjectChangeTrackingPolicy` provides improved unit of work commit performance for objects with few attributes, or with many attributes and many

changed attributes. For more information, see "Object-Level Change Tracking Policy" on page 97-7).

For CMP applications and EJB 3.0 persistent applications deployed to an application server, for which TopLink provides CMP integration (see "Application Server Support" on page 7-1), when you configure a entity bean's descriptor with an `ObjectLevelChangeTrackingPolicy`, TopLink automatically generates code of a concrete subclass to implement the TopLink `ChangeTracker` interface at deploy time. Configuring an `ObjectLevelChangeTrackingPolicy` prevents TopLink from automatically applying an `AttributeChangeTrackingPolicy` (see "Configuring Attribute Change Tracking Policy" on page 25-75).

To configure TopLink to use an `ObjectChangeTrackingPolicy`, use this procedure:

1. Create a descriptor amendment method (see "Configuring Amendment Methods" on page 25-81) that sets the change policy, as Example 25–19 illustrates.

Example 25–19 Setting the ObjectChangeTrackingPolicy

```
descriptor.setObjectChangePolicy(new ObjectChangeTrackingPolicy());
```

2. For plain Java objects, code each of your persistent classes to implement the `ChangeTracker` interface as Example 25–20 illustrates.

Example 25–20 Implementing the ChangeTracker Interface for the ObjectChangeTrackingPolicy

```
public class Employee implements ChangeTracker {

    PropertyChangeListener listener;

    public PropertyChangeListener getTopLinkPropertyChangeListener() {
      return listener;
    }

    public void setTopLinkPropertyChangeListener(PropertyChangeListener listener)
    {
      this.listener = listener;
    }
...
    public void setFirstName(String firstName) {
      propertyChange("firstName", getFirstName(), firstName);
      this.firstName = firstName;
    }
...
    public void propertyChange(String propertyName, Object oldValue, Object newValue) {
        if (listener != null) {
            if (oldValue != newValue) {
                listener.propertyChange(
                    new PropertyChangeEvent(
                        this, propertyName, oldValue, newValue
                    )
                );
            }
        }
    }
}
```

Configuring Attribute Change Tracking Policy

The `AttributeChangeTrackingPolicy` provides improved unit of work commit performance for objects with many attributes and few changed attributes. In general, this is the most efficient change policy. It is the default change policy for EJB 3.0 persistent applications and EJB 2.*n* CMP applications deployed to OC4J. For more information, see "Attribute Change Tracking Policy" on page 97-8).

> **Note:** You cannot use the `AttributeChangeTrackingPolicy` if you are using any instance of `FieldsLockingPolicy` (see "Optimistic Field Locking Policies" on page 23-20).

When you deploy a TopLink-enabled EJB 3.0 persistent application or EJB 2.*n* CMP application to OC4J, TopLink automatically configures your persistent classes to use the `AttributeChangeTrackingPolicy` and, using bytecode weaving (EJB 3.0) or code generation (EJB 2.*n*), configures your persistence classes to implement the TopLink `ChangeTracker` interface. In this case, you do not need to explicitly configure this change policy.

To configure TopLink to use an `AttributeChangeTrackingPolicy` for plain Java objects or other application servers, use this procedure:

1. Create a descriptor amendment method (see "Configuring Amendment Methods" on page 25-81) that sets the change policy as Example 25–21 illustrates.

Example 25–21 Setting the DeferredChangeDetectionPolicy

```
descriptor.setObjectChangePolicy(new AttributeChangeTrackingPolicy());
```

2. Code each of your persistent classes to implement the `ChangeTracker` interface as Example 25–22 illustrates.

Example 25–22 Implementing the ChangeTracker Interface for the AttributeChangeTrackingPolicy

```
public class Employee implements ChangeTracker {

    PropertyChangeListener listener;

    public PropertyChangeListener getTopLinkPropertyChangeListener() {
      return listener;
    }

    public void setTopLinkPropertyChangeListener(PropertyChangeListener listener) {
      this.listener = listener;
    }
...
    public void setId(long id) {
        _id_change(id);
    }

    protected void id_change(long id) {
        if (listener != null && this.id != id) { // throttle unnecessary events
listener.propertyChange(
                new PropertyChangeEvent(
                    this,
                    "id",
                    new Long(getId()), // primitives must be wrapped
                    new Long(id)
                )
            );
```

```
        }
        this.id = id;
    }
...
}
```

Configuring a History Policy

If you want to use historical sessions (see "Historical Sessions" on page 72-25) to execute historical queries (see "Historical Queries" on page 93-21) against a historical schema of your own design, configure your descriptors with a TopLink `HistoryPolicy` that describes your historical schema.

If you are using an Oracle database platform for Oracle9*i* Database Server (or later), you can query the historical versions of objects automatically maintained by the Oracle database without the need for a history policy. For more information, see "Configuring Historical Sessions Using an Oracle Platform" on page 78-1.

Table 25–35 summarizes which descriptors support history policy configuration.

Table 25–35 Descriptor Support for History Policy Configuration

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors		✔
Object-Relational Descriptors		✔
EIS Descriptors		
XML Descriptors		

There are many ways to configure a historical database schema. TopLink supports several historical schema configurations that you can describe with a `HistoryPolicy` (see "Historical Session Limitations" on page 72-25).

Example Historical Schema

As shown in Table 25–36 and Table 25–37, a common approach is to define a special history table to store past versions of an object: one history table for each regular table that requires historical persistence. The history table typically has the same fields as the corresponding regular table plus fields (such as row start and end) used to define an interval that represents the life time of a particular version.

> **Note:** TopLink assumes that the current version of an object corresponds to the historical table row whose row end field is `NULL`.

TopLink will include the history tables described by a `HistoryPolicy` when you execute a historical query.

Table 25–36 shows the schema for an `EMPLOYEE` table. The table currently contains one `EMPLOYEE` instance.

Table 25–36 Example Table for EMPLOYEE

EMP_ID	F_NAME	L_NAME	SALARY
1	Jane	Doe	55000

Table 25–37 shows one possible history table EMPLOYEE_HIST that stores historical versions of employees. The table contains the current EMPLOYEE (the version with a ROW_END value of NULL) and one historical version.

Table 25–37 Example History Table EMPLOYEE_HIST

EMP_ID	F_NAME	L_NAME	SALARY	ROW_START	ROW_END
1	Jane	Doe	50000	29/08/2004	31/08/2004
1	Jane	Doe	55000	31/08/2004	NULL

Because every record has a start and end interval, the history table can store multiple versions of the same object (with the same primary key). The unique identifier of a particular version is given by the existing primary key, plus the value of the start field. For example, in Table 25–37, the unique identifier of the current version is given by (EMP_ID, START) = (1, 31/08/2004).

Using Java

Example 25–23 shows how to describe the schema shown in Table 25–36 and Table 25–37 using the TopLink HistoryPolicy:

Example 25–23 HistoryPolicy for One Table

```
HistoryPolicy policy = new HistoryPolicy();
policy.addStartFieldName("ROW_START");
policy.addEndFieldName("ROW_END");
policy.addHistoryTableName("EMPLOYEE", "EMPLOYEE_HIST");
// Assuming database triggers or stored procedures update history tables
policy.setShouldHandleWrites(false);

employeeDescriptor.setHistoryPolicy(policy);
```

You can specify more than one table with a HistoryPolicy as shown in Example 25–24. In this example, all history tables have a start field named ROW_START but the EMPLOYEE_HIST and SALARY_HIST tables have different end fields. To avoid ambiguity, the end field names are prefixed with their respective history table names.

Example 25–24 HistoryPolicy for Multiple Tables

```
HistoryPolicy policy = new HistoryPolicy();
policy.addStartFieldName("ROW_START");
policy.addEndFieldName("EMPLOYEE_HIST.ROW_END");
policy.addEndFieldName("SALARY_HIST.VALID_UNTIL");
policy.addHistoryTableName("EMPLOYEE", "EMPLOYEE_HIST");
policy.addHistoryTableName("SALARY", "SALARY_HIST");
// Assuming database triggers or stored procedures update history tables
policy.setShouldHandleWrites(false);

employeeDescriptor.setHistoryPolicy(policy);
```

Configuring Write Responsibility

Use HistoryPolicy method setShouldHandleWrites to specify whether or not TopLink is responsible for writing data to history tables. By default, setShouldHandleWrites is set to true.

Either the database or TopLink can be responsible for writing data to the history tables.

Typically, the database is responsible for writing data to history tables by way of triggers or stored procedures that customize create, insert, and delete operations to modify both the regular table and the history table appropriately.

Alternatively, you can make TopLink responsible by customizing insert, update, and delete queries using the `DescriptorQueryManager` (see "Configuring Default Query Implementations" on page 93-23).

Configuring Wrapper Policy

TopLink lets you use wrappers (or proxies) in cases where the persistent class is not the same class that is to be presented to users.

For example, in the EJB specification prior to 3.0, the entity bean class (the class that implements `javax.ejb.EntityBean`) is persistent, but is hidden from users who interact with a class that implements `javax.ejb.EJBObject` (local or remote interface class). In this example, the `EJBObject` acts as a proxy (or wrapper) for the `EntityBean`.

In cases where such a wrapper is used, TopLink continues to make the class specified in the descriptor persistent, but returns the appropriate instance of the wrapper whenever a persistent object is requested.

Table 25–38 summarizes which descriptors support a wrapper policy.

Table 25–38 Descriptor Support for Wrapper Policy

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors		✔
Object-Relational Descriptors		✔
EIS Descriptors		✔
XML Descriptors		

Use a wrapper policy to tell TopLink how to create wrappers for a particular persistent class, and how to obtain the underlying persistent object from a given wrapper instance.

If you specify a wrapper policy, TopLink uses the policy to *wrap* and *unwrap* persistent objects as required:

- Wrapper policies implement the interface `oracle.toplink.descriptors.WrapperPolicy`.
- A wrapper policy is specified by setting the wrapper policy for the TopLink descriptor.
- By default, no wrapper policy is used (the wrapper policy for a descriptor is null by default).

Note: Wrapper policies are advanced TopLink options. Using a wrapper policy may not be compatible with some TopLink Workbench features.

For CMP descriptors, the EJB wrapper policy is automatically configured during deployment, so does not need to be set in the descriptor (see "Configuring a Descriptor With EJB Information" on page 25-45).

For BMP descriptors, you must set a `BMPWrapperPolicy` on the descriptor. The `BMPWrapperPolicy` includes the bean's information including the `bean-name`, `primary-key-class`, `home-interface`, and `remote-interface`.

Wrapper policies cannot be set using TopLink Workbench and can be set only using Java code (see "Using Java" on page 25-79).

Using Java

Use the `ClassDescriptor` method `setWrapperPolicy` to set the appropriate instance of `WrapperPolicy`.

Example 25–25 shows how to amend a BMP descriptor with the required BMP information.

Example 25–25 Configuring a BMP Wrapper Policy

```
public static void addToDescriptor(ClassDescriptor descriptor) {
    BMPWrapperPolicy policy = new BMPWrapperPolicy(
        "employee",
        EmployeeHome.class,
        EmployeePK.class,
        Employee.class,
        new Hashtable());
    descriptor.setWrapperPolicy(policy);
```

Configuring Fetch Groups

By default, when you execute an object-level read query for a particular object class, TopLink returns all the persistent attributes mapped in the object's descriptor. With this single query, all the object's persistent attributes are defined, and calling their `get` methods returns the value directly from the object.

When you are interested in only some of the attributes of an object, it may be more efficient to return only a subset of the object's attributes using a fetch group.

Using a fetch group, you can define a subset of an object's attributes and associate the fetch group with either a `ReadObjectQuery` or `ReadAllQuery` query. When you execute the query, TopLink retrieves only the attributes in the fetch group. TopLink automatically executes a query to fetch all the attributes excluded from this subset when and if you call a `get` method on any one of the excluded attributes.

You can define more than one fetch group for a class. You can optionally designate at most one such fetch group as the default fetch group. If you execute either a `ReadObjectQuery` or `ReadAllQuery` query without specifying a fetch group, TopLink will use the default fetch group, unless you configure the query otherwise (see "Configuring Default Fetch Group Behavior" on page 96-3).

Currently, you can use fetch groups only in CMP projects for EJB objects. For non-CMP classes, use partial object querying (see "Partial Object Queries" on page 93-11).

Before using fetch groups, Oracle recommends that you perform a careful analysis of system use. In many cases, the extra queries required to load attributes not in the fetch group could well offset the gain from the partial attribute loading. For more

information about optimizing read performance, see "Read Optimization Examples" on page 11-19.

Table 25–39 summarizes which descriptors support fetch group configuration.

Table 25–39 Descriptor Support for Fetch Group Configuration

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors		✔
Object-Relational Descriptors		✔
EIS Descriptors		
XML Descriptors		

This section describes how to create a fetch group, store it in a descriptor, and optionally designate a fetch group as the default fetch group for its descriptor reference class.

For more information, see the following:

- "Fetch Groups" on page 23-5
- "Fetch Groups and Object-Level Read Queries" on page 93-13
- "Configuring Default Fetch Group Behavior" on page 96-3

Using Java

To configure a fetch group, use a descriptor amendment method (see "Configuring Amendment Methods" on page 25-81) as Example 25–26 shows.

Example 25–26 Configuring a Fetch Group

```
//Create a FetchGroupManager for the descriptor
descriptor.setFetchGroupManager(new FetchGroupManager());
// Create a FetchGroup
FetchGroup group = new FetchGroup("nameOnly");
// Add attributes to FetchGroup. Alternatively, use
// FetchGroup method addAttributes, passing in a Set of String attribute names
group.addAttribute("firstName");
group.addAttribute("lastName");
// Add the FetchGroup to the FetchGroupManager
descriptor.getFetchGroupManager().addFetchGroup(group);
//Set the default fetch group
descriptor.getFetchGroupManager().setDefaultFetchGroup(group);
```

Each instance of `FetchGroup` that you store in a descriptor must be configured with a fetch group name that is unique for that descriptor (that is, each descriptor owns a set of named fetch groups).

When configuring fetch groups, note that the primary key fields and other required fields (such as inheritance type and optimistic lock version) are always included in all fetch groups.

Fetch groups can include direct and relationship attributes. Including a relationship attribute in a fetch group does not cause the relationship to be joined or instantiated: joining and indirection are set independently of fetch groups.

After you add a fetch group to a descriptor, you can configure a `ReadObjectQuery` or `ReadAllQuery` query with this fetch group by name (`nameOnly`) or rely on TopLink to use this fetch group by default. For more information, see "Using Queries With Fetch Groups" on page 96-2.

Configuring Amendment Methods

Some TopLink descriptor features cannot be configured from TopLink Workbench. To use these features, you must write a Java method to amend the descriptor after it is loaded as part of the project. This method must have the following characteristics:

- Be public static.
- Take a single parameter of type `oracle.toplink.descriptors.ClassDescriptor`.

In the implementation of this method, you can configure advanced features of the descriptor using any of the public descriptor and mapping API.

Table 25–40 summarizes which descriptors support amendment methods.

Table 25–40 Descriptor Support for Amendment Methods

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors	✔	
Object-Relational Descriptors		
EIS Descriptors	✔	
XML Descriptors	✔	

This section describes how to associate an amendment method with a descriptor.

For more information about how to implement an amendment method, see "Amendment and After-Load Methods" on page 23-5.

To customize a session, use a session customizer class (see "Configuring Customizer Class" on page 74-13).

Using TopLink Workbench

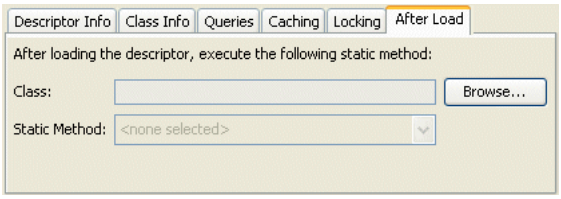
To use an amendment method with a descriptor (after it is loaded as part of the project) use this procedure:

1. Select a descriptor in the **Navigator**. Its properties appear in the Editor.

   If the **After load** advanced property is not visible for the descriptor, right-click the descriptor and choose **Select Advanced Properties** > **After Load** from context menu or from the **Selected** menu.

2. Click the **After Load** tab in the **Editor**.

Figure 25–42 After Load Tab



Field	Description
Class	Click **Browse** and choose the class of the method to execute.
Static Method	Use the **Static Method** list to choose the static method to execute at run time, after loading the descriptor. The method must be public static and take a single attribute of type `oracle.toplink.descriptors.ClassDescriptor`.

26

Configuring a Relational Descriptor

This chapter describes how to configure a relational descriptor.

For more information, see the following:

- "Descriptor Creation Overview" on page 24-1
- "Relational Descriptors" on page 23-11

Relational Descriptor Configuration Overview

Table 26–1 lists the default configurable options for a relational descriptor.

Table 26–1 Configurable Options for Relational Descriptor

Option	Type	TopLink Workbench	Java
"Configuring Associated Tables" on page 26-2	Basic	✔	✔
"Configuring Primary Keys" on page 25-3	Basic	✔	✔
"Configuring Sequencing at the Descriptor Level" on page 26-3	Basic	✔	✔
"Configuring Read-Only Descriptors" on page 25-5	Advanced	✔	✔
"Configuring Unit of Work Conforming at the Descriptor Level" on page 25-6	Advanced	✔	✔
"Configuring Descriptor Alias" on page 25-7	Advanced	✔	
"Configuring Descriptor Comments" on page 25-9	Advanced	✔	
"Configuring Classes" on page 4-41	Basic	✔	
"Configuring Named Queries at the Descriptor Level" on page 25-10	Advanced	✔	✔
"Configuring Custom SQL Queries for Basic Persistence Operations" on page 26-6	Advanced	✔	✔
"Configuring Query Timeout at the Descriptor Level" on page 25-26	Advanced	✔	✔
"Configuring Cache Refreshing" on page 25-27	Advanced	✔	✔
"Configuring Query Keys" on page 25-30	Advanced	✔	✔
"Configuring Interface Query Keys" on page 25-33	Advanced	✔	✔
"Configuring Interface Alias" on page 26-10	Advanced	✔	✔
"Configuring Cache Type and Size at the Descriptor Level" on page 25-35	Advanced	✔	✔
"Configuring Cache Isolation at the Descriptor Level" on page 25-37	Advanced	✔	✔
"Configuring Cache Coordination Change Propagation at the Descriptor Level" on page 25-40	Advanced	✔	✔

Table 26–1 (Cont.) Configurable Options for Relational Descriptor

Option	Type	TopLink Workbench	Java
"Configuring Cache Expiration at the Descriptor Level" on page 25-42	Advanced	✔	✔
"Configuring Cache Existence Checking at the Descriptor Level" on page 25-43	Advanced	✔	✔
"Configuring a Descriptor With EJB Information" on page 25-45	Advanced	✔	✔
"Configuring a Relational Descriptor as a Class or Aggregate Type" on page 26-11	Advanced	✔	✔
"Configuring Reading Subclasses on Queries" on page 25-49	Advanced	✔	✔
"Configuring Inheritance for a Child (Branch or Leaf) Class Descriptor" on page 25-51	Advanced	✔	✔
"Configuring Inheritance for a Parent (Root) Descriptor" on page 25-52	Advanced	✔	✔
"Configuring Inheritance Expressions for a Parent (Root) Class Descriptor" on page 25-55	Advanced	✔	✔
"Configuring Inherited Attribute Mapping in a Subclass" on page 25-58	Advanced	✔	✔
"Configuring Multitable Information" on page 26-13	Advanced	✔	✔
"Configuring a Domain Object Method as an Event Handler" on page 25-59	Advanced	✔	✔
"Configuring a Descriptor Event Listener as an Event Handler" on page 25-62	Advanced	✔	✔
"Configuring Locking Policy" on page 25-64	Advanced	✔	✔
"Configuring Returning Policy" on page 25-67	Advanced	✔	✔
"Configuring Instantiation Policy" on page 25-70	Advanced	✔	✔
"Configuring Copy Policy" on page 25-71	Advanced	✔	✔
"Configuring Change Policy" on page 25-73	Advanced		✔
"Configuring a History Policy" on page 25-76	Advanced		✔
"Configuring Wrapper Policy" on page 25-78	Advanced		✔
"Configuring Fetch Groups" on page 25-79	Advanced		✔
"Configuring Amendment Methods" on page 25-81	Advanced	✔	
"Configuring a Mapping" on page 32-1	Basic	✔	✔

Configuring Associated Tables

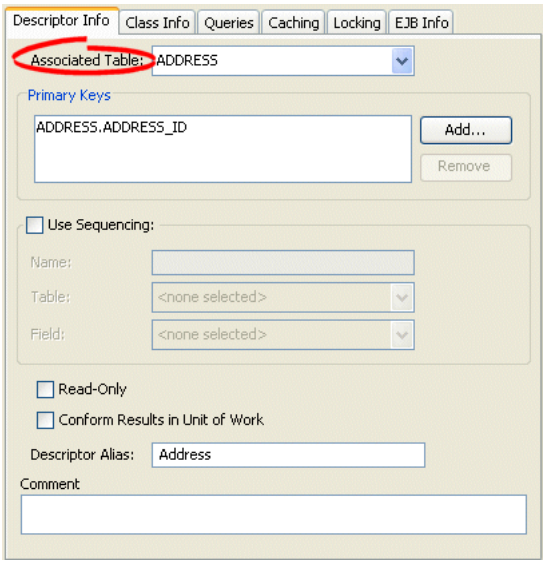
Each relational class descriptor (see "Relational Class Descriptors" on page 24-2) must be associated with a database table for storing instances of that class. This does not apply to relational aggregate descriptors (see "Relational Aggregate Descriptors" on page 24-2).

Using TopLink Workbench

To associate a descriptor with a database table, use this procedure:

1. Select a descriptor in the **Navigator**. Its properties appear in the Editor.
2. Click the **Descriptor Info** tab. The Descriptor Info tab appears.

Figure 26–1 Descriptor Info Tab, Associated Table Options



Use the **Associated Table** list to select a database table for the descriptor. You must associate a descriptor with a database table *before* specifying primary keys.

Using Java

To configure a descriptor's associated table(s) using Java, use `RelationalDescriptor` methods `setTableName` or `addTableName`.

Configuring Sequencing at the Descriptor Level

Sequencing allows TopLink to automatically assign the primary key or ID of an object when the object is inserted.

You configure TopLink sequencing at the project level ("Configuring Sequencing at the Project Level" on page 20-3) or session level (see "Configuring Sequencing at the Session Level" on page 83-4) to tell TopLink how to obtain sequence values: that is, what type of sequences to use.

To enable sequencing, you must then configure TopLink sequencing at the descriptor level to tell TopLink into which table and column to write the sequence value when an instance of a descriptor's reference class is created.

Only descriptors that have been configured with a sequence field and a sequence name will be assigned sequence numbers.

The sequence field is the database field that the sequence number will be assigned to: this is almost always the primary key field (see "Configuring Primary Keys" on page 25-3). The sequence name is the name of the sequence to be used for this descriptor. The purpose of the sequence name depends on the type of sequencing you are using:

When using table sequencing, the sequence name refers to the row's SEQ_NAME value used to store this sequence.

When using Oracle native sequencing, the sequence name refers to the Oracle sequence object that has been created in the database. When using native sequencing on other databases, the sequence name does not have any direct meaning, but should still be set for compatibility.

The sequence name can also refer to a custom sequence defined in the project.

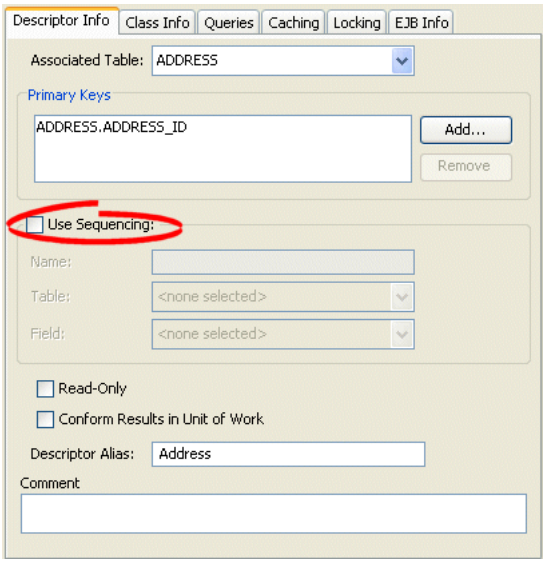
For more information, see "Understanding Sequencing in Relational Projects" on page 17-14.

Using TopLink Workbench

To configure sequencing for a descriptor, use this procedure:

1. Select a descriptor in the **Navigator**. Its properties appear in the Editor.
2. Click the **Descriptor Info** tab. The Descriptor Info tab appears.

Figure 26–2 Descriptor Info Tab, Sequencing Options



Use the following information to specify sequencing options:

Field	Description
Use Sequencing	Specify if this descriptor uses sequencing. If selected, specify the Name, Table, and Field for sequencing.
Name	Enter the name of the sequence. ■ **For table sequencing:** Enter the name of the value in the sequence name column (for default table sequencing, the column named `SEQ_NAME`) of the sequence table (for default table sequencing, the table named `SEQUENCE`) that TopLink uses to look up the corresponding sequence count value (for default table sequencing, the corresponding value in the `SEQ_COUNT` column) for this descriptor's reference class. For more information, see "Table Sequencing" on page 17-16. ■ **For native sequencing (Oracle platform):** Enter the name of the sequence object that Oracle Database creates to manage sequencing for this descriptor's reference class. For more information, see "Native Sequencing With an Oracle Database Platform" on page 17-18 ■ **For native sequencing (non-Oracle platform):** For database compatibility, enter a generic name for the sequence, such as `SEQ`. For more information, see "Native Sequencing With a Non-Oracle Database Platform" on page 17-19.

Field	Description
Table	Specify the name of the database table that contains the field (see **Field**) into which TopLink is to write the sequence value when a new instance of this descriptor's reference class is created. This is almost always this descriptor's primary table.
Field	Specify the name of the field in the specified table (see **Table**) into which TopLink is to write the sequence value when a new instance of this descriptor's reference class is created. This field is almost always the class's primary key (see "Configuring Primary Keys" on page 25-3). ■ **For native sequencing (non-Oracle platform)**: Ensure that your database schema specifies the correct type for this field (see "Native Sequencing With a Non-Oracle Database Platform" on page 17-19).

Using Java

Using Java, you can configure sequencing to use multiple different types of sequence for different descriptors. You configure the sequence objects on the session's login and reference them from the descriptor by their name. The descriptor's sequence name refers to the sequence object's name you register in the session's login.

The following examples assume the session sequence configuration shown in Example 26–1.

Example 26–1 Example Sequences

```
dbLogin.addSequence(new TableSequence("EMP_SEQ", 25));
dbLogin.addSequence(new DefaultSequence("PHONE_SEQ", 30));
dbLogin.addSequence(new UnaryTableSequence("ADD_SEQ", 55));
dbLogin.addSequence(new NativeSequence("NAT_SEQ", 10));
```

Using Java code, you can perform the following sequence configurations:

- Configuring a Sequence by Name
- Configuring the Same Sequence for Multiple Descriptors
- Configuring the Platform Default Sequence

Configuring a Sequence by Name

As Example 26–2 shows, you associate a sequence with a descriptor by sequence name. The sequence `EMP_SEQ` was added to the login for this project in Example 26–1. When a new instance of the `Employee` class is created, the TopLink runtime will use the sequence named `EMP_SEQ` (in this example, a `TableSequence`) to obtain a value for the `EMP_ID` field.

Example 26–2 Associating a Sequence with a Descriptor

```
empDescriptor.setSequenceNumberFieldName("EMP_ID"); // primary key field
empDescriptor.setSequenceNumberName("EMP_SEQ");
```

Configuring the Same Sequence for Multiple Descriptors

As Example 26–3 shows, you can associate the same sequence with more than one descriptor. In this example, both the `Employee` descriptor and `Phone` descriptor use the same `NativeSequence`. Having descriptors share the same sequence can

improve pre-allocation performance. For more information on pre-allocation, see "Sequencing and Preallocation Size" on page 17-20.

Example 26–3 Configuring a Sequence for Multiple Descriptors

```
empDescriptor.setSequenceNumberFieldName("EMP_ID"); // primary key field
empDescriptor.setSequenceNumberName("NAT_SEQ");
phoneDescriptor.setSequenceNumberFieldName("PHONE_ID"); // primary key field
phoneDescriptor.setSequenceNumberName("NAT_SEQ");
```

Configuring the Platform Default Sequence

In Example 26–4, you associate a nonexistent sequence (`NEW_SEQ`) with a descriptor. Because you did not add a sequence named `NEW_SEQ` to the login for this project in Example 26–1, the TopLink runtime will create a `DefaultSequence` named `NEW_SEQ` for this descriptor. For more information about `DefaultSequence`, see "Default Sequencing" on page 17-18.

Example 26–4 Configuring a Default Sequence

```
descriptor.setSequenceNumberFieldName("EMP_ID"); // primary key field
descriptor.setSequenceNumberName("NEW_SEQ");
```

Configuring Custom SQL Queries for Basic Persistence Operations

You can use TopLink to define an SQL query for each basic persistence operation (insert, update, delete, read-object, read-all, or does-exist) so that when you query and modify your relational-mapped objects, the TopLink runtime will use the appropriate SQL query instead of the default SQL query.

SQL strings can include any fields that the descriptor maps, as well as arguments. You specify arguments in the SQL string using `#<arg-name>`, such as:

```
select * from EMP where EMP_ID = #EMP_ID
```

The insert and update SQL strings can take any field that the descriptor maps as an argument.

The read-object, delete and does-exist SQL strings can only take the primary key fields as arguments.

The read-all SQL string must return all instances of the class and thus can take no arguments.

You can define a custom SQL string for insert, update, delete, read-object, and read-all using TopLink Workbench (see "Using TopLink Workbench" on page 26-7).

You can define a custom SQL string or `Call` object for insert, update, delete, read-object, read-all, and does-exist using Java (see "Using Java" on page 26-8). Using a `Call`, you can define more complex SQL strings and invoke custom stored procedures.

For CMP projects, the `ejb-jar.xml` file stores query lists. You can define the queries in the file and then read them into TopLink Workbench (see "Reading From the ejb-jar.xml File" on page 18-16), or define them on the **Queries** tab and write them to the file (see "Writing to the ejb-jar.xml File" on page 18-16).

Note: When you customize the update persistence operation for an application that uses optimistic locking (see "Configuring Locking Policy" on page 25-64), 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.

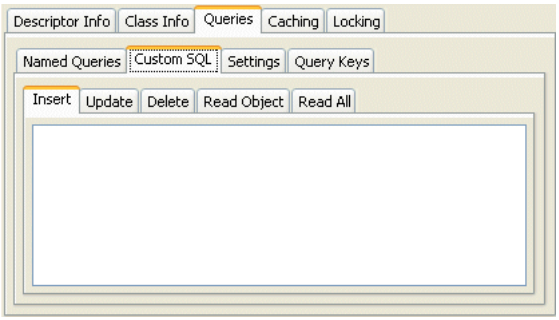
Note: TopLink does not validate the SQL code that you enter. Enter the SQL code appropriate for your database platform (see "Data Source Platform Types" on page 81-3).

Using TopLink Workbench

To configure custom SQL queries for basic persistence operations:

1. In the **Navigator**, select a descriptor in a relational database project.
2. Click the **Queries** tab in the **Editor**.
3. Click the **Custom SQL** tab.

Figure 26–3 Queries, Custom SQL Tab



Click the appropriate SQL function tab and type your own SQL string to control these actions for a descriptor. Use the following information to complete the tab:

Tab	Description
Insert	Defines the insert SQL that TopLink uses to insert a new object's data into the database.

Tab	Description
Update	Defines the update SQL that TopLink uses to update any changed existing object's data in the database. When you define a descriptor's **update** query, you must conform to the following: ■ If the application uses optimistic locking, you must ensure that the row is not written if the version field has changed since the object was read. ■ The update query must increment the version field if the row is written. ■ The update string must maintain the row count of the database.
Delete	Defines the delete SQL that TopLink uses to delete an object.
Read Object	Defines the read SQL that TopLink uses in any `ReadObjectQuery`, whose selection criteria is based on the object's primary key. When you define a descriptor's **read-object** query, your implementation overrides any `ReadObjectQuery`, whose selection criteria is based on the object's primary key. TopLink generates dynamic SQL for all other `Session readObject` method signatures. To customize other `Session readObject` method signatures, define additional named queries and use them in your application instead of the `Session` methods.
Read All	Defines the read-all SQL that TopLink uses when you call `Session` method `readAllObjects(java.lang.Class)` passing in the `java.lang.Class` that this descriptor represents. When you define a descriptor's **read-all** query, your implementation overrides only the `Session` method `readAll(java.lang.Class)`, not the version that takes a `Class` and `Expression`. As a result, this query reads every single instance. TopLink generates dynamic SQL for all other `Session readAll` method signatures. To customize other `Session readAll` method signatures, define additional named queries and use them in your application instead of the Session methods.

Using Java

The `DescriptorQueryManager` generates default SQL for the following persistence operations:

- Insert
- Update
- Delete
- Read-object
- Read-all
- Does-exist

Using Java code, you can use the descriptor query manager to provide custom SQL strings to perform these functions on a class-by-class basis.

Use `ClassDescriptor` method `getQueryManager` to acquire the `DescriptorQueryManager`, and then use the `DescriptorQueryManager` methods that Table 26–2 lists.

Table 26–2 Descriptor Query Manager Methods for Configuring Custom SQL

To Change the Default SQL for ...	Use Descriptor Query Manager Method ...
Insert	`setInsertQuery (InsertObjectQuery query)`
	`setInsertSQLString (String sqlString)`
	`setInsertCall(Call call)`
Update	`setUpdateQuery (UpdateObjectQuery query)`
	`setUpdateSQLString (String sqlString)`
	`setUpdateCall(Call call)`
Delete	`setDeleteQuery (DeleteObjectQuery query)`
	`setDeleteSQLString (String sqlString)`
	`setDeleteCall(Call call)`
Read	`setReadObjectQuery (ReadObjectQuery query)`
	`setReadObjectSQLString (String sqlString)`
	`setReadObjectCall(Call call)`
Read all	`setReadAllQuery (ReadAllQuery query)`
	`setReadAllSQLString (String sqlString)`
	`setReadAllCall(Call call)`
Does exist	`setDoesExistQuery(DoesExistQuery query)`
	`setDoesExistSQLString(String sqlString)`
	`setDoesExistCall(Call call)`

Example 26–5 shows how to implement an amendment method to configure a descriptor query manager to use custom SQL strings. Alternatively, using an `SQLCall`, you can specify more complex SQL strings using features such as in, out, and in-out parameters and parameter types (see "Using SQL Calls" on page 94-19).

Example 26–5 Configuring a Descriptor Query Manager with Custom SQL Strings

```
public static void addToDescriptor(ClassDescriptor descriptor) {

    // Read-object by primary key procedure
    descriptor.getQueryManager().setReadObjectSQLString(
        "select * from EMP where EMP_ID = #EMP_ID"
    );

    // Read-all instances procedure
    descriptor.getQueryManager().setReadAllSQLString(
        "select * from EMP"
    );

    // Insert procedure
    descriptor.getQueryManager().setInsertSQLString(
```

```
        "insert into EMP (EMP_ID, F_NAME, L_NAME, MGR_ID) values (#EMP_ID, #F_
NAME, #L_NAME, #MGR_ID)"
    );

    // Update procedure
    descriptor.getQueryManager().setUpdateSQLString(
        "update EMP set (F_NAME, L_NAME, MGR_ID) values (#F_NAME, #L_NAME, #MGR_
ID) where EMP_ID = #EMP_ID"
    );
}
```

Example 26–6 shows how to implement an amendment method to configure a descriptor query manager to use Oracle stored procedures using a `StoredProcedureCall` (see "Using a StoredProcedureCall" on page 94-21). This example uses output cursors to return the result set (see "Handling Cursor and Stream Query Results" on page 96-15).

Example 26–6 Configuring a Descriptor Query Manager with Custom Stored Procedure Calls

```
public static void addToDescriptor(ClassDescriptor descriptor) {

    // Read-object by primary key procedure
    StoredProcedureCall readCall = new StoredProcedureCall();
    readCall.setProcedureName("READ_EMP");
    readCall.addNamedArgument("P_EMP_ID", "EMP_ID");
    readCall.useNamedCursorOutputAsResultSet("RESULT_CURSOR");
    descriptor.getQueryManager().setReadObjectCall(readCall);

    // Read-all instances procedure
    StoredProcedureCall readAllCall = new StoredProcedureCall();
    readAllCall.setProcedureName("READ_ALL_EMP");
    readAllCall.useNamedCursorOutputAsResultSet("RESULT_CURSOR");
    descriptor.getQueryManager().setReadAllCall(readAllCall );

    // Insert procedure
    StoredProcedureCall insertCall = new StoredProcedureCall();
    insertCall.setProcedureName("INSERT_EMP");
    insertCall.addNamedArgument("P_EMP_ID", "EMP_ID");
    insertCall.addNamedArgument("P_F_NAME", "F_NAME");
    insertCall.addNamedArgument("P_L_NAME", "L_NAME");
    insertCall.addNamedArgument("P_MGR_ID", "MGR_ID");
    descriptor.getQueryManager().setInsertCall(insertCall);

    // Update procedure
    StoredProcedureCall updateCall = new StoredProcedureCall();
    updateCall.setProcedureName("UPDATE_EMP");
    updateCall.addNamedArgument("P_EMP_ID", "EMP_ID");
    updateCall.addNamedArgument("P_F_NAME", "F_NAME");
    updateCall.addNamedArgument("P_L_NAME", "L_NAME");
    updateCall.addNamedArgument("P_MGR_ID", "MGR_ID");
    descriptor.getQueryManager().setUpdateCall(updateCall);
}
```

Configuring Interface Alias

An interface alias allows an interface to be used to refer to a descriptor instead of the implementation class. This can be useful for classes that have public interface and the applications desire to refer to the class using the public interface. Specifying the

interface alias allows any queries executed on a TopLink session to use the interface as the reference class instead of the implementation class.

This section includes information on configuring an interface alias. Interfaces cannot be *created* in TopLink Workbench, you must add the Java package or class to your TopLink Workbench project before configuring it.

Using TopLink Workbench

Use the **Interface Alias** tab to specify a descriptor's alias. Each descriptor can have one interface alias. Use the interface in queries and relationship mappings.

> **Note:** If you use an interface alias, do not associate an interface descriptor with the interface.

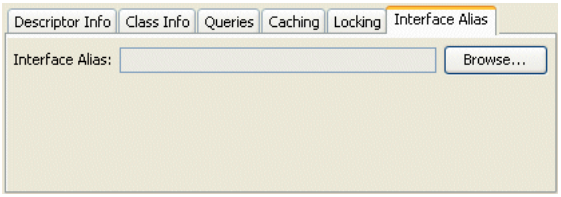
To specify an interface alias, use this procedure:

1. In the **Navigator**, select a descriptor.

   If the **Interface Alias** advanced property is not visible for the descriptor, right-click the descriptor and choose **Select Advanced Properties** > **Interface Alias** from context menu or from the Selected menu.

2. Click the **Interface Alias** tab.

Figure 26–4 Interface Alias Tab



In the **Interface Alias** field, click **Browse** and select an interface.

Using Java

To configure a descriptor with an interface alias using Java, create an amendment method (see "Configuring Amendment Methods" on page 25-81) and use `InterfacePolicy` method `addParentInterface` as Example 26–7 shows.

Example 26–7 Configuring an Interface Alias

```
public static void addToDescriptor(Descriptor descriptor) {
    descriptor.getInterfacePolicy().addParentInterface(MyInterface.class);
}
```

Configuring a Relational Descriptor as a Class or Aggregate Type

By default, when you add a Java class to a relational project (see "Configuring Project Classpath" on page 19-3), TopLink Workbench creates a relational class descriptor for it. A class descriptor is applicable to any persistent object except an object that is owned by another in an aggregate relationship. In this case, you must describe the owned object with an aggregate descriptor. Using a class descriptor, you can configure any relational mapping except aggregate collection and aggregate object mappings.

An aggregate object is an object that is strictly dependent on its owning object. Aggregate descriptors do not define a table, primary key, or many of the standard descriptor options as they obtain these from their owning descriptor. If you want to configure an aggregate mapping to associate data members in a target object with fields in a source object's underlying database tables (see Chapter 41, "Configuring a Relational Aggregate Collection Mapping" and Chapter 43, "Configuring a Relational Aggregate Object Mapping"), you must designate the target object's descriptor as an aggregate.

Alternatively, you can remove the aggregate designation from a relational descriptor and return it to its default type.

You can configure inheritance for a descriptor designated as an aggregate (see "Configuring Inheritance for a Child (Branch or Leaf) Class Descriptor" on page 25-51), however, in this case, *all* the descriptors in the inheritance tree must be aggregates. Aggregate and class descriptors cannot exist in the same inheritance tree. For more information, see "Aggregate and Composite Descriptors and Inheritance" on page 23-17.

If you configure a descriptor as an aggregate, you cannot configure the descriptor with EJB information (see "Configuring a Descriptor With EJB Information" on page 25-45).

For more information, see "Descriptors and Aggregation" on page 23-5.

Using TopLink Workbench

To configure a relational descriptor as class or aggregate, use this procedure.

1. In the **Navigator**, select a relational descriptor.
2. Click the **Class** or **Aggregate** descriptor button on the mapping toolbar.

   You can also select the descriptor and choose **Selected** > **Descriptor Type** > **Class** or **Aggregate** from the menu or by right-clicking on the descriptor in the **Navigator** window and selecting **Descriptor Type** > **Class** or **Aggregate** from the context menu.



3. If you select **Aggregate**, specify each of the aggregate descriptor's attributes as a direct to field mapping. See Chapter 35, "Configuring a Relational Direct-to-Field Mapping" for more information.

Although the attributes of a target class are not mapped directly to a data source until you configure an aggregate object mapping, you must still specify their mapping type in the target class's descriptor. This tells TopLink what type of mapping to use when you do configure the aggregate mapping in the source object's descriptor. For more information, see "Aggregate and Composite Descriptors in Relational Projects" on page 23-5.

Using Java

Using Java, to configure a relational descriptor as an aggregate, use `ClassDescriptor` method `descriptorIsAggregate`.

To configure a relational descriptor for use in an aggregate collection mapping, use `ClassDescriptor` method `descriptorIsAggregateCollection`.

To configure a relational descriptor as a nonaggregate, use `ClassDescriptor` method `descriptorIsNormal`.

Configuring Multitable Information

Descriptors can use multiple tables in mappings. Use multiple tables when either of the following occurs:

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

When a descriptor has multiple tables, you must be able to join a row from the primary table to all the additional tables. By default, TopLink assumes that the primary key of the first, or primary, table is included in the additional tables, thereby joining the tables. TopLink also supports custom methods for joining tables. If the primary key field names of the multiple tables do not match, a foreign key can be used to join the tables. The foreign key can either be from the primary table to the secondary table, or from the secondary table to the primary table, or between two of the secondary tables (see "Using TopLink Workbench" on page 26-13).

For complex multitable situations, a more complex join expression may be required. These include requiring the join to also check a type code, or using an outer-join. TopLink provides support for a multiple-table-join-expression for these cases (see "Using Java" on page 26-14).

Using TopLink Workbench

Use the **Multitable Info** tab to define multiple tables for a descriptor in TopLink Workbench.

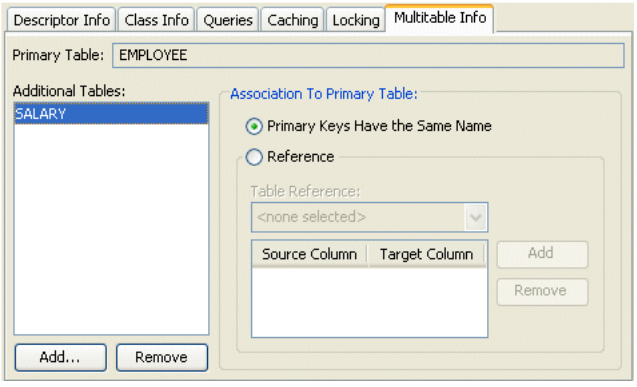
To associate multiple tables with a descriptor, use this procedure.

1. In the **Navigator**, select a descriptor.

   If the **Multitable Info** advanced property is not visible for the descriptor, right-click the descriptor and choose **Select Advanced Properties** > **Multitable Info** from the context menu or from the **Selected** menu.

2. Click the **Multitable Info** tab.

Figure 26–5 Multitable Info Tab



Use the following information to enter data in each field of the tab:

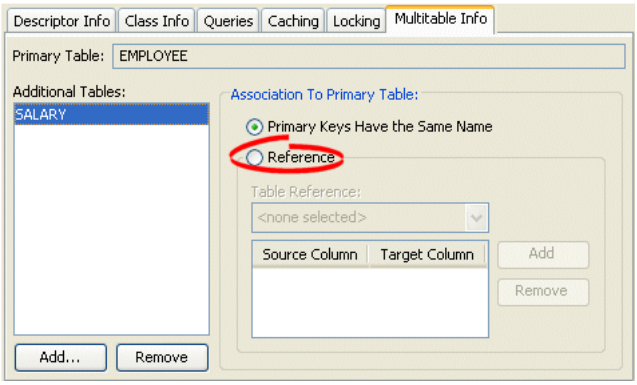
Field	Description
Primary Table	The primary table for this descriptor. This field is for display only.

Field	Description
Additional Tables	Use **Add** and **Remove** to add or remove additional tables.
Association to Primary Table	Specify how each **Additional Table** is associated to the **Primary Table**: ▪ **Primary Keys Have Same Names**–when associating tables by identically named primary keys, TopLink requires no additional configuration. ▪ **Reference**–when associating an additional table to the primary table with a **Reference** (that is, a foreign key), you can specify the **Table Reference**, as well as the **Source** and **Target** fields. Continue with "Associating Tables With References" on page 26-14.

Associating Tables With References

When associating a table using **Reference**, additional options appear. You must choose a reference that relates the correct fields in the primary table to the primary keys in the selected table.

Figure 26–6 Multitable Info Tab, Associated by Reference



Choose a **Table Reference** that defines how the primary keys of the primary table relate to the primary keys of the selected table. Click **Add** to add a primary key association.

Using Java

Using Java, configure a descriptor with multitable information using the following `ClassDescriptor` methods:

- `addTableName`
- `addMultipleTableForeignKeyFieldName`

To specify a complex multiple-table-join-expression, create a descriptor amendment method (see "Configuring Amendment Methods" on page 25-81) and add the join expression using `DescriptorQueryManager` method `setMultipleTableJoinExpression`. For more information, see "Appending Additional Join Expressions" on page 96-4.

27

Configuring an Object-Relational Descriptor

This chapter describes the various components that you must configure to be able to use an object-relational descriptor.

For more information, see the following:

- "Descriptor Creation Overview" on page 24-1
- "Object-Relational Descriptors" on page 23-11

Object-Relational Descriptor Configuration Overview

Table 27–1 lists the configurable options for an object-relational descriptor.

Table 27–1 Configurable Options for Object-Relational Descriptor

Option	Type	TopLink Workbench	Java
"Configuring Field Ordering" on page 27-2	Basic		✔
"Configuring Primary Keys" on page 25-3	Basic		✔
"Configuring Read-Only Descriptors" on page 25-5	Advanced		✔
"Configuring Unit of Work Conforming at the Descriptor Level" on page 25-6	Advanced		✔
"Configuring Query Keys" on page 25-30	Advanced		✔
"Configuring Cache Expiration at the Descriptor Level" on page 25-42	Advanced		✔
"Configuring Amendment Methods" on page 25-81	Advanced		✔
"Configuring Reading Subclasses on Queries" on page 25-49	Advanced		✔
"Configuring Inheritance for a Child (Branch or Leaf) Class Descriptor" on page 25-51	Advanced		✔
"Configuring Inheritance for a Parent (Root) Descriptor" on page 25-52	Advanced		✔
"Configuring Inheritance Expressions for a Parent (Root) Class Descriptor" on page 25-55	Advanced		✔
"Configuring Inherited Attribute Mapping in a Subclass" on page 25-58	Advanced		✔
"Configuring Cache Type and Size at the Descriptor Level" on page 25-35	Advanced		✔
"Configuring a Domain Object Method as an Event Handler" on page 25-59	Advanced		✔
"Configuring a Descriptor Event Listener as an Event Handler" on page 25-62	Advanced		✔
"Configuring Locking Policy" on page 25-64	Advanced		✔

Table 27–1 (Cont.) Configurable Options for Object-Relational Descriptor

Option	Type	TopLink Workbench	Java
"Configuring Copy Policy" on page 25-71	Advanced		✔
"Configuring Instantiation Policy" on page 25-70	Advanced		✔
"Configuring Wrapper Policy" on page 25-78	Advanced		✔
"Configuring a History Policy" on page 25-76	Advanced		✔
"Configuring Returning Policy" on page 25-67	Advanced		✔

Configuring Field Ordering

If your object-relational data source driver uses JDBC indexed arrays, you can specify the order in which TopLink persists object attributes to define the field index.

Using Java

Use `ObjectRelationalDescriptor` method `addFieldOrdering` to specify the field ordering. Example 27–1 shows how to specify the order of the object-relational database fields `OBJECT_ID`, `F_NAME`, and `L_NAME` for the `Employee` descriptor.

Example 27–1 Field Ordering

```
descriptor.addFieldOrdering("ID");
descriptor.addFieldOrdering("F_NAME");
descriptor.addFieldOrdering("L_NAME");
```

28

Configuring an EIS Descriptor

This chapter describes the various components that you must configure in order to use an enterprise information system (EIS) descriptor.

For more information, see the following:

- "Descriptor Creation Overview" on page 24-1
- "EIS Descriptors" on page 23-12

EIS Descriptor Configuration Overview

Table 28–1 lists the default configurable options for an EIS descriptor.

Table 28–1 Configurable Options for EIS Descriptor

Option	Type	TopLink Workbench	Java
"Configuring XML Schema Namespace" on page 4-37	Basic	✔	✔
"Configuring XML Schema Reference" on page 4-36	Basic	✔	✔
"Configuring Schema Context for an EIS Descriptor" on page 28-2	Basic	✔	✔
"Configuring Default Root Element" on page 28-3	Basic	✔	
"Configuring Primary Keys" on page 25-3[1]	Basic	✔	✔
"Configuring Read-Only Descriptors" on page 25-5[1]	Basic	✔	✔
"Configuring Unit of Work Conforming at the Descriptor Level" on page 25-6[1]	Advanced	✔	✔
"Configuring Descriptor Alias" on page 25-7	Advanced	✔	
"Configuring Descriptor Comments" on page 25-9	Advanced	✔	
"Configuring Record Format" on page 28-5	Basic		✔
"Creating Classes" on page 4-40	Basic	✔	
"Configuring Named Queries at the Descriptor Level" on page 25-10[1]	Advanced	✔	✔
"Configuring Custom EIS Interactions for Basic Persistence Operations" on page 28-6[1]	Advanced	✔	✔
"Configuring Cache Refreshing" on page 25-27[1]	Advanced	✔	✔
"Configuring Cache Type and Size at the Descriptor Level" on page 25-35[1]	Advanced	✔	✔
"Configuring Cache Isolation at the Descriptor Level" on page 25-37	Advanced	✔	✔
"Configuring Cache Coordination Change Propagation at the Descriptor Level" on page 25-40	Advanced	✔	✔

Table 28–1 (Cont.) Configurable Options for EIS Descriptor

Option	Type	TopLink Workbench	Java
"Configuring Cache Expiration at the Descriptor Level" on page 25-42	Advanced	✔	✔
"Configuring Cache Existence Checking at the Descriptor Level" on page 25-43	Advanced	✔	✔
"Configuring a Descriptor With EJB Information" on page 25-45	Advanced	✔	✔
"Configuring an EIS Descriptor as a Root or Composite Type" on page 28-8	Basic	✔	✔
"Configuring Inheritance for a Child (Branch or Leaf) Class Descriptor" on page 25-51	Advanced	✔	✔
"Configuring Inheritance for a Parent (Root) Descriptor" on page 25-52	Advanced	✔	✔
"Configuring Inherited Attribute Mapping in a Subclass" on page 25-58	Advanced	✔	✔
"Configuring a Domain Object Method as an Event Handler" on page 25-59	Advanced	✔	✔
"Configuring a Descriptor Event Listener as an Event Handler" on page 25-62	Advanced	✔	✔
"Configuring Locking Policy" on page 25-64[1]	Advanced	✔	✔
"Configuring Returning Policy" on page 25-67	Advanced	✔	✔
"Configuring Instantiation Policy" on page 25-70	Advanced	✔	✔
"Configuring Copy Policy" on page 25-71	Advanced	✔	✔
"Configuring Change Policy" on page 25-73	Advanced		✔
"Configuring Wrapper Policy" on page 25-78	Advanced		✔
"Configuring Amendment Methods" on page 25-81	Advanced	✔	✔
"Configuring a Mapping" on page 32-1	Basic	✔	✔

[1] EIS root descriptors only (see "Configuring an EIS Descriptor as a Root or Composite Type" on page 28-8).

Configuring Schema Context for an EIS Descriptor

TopLink Workbench uses the schema context to associate the class that the EIS descriptor describes with a simple or complex type in one of the schemas associated with the EIS project (see "Configuring XML Schema Reference" on page 4-36). This allows TopLink Workbench to display the appropriate attributes available for mapping in that context.

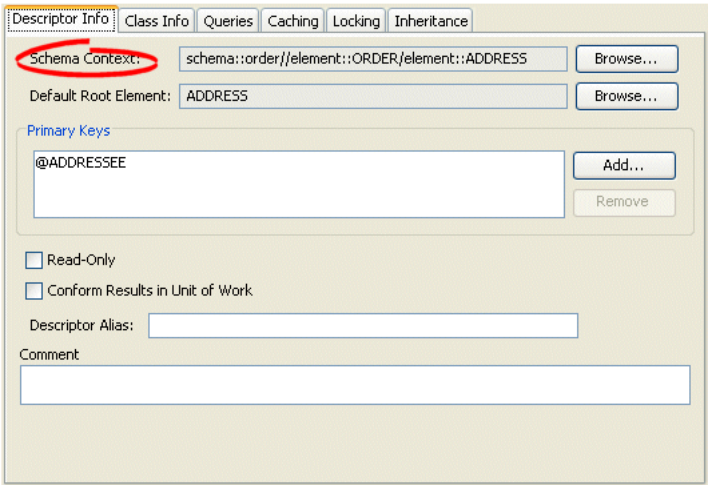
You must configure the schema context for an EIS root descriptor (see "Configuring an EIS Descriptor as a Root or Composite Type" on page 28-8) only if you are using TopLink Workbench.

Using TopLink Workbench

To associate an EIS descriptor with a simple or complex type in this project's schema, use this procedure:

1. Select an EIS descriptor in the **Navigator**. Its properties appear in the Editor.
2. Click the **Descriptor Info** tab. The Descriptor Info tab appears.

Figure 28–1 Descriptor Info Tab, Schema Context Option

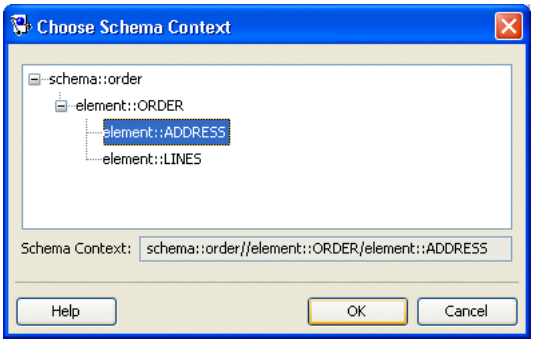


Click **Browse** to select the schema element to associate with this descriptor. For more information, see "Choosing a Schema Context" on page 28-3.

Choosing a Schema Context

Use the Choose Schema Context dialog box to select a specific schema element (such as when mapping an element).

Figure 28–2 Choose Schema Context Dialog Box



Select the schema element and click **OK**.

Using Java

For an EIS descriptor, the TopLink runtime does not need the schema context: the runtime can determine the schema context based on the mappings you configure on the descriptor. No further configuration is required.

Configuring Default Root Element

You must configure the default root element for an EIS root descriptor (see "EIS Root Descriptors" on page 24-5) so that the TopLink runtime knows the data source data type associated with the class the descriptor describes. Descriptors used only in composite relationship mappings do not require a default root element.

Note: Although you select an element from your project's schema to configure this attribute, you are choosing the element's simple or complex type.

For more information, see "Default Root Element" on page 23-9.

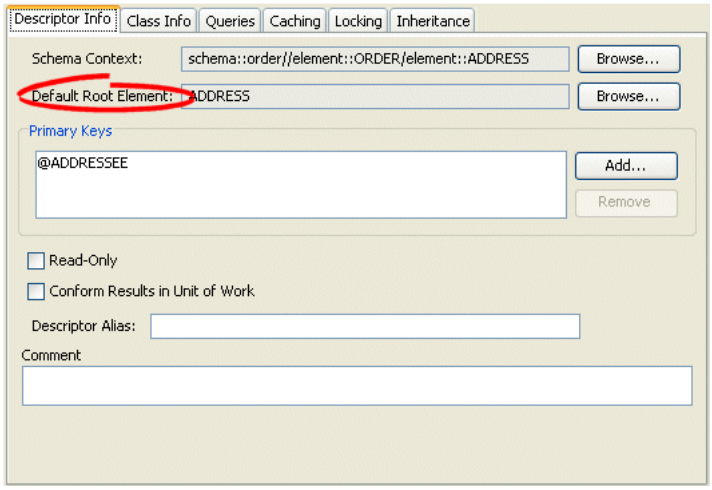
Using TopLink Workbench

When you create an EIS project using TopLink Workbench, you must use XML records. Consequently, you must configure a default root element so that TopLink Workbench knows what element to start with when persisting an instance of the class that the EIS descriptor describes.

To specify a schema element as the default root element for the descriptor, use this procedure:

1. Select a descriptor in the **Navigator**. Its properties appear in the Editor.
2. Click the **Descriptor Info** tab. The Descriptor Info tab appears.

Figure 28–3 Descriptor Info Tab, Default Root Element Option



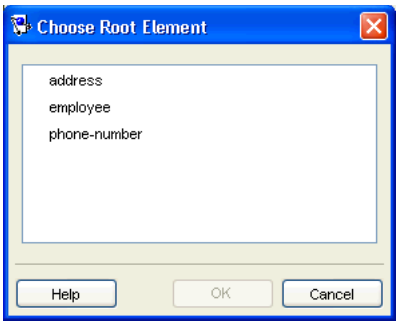
Use the **Default Root Element** option to select the root element for this descriptor.

Click **Browse** to select the schema element to identify as the root element. See "Choosing a Root Element" on page 28-4 for more information.

Choosing a Root Element

Use the Choose Root Element dialog box to select a specific root element.

Figure 28–4 Choose Root Element Dialog Box



Select the root element and click **OK**.

Using Java

When you create an EIS project using Java code, use the `EISDescriptor` method `setDataTypeName` to specify the XML schema complex type name (if you are using XML records) or the J2C record name (if you are using indexed or mapped records) corresponding to the class that the EIS descriptor describes. For more information, see Oracle TopLink API Reference.

Configuring Record Format

The EIS descriptor record format determines the EIS record type to which the descriptor's EIS mappings map.

When you create an EIS project using TopLink Workbench, TopLink configures all EIS descriptors with a record format of XML.

When you create an EIS project in Java, you can configure the EIS descriptor record type to any of the supported types, as Table 28–2 shows.

Table 28–2 EIS Record Formats

EISDescriptor Method	EIS Record Type
`useMappedRecordFormat`	All EIS mappings owned by this descriptor map to EIS mapped records.
`useIndexedRecordFormat`	All EIS mappings owned by this descriptor map to EIS indexed records.
`useXMLRecordFormat`	All EIS mappings owned by this descriptor map to EIS XML records. If you use the XML record format, you must specify one or more XML schemas in your EIS project (see "Importing an XML Schema" on page 4-34). The TopLink runtime performs XML data conversion based on one or more XML schemas. In an EIS XML project, TopLink Workbench does not directly reference schemas in the deployment XML, but instead exports mappings configured with respect to the schemas you specify. For information on TopLink support for XML namespaces, see "XML Namespaces" on page 17-5.

For more information, see "EIS Record Type" on page 53-2.

Using Java

To configure the EIS record format for an EIS descriptor, use one of the `EISDescriptor` methods listed in Table 28–2 as shown in Example 28–1.

Example 28–1 Configuring EISDescriptor Record Format

```
EISDescriptor descriptor = new EISDescriptor();
descriptor.useIndexedRecordFormat();
```

Configuring Custom EIS Interactions for Basic Persistence Operations

You can use TopLink to define an interaction for each basic persistence operation (**insert**, **update**, **delete**, **read object**, **read all**, or **does exist**) so that when you query and modify your EIS-mapped objects, the TopLink runtime will use the appropriate EIS interaction instead of the default EIS interaction.

You can configure custom EIS interactions for basic persistence operations only for EIS descriptors designated as root descriptors ("Configuring an EIS Descriptor as a Root or Composite Type" on page 28-8).

For CMP projects, the `ejb-jar.xml` file stores query lists. You can define the queries in the file and then read them into TopLink Workbench, or define them on the Queries tab and write them to the file. For more information, see "Writing to the ejb-jar.xml File" on page 18-16 and "Reading From the ejb-jar.xml File" on page 18-16.

Using TopLink Workbench, you can create `XMLInteraction` objects, in which there is a single query per interaction (see "Using TopLink Workbench" on page 28-6).

Using Java, you can create any `EISInteraction` type. For some EIS projects, it is common for multiple interactions to be used in a single query. For example, one interaction–to enqueue a request, and another–to dequeue the response. Because TopLink Workbench does not support setting multiple interactions on a single query, you must use an amendment method to create and configure the interaction in Java (see "Using Java" on page 28-8).

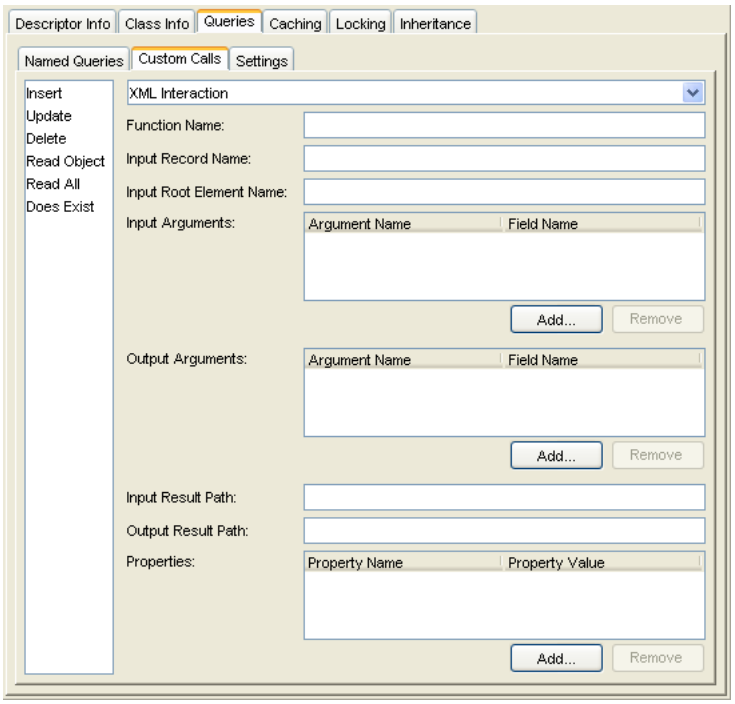
Note: In a one-to-one or one-to-many EIS mapping, you must also specify a selection interaction that TopLink uses to acquire target objects. You can use either the target object's read interaction (the default) or specify a separate selection interaction, if necessary. For more information, see "Configuring Selection Interaction" on page 54-3).

Using TopLink Workbench

To configure custom EIS interactions for basic persistence operations, use the following procedure:

1. In the **Navigator**, select an EIS root descriptor in a EIS project.
2. Click the **Queries** tab in the **Editor**. The Queries tab appears.
3. Click the **Custom Calls** tab. The Custom Calls tab appears.

Figure 28–5 Queries, Custom Calls Tab for EIS Calls



Click the appropriate interaction type from the list (**Insert**, **Update**, **Delete**, **Read Object**, **Read All**, or **Does Exist**) and use the following table to enter data in each field

Field	Description
Interaction Type	Using TopLink Workbench, you can only use XML Interactions. You cannot change this field.
Function Name	The name of the EIS function that this call type (Read Object or Read All) invokes on the EIS.
Input Record Name	The name passed to the J2C adapter when creating the input record.
Input Root Element	The root element name to use for the input DOM.
Input Arguments	The query argument name to map to the interaction field or XPath nodes in the argument record. For example, if you are using XML records, use this option to map input argument `name` to the XPath `name/first-name`.
Output Arguments	The result record field or XPath nodes to map to the correct nodes in the record used by the descriptor's mappings. For example, if you are using XML records, use this option to map the output `fname` to `name/first-name`. Output arguments are not required if the interaction returns an XML result that matches the descriptor's mappings.
Input Result Path	Use this option if the EIS interaction expects the interaction arguments to be nested in the XML record. For example, specify `arguments`, if the arguments were to be nested under the root element `exec-find-order`, then under an `arguments` element.

Field	Description
Output Result Path	Use this option if the EIS interaction result record contains the XML data that maps to the objects in a nested structure. For example, specify `order`, if the results were return under a root element `results`, then under an `order` element.
Properties	Any properties required by your EIS platform. For example, property name `operation` (from `AQPlatform.QUEUE_OPERATION`) and property value `enqueue` (from `AQPlatform.ENQUEUE`).

Using Java

Using Java, you can create any type of EIS interaction that TopLink supports (see "Using EIS Interactions" on page 94-24).

For some EIS projects, it is common for multiple interactions to be used in a single query: for example, one interaction to enqueue a request and another to dequeue the response. Because TopLink Workbench does not support setting multiple interactions on a single query, you must use an amendment method to create and configure the interaction in Java as Example Example 28–2 shows.

Example 28–2 Creating an XML Interaction for an AQ Platform

```
public static void addXMLInteractions(ClassDescriptor descriptor) {
    // find order interaction
    XMLInteraction request = new XMLInteraction();
    request.setProperty(AQPlatform.QUEUE_OPERATION, AQPlatform.ENQUEUE);
    request.setProperty(AQPlatform.QUEUE, "ORDER_INBOUND_QUEUE");
    request.setProperty(AQPlatform.SCHEMA, "AQUSER");
    request.setInputRootElementName("READ_ORDER");
    request.addArgument("@id");

    XMLInteraction response = new XMLInteraction();
    response.setProperty(AQPlatform.QUEUE_OPERATION, AQPlatform.DEQUEUE);
    response.setProperty(AQPlatform.QUEUE, "ORDER_OUTBOUND_QUEUE");
    response.setProperty(AQPlatform.SCHEMA, "AQUSER");

    ReadObjectQuery query = new ReadObjectQuery();
    query.addCall(request);
    query.addCall(response);
    descriptor.getQueryManager().setReadObjectQuery(query);

    // place order interaction
    XMLInteraction insert = new XMLInteraction();
    insert.setProperty(AQPlatform.QUEUE_OPERATION, AQPlatform.ENQUEUE);
    insert.setProperty(AQPlatform.QUEUE, "ORDER_INBOUND_QUEUE");
    insert.setProperty(AQPlatform.SCHEMA, "AQUSER");
    insert.setInputRootElementName("INSERT_ORDER");

    descriptor.getQueryManager().setInsertCall(insert);
}
```

Configuring an EIS Descriptor as a Root or Composite Type

You can designate an EIS descriptor as root (see "EIS Root Descriptors" on page 24-5) or composite (see "EIS Composite Descriptors" on page 24-5).

When you designate an EIS descriptor as a root, you tell the TopLink runtime that the EIS descriptor's reference class is a parent classš–no other class will reference it by way of a composite object mapping or composite collection mapping. Using an EIS root descriptor, you can configure all supported mappings and you can configure the descriptor with EIS interactions (see "Using EIS Interactions" on page 94-24). However, if you configure the EIS root descriptor with a composite object mapping or composite collection mapping, the reference descriptor you define must be an EIS composite descriptor; it cannot be another EIS root descriptor.

When you designate an EIS descriptor as a composite (the default), you tell the TopLink runtime that the EIS descriptor's reference class may be referenced by a composite object or composite collection mapping (see Chapter 57, "Configuring an EIS Composite Object Mapping" and Chapter 58, "Configuring an EIS Composite Collection Mapping"). Using an EIS composite descriptor, you can configure all supported mappings, but you cannot configure it with EIS interactions.

You can configure inheritance for a descriptor designated as a composite (see "Configuring Inheritance for a Child (Branch or Leaf) Class Descriptor" on page 25-51), however, in this case, *all* the descriptors in the inheritance tree must be aggregates. Aggregate and class descriptors cannot exist in the same inheritance tree. For more information, see "Aggregate and Composite Descriptors and Inheritance" on page 23-17.

If you configure a descriptor as a composite using TopLink Workbench, you cannot configure the descriptor with EJB information (see "Configuring a Descriptor With EJB Information" on page 25-45).

For more information, see the following:

- "Descriptors and Aggregation" on page 23-5
- "Composite and Reference EIS Mappings" on page 53-4

Using TopLink Workbench

Configuring a Descriptor as a Root or Composite

To configure an EIS descriptor as a root or composite EIS descriptor, use this procedure:

1. In the **Navigator**, select an EIS composite descriptor.
2. Click the **Root** or **Composite** descriptor button on the mapping toolbar.

   You can also select the descriptor and choose **Selected** > **Descriptor Type** > **Root** or **Composite** from the menu or by right-clicking on the descriptor in the **Navigator** and selecting **Descriptor Type** > **Root** or **Composite** from the context menu.

Using Java

To configure an EIS descriptor as root or composite using Java, create a descriptor amendment method (see "Configuring Amendment Methods" on page 25-81) and use the following `EISDescriptor` methods:

- To designate an EIS descriptor as a root descriptor, use `EISDescriptor` method `descriptorIsNormal`.
- To designate an EIS descriptor as a composite (nonroot) descriptor, use `EISDescriptor` method `descriptorIsAggregate`.

29

Configuring an XML Descriptor

This chapter describes the various components that you must configure in order to use an XML descriptor.

For more information, see the following:

- "Descriptor Creation Overview" on page 24-1
- "XML Descriptors" on page 23-12

XML Descriptor Configuration Overview

Table 29–1 lists the default configurable options for an XML descriptor.

Table 29–1 Configuration Options for XML Descriptors

Option	Type	TopLink Workbench	Java
"Configuring XML Schema Namespace" on page 4-37	Basic	✔	✔
"Configuring XML Schema Reference" on page 4-36	Basic	✔	✔
"Configuring Schema Context for an XML Descriptor" on page 29-1	Basic	✔	✔
"Configuring for Complex Type of anyType" on page 29-3	Advanced	✔	
"Configuring Default Root Element" on page 29-5	Basic	✔	✔
"Configuring Document Preservation" on page 29-6	Basic	✔	✔
"Configuring Descriptor Comments" on page 25-9	Advanced	✔	
"Configuring Classes" on page 4-41	Basic	✔	
"Configuring Inheritance for a Child (Branch or Leaf) Class Descriptor" on page 25-51	Advanced	✔	✔
"Configuring Inheritance for a Parent (Root) Descriptor" on page 25-52	Advanced	✔	✔
"Configuring Inherited Attribute Mapping in a Subclass" on page 25-58	Advanced	✔	✔
"Configuring Instantiation Policy" on page 25-70	Advanced	✔	✔
"Configuring Copy Policy" on page 25-71	Advanced	✔	✔
"Configuring Amendment Methods" on page 25-81	Advanced	✔	✔
"Configuring a Mapping" on page 32-1	Basic	✔	✔

Configuring Schema Context for an XML Descriptor

TopLink Workbench uses the schema context to associate the XML descriptor reference class with a simple or complex type in one of the schemas associated with

the XML project (see "Configuring XML Schema Reference" on page 4-36). This allows TopLink Workbench to display the appropriate attributes available for mapping in that context.

You must configure the schema context for an XML descriptor regardless of whether or not you are using TopLink Workbench.

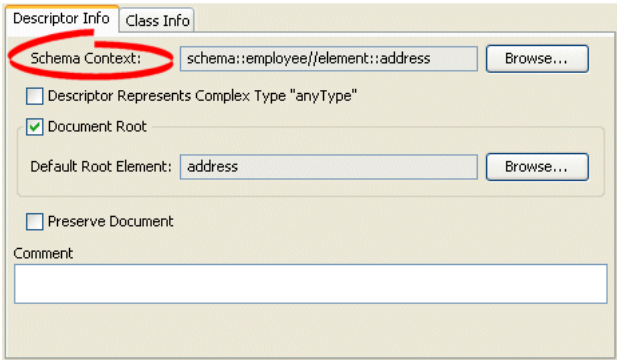
The TopLink runtime uses the schema context to validate XML fragments.

Using TopLink Workbench

To associate an XML descriptor with a specific schema complex type, use this procedure:

1. Select an XML descriptor in the **Navigator**. Its properties appear in the Editor.
2. Click the **Descriptor Info** tab. The Descriptor Info tab appears.

Figure 29–1 Descriptor Info Tab, Schema Context Option

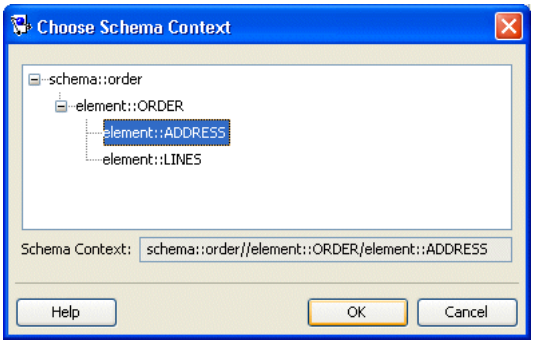


Click **Browse** to select the schema element to associate with this descriptor. For more information, see "Choosing a Schema Context" on page 29-2.

Choosing a Schema Context

Use the Choose Schema Context dialog box to select a specific schema element (such as when mapping an element).

Figure 29–2 Choose Schema Context Dialog Box



Select a schema element and click **OK**.

Using Java

To configure an XML descriptor with a schema context using Java, create a descriptor amendment method (see "Configuring Amendment Methods" on page 25-81) that uses `XMLSchemaReference` method `setSchemaContext`, as Example 29–1 shows.

Example 29–1 Configuring Schema Context

```
public void addToDescriptor(ClassDescriptor descriptor) {
    descriptor.getSchemaReference().setSchemaContext(xPath);
}
```

Configuring for Complex Type of anyType

This attribute applies only to TopLink Workbench. Use this option to solve "No schema context is specified" problems (see "Using the Problems Window" on page 4-11) for an XML descriptor that does not represent an element in your XML schema.

In general, TopLink Workbench assumes that every XML descriptor must have a schema context (see "Configuring Schema Context for an XML Descriptor" on page 29-1). However, if a class in your project does not relate to an element in your schema, then it does not have a schema context.

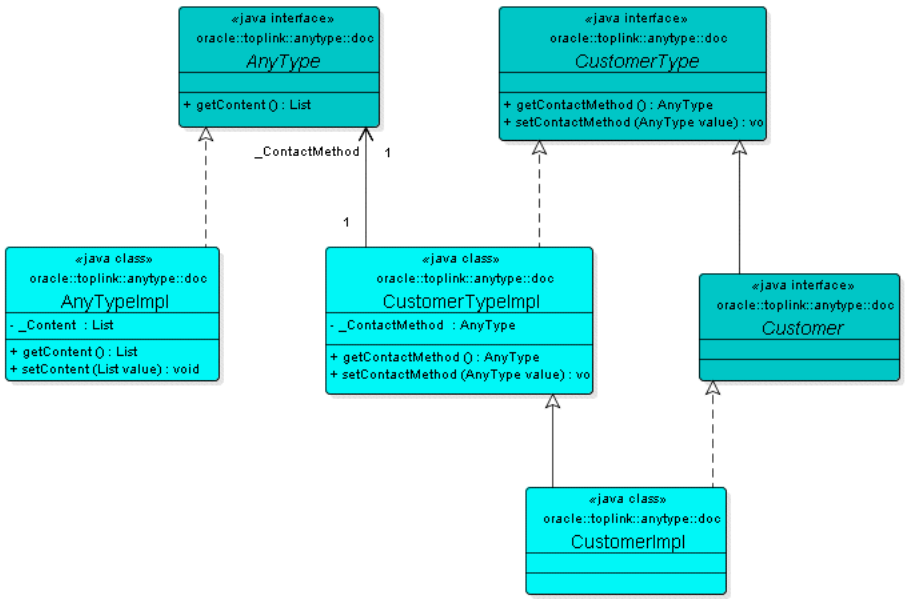
For example, consider the schema that Example 29–2 shows.

Example 29–2 Schema Using xsd:anyType

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="customer" type="customer-type"/>
    <xsd:complexType name="customer-type">
        <xsd:sequence>
            <xsd:element name="contact-method" type="xsd:anyType"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="address">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="street" type="xsd:string"/>
                <xsd:element name="city" type="xsd:string"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="phone-number" type="xsd:string"/>
</xsd:schema>
```

Because element `contact-method` is of type `xsd:anyType`, your project requires a class to represent that type, such as class `AnyTypeImpl` shown in Figure 29–3. Because this class does not relate to any complex type in your schema, it has no schema context. In this example, you would select this option for the `AnyTypeImpl` class.

Figure 29–3 Class Representing xsd:anyType



Note: See also "Configuring Maps to Wildcard" on page 63-3.

For more information, see "xs:any and xs:anyType Support" on page 62-4.

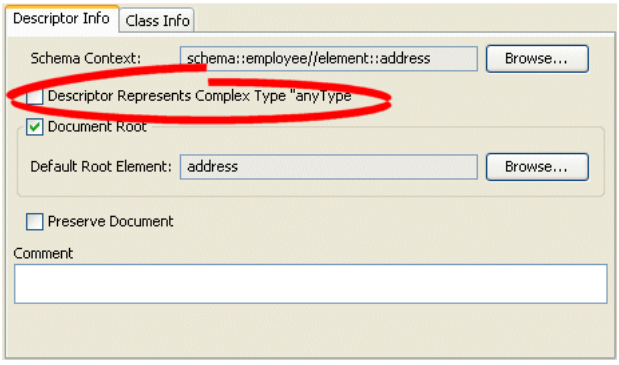
Using TopLink Workbench

Complex Type of anyType Option

To specify that the descriptor represents a complex type of `anyType`, use this procedure:

1. Select a descriptor in the **Navigator**. Its properties appear in the Editor.
2. Click the **Descriptor Info** tab. The Descriptor Info tab appears.

Figure 29–4 Descriptor Info Tab, Complex Type "anyType" Option



Select the **Descriptor Represents Complex Type "anyType"** option to specify this descriptor as the root element.

Configuring Default Root Element

The default root element is the name that TopLink uses for the root element when marshalling objects for this descriptor to, and unmarshalling from, an XML document. Descriptors used only in composite relationship mappings do not require a default root element.

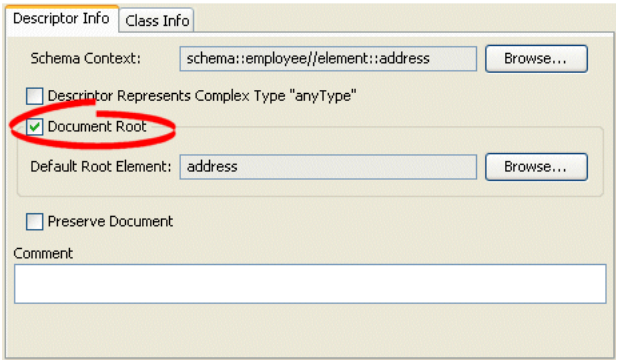
For more information, see "Default Root Element" on page 23-9.

Using TopLink Workbench

To specify a schema element as the default root element for the descriptor, use this procedure:

1. Select a descriptor in the **Navigator**. Its properties appear in the Editor.
2. Click the **Descriptor Info** tab. The Descriptor Info tab appears.

Figure 29–5 Descriptor Info Tab, Default Root Option



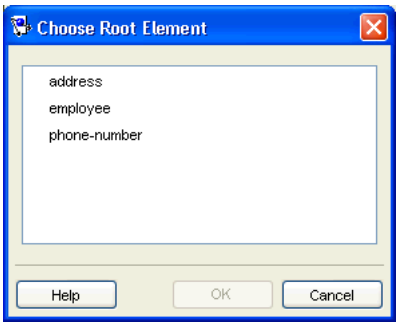
Select the **Default Root Element** option to specify this descriptor as the root element.

Click **Browse** to select the schema element to identify as the root element for this descriptor. See "Choosing a Root Element" on page 29-5 for more information.

Choosing a Root Element

Use the Choose Root Element dialog box to select a specific root element.

Figure 29–6 Choose Root Element Dialog Box



Select the root element and click **OK**.

Configuring Document Preservation

TopLink lets you preserve any "extra" data in your XML source that is not required to map to an object model (such as comments, processing instructions, or unmapped elements).

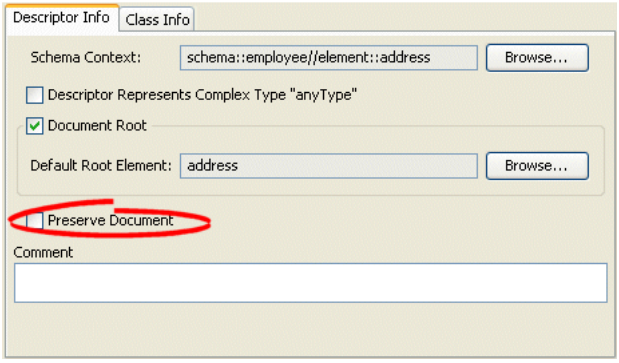
This permits round-tripping from XML to objects and back to XML without losing any data.

Using TopLink Workbench

To preserve the entire XML source document, use this procedure:

1. Select a descriptor in the **Navigator**. Its properties appear in the Editor.
2. Click the **Descriptor Info** tab. The Descriptor Info tab appears.

Figure 29–7 Descriptor Info Tab, Preserve Document Option



Select the **Preserve Document** option to maintain any extra information from the source XML document that TopLink does not require (such as comments).

Using Java

To configure an XML descriptor to maintain any extra information from the source XML document that TopLink does not require (such as comments) using Java, create a descriptor amendment method (see "Configuring Amendment Methods" on page 25-81) that configures the descriptor using `XMLDescriptor` method `setShouldPreserveDocument`.

Part IX

Mappings

This part describes general mapping information. It contains the following chapters:

- Chapter 30, "Understanding Mappings"

  This chapter describes each of the different TopLink mapping types and important mapping concepts.

- Chapter 31, "Creating a Mapping"

  This chapter contains procedures for creating TopLink mappings.

- Chapter 32, "Configuring a Mapping"

  This chapter explains how to configure TopLink mapping options common to two or more mapping types.

For information on specific mapping types, see the following parts:

- Part X, "Relational Mappings"
- Part XI, "Object-Relational Mappings"
- Part XII, "EIS Mappings"
- Part XIII, "XML Mappings"

30

Understanding Mappings

One of the greatest strengths of TopLink is its ability to transform data between an object representation and a representation specific to a data source. This transformation is called mapping and it is the core of a TopLink project.

A mapping corresponds to a single data member of a domain object. It associates the object data member with its data source representation and defines the means of performing the two-way conversion between object and data source.

This chapter includes information on the following

- Mapping Types
- Mapping Concepts
- Understanding the Mapping API

Mapping Types

Table 30–1 describes the mapping types that TopLink supports.

Table 30–1 TopLink Mapping Types

Type	Description	Type	TopLink Workbench	Java
"Relational Mappings" on page 30-25	Mappings that transform any object data member type to a corresponding relational database (SQL) data source representation in any supported relational database. Relational mappings allow you to map an object model into a relational data model.	Basic	✔	✔
"Object-Relational Mappings" on page 30-26	Mappings that transform certain object data member types to structured data source representations optimized for storage in specialized object-relational databases such as Oracle Database. Object-relational mappings let you map an object model into an object-relational data model.	Advanced		✔
"EIS Mappings" on page 30-27	Mappings that transform object data members to the EIS record format defined by the object's descriptor.	Advanced	✔	✔
"XML Mappings" on page 30-27	Mappings that transform object data members to the XML elements of an XML document whose structure is defined by an XML schema document (XSD).	Advanced	✔	✔

For more information, see the following:

- "Creating a Mapping" on page 31-1
- "Configuring a Mapping" on page 32-1

Mapping Concepts

This section describes concepts unique to TopLink mappings, including the following:

- Mapping Architecture (applicable to relational and nonrelational mappings)
- Example Mapping (applicable to relational and nonrelational mappings)
- Automatic Mappings
  - Automapping With TopLink Workbench at Development Time (applicable to relational and nonrelational mappings)
  - Default Mapping in CMP Projects Using OC4J at Run Time (applicable to relational mappings)
  - JAXB Project Generation at Development Time (applicable to XML mappings)
- Indirection (applicable to object-relational mappings)
- Method Accessors and Attribute Accessors (applicable to relational and nonrelational mappings)
- Mapping Converters and Transformers
  - Serialized Object Converter (applicable to relational and nonrelational mappings)
  - Type Conversion Converter (applicable to relational and nonrelational mappings)
  - Object Type Converter (applicable to XML mappings)
  - Simple Type Translator (applicable to XML mappings)
  - Transformation Mappings (applicable to relational and nonrelational mappings)
- Mappings and XPath (applicable to XML mappings)
- Mappings and xsd:list and xsd:union Types (applicable to XML mappings)
- Mappings and the jaxb:class Customization (applicable to XML mappings)
- Mappings and JAXB Typesafe Enumerations (applicable to XML mappings)

Mapping Architecture

To define a mapping, you draw upon the following components:

- The data representation specific to the data source (such as a relational database table or schema-defined XML element) in which you store the object's data
- A descriptor for a particular object class
- An object class to map

> **Note:** A mapping is the same regardless of whether your project is persistent or nonpersistent.

For an example of a typical TopLink mapping, see "Example Mapping" on page 30-3.

The type of data source you define in your TopLink project determines the type of mappings you can use and how you configure them. In a persistent project, you use mappings to persist to a data source. In a nonpersistent project, you use mappings

simply to transform between the object format and some other data representation (such as XML). For more information about data source and project types, see "TopLink Project Types" on page 17-1.

A descriptor represents a particular domain object: it describes the object's class. It owns mappings: one mapping for each of the class data members that you intend to persist or transform in memory.

> **Note:** Persistence is applicable at the descriptor level.

For more information about descriptors, see "Understanding Descriptors" on page 23-1.

TopLink provides mappings to handle a wide variety of data types and data representations. For more information, see "Mapping Types" on page 30-1.

All mappings are subclasses of the `oracle.toplink.mappings.DatabaseMapping` class. For more information about the mapping API, see "Understanding the Mapping API" on page 30-25.

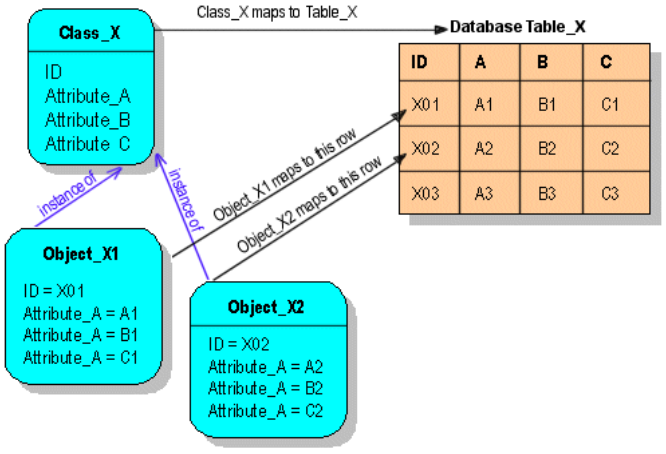
Example Mapping

Although TopLink supports more complex mappings, most TopLink classes map to a single database table or XML element that defines the type of information available in the class. Each object instance of 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 30–1 illustrates the simplest database mapping case in which:

- **Table_X** in the database represents **Class_X**.
- **Object_X1** and **Object_X2** are instances of **Class_X**.
- Individual rows in **Table_X** represent **Object_X1** and **Object_X2**, as well as any other instances of **Class_X**.

Figure 30–1 How Classes and Objects Map to a Database Table



TopLink provides you with the tools to build these mappings, from the simple mappings illustrated in Figure 30–1, to complex mappings.

For an additional example of a relational mapping, see Figure 33–1, "Direct-to-Field Mapping" on page 33-4.

For an example of a nonrelational mapping, see Figure 62–34, "XML Transformation Mappings" on page 62-31.

Automatic Mappings

Typically, you use TopLink Workbench to define mappings on a class-by-class and data member-by-data-member basis manually (see "Creating Mappings Manually During Development" on page 31-1).

Alternatively, you can take advantage of the following:

- Automapping With TopLink Workbench at Development Time
- Default Mapping in CMP Projects Using OC4J at Run Time
- JAXB Project Generation at Development Time

Automapping With TopLink Workbench at Development Time

You can use TopLink Workbench **Automap** feature to automatically define default mappings for every class and data member in your project (see "Creating Mappings Automatically During Development" on page 31-2).

TopLink Workbench automapping is available for all project types and assumes that both the object model and database schema are already defined.

Default Mapping in CMP Projects Using OC4J at Run Time

Default mapping is a relational persistence framework term that refers to making the framework automatically generate the object descriptor metadata (including such things as mappings, login data, database platform, locking, and foreign keys).

Note: You can apply default mapping to relational projects only.

TopLink can also optionally create or drop-and-create the tables associated with the entities during the deployment. This means that TopLink can handle the whole deployment process with the minimum requirements: a compliant EAR file and a valid data source. This frees you from creating tables and specifying mappings before you deploy your application.

TopLink default mapping supports the following features:

- Direct-to-field mapping support for standard CMP (that is, not dependent object) fields.
- Serialized object mapping support for dependent objects.
- One-to-one, one-to-many, and many-to-many mappings support for CMR fields.
- Self-referencing, unidirectional and bidirectional relationship mappings (see "Directionality" on page 33-2).
- Optimistic version locking for each entity.
- Automatic table drop and create, platform-specified supported types, default size and subsize, and database-reserved keywords.
- EJB QL (EJB query language) queries, such as **finder** and **ejbSelect**.

- Unknown primary key class case (primary key-class type `java.lang.Object.class`).

Default mapping is available for CMP relational projects deployed to OC4J configured to use TopLink as the default persistence manager. In this configuration, the EJB container provides the entity bean descriptor data (from `ejb-jar.xml`) required by the persistence manager to generate the persistence descriptor file.

If a `toplink-ejb-jar.xml` descriptor file is not present, TopLink, working as the OC4J default persistence manager, generates a default persistence descriptor file for any CMP project during deployment. In this case, TopLink applies default mappings and, optionally, automatic table generation. The generated descriptor file includes the following:

- Mapping for each entity CMP and CMR field
- Optimistic locking, foreign keys, target foreign key, and relation table
- Transparent indirection for relationships
- Database login and platform metadata

If a `toplink-ejb-jar.xml` descriptor file is present and specified in the `orion-ejb-jar.xml` file, TopLink does not apply default mapping: it honours the mappings specified in the `toplink-ejb-jar.xml` file. In this case, you can still configure automatic table generation.

For more information, see "Configuring default-mapping Properties" on page 8-12.

Note: The preceding information is also applicable to EJB 3.0 persistent relational projects.

EJB 3.0 annotations often replace descriptor files. For example, TopLink does not apply default mapping if annotations that are equivalent to `toplink-ejb-jar.xml` descriptor file are specified.

JAXB Project Generation at Development Time

JAXB provides an API and a tool that allow automatic two-way mapping between XML documents and Java objects. The JAXB compiler generates all the Java classes and mappings based on the provided Document Type Definition (DTD) and a schema definition.

For more information on JAXB, see *Architecture for XML Binding (JAXB): A Primer* at `http://java.sun.com/developer/technicalArticles/xml/jaxb/index.html`

For more information on XML mappings, see Chapter 62, "Understanding XML Mappings".

Indirection

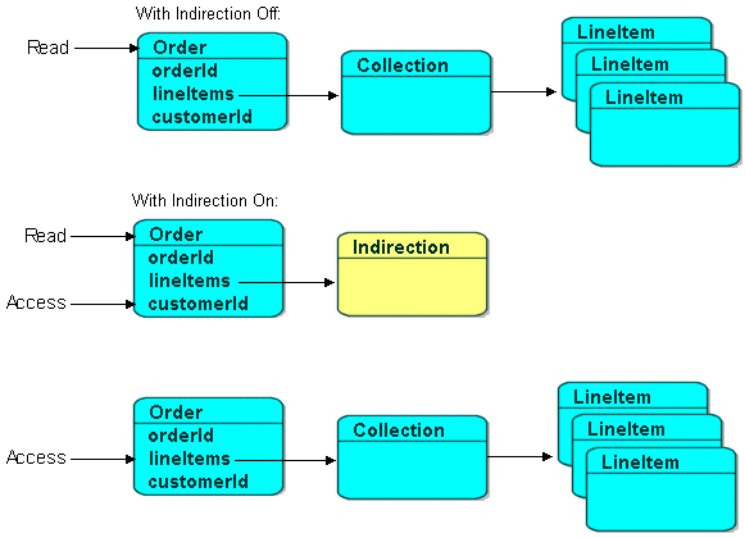
By default, when TopLink retrieves a persistent object, it retrieves all of the dependent objects to which it refers. When you configure indirection (also known as lazy reading, lazy loading, and just-in-time reading) for an attribute mapped with a relationship mapping, TopLink uses an indirection object as a place holder for the referenced object: TopLink defers reading the dependent object until you access that specific attribute. This can result in a significant performance improvement, especially if the application is interested only in the contents of the retrieved object, rather than the objects to which it is related.

Oracle strongly recommends using indirection for all relationship mappings. Not only does this lets you optimize data source access, but it also allows TopLink to optimize the unit of work processing, cache access, and concurrency.

> **Note:** The use of indirection is especially important for providing a proper maintenance of bidirectional relationships (see "Maintaining Bidirectional Relationships" on page 2-29). In this case, you must use indirection. If you are operating with collections, you must use transparent indirection (see "Transparent Indirect Container Indirection" on page 30-8).

Figure 30–2 shows an indirection example. Without indirection, reading the `Order` object also reads the dependent collection of `LineItem` objects. With indirection, reading the `Order` object does not read the dependent collection of `LineItem` objects: the `lineItems` attribute refers to an indirection object. You can access other attributes (such as `customerId`), but TopLink reads the dependent `LineItem` objects only if and when you access the `lineItems` attribute.

Figure 30–2 TopLink Indirection



TopLink supports the following types of indirection:

- Value Holder Indirection
- Transparent Indirect Container Indirection
- Proxy Indirection

When using indirection with EJB, the version of EJB and application server you use affects how indirection is configured and what types of indirection are applicable (see "Indirection and EJB" on page 30-9).

When using indirection with an object that your application serializes, you must consider the effect of any untriggered indirection objects at deserialization time (see "Indirection, Serialization, and Detachment" on page 30-9).

For information on configuring indirection, see "Configuring Indirection" on page 32-3.

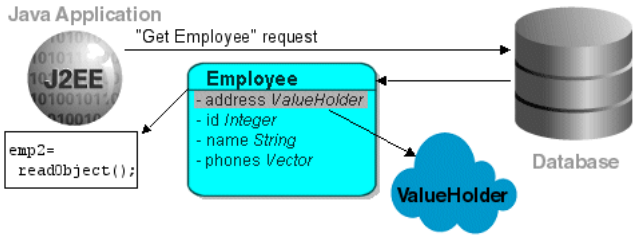
Value Holder Indirection

Persistent classes that use indirection must replace relationship attributes with value holder attributes. A value holder is an instance of a class that implements the `ValueHolderInterface` interface, such as `ValueHolder`. This object stores the information necessary to retrieve the object it is replacing from the database. If the application does not access the value holder, the replaced object is never read from the database.

To obtain the object that the value holder replaces, use the `getValue` and `setValue` methods of the `ValueHolderInterface`. A convenient way of using these methods is to hide the `getValue` and `setValue` methods of the `ValueHolderInterface` inside `get` and `set` methods, as shown in the following illustrations.

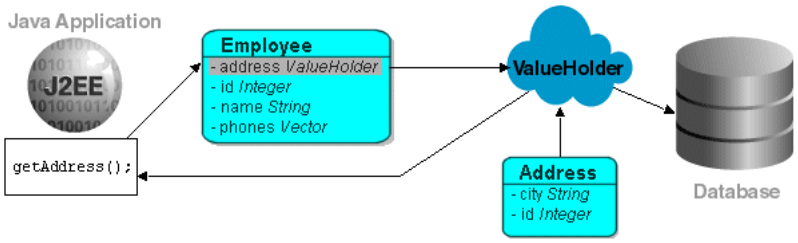
Figure 30–3 shows the `Employee` object being read from the database. The `Address` object is not read and will not be created unless it is accessed.

Figure 30–3 Address Object Not Read



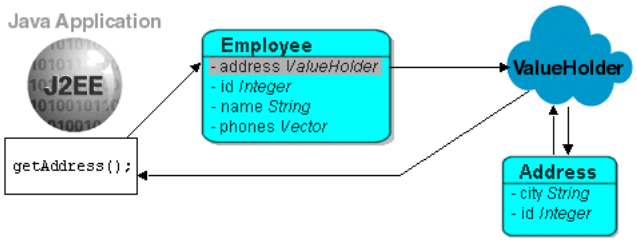
The first time the address is accessed, as in Figure 30–4, the `ValueHolder` reads and returns the `Address` object.

Figure 30–4 Initial Request



Subsequent requests for the address do not access the database, as shown in Figure 30–5.

Figure 30–5 Subsequent Requests



If you are using method access ("Configuring Method Accessing" on page 32-14), the get and set methods specified in the mapping must access the instance of `ValueHolderInterface`, rather than the object referenced by the value holder. The

application should not use these getter and setter, but use the getter and setter that hide the usage of value holders. For more information, see "Configuring ValueHolder Indirection With Method Accessing" on page 32-7.

Transparent Indirect Container Indirection

Transparent indirect container (see "Configuring Container Policy" on page 32-26) indirection lets you declare any relationship attribute of a persistent class that holds a collection of related objects as any of the following:

- `java.util.Collection`
- `java.util.Hastable`
- `java.util.List`
- `java.util.Map`
- `java.util.Set`
- `java.util.Vector`

TopLink will use an indirection object that implements the appropriate interface and also performs just-in-time reading of the related objects. When using transparent indirection, you do not have to declare the attributes as `ValueHolderInterface`.

Newly created collection mappings use transparent indirection by default if their attribute *is not* a `ValueHolderInterface`.

Proxy Indirection

Introduced in JDK 1.3, the Java class `Proxy` lets you to use dynamic proxy objects as place-holders for a defined interface. Certain TopLink mappings (see Table 32–4) can be configured to use proxy indirection, which gives you the benefits of TopLink indirection without the need to include TopLink classes in your domain model. Proxy indirection is to one-to-one relationship mappings as indirect containers are to collection mappings.

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

- The target class of the one-to-one relationship must implement a public interface.
- The one-to-one attribute on the source class must be of the `interface` type.
- If you employ method accessing ("Configuring Method Accessing" on page 32-14), then the getter and setter methods must use the interface.

Before using proxy indirection, be aware of the restrictions it places on how you use the unit of work (see "Proxy Indirection Restrictions" on page 30-8).

To configure proxy indirection, you can use TopLink Workbench (see "Using TopLink Workbench" on page 32-15) or Java in an amendment method (see "Configuring Proxy Indirection" on page 32-9).

Proxy Indirection Restrictions Proxy objects in Java are only able to intercept messages sent. If a primitive operation such as `==`, `instanceof`, or `getClass` is used on a proxy, it will not be intercepted. This limitation can require the application to be somewhat aware of the usage of proxy objects.

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 lets you retrieve a target object clone with a call to a getter on the source object 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 Chapter 97, "Understanding TopLink Transactions".

Indirection and EJB

When using indirection with EJB, how TopLink handles indirection depends on the EJB version and application server you are using.

In addition, you cannot use proxy indirection (see "Proxy Indirection" on page 30-8) for relationships to an enterprise bean, because EJB do not directly implement their remote or local interfaces.

When using indirection with an enterprise bean that your application serializes, you must consider the effect of any untriggered indirection objects at deserialization time (see "Indirection, Serialization, and Detachment" on page 30-9).

CMP When using CMP with any of the application servers for which TopLink provides CMP integration (see "Application Server Support" on page 7-1), TopLink uses code generation to automatically configure the usage of value holder indirection ("Value Holder Indirection" on page 30-7) for all CMR.

Note: When using EJB 3.0 persistence with OC4J, if you specify the `@Bean` annotation attribute `fetch=lazy`, TopLink uses bytecode weaving to automatically configure the usage of value holder indirection (see "Value Holder Indirection" on page 30-7) for all container-managed relationships.

Indirection, Serialization, and Detachment

When using indirection, it is likely that a graph of persistent objects will contain untriggered indirection objects. Because indirection objects are transient and do not survive serialization between one JVM and another, untriggered indirection objects may appear as null values at deserialization time. This section describes how to address this depending on the EJB version and application server you are using.

For more information on serialization and TopLink, see "Merging Changes in Working Copy Clones" on page 99-13.

CMP When using CMP with any of the application servers for which TopLink provides CMP integration (see "Application Server Support" on page 7-1), indirection does not affect enterprise beans' passivation and activation within the same JVM.

However, when you serialize an indirection-enabled enterprise bean from one JVM to another (for example, from server to client), untriggered indirection objects appear as null values at deserialization time. You must code your client to handle this scenario and to distinguish between an attribute that is `null` because it was null in the data source, and an attribute that is null because of an untriggered indireciton object.

Note: When using EJB 3.0 persistence with OC4J, indirection does not affect enterprise beans' passivation and activation, nor does it affect detachment from and attachment to a persistence manager within the same JVM.

However, when you serialize an indirection-enabled EJB 3.0 beans from one JVM to another, (for example, from server to client), untriggered indirection objects appear as null values at deserialization time. To avoid this, configure your client to use the same Java agent that OC4J uses (see "Configuring ValueHolder Indirection With EJB 3.0 on OC4J" on page 32-8).

Method Accessors and Attribute Accessors

By default, TopLink uses direct access to access public attributes. Using TopLink, you can configure field access at the project level (see "Configuring Mapped Field Access at the Project Level" on page 19-4) and at the method level ("Configuring Method Accessing" on page 32-14).

Mapping Converters and Transformers

If existing TopLink mappings do not meet your needs, you can create custom mappings using mapping extensions. These extensions include the following:

- Serialized Object Converter
- Type Conversion Converter
- Object Type Converter
- Simple Type Translator
- Transformation Mappings

Note: You can use the mapping converters and transformers regardless of whether your data source is relational or nonrelational.

Serialized Object Converter

The serialized object converter is an extension of direct and direct collection mappings that lets you map complex objects into binary fields through Java object serialization. Serialized objects are normally stored in `RAW` or Binary Large Object (`BLOB`) fields in the database, or `HEX` or `BASE64` elements in an XML document.

Figure 30–6 shows an example of a direct-to-field mappings that uses a serialized object converter. The attribute `jobDescription` contains a formatted text document that is stored in the JOB_DESC field of the database.

Figure 30–6 Serialized Object Converter (relational)

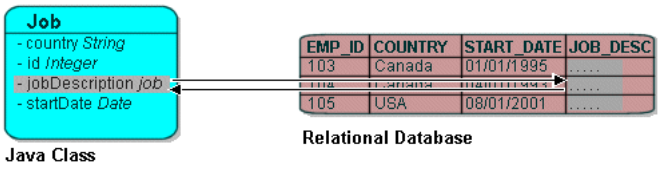
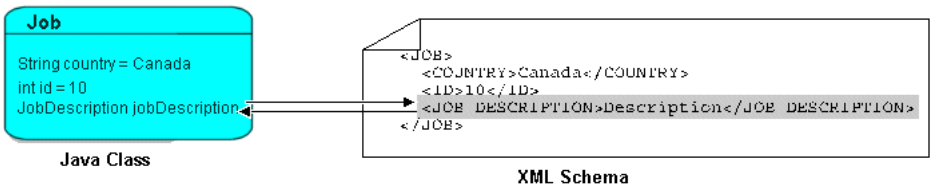


Figure 30–8 demonstrates an example of a nonrelational mapping that uses a serialized object converter. The attribute `jobDescription` contains a formatted text document that TopLink stores in the `JOB DESCRIPTION` element of an XML schema.

Figure 30–7 Serialized Object Converter (nonrelational)



The serialized object converter relies on the Java serializer. Before you map a domain object with the serialized object converter, ensure that the domain object implements the `java.io.Serializable` interface (or inherits that implementation) and marks all nonserializable fields transient.

For more information, see "Configuring a Serialized Object Converter" on page 32-18.

Type Conversion Converter

The type conversion converter is an extension of direct and direct collection mappings that lets you explicitly map a data source type to a Java type. For example, a `Number` in the data source can be mapped to a `String` in Java, or a `java.util.Date` in Java can be mapped to a `java.sql.Date` in the data source.

Figure 30–8 illustrates a type conversion mapping (relational). Because the `java.util.Date` class is stored by default as a `Timestamp` in the database, it must first be converted to an explicit database type such as `java.sql.Date` (required only for DB2–most other databases have a single date data type that can store any date or time).

Figure 30–8 Type Conversion Mapping (relational)

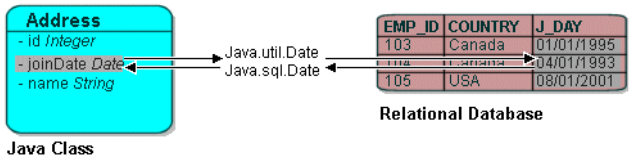
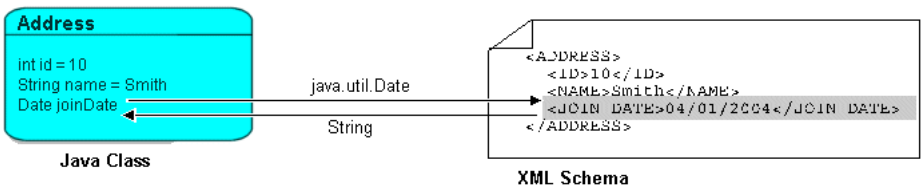


Figure 30–9 illustrates a type conversion mapping (nonrelational). `java.util.Date` object is mapped to a String in a XML schema.

Figure 30–9 Type Conversion Mapping (nonrelational)



You can use a type conversion converter to specify the specific database type when that type must be handled specially for the database. This includes support for the

special Oracle JDBC binding options required for NCHAR, NVARCHAR2, and NCLOB fields as well as the special Oracle Thin JDBC insert and update requirements for handling BLOB and CLOB fields greater than 5K.

TopLink uses the NCharacter, NClob and NString types in the oracle.toplink.platform.database.oracle package as the converter data type to support the NCHAR, NCLOB and NVARCHAR2 types. TopLink uses the java.sql.Blob and Clob types as the converter data type to support BLOB and CLOB values greater than 5K.

You can configure a type conversion converter to map a data source time type (such as TIMESTAMP) to a java.lang.String provided that the String value conforms to the following formats:

- YYYY/MM/DD HH:MM:SS
- YY/MM/DD HH:MM:SS
- YYYY-MM-DD HH:MM:SS
- YY-MM-DD HH:MM:SS

For more complex String to TIMESTAMP type conversion, consider a transformation mapping (see "Transformation Mappings" on page 30-14).

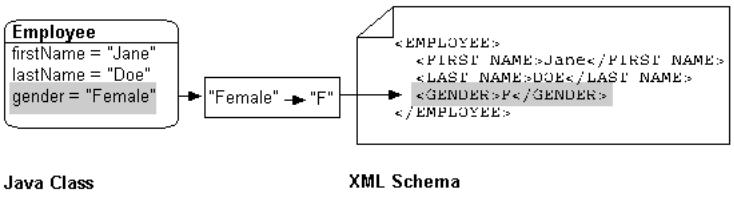
For more information, see "Configuring a Type Conversion Converter" on page 32-20.

Object Type Converter

The object type converter is an extension of direct and direct collection mappings that lets you match a fixed number of XML values to Java objects. Use this converter when the values in the schema differ from those in Java.

Figure 30–10 illustrates an object type conversion between the Employee attribute gender and the XML element gender. If the value of the Java object attribute is Female, TopLink stores it in the XML element as F.

Figure 30–10 Object Type XML Converter



For more information, see "Configuring an Object Type Converter" on page 32-22.

Simple Type Translator

The simple type translator is an extension of direct and direct collection mappings that lets you automatically translate an XML element value to an appropriate Java type based on the element's <type> attribute as defined in your XML schema.

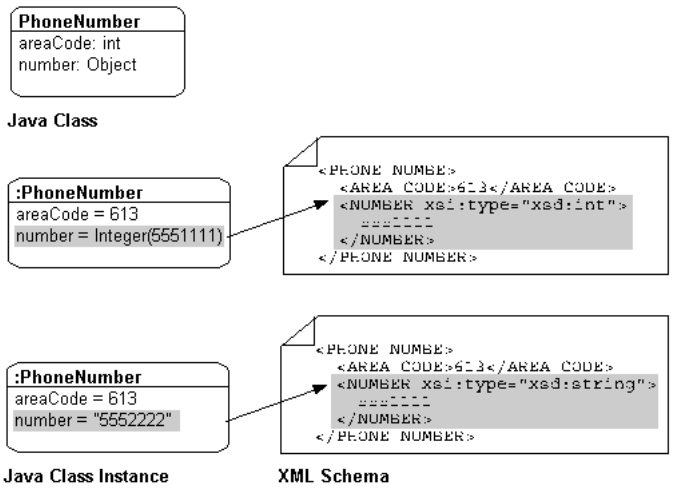
You can use a simple type translator only when the mapping's XPath goes to a text node. You cannot use a simple type translator if the mapping's XPath goes to an attribute.

Using a simple type translator, you can make the XML document preserve type information. This is useful when your object model specifies generic object attributes

such as `java.lang.Object` and `java.io.Serializable`, since they do not trigger specific type conversions in TopLink as do specific object attributes such as `java.lang.Integer` or `java.util.Calendar`.

Figure 30–11 illustrates a type translation XML mapping for the `number` attribute of the `PhoneNumber` class. Notice that the Java attribute is not specific enough to preserve the typing. The simple type translator adds the type information to the resulting document to preserve the typing.

Figure 30–11 Simple Type Translator



By default, TopLink uses built-in read and write conversion pairs (see "Default Read Conversions" on page 30-13 and "Default Write Conversions" on page 30-14).

You can override this behavior by specifying and configuring your own simple type translator, for example, to write XML binary data as `Base64`.

For more information, see "Configuring a Simple Type Translator" on page 32-23.

Default Read Conversions Table 30–2 lists the built-in conversion pairs for reading XML elements. When the schema `<type>` attribute is specified and the simple type translator is enabled, the value read is converted to the corresponding Java type.

Table 30–2 Simple Type Translator Read Conversions

Schema Type	Java Type
base64Binary	Byte[]
boolean	Boolean
byte	Byte
date	Calendar
dateTime	Calendar
double	Double
float	Float
hexBinary	Byte[]
int	int
integer	BigInteger

Table 30–2 (Cont.) Simple Type Translator Read Conversions

Schema Type	Java Type
long	Long
short	Short
string	String
time	Calendar
unsignedByte	Short
unsignedInt	Long
unsignedShort	Integer

Default Write Conversions Table 30–3 lists the built-in conversion pairs for writing XML. When a Java class attribute is of a type in Table 30–3 and the simple type translator is enabled, the corresponding schema type is specified on the element written.

Table 30–3 Simple Type Translator Write Conversions

Java Type	Schema Type
Byte[]	hexBinary
BigInteger	integer
Boolean	boolean
Byte	byte
Calendar	dateTime
Gregorian_Calendar	dateTime
Double	double
Float	float
Integer	int
Long	long
int	int
short	short
String	string

Transformation Mappings

In some special circumstances, existing mapping types and their default Java to data source type handling may be insufficient. In these special cases, you can consider using a transformation mapping to perform specialized translations between how a value is represented in Java and in the data source.

A transformation mapping is made up of the following two components:

- attribute transformer (see "Configuring Attribute Transformer" on page 32-29): performs the object attribute transformation at read (unmarshall) time
- field transformer (see "Configuring Field Transformer Associations" on page 32-31): performs the object attribute-to-field transformation at write (marshal) time

You can implement a transformer as either a separate class or as a method on your domain object.

Within your implementation of the attribute and field transformer, you can take whatever actions are necessary to transform your application data to suit your data source, and vise versa.

For more information, see the following:

- "Transformation Mapping" on page 33-15
- "EIS Transformation Mapping" on page 53-17
- "XML Transformation Mapping" on page 62-31

Mappings and XPath

TopLink uses XPath statements to efficiently map the attributes of a Java object in EIS mappings to XML records and in XML mappings to XML documents. When you create such a mapping, you can specify the following:

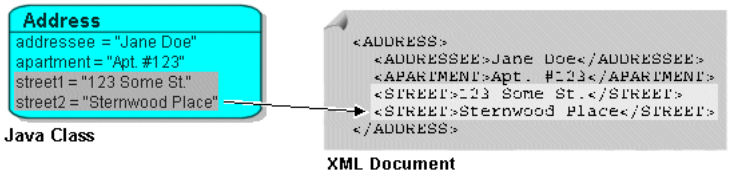
- XPath by Position
- XPath by Path and Name
- XPath by Name
- Self XPath

XPath by Position

In a relational database table, columns are uniquely identified by name. In an XML document, elements are uniquely identified by name and position. Figure 30–12 illustrates mapping to an XML document in which the first instance of the `street` element stores apartment information and the second instance of the `street` element stores street information. Figure 30–12 shows that TopLink XML mappings preserve the order in which mappings are persisted and allow you to map Java object attributes to XML elements by position using an XPath like `street[2]/text()`.

Other XML technologies only recognize the name of XML elements (not their position) and force you to store the simple values from elements with the same name in a collection.

Figure 30–12 Mapping to an XML Document by Position

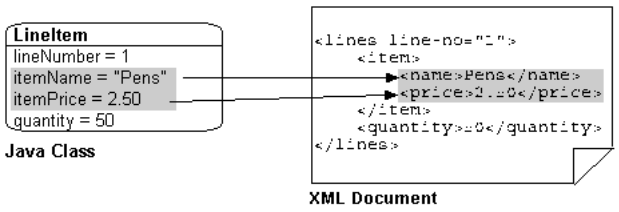


XPath by Path and Name

In an XML document, attributes and elements are uniquely identified by a combination of name and path. Figure 30–13 illustrates that TopLink XML mappings can uniquely identify an XML element by name and path using an XPath such as `item/name/text()`. TopLink does not require a formal object relationship between XML elements `lines` and `item`.

Other XML technologies force you to provide an object relationship for every level of nesting, resulting in the inclusion of many XML elements and classes simply to organize the data to satisfy this restriction. This produces an unnecessarily large object model that does not properly reflect the domain space.

Figure 30–13 Mapping to an XML Document by Path and Name

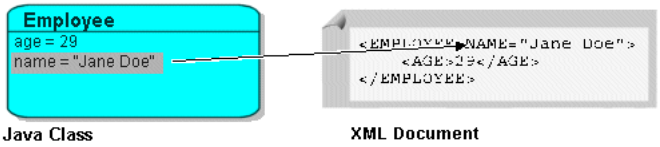


XPath by Name

For simple XML documents, TopLink XML mappings can correctly place data in an XML document given an XPath of only an attribute or element name.

Figure 30–14 illustrates mapping to a simple XML document by name. You can map Java object attribute `name` to XML attribute `name` by specifying an XPath of only `@NAME`. Similarly, you can map Java object attribute `age` to XML text node `AGE` by specifying an XPath of only `AGE`.

Figure 30–14 Mapping to a Simple XML Document by Name



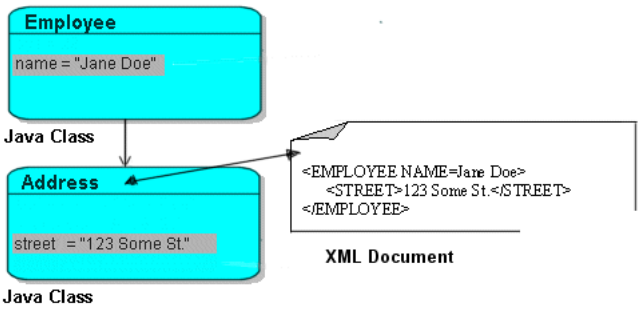
Specifying an XPath by name provides the worst performance of the XPath mapping options. Oracle recommends that you use XPath by position (see "XPath by Position" on page 30-15) or XPath by path and name (see "XPath by Path and Name" on page 30-15) instead.

Self XPath

For composite relationships, TopLink XML mappings can place data in the parent's element rather than an element nested within it given the self XPath (`"."`).

Figure 30–15 illustrates mapping to an XML document using the self XPath.

Figure 30–15 Mapping to a XML Document Using Self XPath



Note that in the preceding example represented by Figure 30–15, `name` attribute of the `Employee` class is mapped using the `@name` annotation.

Using the self XPath, you can make TopLink perform all read and write operations in the parent's element and not an element nested within it (see "Mappings and the jaxb:class Customization" on page 30-20).

Mappings and xsd:list and xsd:union Types

TopLink supports mapping to `xsd:list` and `xsd:union` types in EIS mappings to XML records and XML mappings to XML documents as Table 30–4 shows.

Table 30–4 TopLink Support for xsd:list and xsd:union Types

XSD	EIS Direct Mapping XML Direct Mapping	EIS Composite Direct Collection Mapping XML Composite Direct Collection Mapping
Mapping an xsd:union Type	✔	
Mapping an xsd:list Type	✔	✔
Mapping a List of Unions		✔
Mapping a Union of Lists	✔	✔
Mapping a Union of Unions	✔	

Mapping an xsd:union Type

Use an `EISDirectMapping` (with XML records) or an `XMLDirectMapping` to map a Java attribute to an `xsd:union` type, such as:

```
<xsd:simpleType name="size-type">
    <xsd:union memberTypes="xsd:decimal xsd:string"/>
</xsd:simpleType>
```

When TopLink marshalls (writes) an object to XML, it uses its default conversion pairs to convert from the Java type to the appropriate `xsd` type.

In the case where the `memberTypes` map to the same Java type, TopLink marshalls using the first `memberType` in the union which allows a successful conversion. For example, if you map a Java type of `byte[]` to an `xsd:union` with `memberTypes` of `hexBinary` and `base64Binary`, then TopLink marshalls using the first `memberType`: `hexBinary`.

You can customize the default conversion pairs to control the Java type to `xsd` type conversion using `XMLField` method `addConversion` and configuring your mapping with that `XMLField` using `EISDirectMapping` or `XMLDirectMapping` method `setField`. For example, if the `memberTypes` were `xsd:date` and `xsd:time` and the Java attribute was of type `java.util.Date` instead of the JAXB 1.0 standard `java.util.Calendar`, you can modify the conversion pair for `xsd:date` to be `java.util.Date`.

When TopLink unmarshalls (reads) XML into an object, it tries each `memberType` in the order specified in the XSD until the first successful conversion is made.

If your XML document specifies the `xsi:type` attribute on an element, then TopLink converts according to the `xsi:type` instead of trying the `memberTypes`.

For more information, see "Mapping to a Union Field With an XML Direct Mapping" on page 62-10. The same applies to an `EISDirectMapping` with XML records (see "EIS Direct Mapping" on page 53-5).

Mapping an xsd:list Type

You can map a Java attribute to an `xsd:list` type, such as:

```
<xsd:simpleType name="sizes">
    <xsd:list itemType="xsd:int"/>
</xsd:simpleType>
```

If you represent the `xsd:list` in your object model as a Java `List` type, use an `EISCompositeDirectCollectionMapping` (with XML records) or an `XMLCompositeDirectCollectionMapping` and use mapping method `useCollectionClass` to specify the `List` type of the Java attribute.

If you represent the list in your object model as a `String` of white space delimited tokens (for example, `"aaa bbb ccc"`), use an `EISDirectMapping` (with XML records) or an `XMLDirectMappng` to map this Java attribute to an `xsd:list` (for example, `<item>aaa bbb ccc</item>`).

In either case, you can configure whether or not the mapping unmarshalls (writes) the list to a single node, like `<item>aaa bbb ccc</item>`, or to multiple nodes, such as the following:

```
<item>aaa</item>
<item>bbb</item>
<item>ccc</item>
```

For more information on mapping to an `xsd:list` type using an `XMLCompositeDirectCollectionMapping`, see:

- "Mapping to a Single Text Node With an XML Composite Direct Collection Mapping" on page 62-17
- "Mapping to a Single Attribute With an XML Composite Direct Collection Mapping" on page 62-18
- "Specifying the Content Type of a Collection With an XML Composite Direct Collection Mapping" on page 62-20

The same applies to an `EISCompositeDirectCollectionMapping` (with XML records).

For more information about mapping to an `xsd:list` type using an `XMLDirectMapping`, see "Mapping to a List Field With an XML Direct Mapping" on page 62-10. The same applies to an `EISDirectMapping` with XML records (see "EIS Direct Mapping" on page 53-5).

Mapping a List of Unions

Use an `EISCompositeDirectCollectionMapping` (with XML records) or an `XMLCompositeDirectCollectionMapping` to map a Java attribute to an `xsd:list` that contains `xsd:union` types, such as:

```
<xsd:element name="listOfUnions" type="listOfUnions"/>
    <xsd:simpleType name="listOfUnions">
        <xsd:list>
            <xsd:simpleType>
                <xsd:union memberTypes="xsd:date xsd:integer"/>
            </xsd:simpleType>
        </xsd:list>
    </xsd:simpleType>
```

When TopLink marshalls (writes) an object to XML, it does not rely on a single `xsd:list itemType`. Instead, for each item in the list, TopLink tries each `memberType` until the first successful conversion.

For more information, see "Mapping to a List of Unions With an XML Composite Direct Collection Mapping" on page 62-18. The same applies to an `EISCompositeDirectCollectionMapping` with XML records (see "EIS Composite Direct Collection Mapping" on page 53-6).

Mapping a Union of Lists

You can map a Java attribute to an `xsd:union` type whose `memberTypes` are `xsd:list` types where each `xsd:list` contains items of a single type, such as:

```
<xsd:element name="listOfUnions" type="UnionOfLists"/>
    <xsd:simpleType name="UnionOfLists">
        <xsd:union memberTypes="xsd:double">
            <xsd:simpleType>
                <xsd:list itemType="xsd:date"/>
            </xsd:simpleType>
            <xsd:simpleType>
                <xsd:list itemType="xsd:integer"/>
            </xsd:simpleType>
        </xsd:union>
    </xsd:simpleType>
```

Note that in this example, valid XML documents contain either all `xsd:double`, all `xsd:date`, or all `xsd:integer` values.

If you represent the list in your object model as a `String` of white space delimited tokens (for example, `"aaa bbb ccc"`), use an `EISDirectMapping` (with XML records) or an `XMLDirectMappng` to map this Java attribute to an `xsd:list` (for example, `<item>aaa bbb ccc</item>`).

If you represent the list in your object model as a Java `List` type, use an `EISCompositeDirectCollectionMapping` (with XML records) or an `XMLCompositeDirectCollectionMapping`.

For more information, see the following:

- "Mapping to a Union of Lists With an XML Direct Mapping" on page 62-12. The same applies to an EISDirectMapping with XML records (see "EIS Direct Mapping" on page 53-5).
- "Mapping to a Union of Lists With an XML Composite Direct Collection Mapping" on page 62-19. The same applies to an EISCompositeDirectCollectionMapping with XML records (see "EIS Composite Direct Collection Mapping" on page 53-6).

Mapping a Union of Unions

Use an `EISDirectMapping` (with XML records) or an `XMLDirectMapping` to map a Java attribute to an `xsd:union` that contains `xsd:union` types, such as:

```
<xsd:simpleType name="UnionOfUnions">
    <xsd:union>
        <xsd:simpleType>
            <xsd:union>
              <xsd:simpleType>
                <xsd:list itemType="xsd:date"/>
              </xsd:simpleType>
              <xsd:simpleType>
                <xsd:list itemType="xsd:integer"/>
              </xsd:simpleType>
              </xsd:union>
```

```
      </xsd:simpleType>
      <xsd:simpleType>
            <xsd:union>
              <xsd:simpleType>
                <xsd:list itemType="xsd:string"/>
              </xsd:simpleType>
              <xsd:simpleType>
                <xsd:list itemType="xsd:float"/>
              </xsd:simpleType>
              </xsd:union>
      </xsd:simpleType>
    </xsd:union>
</xsd:simpleType>
```

Note that in this example, valid XML documents may contain any of `xsd:date`, `xsd:integer`, `xsd:string`, or `xsd:float`.

For more information, see "Mapping to a Union of Unions With an XML Direct Mapping" on page 62-12. The same applies to an `EISDirectMapping` with XML records (see "EIS Direct Mapping" on page 53-5).

Mappings and the jaxb:class Customization

Using the `jaxb:class` customization, you can declaratively specify an application-specific subclass of a schema-derived implementation class. This lets you write your own classes that extend JAXB's generated implementation classes. The JAXB runtime binding framework can then access your subclasses.

When you create an EIS composite object mapping to XML records or an XML composite object mapping to XML documents, you can configure the mapping's XPath ("Configuring XPath" on page 32-10) to accommodate `jaxb:class` customizations with the following XSD structures:

- all, choice, or sequence Structure
- group Structure
- sequence or choice Structure Containing a group
- group Structure Containing a sequence or choice
- group Structure Containing a sequence or choice

When mapping to `jaxb:class` customized structures, consider the limitations of TopLink support for this customization (see "Limitations of jaxb:class Customization Support" on page 30-23).

all, choice, or sequence Structure

You can use the `jaxb:class` customization with an `all`, `choice`, or `sequence` structure. Example 30–1 shows a `jaxb:class` customization of an `all` structure.

Example 30–1 jaxb:class Customization of an all Structure

```
<xsd:element name="employee">
    <xsd:complexType>
        <xsd:all>
            <xsd:annotation>
                <xsd:appinfo>
                    <jaxb:class name="period"/>
                </xsd:appinfo>
            </xsd:annotation>
            <xsd:element name="startDate" type="xsd:date"/>
            <xsd:element name="endDate" type="xsd:date"/>
```

```
        </xsd:all>
    </xsd:complexType>
</xsd:element>
```

This directs the JAXB compiler to create an inner class named `Period` in the owning element's class for the `all` structure. Use an `EISCompositeObjectMapping` (with XML records) or an `XMLCompositeObjectMapping` to map a Java attribute to this inner class.

For more information, see "XML Composite Object Mapping" on page 62-21. The same applies to an `EISCompositeObjectMapping` with XML records (see "EIS Composite Object Mapping" on page 53-7).

group Structure

You can use the `jaxb:class` customization with a `group` structure as Example 30–2 shows.

Example 30–2 jaxb:class Customization of a group Structure

```
<xsd:group name="G1">
    <xsd:annotation>
        <xsd:appinfo>
            <jaxb:class name="period"/>
        </xsd:appinfo>
    </xsd:annotation>
    <xsd:sequence>
        <xsd:element name="startDate" type="xsd:date"/>
        <xsd:element name="endDate" type="xsd:date"/>
    </xsd:sequence>
</xsd:group>

<xsd:element name="employee">
    <xsd:complexType>
        <xsd:group ref="G1"/>
    </xsd:complexType>
</xsd:element>
```

This directs the JAXB compiler to create an external wrapper class named `Period` for the `group` structure. Use an `EISCompositeObjectMapping` (with XML records) or an `XMLCompositeObjectMapping` to map a Java attribute to this external wrapper class.

For more information, see "XML Composite Object Mapping" on page 62-21. The same applies to an `EISCompositeObjectMapping` with XML records (see "EIS Composite Object Mapping" on page 53-7).

sequence or choice Structure Containing a group

You can use the `jaxb:class` customization with a `sequence` or `choice` structure that contains a `group`. Example 30–3 shows a `jaxb:class` customization of a `sequence` structure containing a `group` structure.

Example 30–3 jaxb:class Customization of a sequence Structure Containing a group

```
<xsd:element name="employee">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:annotation>
                <xsd:appinfo>
                    <jaxb:class name="EmploymentInfo"/>
                </xsd:appinfo>
            </xsd:annotation>
```

```
            <xsd:element name="id" type="xsd:int"/>
            <xsd:group ref="G1"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<xsd:group name="G1">
    <xsd:annotation>
        <xsd:appinfo>
            <jaxb:class name="Period"/>
        </xsd:appinfo>
    </xsd:annotation>
    <xsd:sequence>
        <xsd:element name="startDate" type="xsd:date"/>
        <xsd:element name="endDate" type="xsd:date"/>
    </xsd:sequence>
</xsd:group>
```

This directs the JAXB compiler to create an inner class named `EmploymentInfo` in the owning element's class for the `sequence` structure and an external wrapper class named `Period` for the `group` structure. The inner class references the external wrapper class. Use an `EISCompositeObjectMapping` (with XML records) or an `XMLCompositeObjectMapping` to map a Java attribute to this inner class.

For more information, see "XML Composite Object Mapping" on page 62-21. The same applies to an `EISCompositeObjectMapping` with XML records (see "EIS Composite Object Mapping" on page 53-7.

group Structure Containing a sequence or choice

You can use the `jaxb:class` customization with a `group` structure that contains a `sequence` or `choice`. Example 30–4 shows a `jaxb:class` customization of a `group` structure containing a `sequence` structure.

Example 30–4 jaxb:class Customization of a group Structure Containing a sequence

```
<xsd:group name="G1">
    <xsd:annotation>
        <xsd:appinfo>
            <jaxb:class name="EmploymentInfo"/>
        </xsd:appinfo>
    </xsd:annotation>
    <xsd:sequence>
        <xsd:annotation>
            <xsd:appinfo>
                <jaxb:class name="Period"/>
            </xsd:appinfo>
        </xsd:annotation>
        <xsd:element name="startDate" type="xsd:date"/>
        <xsd:element name="endDate" type="xsd:date"/>
    </xsd:sequence>
</xsd:group>

<xsd:element name="employee">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="id" type="xsd:int"/>
            <xsd:group ref="G1"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

This directs the JAXB compiler to create an external wrapper class named `EmploymentInfo` for the `group` structure and an inner class named `Period` in the

external wrapper class for the `sequence` structure. The owning element references the external wrapper class. Use an `EISCompositeObjectMapping` (with XML records) or an `XMLCompositeObjectMapping` to map a Java attribute to this external wrapper class.

For more information, see "XML Composite Object Mapping" on page 62-21. The same applies to an `EISCompositeObjectMapping` with XML records (see "EIS Composite Object Mapping" on page 53-7).

group Structure Containing a group

You can use the `jaxb:class` customization with a `group` structure that contains another `group` structure as Example 30–5 shows.

Example 30–5 jaxb:class Customization of a group Structure Containing a group

```
<xsd:group name="G1">
    <xsd:annotation>
        <xsd:appinfo>
            <jaxb:class name="EmploymentInfo"/>
        </xsd:appinfo>
    </xsd:annotation>
    <xsd:sequence>
        <xsd:element name="id" type="xsd:int"/>
        <xsd:group ref="G2"/>
    </xsd:sequence>
</xsd:group>

<xsd:group name="G2">
    <xsd:annotation>
        <xsd:appinfo>
            <jaxb:class name="Period"/>
        </xsd:appinfo>
    </xsd:annotation>
    <xsd:sequence>
        <xsd:element name="startDate" type="xsd:date"/>
        <xsd:element name="endDate" type="xsd:date"/>
    </xsd:sequence>
</xsd:group>

<xsd:element name="employee">
    <xsd:complexType>
        <xsd:group ref="G1"/>
    </xsd:complexType>
</xsd:element>
```

This directs the JAXB compiler to create a wrapper class named `EmploymentInfo` for the `group` structure that the owning element's class references and another wrapper class named `Period` for the `group` structure that the `EmploymentInfo` class references. Use an `EISCompositeObjectMapping` (with XML records) or an `XMLCompositeObjectMapping` to map a Java attribute to these wrapper classes.

For more information, see "XML Composite Object Mapping" on page 62-21. The same applies to an `EISCompositeObjectMapping` with XML records (see "EIS Composite Object Mapping" on page 53-7).

Limitations of jaxb:class Customization Support

When mapping to jaxb:class customized structures, consider the following limitations:

- Unbounded structures are not supported.
- Partial validation is not supported.

- When mapping sequence elements to a composite object, the XML schema must order the elements so that the elements you map to the composite object are kept together.

  The `sequence` structure forces all elements to occur in the order in which they are specified in the XML schema. Consider the XML schema shown in Example 30–6. A valid XML instance must contain the sequence elements in the specified order:

  `street`, `customerName`, `city`

  In this example, you want to map the `customerName` attribute with a direct mapping and you want to map the `street` and `city` attributes to a composite `Address` object. Depending on the order in which you define the mappings, TopLink will marshall invalid XML document instances in the order

  `customerName`, `street`, `city`

  or

  `street`, `city`, `customerName`.

Example 30–6 XML Schema With Unsupported Sequence Element Order

```
<xs:element name="customer">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="street" type="xs:string"/>
            <xs:element name="customerName" type="xs:string" />
            <xs:element name="city" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

To correct this problem, modify the XML schema to keep the elements you want to map to the composite object together (see Example 30–7) and define the mappings in the order specified by the XML schema.

Example 30–7 XML Schema With Supported Sequence Element Order

```
<xs:element name="customer">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="customerName" type="xs:string" />
            <xs:element name="street" type="xs:string"/>
            <xs:element name="city" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Mappings and JAXB Typesafe Enumerations

JAXB binds a typesafe enumeration class to a named simple type definition with a `basetype` that derives from `xsd:NCName` and has `enumeration` facets (see Example 30–8).

Example 30–8 Schema Fragment with Typesafe Enumeration Declaration

```
<simpleType name="NISTSchema-NCName-enumeration-1-Type">
    <restriction base="NCName">
        <enumeration value="qbandwidth-and.software-use.too"/>
```

```
        <enumeration value="_effort-disseminate_and-devices.com"/>
    </restriction>
</simpleType>
```

You can map a Java attribute to such an enumeration using the `JAXBTypesafeEnumConverter` with an `EISDirectMapping` or `EISCompositeDirectCollectionMapping` with XML records or with an `XMLDirectMapping` or `XMLCompositeDirectCollectionMapping` with XML documents.

TopLink Workbench does not support the `JAXBTypesafeEnumConverter` directly: to configure a mapping with this converter, you must use a descriptor amendment method (see "Configuring a JAXB Typesafe Enumeration Converter" on page 32-25).

If you create a project and object model using the TopLink JAXB compiler (see "Creating an XML Project From an XML Schema" on page 18-6), the compiler will create the type safe enumeration class and a class with descriptor amendment methods and register the required amendment methods automatically (see "Typesafe Enumeration Converter Amendment Method DescriptorAfterLoads Class" on page 17-12).

Understanding the Mapping API

All the mapping classes are derived from the `DatabaseMapping` class.

Table 30–5 Platform and Mapping Package Compatibility

Platform	Mapping Package
`DatabasePlatform`	`oracle.toplink.mappings`
For relational projects	`oracle.toplink.xdb`
	`oracle.toplink.objectrelational`
`EISPlatform`	`oracle.toplink.eis`
For EIS projects	`oracle.toplink.mappings.TransformationMapping`
`XMLPlatform`	`oracle.toplink.ox`
For XML projects	`oracle.toplink.mappings.TransformationMapping`

Relational Mappings

A relational mapping transforms any object data member type to a corresponding relational database (SQL) data source representation in any supported relational database. Relational mappings allow you to map an object model into a relational data-model.

Relational mappings can also transform object data members that reference other domain objects that are stored in other tables in the database and are related through foreign keys.

Use relational mappings in relational projects. For more information, see "Relational Projects" on page 17-6.

For more information about relational mappings, see Chapter 33, "Understanding Relational Mappings" and Chapter 34, "Configuring a Relational Mapping".

Note: Do not confuse relational mappings with object-relational mappings (see "Object-Relational Mappings" on page 30-26). An object-relational mapping transforms certain object data member types to structured data source representations optimized for storage in specialized object-relational databases such as Oracle Database. Object-relational mappings allow you to map an object model into an object-relational data-model. In general, you can use relational mappings with any supported relational database. You can only use Object-relational mappings with specialized object-relational databases optimized to support object-relational data source representations.

TopLink provides the following relational mappings:

- Chapter 35, "Configuring a Relational Direct-to-Field Mapping"
- Chapter 36, "Configuring a Relational Direct-to-XMLType Mapping"
- Chapter 37, "Configuring a Relational One-to-One Mapping"
- Chapter 38, "Configuring a Relational Variable One-to-One Mapping"
- Chapter 39, "Configuring a Relational One-to-Many Mapping"
- Chapter 40, "Configuring a Relational Many-to-Many Mapping"
- Chapter 41, "Configuring a Relational Aggregate Collection Mapping"
- Chapter 42, "Configuring a Relational Direct Collection Mapping"
- Chapter 44, "Configuring a Relational Direct Map Mapping"
- Chapter 43, "Configuring a Relational Aggregate Object Mapping"
- Chapter 45, "Configuring a Relational Transformation Mapping"

Object-Relational Mappings

An object-relational mapping transforms certain object data member types to structured data source representations optimized for storage in specialized object-relational databases such as Oracle Database. Object-relational mappings allow you to map an object model into an object-relational data-model.

Use object-relational mappings in relational projects. For more information, see "Relational Projects" on page 17-6.

For more information about object-relational mappings, see Chapter 46, "Understanding Object-Relational Mappings" and Chapter 47, "Configuring an Object-Relational Mapping".

Note: Do not confuse object-relational mappings with relational mappings (see "Relational Mappings" on page 30-25). A relational mapping transforms any object data member type to a corresponding relational database (SQL) data source representation in any supported relational database. Relational mappings allow you to map an object model into a relational data-model. In general, you can use relational mappings with any supported relational database. You can only use Object-relational mappings with specialized object-relational databases optimized to support object-relational data source representations.

TopLink provides the following object-relational mappings:

- Chapter 48, "Configuring an Object-Relational Structure Mapping"
- Chapter 49, "Configuring an Object-Relational Reference Mapping"
- Chapter 50, "Configuring an Object-Relational Array Mapping"
- Chapter 51, "Configuring an Object-Relational Object Array Mapping"
- Chapter 52, "Configuring an Object-Relational Nested Table Mapping"

XML Mappings

An XML mapping transforms object data members to the XML elements of an XML file whose structure is defined by an XML schema document (XSD).

Use XML mappings in XML projects. For more information, see "XML Projects" on page 17-9.

For more information about XML mappings, see Chapter 62, "Understanding XML Mappings" and Chapter 63, "Configuring an XML Mapping".

TopLink provides the following XML mappings:

- Chapter 64, "Configuring an XML Direct Mapping"
- Chapter 65, "Configuring an XML Composite Direct Collection Mapping"
- Chapter 66, "Configuring an XML Composite Object Mapping"
- Chapter 67, "Configuring an XML Composite Collection Mapping"
- Chapter 68, "Configuring an XML Any Object Mapping"
- Chapter 69, "Configuring an XML Any Collection Mapping"
- Chapter 70, "Configuring an XML Transformation Mapping"

EIS Mappings

An EIS mapping transforms object data members to the EIS record format defined by the object's descriptor.

Note: If you understand the concept of relational mappings, you understand the EIS mappings. For more information about relational mappings, see "Relational Mappings" on page 30-25.

Use EIS mappings in EIS projects. For more information, see "EIS Projects" on page 17-7.

For more information about EIS mappings, see Chapter 53, "Understanding EIS Mappings" and Chapter 54, "Configuring an EIS Mapping".

TopLink provides the following EIS mappings:

- Chapter 55, "Configuring an EIS Direct Mapping"
- Chapter 56, "Configuring an EIS Composite Direct Collection Mapping"
- Chapter 57, "Configuring an EIS Composite Object Mapping"
- Chapter 58, "Configuring an EIS Composite Collection Mapping"
- Chapter 59, "Configuring an EIS One-to-One Mapping"
- Chapter 60, "Configuring an EIS One-to-Many Mapping"
- Chapter 61, "Configuring an EIS Transformation Mapping"

31

Creating a Mapping

This chapter includes information on the following:

- Mapping Creation Overview
- Creating Mappings Manually During Development
- Creating Mappings Automatically During Development
- Creating Mappings Automatically During Deployment
- Creating Mappings to Oracle LOB Database Objects
- Removing Mappings

Mapping Creation Overview

You can create a database mapping using TopLink Workbench or Java code. Oracle recommends using TopLink Workbench to create and manage your mappings.

For more information on creating mappings in Java, see Oracle TopLink API Reference

For complete information on the various types of mapping that TopLink supports, see "Mapping Types" on page 30-1.

During development, you can create mappings individually (see "Creating Mappings Manually During Development" on page 31-1) or you can allow TopLink Workbench to automatically map all attributes (see "Creating Mappings Automatically During Development" on page 31-2).

For CMP projects using OC4J, you can also configure TopLink to create mappings automatically at deployment time (see "Creating Mappings Automatically During Deployment" on page 31-2).

After you create a mapping, you must configure its various options (see Chapter 32, "Configuring a Mapping").

Creating Mappings Manually During Development

You can manually create a mapping from each persistent field of a class to a data source using TopLink Workbench or Java code. Oracle recommends that you use TopLink Workbench.

Using TopLink Workbench

To manually create a mapping using TopLink Workbench, use this procedure:

1. Select a descriptor in the **Navigator**. Its properties appear in the Editor.

2. On the **Descriptor Info** tab, associate the descriptor with the data source (see Chapter 25, "Configuring a Descriptor"). You must associate descriptors with a database table or schema context before mapping their attributes.
3. In the **Navigator**, expand the descriptor to display its attributes.
4. Select an attribute and click the appropriate mapping on the toolbar (see "Context Toolbar" on page 4-7), or right-click the attribute and select **Map As >** *specific mapping* from the menu (see "Context Menus" on page 4-5).

Continue with Chapter 32, "Configuring a Mapping" to complete the mapping.

Creating Mappings Automatically During Development

For relational database projects, TopLink Workbench can automatically map class attributes to a similarly named database field. For example, TopLink Workbench can map the attribute `province` to the database field `PROV`, the attribute `street` to the field `ST`, and the attribute `postalCode` to the field `POSTAL_CODE`.

The Automap function creates mappings only for unmapped attributes–it does not change previously defined mappings.

You can automatically map classes for an entire project or for specific classes or descriptors. If TopLink cannot

---

Note: Associating a descriptor with a database table (see "Configuring Associated Tables" on page 26-2) before using the Automap function can aid the automapper if it cannot guess the correct table for a class.

Using TopLink Workbench

To automatically map *all* the descriptors in a project, right-click the project icon in the **Navigator** window and choose **Automap** from the context menu or choose **Selected > Automap** from the menu.

To automatically map a *specific* descriptor or attribute, choose the descriptor or attributes and right-click, and then select **Automap** from the context menu or choose **Selected > Automap** from the menu.

Creating Mappings Automatically During Deployment

If you create a project from an OC4J EJB CMP EAR at deployment time, you can take advantage of the TopLink default mapping feature to automatically create mappings at deployment time.

For more information, see the following:

- "Creating a Project From an OC4J EJB CMP EAR at Deployment Time" on page 18-10
- "Default Mapping in CMP Projects Using OC4J at Run Time" on page 30-4

Creating Mappings to Oracle LOB Database Objects

In an Oracle Database, large amounts of binary or character data is stored as a `BLOB` (binary large object) or `CLOB` (character large object), respectively. Depending on the

size of the LOB value and your Oracle Database database version, the value may be stored either inside or outside of the table.

- With Oracle8i version 8.1.6 and earlier, LOB values less than 4K are stored inline; values more than 4K are stored outside the table.
- With Oracle8i version 8.1.7 and later, LOB values less than 5.9K are stored inline; values more than 5.9K are stored outside the table.

A client application (such as Oracle TopLink) must use a LOB locator to write a LOB value, if the value is stored outside of the database. The Oracle JDBC OCI driver and server driver handle these LOB (large object) values differently than the Oracle JDBC thin driver.

Using the Oracle JDBC OCI Driver or Server Driver

When using the Oracle JDBC OCI or Server driver, the driver is responsible for acquiring the LOB locator before writing the value–not TopLink. You can use any TopLink mapping type to read and write a LOB value.

Use this procedure to map LOB values using the Oracle JDBC OCI driver:

1. Use any mapping type to map the attributes of a TopLink descriptor to a LOB value. Example 31–1 shows an example of a direct-to-field mapping to a BLOB value.

Example 31–1 Mapping a BLOB with a Direct-To-Field Mapping

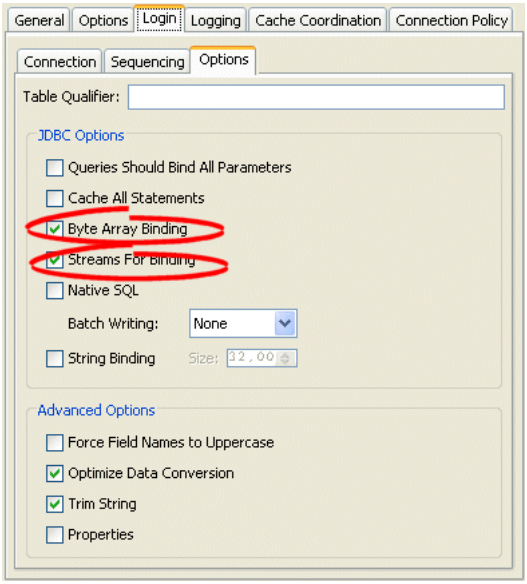
```
DirectToField pictureMapping = new DirectToField();
pictureMapping.setAttributeName("picture");
pictureMapping.setFieldName("IMAGE.PICTURE");
descriptor.addMapping(pictureMapping);
```

2. Acquire the DatabaseLogin from the session.

   ```
   DatabaseLogin login = session.getLogin();
   ```

3. Configure the parameter bindings for the specific LOB value:
   - For BLOB values, enable byte array binding to ensure that byte arrays are bound (in the event that `shouldBindAllParamters` is false) and enable the streams for binding to ensure that stream binding is used for byte arrays. Example 31–2 shows configuring the parameter bindings in TopLink Workbench. Example 31–3 shows configuring the parameter bindings in TopLink Workbench.

Example 31–2 Configuring Parameter Bindings in TopLink Workbench

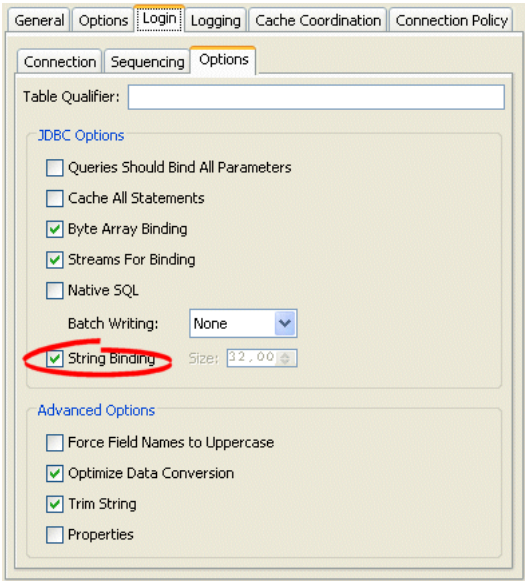


Example 31–3 Configuring Parameter Bindings in Java Code

```
login.setUsesByteArrayBinding(true);
login.setUsesStreamsForBinding(true);
```

- For `CLOB` values, enable string binding to ensure that long string values (more than 255 characters) are bound as character streams. Example 31–4 shows configuring string binding in TopLink Workbench. Example 31–5 shows configuring string binding in TopLink Workbench.

Example 31–4 Configuring String Binding in TopLink Workbench



Example 31–5 Configuring String Binding in Java Code

```
login.useStringBinding();
```

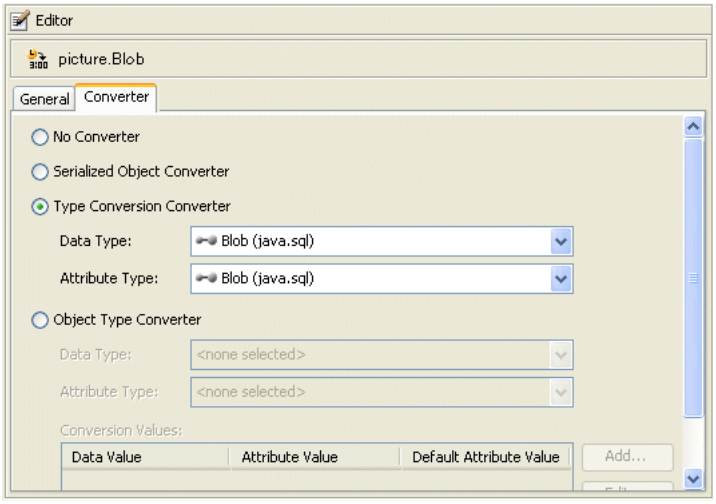
Using the Oracle JDBC Thin Driver

When using the Oracle JDBC thin driver, TopLink is responsible for acquiring the LOB locator before writing the value. You can use a type conversion mapping (see "Configuring a Type Conversion Converter" on page 32-20) to retrieve the locator during a commit operation.

Use this procedure to map `LOB` values using the Oracle JDBC thin driver:

1. Use a type conversion mapping to map the attributes of a TopLink descriptor to a `LOB` value. Example 31–6 shows a type conversion mapping to a `BLOB` value in TopLink Workbench. Example 31–6 shows the Java code for the same mapping.

Example 31–6 Mapping a BLOB in TopLink Workbench



Example 31–7 Mapping a BLOB in Java Code

```
TypeConversionMapping pictureMapping = new TypeConversionMapping();
pictureMapping.set.AttributeName("picture");
pictureMapping.setFieldName("IMAGE.PICTURE");
pictureMapping.setFieldClassification(java.sql.Blob.class);
descriptor.addMapping(pictureMapping);
```

2. Acquire the DatabaseLogin from the session.

   ```
   DatabaseLogin login = session.getLogin();
   ```

3. Configure a platform that provides locator support.

   - For Oracle8i, use the `oracle.toplink.oraclespecific.Oracle8Platform` class:

     ```
     login.usePlatform(new Oracle8Platform());
     ```

   - For Oracle9*i* Database Server, use the `oracle.toplink.oraclespecific.Oracle9Platform` class:

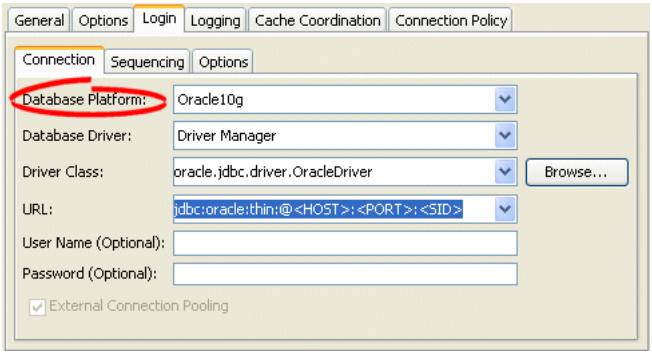
     ```
     login.usePlatform(new Oracle9Platform());
     ```

   - Oracle Database 10*g*, use the `oracle.toplink.oraclespecific.Oracle10Platform` class:

     ```
     login.usePlatform(new Oracle10Platform());
     ```

In TopLink Workbench, select the appropriate platform in the Database editor.

Figure 31–1 Selecting Database Platform in TopLink Workbench



Removing Mappings

If you are using TopLink Workbench, you can unmap any mapped attribute.

Using TopLink Workbench

To unmap an attribute (that is, remove its mapping), use this procedure:

- Select the attribute in the **Navigator** window and click **Unmap**. You can also unmap the attribute by right-clicking the attribute and selecting **Map As > Unmapped** from the context menu.

To unmap all the attributes in a descriptor or Java package, use this procedure:

- Right-click the descriptor or Java package in the **Navigator** window and select **Unmap > Unmap Selected Descriptor** or **Unmap All Descriptors in Package** from the context menu.

32

Configuring a Mapping

This chapter describes how to configure TopLink mappings.

Table 32–1 lists the types of TopLink mappings that you can configure and provides a cross-reference to the type-specific chapter that lists the configurable options supported by that type.

Table 32–1 Configuring TopLink Mappings

If you are creating...	See...
Relational Mappings	Chapter 34, "Configuring a Relational Mapping"
Object-Relational Mappings	Chapter 47, "Configuring an Object-Relational Mapping"
EIS Mappings	Chapter 54, "Configuring an EIS Mapping"
XML Mappings	Chapter 63, "Configuring an XML Mapping"

Table 32–2 lists the configurable options shared by two or more TopLink mapping types.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "Understanding Mappings" on page 30-1

Configuring Common Mapping Options

Table 32–2 lists the configurable options shared by two or more TopLink mapping types. In addition to the configurable options described here, you must also configure the options described for the specific Mapping Types, as shown in Table 32–1

Table 32–2 Common Mapping Options

Option	Type	TopLink Workbench	Java
"Configuring Read-Only Mappings" on page 32-2	Basic	✔	✔
"Configuring Indirection" on page 32-3	Basic	✔	✔
"Configuring XPath" on page 32-10	Basic	✔	✔
"Configuring a Default Null Value at the Mapping Level" on page 32-12	Basic	✔	✔
"Configuring Method Accessing" on page 32-14	Advanced	✔	✔
"Configuring Private or Independent Relationships" on page 32-16	Advanced	✔	✔
"Configuring Mapping Comments" on page 32-18	Advanced	✔	

Table 32–2 (Cont.) Common Mapping Options

Option	Type	TopLink Workbench	Java
"Configuring a Serialized Object Converter" on page 32-18	Advanced	✔	✔
"Configuring a Type Conversion Converter" on page 32-20	Advanced	✔	✔
"Configuring an Object Type Converter" on page 32-22	Advanced	✔	✔
"Configuring a Simple Type Translator" on page 32-23	Advanced	✔	✔
"Configuring a JAXB Typesafe Enumeration Converter" on page 32-25	Advanced		✔
"Configuring Container Policy" on page 32-26	Advanced	✔	✔
"Configuring Attribute Transformer" on page 32-29	Advanced	✔	✔
"Configuring Field Transformer Associations" on page 32-31	Advanced	✔	✔
"Configuring Mutable Mappings" on page 32-33	Advanced	✔	✔
"Configuring Bidirectional Relationship" on page 32-34	Advanced	✔	✔
"Configuring the Use of a Single Node" on page 32-36	Advanced	✔	✔

Configuring Read-Only Mappings

Mappings that are read-only will not be affected during insert, update, and delete operations.

Use read-only mappings when multiple attributes in an object map to the same fields in the database but only one of the mappings can write to the field.

You can also use read-only mappings with bi-directional many-to-many mappings to designate which mapping will be responsible for updating the many-to-many join table.

Note: The primary key mappings cannot not be 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 or the class indicator is part of the primary key.

Use read-only mappings only if specific mappings in a descriptor are read-only. If the *entire descriptor* is read-only, use the descriptor-level setting (see "Configuring Read-Only Descriptors" on page 25-5).

Table 32–3 summarizes which mappings support this option.

Table 32–3 Mapping Support for Read-Only

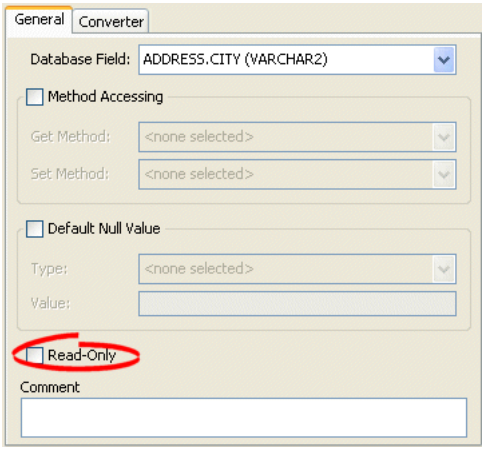
Mapping	Using TopLink Workbench	Using Java
Relational Mappings	✔	✔
Object-Relational Mappings		✔
EIS Mappings	✔	✔
XML Mappings	✔	✔

Using TopLink Workbench

To specify a mapping as read-only, use this procedure:

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.

Figure 32–1 General Tab, Read-Only Option



Select the **Read-Only** option to set the mapping to be read-only and not affected during update and delete operations.

Using Java

Use the following `DatabaseMapping` methods to configure the read access of a mapping:

- `readOnly`–configures mapping read access to read-only
- `readWrite`–configures mapping read access to read and write (default)

Example 32–1 shows how to use these methods with a class that has a read-only attribute named `phones`.

Example 32–1 Configuring Read Only Mappings in Java

```
// Map the phones attribute
phonesMapping.setAttributeName("phones");

// Specify read-only
phonesMapping.readOnly();
```

Configuring Indirection

By default, when TopLink retrieves a persistent object, it retrieves all of the dependent objects to which it refers. When you enable indirection for an attribute mapped with a relationship mapping, TopLink uses an indirection object as a placeholder for the referenced object: TopLink defers reading the dependent object until you access that specific attribute. This can result in a significant performance improvement, especially if the application is interested only in the contents of the retrieved object rather than the objects to which it refers.

Oracle strongly recommends using indirection for all relationship mappings. Not only does this allow you to optimize data source access, but it also allows TopLink to optimize the unit of work processing, cache access, and concurrency.

Table 32–4 summarizes which mappings support this option.

Table 32–4 Mapping Support for Indirection

Mapping	Value Holder Indirection	Transparent Indirect Container Indirection	Proxy Indirection	Using TopLink Workbench	Using Java
Relational Mappings					
Direct-to-Field Mapping	✔	✔	✔		✔
Transformation Mapping	✔	✔		✔	✔
One-to-One Mapping	✔	✔	✔	✔	✔
Variable One-to-One Mapping	✔	✔	✔	✔	✔
One-to-Many Mapping	✔	✔	✔	✔	✔
Many-to-Many Mapping	✔	✔	✔	✔	✔
Aggregate Collection Mapping	✔	✔			✔
Direct Collection Mapping	✔	✔	✔	✔	✔
Direct Map Mapping	✔	✔		✔	✔
Object-Relational Mappings					
Object-Relational Reference Mapping	✔	✔	✔		✔
Object-Relational Nested Table Mapping	✔	✔	✔		✔
EIS Mappings					
EIS One-to-One Mapping	✔	✔	✔	✔	✔
EIS One-to-Many Mapping	✔	✔	✔	✔	✔
XML Mappings					
XML Transformation Mapping	✔	✔			✔

In general, Oracle recommends that you use value holder indirection (see "Value Holder Indirection" on page 30-7) for one-to-one mappings and transparent indirect container indirection (see "Transparent Indirect Container Indirection" on page 30-8) for collection mappings. Enable indirection for transformation mappings if the execution of the transformation is a resource-intensive task (such as accessing a database, in a relational project).

When using indirection with EJB, the version of EJB and application server you use affects how indirection is configured and what types of indirection are applicable (see "Indirection and EJB" on page 30-9).

When using indirection with an object that your application serializes, you must consider the effect of any untriggered indirection objects at deserialization time (see "Indirection, Serialization, and Detachment" on page 30-9).

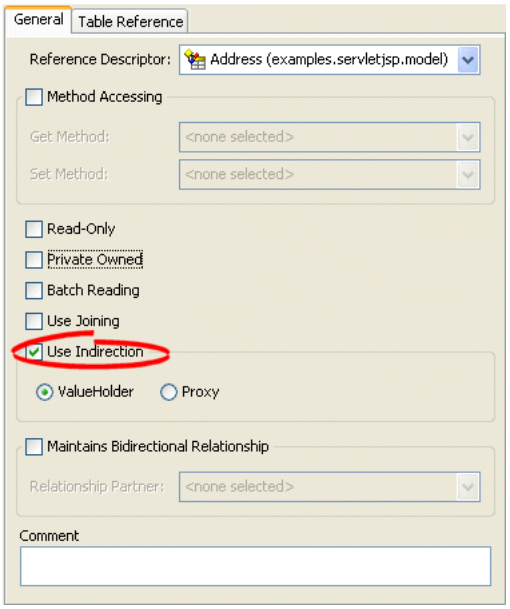
For more information, see "Indirection" on page 30-5.

Using TopLink Workbench

To complete the indirection options on a mapping's **General** tab use this procedure:

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.

Figure 32–2 General Tab, Indirection Options



Use the following information to complete the **Indirection** fields on the tab:

Field	Description
Use Indirection	Specify if this mapping uses indirection.
ValueHolder	Specify that the mapping uses **Value Holder** indirection. See "Value Holder Indirection" on page 30-7 for more information.
Proxy	Specify that the mapping uses **Proxy** indirection. See "Proxy Indirection" on page 30-8 for more information.

Using Java

When creating mappings through the Java API, all foreign reference mappings default to using value-holder indirection and all transformation mappings default to not using indirection.

To disable indirection use `ForeignReferenceMapping` method `dontUseIndirection`.

To enable value holder indirection, use `ForeignReferenceMapping` method `useBasicIndirection`.

To enable transparent container indirection, use one of the following `CollectionMapping` methods:

- `useTransparentCollection`
- `useTransparentList`
- `useTransparentMap`
- `useTransparentSet`

To enable proxy indirection, use `ObjectReferenceMapping` method `useProxyIndirection`.

This section provides additional information on the following:

- Configuring ValueHolder Indirection
- Configuring ValueHolder Indirection With Method Accessing
- Configuring ValueHolder Indirection With EJB 3.0 on OC4J
- Configuring IndirectContainer Indirection
- Configuring Proxy Indirection

Configuring ValueHolder Indirection

Instances of `oracle.toplink.mappings.ForeignReferenceMapping` and `oracle.toplink.mappings.foundation.AbstractTransformationMapping` provide the `useBasicIndirection` method to configure a mapping to an attribute that you code with an `oracle.toplink.indirection.ValueHolderInterface` between it and the real object.

If the attribute is of a `Collection` type (such as a `Vector`), then you can either use an `IndirectContainer` (see "Configuring IndirectContainer Indirection" on page 32-8) or define the `ValueHolder` in the constructor as follows:

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

Example 32–2 illustrates the `Employee` class using `ValueHolder` indirection. The class definition conceals the use of ValueHolder within the existing getter and setter methods.

Example 32–2 Class Using ValueHolder Indirection

```
public class Employee {

    protected ValueHolderInterface address;

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

    public Address getAddress() {
        return (Address) this.addressHolder.getValue();
    }

    public void setAddress(Address address) {
        this.addressHolder.setValue(address);
    }
}
```

Example 32–3 shows how to configure a one-to-one mapping to the `address` attribute.

Example 32–3 Mapping Using ValueHolder Indirection

```
OneToOneMapping mapping = new OneToOneMapping();
mapping.useBasicIndirection();
mapping.setReferenceClass(Employee.class);
mapping.setAttributeName("address");
```

The application uses `Employee` methods `getAddress` and `setAddress` to access the `Address` object. Because basic indirection is enabled, TopLink expects the persistent fields to be of type `ValueHolderInterface`.

Configuring ValueHolder Indirection With Method Accessing

If you are using `ValueHolder` indirection with method accessing (see "Configuring Method Accessing" on page 32-14), in addition to changing your attributes types in your Java code to `ValueHolderInterface`, you must also provide TopLink with two pairs of getter and setter methods:

- getter and setter of the *indirection* object that are registered with the mapping and used only by TopLink. They include a `get` method that returns an instance that conforms to `ValueHolderInterface`, and a `set` method that accepts one argument that conforms to the same interface.
- getter and setter of the actual attribute value used by the application

Example 32–2 illustrates the `Employee` class using `ValueHolder` indirection with method access. The class definition is modified so that the `address` attribute of `Employee` is a `ValueHolderInterface` instead of an `Address`, and appropriate getter and setter methods are supplied.

Example 32–4 Class Using ValueHolder Indirection with Method Accessing

```
public class Employee {

    protected ValueHolderInterface address;

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

    // getter and setter registered with the mapping and used only by TopLink
    public ValueHolderInterface getAddressHolder() {
        return address;
    }
    public void setAddressHolder(ValueHolderInterface holder) {
        address = holder;
    }

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

Example 32–3 shows how to configure a one-to-one mapping to the `address` attribute.

Example 32–5 Mapping Using ValueHolder Indirection with Method Accessing

```
OneToOneMapping mapping = new OneToOneMapping();
mapping.useBasicIndirection();
mapping.setReferenceClass(Employee.class);
mapping.setAttributeName("address");
mapping.setGetMethodName("getAddressHolder");
mapping.setSetMethodName("setAddressHolder");
```

The application uses `Employee` methods `getAddress` and `setAddress` to access the `Address` object. Because basic indirection is enabled, TopLink uses `Employee` methods `getAddressHolder` and `setAddressHolder` methods when performing persistence operations on instances of `Employee`.

Configuring ValueHolder Indirection With EJB 3.0 on OC4J

When using indirection with EJB 3.0 persistence on OC4J (see "Indirection and EJB" on page 30-9), if your application serializes any indirection-enabled enterprise bean (see "Indirection, Serialization, and Detachment" on page 30-9), then, to preserve untriggered indirection objects on deserialization, configure your client to use the same Java agent that OC4J uses, as follows:

1. Include the following JAR files (from `<TOPLINK_HOME>\jlib`) in your client classpath:
   - `toplink.jar`
   - `toplink-agent.jar`
   - `asm.jar`
   - `asm-util.jar`
   - `ejb3-toplink-session.xml` (and whatever project deployment XML it refers to)
2. Add the following argument to the Java command line you use to start your client:

```
-javaagent:toplink-agent.jar
```

Configuring IndirectContainer Indirection

Instances of `oracle.toplink.mappings.ForeignReferenceMapping` and `oracle.toplink.mappings.foundation.AbstractTransformationMapping` provide the `useContainerIndirection` method to configure a mapping to an attribute that you code with an `oracle.toplink.indirection.IndirectContainer` between it and the real object.

Using an `IndirectContainer`, a `java.util.Collection` class can act as a TopLink indirection object: the `Collection` will only read its contents from the database when necessary (typically, when a `Collection` accessor is invoked). Without an `IndirectContainer`, all members of the `Collection` must be retrieved when the `Collection` attribute is accessed.

Example 32–2 illustrates the `Employee` class using `IndirectContainer` indirection with method access. The class definition is modified so that the `addresses` attribute of `Employee` is an `IndirectContainer` instead of an `Addresses`, and appropriate getter and setter methods are supplied. In this example, `addresses` is a `java.util.List`, so an instance of `oracle.toplink.indirection.IndirectList` is used.

Example 32–6 Class Using IndirectContainer Indirection

```
public class Employee {

    protected IndirectList addresses;

    // Initialize ValueHolders in constructor
    public Employee() {
        addresses = new IndirectList();
    }

    // getter and setter methods registered with the mapping and used by TopLink
    public ValueHolderInterface getAddressesHolder() {
        return addresses.getValueHolder();
    }
    public void setAddressesHolder(ValueHolderInterface holder) {
```

```
        addresses = addresses.setValueHolder(holder);
    }

    // getter and setter methods used by the application to access the attribute
    public List getAddresses() {
        return (List) addresses.getValue();
    }
    public void setAddresses(List newAddresses) {
        addresses.removeAll();
        addresses.addAll(newAddresses);
    }
}
```

Example 32–3 shows how to configure a one-to-one mapping to the `addresses` attribute.

Example 32–7 Mapping Using IndirectContainer Indirection

```
OneToOneMapping mapping = new OneToOneMapping();
mapping.useBasicIndirection();
mapping.setReferenceClass(Employee.class);
mapping.setAttributeName("addresses");
mapping.setGetMethodName("getAddressesHolder");
mapping.setSetMethodName("setAddressesHolder");
```

Configuring Proxy Indirection

Example 32–8 illustrates an `Employee` to `Address` one-to-one relationship.

Example 32–8 Proxy indirection Examples

```
public interface Employee {
    public String getName();
    public Address getAddress();
    public void setName(String value);
    public void setAddress(Address value);
    . . .
}
public class EmployeeImpl implements Employee {
    public String name;
    public Address address;
    . . .
    public Address getAddress() {
        return this.address;
    }
    public void setAddress(Address value) {
        this.address = value;
    }
}
public interface Address {
    public String getStreet();
    public void setStreet(String value);
    . . .
}
public class AddressImpl implements Address {
    public String street;
    . . .
}
```

In Example 32–8, both the `EmployeeImpl` and the `AddressImpl` classes implement public interfaces (`Employee` and `Address` respectively). Therefore, because the `AddressImpl` class is the target of the one-to-one relationship, it is the only class that must implement an interface. However, if the `EmployeeImpl` is ever to be the target

of another one-to-one relationship using transparent indirection, it must also implement an interface, as shown in the following example:

```
Employee emp = (Employee) session.readObject(Employee.class);
System.out.println(emp.toString());
System.out.println(emp.getAddress().toString());
// Would print:
[Employee] John Smith
{ IndirectProxy: not instantiated }
String street = emp.getAddress().getStreet();
// Triggers database read to get Address information
System.out.println(emp.toString());
System.out.println(emp.getAddress().toString());
// Would print:
[Employee] John Smith
{ [Address] 123 Main St. }
```

Using proxy indirection does not change how you instantiate your own domain objects for an insert operation. You still use the following code:

```
Employee emp = new EmployeeImpl("John Smith");
Address add = new AddressImpl("123 Main St.");
emp.setAddress(add);
```

Configuring XPath

TopLink uses XPath statements to map the attributes of a Java object to locations in an XML document. When you create an XML mapping or EIS mapping using XML records, you can specify the XPath based on any of the following:

- Name
- Position
- Path and name

Table 32–5 summarizes which mappings support this option.

Table 32–5 Mapping Support for XPath

Mapping	Using TopLink Workbench	Using Java
EIS Mappings[1]		
EIS Direct Mapping	✔	✔
EIS Composite Direct Collection Mapping	✔	✔
EIS Composite Object Mapping[2]	✔	✔
EIS Composite Collection Mapping	✔	✔
XML Mappings		
XML Direct Mapping	✔	✔
XML Composite Direct Collection Mapping	✔	✔
XML Composite Object Mapping[2]	✔	✔
XML Composite Collection Mapping	✔	✔
XML Any Object Mapping	✔	✔
XML Any Collection Mapping	✔	✔

[1] When used with XML records only (see "Configuring Record Format" on page 28-5).

[2] Supports the self XPath (".") so that the TopLink runtime performs all read and write operations in the parent's element and not an element nested within it (see "Mappings and the jaxb:class Customization" on page 30-20).

Before you can select an XPath for a mapping, you must associate the descriptor with a schema context (see "Configuring Schema Context for an EIS Descriptor" on page 28-2 or "Configuring Schema Context for an XML Descriptor" on page 29-1).

For more information, see "Mappings and XPath" on page 30-15.

Using TopLink Workbench

Use this table to select the XPath for an XMl mapping or EIS mapping using XML records:

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.
2. If necessary, click the **General** tab. The General tab appears.

Figure 32–3 General Tab, XPath Options

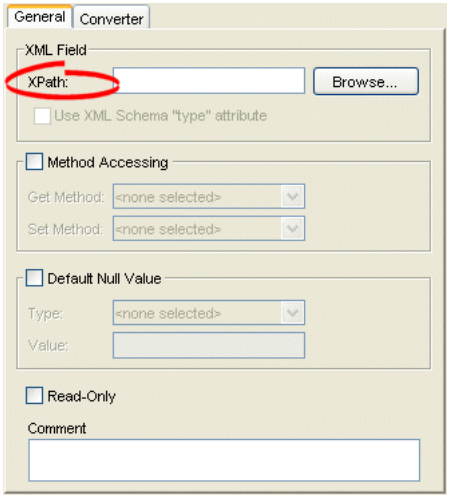
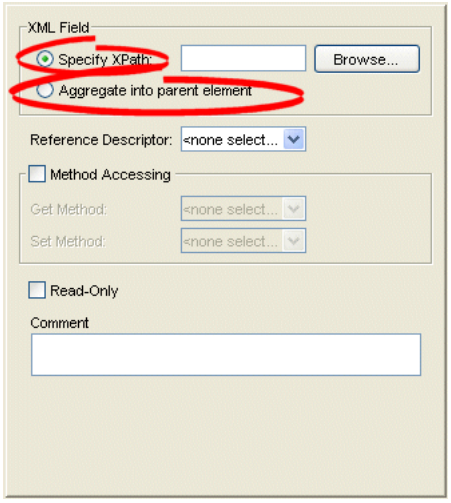


Figure 32–4 XPath Options for Composite Object Mappings



Click **Browse** and select the XPath to map to this attribute (see "Choosing the XPath").

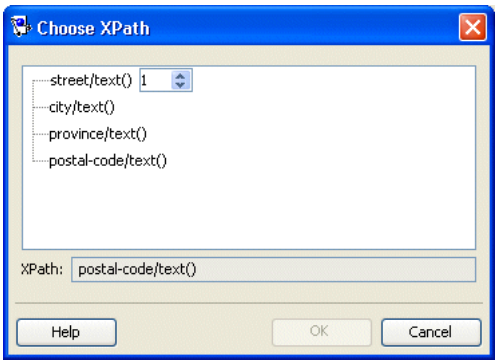
For an EIS composite object mapping using XML records or an XML composite object mapping, you can choose one of the following:

- **Specify XPath**: select the XPath to map to this attribute (see "Choosing the XPath").
- **Aggregate into parent element**: select the self XPath (`"."`) (see "Self XPath" on page 30-16) so that the TopLink runtime performs all read and write operations in the parent's element, and not an element nested within it (see "Mappings and the jaxb:class Customization" on page 30-20).

Choosing the XPath

From the Choose XPath dialog box, select the XPath and click **OK**. TopLink Workbench builds the complete XPath name.

Figure 32–5 Choose XPath Dialog Box



Configuring a Default Null Value at the Mapping Level

A default null value is the Java `Object` type and value that TopLink uses instead of `null` when TopLink reads a null value from a data source.

When you configure a default null value at the mapping level, TopLink uses it to translate in two directions:

- When TopLink reads `null` from the data source, it converts this `null` to the specified type and value.
- When TopLink writes or queries to the data source, it converts the specified type and value back to `null`.

Table 32–6 summarizes which mappings support this option.

Table 32–6 Mapping Support for Default Null Values

Mapping	Using TopLink Workbench	Using Java
Relational Mappings		
Direct-to-Field Mapping	✔	✔
Direct-to-XMLType Mapping	✔	✔
EIS Mappings		
EIS Direct Mapping	✔	✔
XML Mappings		

Table 32–6 (Cont.) Mapping Support for Default Null Values

Mapping	Using TopLink Workbench	Using Java
XML Direct Mapping	✔	✔
XML Composite Direct Collection Mapping	✔	✔

Note: A default null value must be an `Object`. To specify a primitive value (such as `int`), you must use the corresponding `Object` wrapper (such as `Integer`).

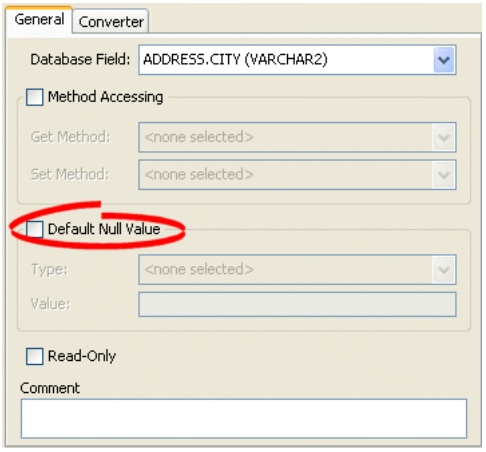
You can also use TopLink to set a default null value for all mappings used in a session (see "Configuring a Default Null Value at the Login Level" on page 82-5).

Using TopLink Workbench

To configure a default null value for a mapping, use this procedure.

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.

Figure 32–6 General Tab, Default Null Value Options



Use the following information to complete the **Default Null Value** fields on the tab:

Field	Description
Default Null Value	Specify if this mapping contains a default value in the event that the data source is `null`. If selected, you must enter both the **Type** and **Value** of the default.
Type	Select the Java type of the default value.
Value	Enter the default value.

Using Java

To configure a mapping null value using Java API, use the `AbstractDirectMapping` method `setNullValue`.

For example:

```
// Defaults a null salary to 0
salaryMapping.setNullValue(new Integer(0));
```

Configuring Method Accessing

By default, TopLink uses direct access to access public attributes. Alternatively, you can use getter and setter methods to access object attributes when writing the attributes of the object to the database, or reading the attributes of the object from the database. This is known as method access.

The attribute's visibility (public, protected, private, or package visibility) and the supported version of JDK may restrict the type of access that you can use.

Using private, protected or package variable or method access requires you to enable the Java reflect security setting. This is enabled by default in most application servers (see "Security Permissions" on page 7-4), but may need to be enabled explicitly in certain JVM configurations. If necessary, use the `java.policy` file to grant `ReflectPermission` to the entire application or the application's code base. For example:

```
grant{
     permission java.lang.reflect.ReflectPermission;
};
```

Oracle recommends using *direct access* whenever possible to improve performance and avoid executing any application-specific behavior while building objects.

Table 32–7 summarizes which mappings support this option.

Table 32–7 Mapping Support for Method Accessing

Mapping	Using TopLink Workbench	Using Java
Relational Mappings	✔	✔
Object-Relational Mappings		✔
EIS Mappings	✔	✔
XML Mappings	✔	✔

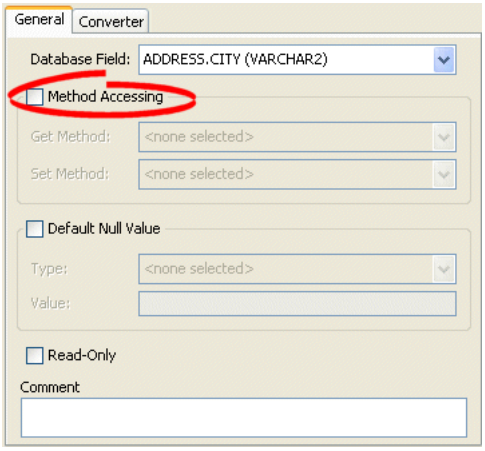
For information on configuring method accessing at the project level, see "Configuring Mapped Field Access at the Project Level" on page 19-4.

Using TopLink Workbench

To complete the field access method for a mapping, use this procedure:

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.

Figure 32–7 General Tab, Method Accessing Options



Use the following information to complete the **Method Accessing** fields on this tab:

Field	Description
Method Accessing	Specify if this mapping uses specific accessor methods instead directly accessing public attributes. By default, this option is not selected (that is, the mapping uses direct access).
Get Method	Select a specific `get` method.
Set Method	Select a specific `set` method.

To change the default access type used by all new mappings, use the Defaults tab on the project Editor window. See "Configuring Mapped Field Access at the Project Level" on page 19-4 for more information.

Using Java

Use the following `DatabaseMapping` methods to configure the user-defined getters and setters that TopLink will use to access the mapped attribute:

For mappings not supported in TopLink Workbench, use the `setGetMethodName` and `setSetMethodName` methods to access the attribute through user-defined methods, rather than directly.

- `setGetMethodName`–set the `String` name of the user-defined method to get the mapped attribute
- `setSetMethodName`–set the `String` name of the user-defined method to set the mapped attribute

Example 32–9 shows how to use these methods with a class that has an attribute `phones` and accessor methods `getPhones` and `setPhones` in an object-relational mapping.

Example 32–9 Configuring Access Method in Java

```
// Map the phones attribute
phonesMapping.setAttributeName("phones");

// Specify access method
phonesMapping.setGetMethodName("getPhones");
phonesMapping.setSetMethodName("setPhones");
```

Configuring Private or Independent Relationships

In TopLink, object relationships can be either private or independent:

- In a private relationship, the target object is a private component of the source object. The target object cannot exist without the source and is accessible only through the source object. Destroying the source object will also destroy the target object.
- In an independent relationship, the source and target objects are public ones that exist independently. Destroying one object does not necessarily imply the destruction of the other.

> **Tip:** TopLink automatically manages private relationships. Whenever an object is written to the database, any private objects it owns are also written to the database. When an object is removed from the database, any private objects it owns are also removed. Be aware of this when creating new systems, since it may affect both the behavior and the performance of your application.

Table 32–8 summarizes which mappings support this option.

Table 32–8 Mapping Support for Private or Independent Relationships

Mapping	Implicitly Private	Private or Independent	Using TopLink Workbench	Using Java
Relational Mappings				
One-to-One Mapping		✔	✔	✔
Variable One-to-One Mapping		✔	✔	✔
One-to-Many Mapping		✔	✔	✔
Many-to-Many Mapping		✔	✔	✔
Aggregate Collection Mapping	✔			
Direct Collection Mapping	✔			
Direct Map Mapping		✔	✔	✔
Aggregate Object Mapping	✔			
Object-Relational Mappings				
Object-Relational Structure Mapping	✔			
Object-Relational Reference Mapping		✔		✔
Object-Relational Array Mapping	✔			
Object-Relational Object Array Mapping	✔			
Object-Relational Nested Table Mapping		✔		✔
EIS Mappings				
EIS Composite Direct Collection Mapping	✔			
EIS Composite Object Mapping	✔			
EIS Composite Collection Mapping	✔			
EIS One-to-One Mapping		✔	✔	✔
EIS One-to-Many Mapping		✔	✔	✔
XML Mappings				
XML Composite Direct Collection Mapping	✔			

Table 32–8 (Cont.) Mapping Support for Private or Independent Relationships

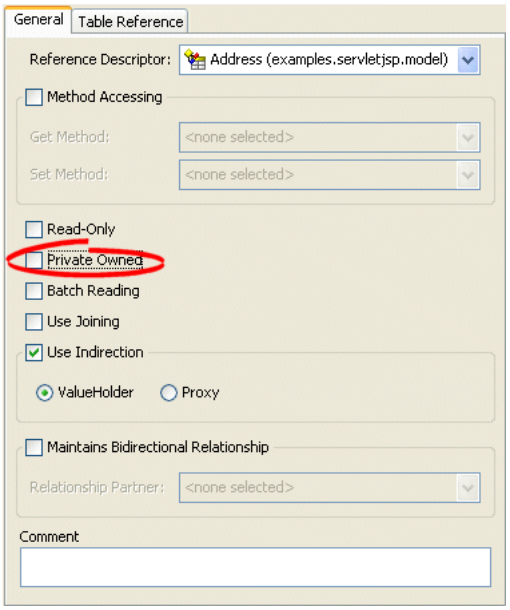
Mapping	Implicitly Private	Private or Independent	Using TopLink Workbench	Using Java
XML Composite Object Mapping	✔			
XML Composite Collection Mapping	✔			

Using TopLink Workbench

To create a privately owned mapping, use this procedure:

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.

Figure 32–8 General Tab, Private Owned option



To create private ownership, select the **Private Owned** option.

Using Java

For mappings not supported in TopLink Workbench, use the `independentRelationship` (default), `privateOwnedRelationship`, and `setIsPrivateOwned` methods.

Example 32–9 shows how to use these methods with a class that has a privately owned attribute, `phones`, in a mapping.

Example 32–10 Configuring Access Method in Java

```
// Map the phones attribute
phonesMapping.setAttributeName("phones");

// Specify as privately owned
phonesMapping.privateOwnedRelationship();
```

Configuring Mapping Comments

You can define a free-form textual comment for each mapping. You can use these comments however you whish: for example, to record important project implementation details such as the purpose or importance of a mapping.

Comments are stored in the TopLink Workbench project, in the TopLink deployment XML file. There is no Java API for this feature.

Table 32–9 summarizes which mappings support this option.

Table 32–9 Mapping Support for Comments

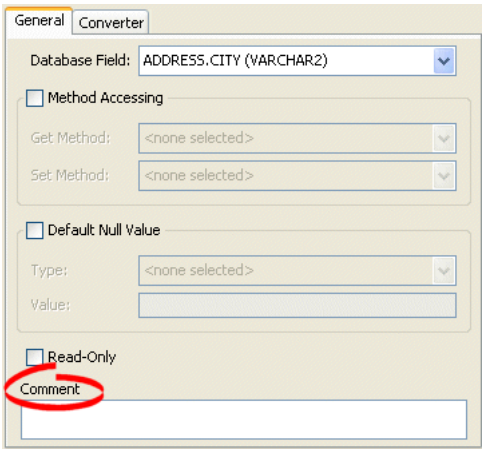
Mapping	Using TopLink Workbench	Using Java
Relational Mappings	✔	
EIS Mappings	✔	
XML Mappings	✔	

Using TopLink Workbench

To add a comment for a mapping, use this procedure.

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.

Figure 32–9 General Tab, Comment



Enter a comment that describes this mapping.

Configuring a Serialized Object Converter

A serialized object converter can be used to store an arbitrary object or set of objects into a data source binary large object (BLOB) field. It uses the Java serializer so the target must be serializable.

For more information about the serialized object converter, see "Serialized Object Converter" on page 30-10.

Table 32–10 summarizes which mappings support this option.

Table 32–10 Mapping Support for Serialized Object Converter

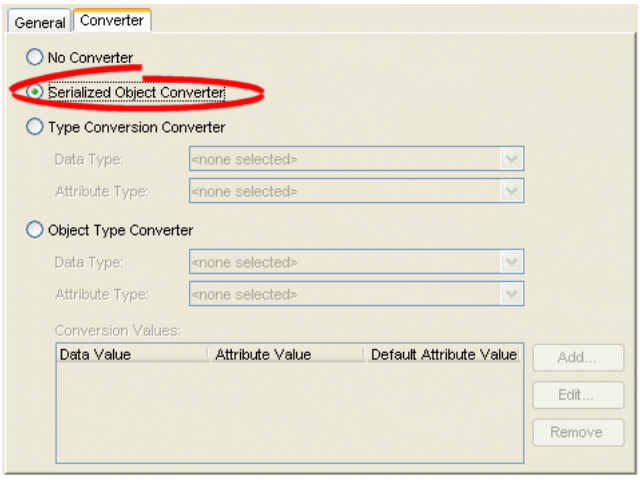
Mapping	Using TopLink Workbench	Using Java
Relational Mappings		
Direct-to-Field Mapping	✔	✔
Object-Relational Mappings		
Object-Relational Array Mapping		✔
EIS Mappings		
EIS Direct Mapping	✔	✔
EIS Composite Direct Collection Mapping	✔	✔
XML Mappings		
XML Direct Mapping	✔	✔
XML Composite Direct Collection Mapping	✔	✔

Using TopLink Workbench

To create an serialized object direct mapping, use this procedure:

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.
2. Click the **Converter** tab. The Converter tab appears.

Figure 32–10 Converter Tab, Serialized Object Converter Option



To specify a serialized object converter, select the **Serialized Object Converter** option.

Using Java

You can set an `oracle.toplink.converters.SerializedObjectConverter` on any instance of `oracle.toplink.mappings.foundation.AbstractCompositeDirectCollectionMapping` using `AbstractCompositeDirectCollectionMapping` method `setValueConverter` as Example 32–11 shows.

Example 32–11 Configuring a SerializedObjectConverter in Java

```
// Create SerializedObjectConverter instance
SerializedObjectConverter serializedObjectConvter = new SerializedObjectConverter();

// Set SerializedObjectConverter on ArrayMapping
ArrayMapping arrayMapping = new ArrayMapping();
arrayMapping.setValueConverter(serializedObjectConvter);
arrayMapping.setAttributeName("responsibilities");
arrayMapping.setStructureName("Responsibilities_t");
arrayMapping.setFieldName("RESPONSIBILITIES");
orDescriptor.addMapping(arrayMapping);
```

Configuring a Type Conversion Converter

A type conversion converter is used to explicitly map a data source type to a Java type.

For more information about the type conversion converter, see "Type Conversion Converter" on page 30-11.

Table 32–11 summarizes which mappings support this option.

Table 32–11 Mapping Support for Type Conversion Converter

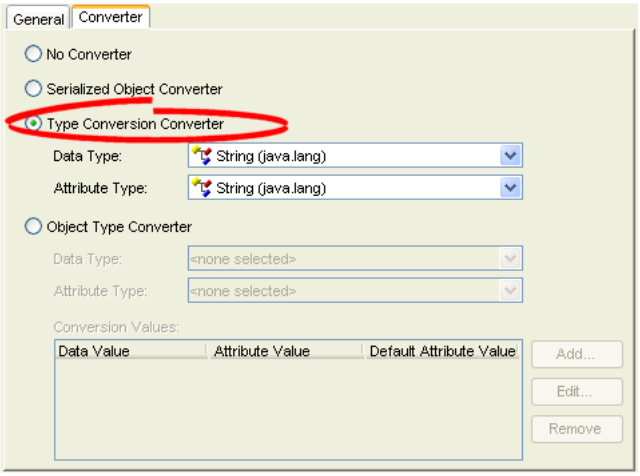
Mapping	Using TopLink Workbench	Using Java
Relational Mappings		
Direct-to-Field Mapping	✔	✔
Object-Relational Mappings		
Object-Relational Array Mapping		✔
EIS Mappings		
EIS Direct Mapping	✔	✔
EIS Composite Direct Collection Mapping	✔	✔
XML Mappings		
XML Direct Mapping	✔	✔
XML Composite Direct Collection Mapping	✔	✔

Using TopLink Workbench

To create an type conversion direct mapping, use this procedure:

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.
2. Click the **Converter** tab. The Converter tab appears.
3. Select the **Type Conversion Converter** option.

Figure 32–11 Converter Tab, Type Conversion Converter Option



Use the following information to complete the Type Conversion Converter fields on the Converter tab:

Field	Description
Data Type	Select the Java type of the data in the data source.
Attribute Type	Select the Java type of the attribute in the Java class.

Using Java

You can set an `oracle.toplink.converters.TypeConversionConverter` on any instance of `oracle.toplink.mappings.foundation.AbstractCompositeDirectCollectionMapping` using `AbstractCompositeDirectCollectionMapping` method `setValueConverter` as Example 32–12 shows.

Example 32–12 Configuring a TypeConversionConverter in Java

```
// Create TypeConversionConverter instance
TypeConversionConverter typeConversionConverter = new TypeConversionConverter();
typeConversionConverter.setDataClass();
typeConversionConverter.setObjectClass();

// Set TypeConversionConverter on ArrayMapping
ArrayMapping arrayMapping = new ArrayMapping();
arrayMapping.setValueConverter(typeConversionConverter);
arrayMapping.setAttributeName("responsibilities");
arrayMapping.setStructureName("Responsibilities_t");
arrayMapping.setFieldName("RESPONSIBILITIES");
orDescriptor.addMapping(arrayMapping);
```

Configure the `TypeConversionConverter` instance using the following API:

- `setDataClass(java.lang.Class dataClass)`–to specify the data type class
- `setObjectClass(java.lang.Class objectClass)`–to specify the object type class

Configuring an Object Type Converter

An object type converter is used to match a fixed number of data source data values to Java object values. It can be used when the values in the data source and in Java differ.

For more information about the object type converter, see "Object Type Converter" on page 30-12.

Table 32–12 summarizes which mappings support this option.

Table 32–12 Mapping Support for Object Type Converter

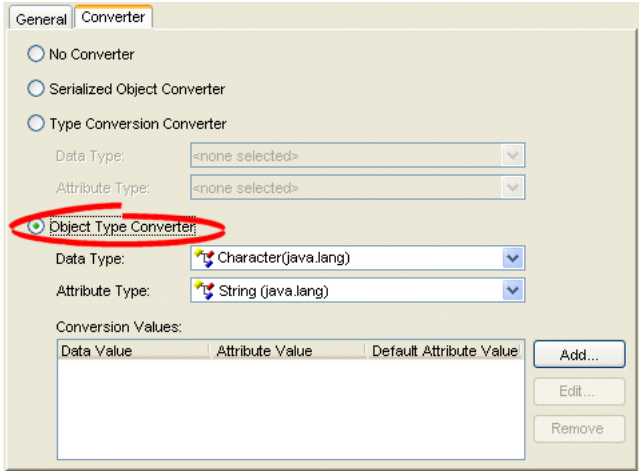
Mapping	Using TopLink Workbench	Using Java
Relational Mappings	✔	✔
Object-Relational Mappings		
Object-Relational Array Mapping		✔
EIS Mappings		
EIS Direct Mapping	✔	✔
EIS Composite Direct Collection Mapping	✔	✔
XML Mappings		
XML Direct Mapping	✔	✔
XML Composite Direct Collection Mapping	✔	✔

Using TopLink Workbench

To add an object type converter to a direct mapping, use this procedure:

1. Select the mapping in the **Navigator**. Its properties appear in the Editor.
2. Click the **Converter** tab. The Converter tab appears.

Figure 32–12 Converter Tab, Object Type Converter



Use the following fields on the mapping's **Converter** tab to specify the object type converter options:

Field	Description
Data Type	Select the Java type of the data in the data source.
Attribute Type	Select the Java type of the attribute in the Java class.
Conversion Values	Click **Add** to add a new conversion value. Click **Edit** to modify an existing conversion value. Click **Remove** to delete an existing conversion value. Use to specify the selected value as the default value. If TopLink retrieves a value from the database that is not mapped as a valid Conversion Value, the default value will be used.
Data Value	Specify the value of the attribute in the data source.
Attribute Value	Specify the value of the attribute in the Java class
Default Attribute Value	Specify whether or not to use the selected value as the default value. If TopLink retrieves a value from the database that is not mapped as a valid Conversion Value, the default value will be used.

Using Java

You can set an `oracle.toplink.converters.ObjectTypeConverter` on any instance of `oracle.toplink.mappings.foundation.AbstractCompositeDirectCollectionMapping` using `AbstractCompositeDirectCollectionMapping` method `setValueConverter` as Example 32–13 shows.

Example 32–13 Configuring an ObjectTypeConverter in Java

```
// Create ObjectTypeConverter instance
ObjectTypeConverter objectTypeConvter = new ObjectTypeConverter();
objectTypeConverter.addConversionValue("F", "Female");

// Set ObjectTypeConverter on ArrayMapping
ArrayMapping arrayMapping = new ArrayMapping();
arrayMapping.setValueConverter(objectTypeConverter);
arrayMapping.setAttributeName("responsibilities");
arrayMapping.setStructureName("Responsibilities_t");
arrayMapping.setFieldName("RESPONSIBILITIES");
orDescriptor.addMapping(arrayMapping);
```

Configure the `ObjectTypeConverter` instance using the following API:

- `addConversionValue(java.lang.Object fieldValue, java.lang.Object attributeValue)`–to associate data-type values to object-type values
- `addToAttributeOnlyConversionValue(java.lang.Object fieldValue, java.lang.Object attributeValue)`–to add one-way conversion values
- `setDefaultAttributeValue(java.lang.Object defaultAttributeValue)`–to set the default value

Configuring a Simple Type Translator

The simple type translator allows you to automatically translate an XML element value to an appropriate Java type based on the element's `<type>` attribute, as defined in your XML schema. You can use a simple type translator only when the mapping's

XPath goes to an element. You cannot use a simple type translator if the mapping's XPath goes to an attribute.

For more information, see "Simple Type Translator" on page 30-12.

Table 32–13 summarizes which mappings support this option.

Table 32–13 Mapping Support for Simple Type Translator

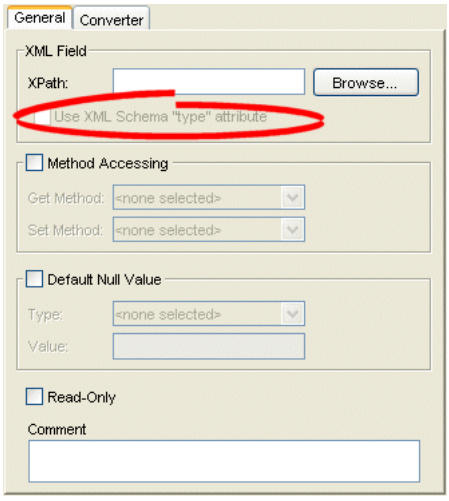
Mapping	Using TopLink Workbench	Using Java
EIS Mappings		
EIS Direct Mapping	✔	✔
EIS Composite Direct Collection Mapping	✔	✔
XML Mappings		
XML Direct Mapping	✔	✔
XML Composite Direct Collection Mapping	✔	✔

Using TopLink Workbench

Use this table to qualify elements from the XML schema

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.

Figure 32–13 General Tab, Use XML Schema "type" Attribute Option



Select the **Field Uses XML Schema "type" attribute** field to qualify elements from the XML schema.

Using Java

To create an XML mapping with a simple type translator with Java code in your IDE, you need the following elements:

- `EISDirectMapping` or `EISCompositeDirectCollectionMapping` or `XMLDirectMapping` or `XMLCompositeDirectCollectionMapping`
- instance of `Converter`
- instance of `TypedElementField`

Example 32–14 shows how to implement your own simple type translator with an `XMLDirectMapping` to override the built-in conversion for writing XML so that TopLink writes a `Byte` array (`ClassConstants.ABYTE`) as a `Base64` (`XMLConstants.BASE64_BINARY`) encoded string.

Example 32–14 Creating a Type Translation XML Mapping

```
XMLDirectMapping mapping = new XMLDirectMapping();
mapping.setConverter(new SerializedObjectConverter());
TypedElementField field = new TypedElementField("element");
field.getSimpleTypeTranslator().addJavaConversion(
    ClassConstants.ABYTE,
    new QName(XMLConstants.SCHEMA_URL, XMLConstants.BASE64_BINARY)
);
mapping.setField(field);
```

Configuring a JAXB Typesafe Enumeration Converter

The JAXB typesafe enumeration converter allows you to automatically translate an XML element value to an appropriate typesafe enumeration value as defined in your XML schema.

For more information, see "Mappings and JAXB Typesafe Enumerations" on page 30-24.

Table 32–14 summarizes which mappings support this option.

Table 32–14 Mapping Support for JAXB Typesafe Enumeration Converter

Mapping	Using TopLink Workbench	Using Java
EIS Mappings[1]		
EIS Direct Mapping		✔
EIS Composite Direct Collection Mapping		✔
XML Mappings		
XML Direct Mapping		✔
XML Composite Direct Collection Mapping		✔

[1] When used with XML records only (see "Configuring Record Format" on page 28-5).

TopLink Workbench does not support the `JAXBTypesafeEnumConverter` directly: to configure a mapping with this converter, you must use Java to create an amendment method (see "Using Java" on page 32-26).

If you create a project and object model using the TopLink JAXB compiler (see "Creating an XML Project From an XML Schema" on page 18-6), the compiler will create the type safe enumeration class and a class with descriptor amendment methods and register the required amendment methods automatically (see "Typesafe

Enumeration Converter Amendment Method DescriptorAfterLoads Class" on page 17-12).

Using Java

To configure a mapping with a `JAXBTypesafeEnumConverter` in Java, use a descriptor amendment method (see "Configuring Amendment Methods" on page 25-81). Example 32–15 illustrates an amendment method that configures an `XMLDirectMapping` with a `JAXBTypesafeEnumConverter`. In this example, attribute `_Val` is mapped to a JAXB typesafe enumeration corresponding to typesafe enumeration class `MyTypesafeEnum`.

Example 32–15 Creating a JAXB Typesafe Enumeration XML Mapping

```
public class DescriptorAfterLoads {
    public static void amendRootImplDescriptor(ClassDescriptor descriptor) {
        DatabaseMapping _ValMapping = descriptor.getMappingForAttributeName("_Val");
        JAXBTypesafeEnumConverter _ValConverter = new JAXBTypesafeEnumConverter();
        ValConverter.setEnumClassName("MyTypesafeEnum");
        ((XMLDirectMapping) _ValMapping).setConverter(_ValConverter);
    }
}
```

Configuring Container Policy

Collection mapping container policy specifies the concrete class TopLink should use when reading target objects from the database.

Collection mappings can use any concrete class that implements the `java.util.List`, `java.util.Set`, `java.util.Collection`, or `java.util.Map` interface. You can map object attributes declared as `List`, `Set`, `Collection`, `Map`, or any subinterface of these interfaces, or as a class that implements one of these interfaces.

By default, the TopLink runtime uses the following concrete classes from the `oracle.toplink.indirection` package for each of these container types:

- `List`–`IndirectList` or `EJBIndirectList`
- `Set`–`IndirectSet` or `EJBIndirectSet`
- `Collection`–`IndirectList` or `EJBIndirectList`
- `Map`–`IndirectMap` or `EJBIndirectMap`

Alternatively, you can specify in the mapping the concrete container class to be used. When TopLink reads objects from the database that contain an attribute mapped with a collection mapping, the attribute is set with an instance of the concrete class specified. For example, TopLink does not sort in memory. If you want to sort in memory, override the default `Set` type (`IndirectList`) with `java.util.TreeSet` as the concrete collection type. By default, a collection mapping's container class is `java.util.Vector`.

Note: If you are using TopLink Workbench and you override the default `Collection` class with a custom `Collection` class of your own, you must put your custom `Collection` class on the TopLink Workbench classpath (see "Configuring the TopLink Workbench Environment" on page 4-2).

Table 32–15 summarizes which mappings support this option.

Table 32–15 Mapping Support for Container Policy

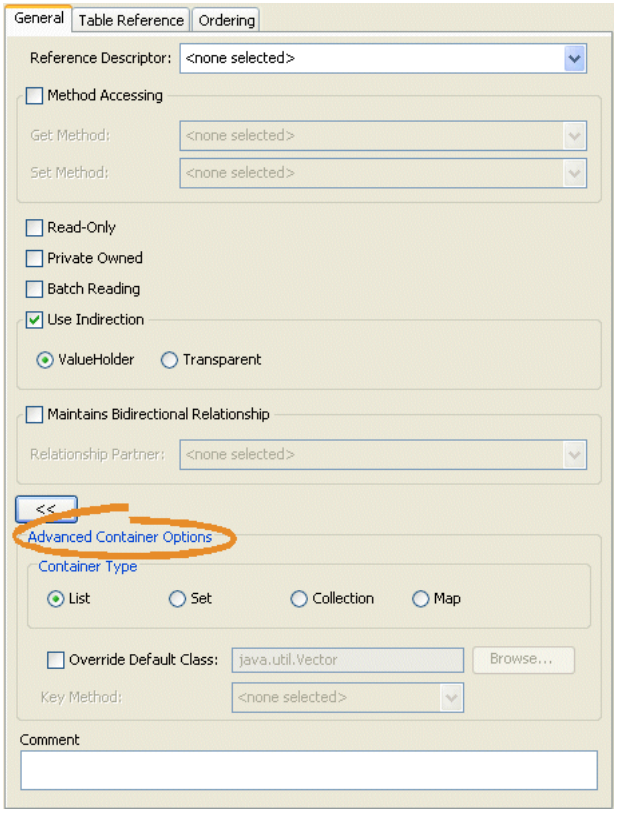
Mapping	List	Set	Collection	Map	Using TopLink Workbench	Using Java
Relational Mappings						
One-to-Many Mapping	✔	✔	✔	✔	✔	✔
Many-to-Many Mapping	✔	✔	✔	✔	✔	✔
Aggregate Collection Mapping	✔	✔	✔	✔		✔
Direct Collection Mapping	✔	✔	✔		✔	✔
Direct Map Mapping				✔	✔	✔
Object-Relational Mappings						
Object-Relational Array Mapping	✔	✔	✔	✔		✔
Object-Relational Object Array Mapping	✔	✔	✔	✔		✔
Object-Relational Nested Table Mapping	✔	✔	✔	✔		✔
EIS Mappings						
EIS Composite Direct Collection Mapping	✔	✔	✔		✔	✔
EIS Composite Collection Mapping	✔	✔	✔	✔	✔	✔
EIS One-to-Many Mapping	✔	✔	✔	✔	✔	✔
XML Mappings						
XML Composite Direct Collection Mapping	✔	✔	✔		✔	✔
XML Composite Collection Mapping	✔	✔	✔	✔	✔	✔
XML Any Collection Mapping	✔	✔	✔		✔	✔

Using TopLink Workbench

To specify a mapping's container policy, use this procedure:

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.
3. Click the **Advanced** button. The Advanced Container Options appear on the General tab.

Figure 32–14 General Tab, Advanced Container Options



Use the following Advanced Container Options fields on the **General** tab to specify the container options:

Field[1]	Description
Container Type	Specify the type of `Collection` class to use: ▪ **List**–use a `java.util.List` ▪ **Set**–use a `java.util.Set` ▪ **Collection**–use a `java.util.Collection` ▪ **Map**–use a `java.util.Map`
Override Default Class	Specify to use a custom class as the mapping's container policy. Click **Browse** to select a different class. The container class must implement (directly or indirectly) the `java.util.Collection` interface.
Key Method	If you configure **Container Type** as **Map**, use this option to specify the name of the zero argument method whose result, when called on the target object, is used as the key in the `Hashtable` or `Map`. This method must return an object that is a valid key in the `Hashtable` or `Map`.

[1] Not all mappings support all options. For more information, see Table 32–15.

Using Java

Classes that implement the `oracle.toplink.mappings.ContainerMapping` interface provide the following methods to set the container policy:

- `useCollectionClass(java.lang.Class concreteClass)`–Configure the mapping to use an instance of the specified `java.util.Collection` container class to hold the target objects.
- `useMapClass(java.lang.Class concreteClass, java.lang.String methodName)`–Configure the mapping to use an instance of the specified `java.util.Map` container class to hold the target objects. The key used to index a value in the `Map` is the value returned by a call to the specified zero-argument method. The method must be implemented by the class (or a superclass) of any value to be inserted into the `Map`.

Classes that extend `oracle.toplink.mappings.CollectionMapping` (which implements the `ContainerMapping` interface) also provide the following methods to set the container policy:

- `useSortedSetClass(java.lang.Class concreteClass, java.util.Comparator comparator)`–Configure the mapping to use an instance of the specified `java.util.SortedSet` container class. Specify the `Comparator` to use to sort the target objects.

Example 32–16 shows how to configure an `ObjectArrayMapping` to use a `java.util.ArrayList` container class.

Example 32–16 Object Array Mapping

```
// Create a new mapping and register it with the source Object-relational 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");
phonesMapping.useCollectionClass(ArrayList.class);
orDescriptor.addMapping(phonesMapping);
```

Configuring Attribute Transformer

A transformation mapping is made up of an attribute transformer for field-to-attribute transformation at read (unmarshall) time and one or more field transformers for attribute-to-field transformation at write (marshall) time (see "Configuring Field Transformer Associations" on page 32-31).

This section describes how to configure the attribute transformer that a transformation mapping uses to perform the field-to-attribute transformation at read (unmarshal) time.

You can do this using either a method or class-based transformer.

A method-based transformer must map to a method in the domain object.

A class-based transformer allows you to place the transformation code in another class, making this approach non-intrusive: that is, your domain object does not need to implement a TopLink interface or provide a special transformation method

Table 32–16 summarizes which mappings support this option.

Table 32–16 Mapping Support for Attribute Transformer

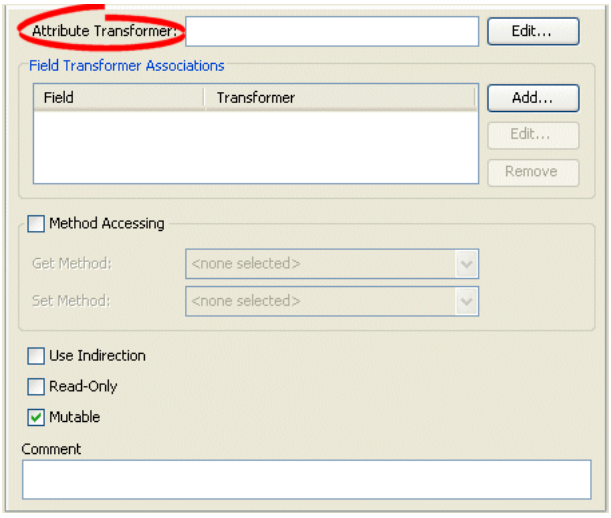
Mapping	Using TopLink Workbench	Using Java
Relational Mappings		
Transformation Mapping	✔	✔
EIS Mappings		
EIS Transformation Mapping	✔	✔
XML Mappings		
XML Transformation Mapping	✔	✔

Using TopLink Workbench

To specify a mapping's attribute transformer, use this procedure:

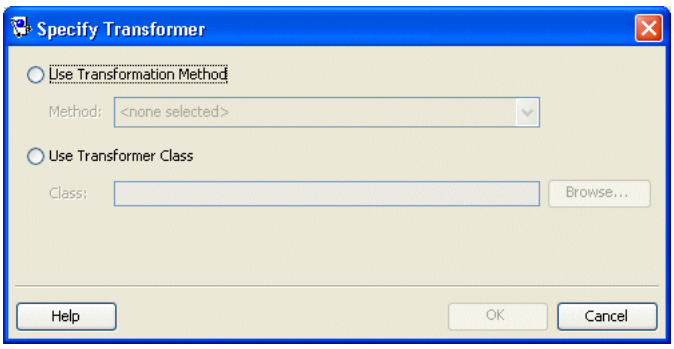
1. Select the transformation mapping in the **Navigator**. Its properties appear in the Editor.

Figure 32–15 Transformation Mapping, Attribute Transformer Field



2. Click **Edit**. The Specify Transformer dialog box appears.

Figure 32–16 Specify Transformer Dialog Box



Use the following information to enter data in each field of the dialog box and click **OK**:

Field	Description
Use Transformation Method	Select a specific method to control the transformation. A method based transformer must map to a method in the domain object.
Use Transformer Class	Select a specific class to control the transformation. The class must be available on the TopLink Workbench application classpath.

Using Java

You can configure a method-based attribute transformer using `AbstractTransformationMapping` method `setAttributeTransformation`, passing in the name of the domain object method to use.

You can configure a class-based attribute transformer using `AbstractTransformationMapping` method `setAttributeTransformer`, passing in an instance of `oracle.toplink.mappings.Transfomers.AttributeTransformer`.

A convenient way to create an `AttributeTransformer` is to extend `AttributeTransformerAdapter`.

Configuring Field Transformer Associations

A transformation mapping is made up of an attribute transformer for field-to-attribute transformation at read (unmarshall) time (see "Configuring Attribute Transformer" on page 32-29) and one or more field transformers for attribute-to-field transformation at write (marshall) time.

This section describes how to configure the field transformers that a transformation mapping uses to perform the object attribute-to-field transformation at write (marshal) time.

You can do this using either a method or class-based transformer.

A method-based transformer must map to a method in the domain object.

A class-based transformer allows you to place the transformation code in another class, making this approach non-intrusive: that is, your domain object does not need to implement a TopLink interface or provide a special transformation method.

Table 32–17 summarizes which mappings support this option.

Table 32–17 Mapping Support for Field Transformer

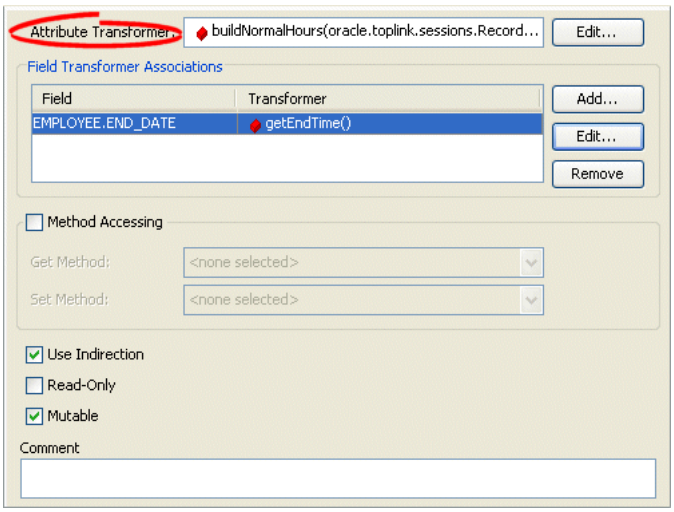
Mapping	Using TopLink Workbench	Using Java
Relational Mappings		
Transformation Mapping	✔	✔
EIS Mappings		
EIS Transformation Mapping	✔	✔
XML Mappings		
XML Transformation Mapping	✔	✔

Using TopLink Workbench

Use this procedure to complete the **Object->Field Method** fields:

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.

Figure 32–17 Transformation Mapping, Field Transformer Associations



To add a new association, click **Add**. Continue with "Specifying Field-to-Transformer Associations" on page 32-32.

To change an existing association, click **Edit**. Continue with "Specifying Field-to-Transformer Associations" on page 32-32.

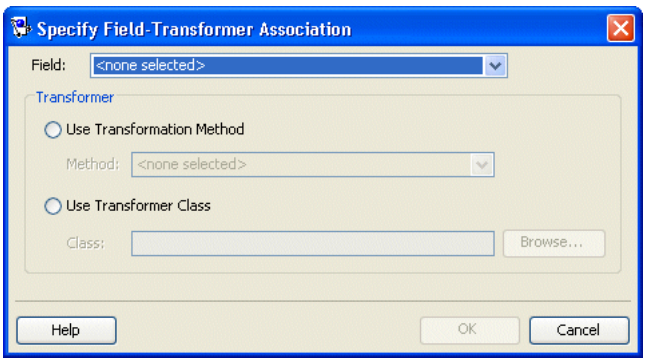
To delete an existing association, select the field transformation association and click **Delete**.

Specifying Field-to-Transformer Associations

To specify the actual transformation method or class used for the field of a transformation mapping, use this procedure.

1. From the Transformation Mapping, Field Transformer Associations, click **Add** or Edit. The Specify Field-Transformer Association dialog box appears.

Figure 32–18 Specify Field-Transformer Association Dialog Box



Use the following information to complete each field on the dialog box:

Field	Description
Field	Select the database field (from the descriptor's associated table) for this transformation.
Transformer	Select one of the following methods to control the transformation:
Use Transformation Method	Select a specific method to control the transformation. A method based transformer must map to a method in the domain object.
Use Transformer Class	Select a specific class to control the transformation. The class must be available on TopLink Workbench application classpath.

Using Java

You can specify a specific transformation method on your domain object or an instance of `oracle.toplink.mappings.Transfomers.FieldTransformer` (you can also extend the `FieldTransformerAdapter`). Using a `FieldTransformer` is non-intrusive: that is, your domain object does not need to implement a TopLink interface or provide a special transformation method.

You can configure a method-based field transformer using `AbstractTransformationMapping` method `addFieldTransformation`, passing in the name of the database field and the name of the domain object method to use.

You can configure a class-based field transformer using `AbstractTransformationMapping` method `addFieldTransformer`, passing in the name of the database field and an instance of `oracle.toplink.mappings.Transfomers.FieldTransformer`.

A convenient way to create a `FieldTransformer` is to extend `FieldTransformerAdapter`.

Configuring Mutable Mappings

Direct mappings typically map simple, non-mutable values such as `String` or `Integer`. Transformation mappings can potentially map complex mutable object values, such as mapping several database field values to an instance of a Java class.

If a transformation mapping maps a mutable value, TopLink must clone and compare the value in a unit of work (see "Configuring Copy Policy" on page 25-71).

By default, TopLink assumes that all transformation mappings are mutable. If the mapping maps a simple nonmutable value, you can improve unit of work performance by configuring the **IsMutable** option to `false`.

Table 32–18 summarizes which mappings support this option.

Table 32–18 Mapping Support for Mutable Mappings

Mapping	Using TopLink Workbench	Using Java
Relational Mappings		
Transformation Mapping	✔	✔
EIS Mappings		
EIS Transformation Mapping	✔	✔

Table 32–18 (Cont.) Mapping Support for Mutable Mappings

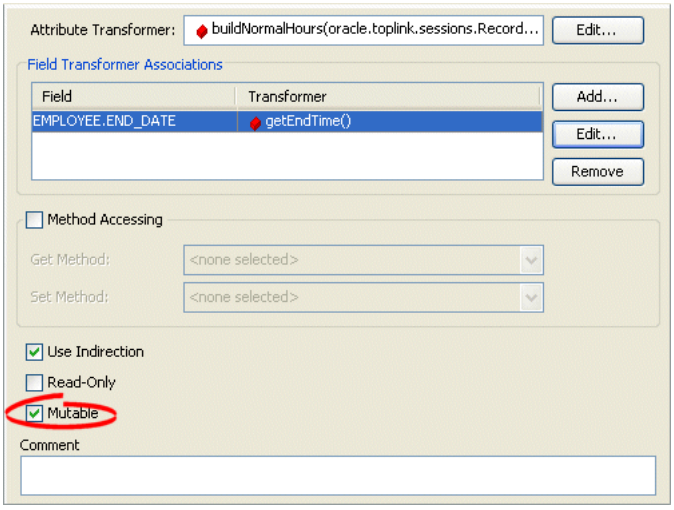
Mapping	Using TopLink Workbench	Using Java
XML Mappings		
XML Transformation Mapping	✔	✔

Using TopLink Workbench

Use this table to complete the **Object->Field Method** fields:

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.

Figure 32–19 Transformation Mapping, Mutable Option



By default, the **IsMutable** option is selected in all transformation mappings. If the mapping maps to a simple atomic value, unselect this option.

Using Java

You can specify whether or not a mapping is mutable using `AbstractTransformationMapping` method `setIsMutable`.

Configuring Bidirectional Relationship

If a 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 call the `setForeignKeyFieldName` method.
- The other must call the `setTargetForeignKeyFieldName` method.

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

Note: 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. For more information, see the following:

- Directionality
- Maintaining Bidirectional Relationships
- Indirection

Table 32–18 summarizes which mappings support this option.

Table 32–19 Mapping Support for Mutable Mappings

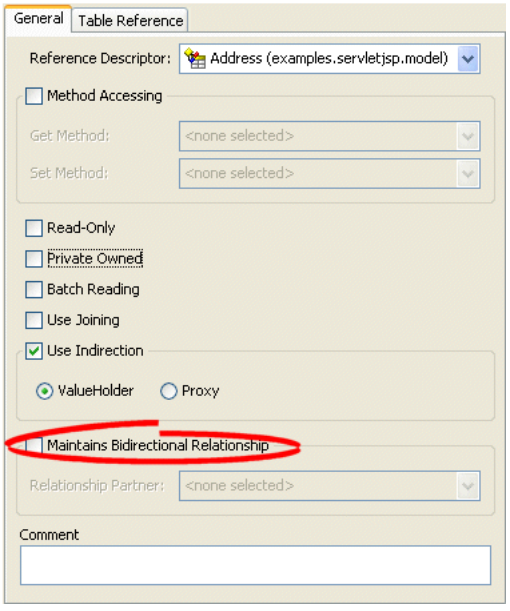
Mapping	Using TopLink Workbench	Using Java
Relational Mappings		
One-to-One Mapping	✔	✔
One-to-Many Mapping	✔	✔
Many-to-Many Mapping	✔	✔
EIS Mappings		
EIS One-to-One Mapping	✔	✔
EIS One-to-Many Mapping	✔	✔

Using TopLink Workbench

To maintain a bidirectional relationship for a mapping, use this procedure:

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.

Figure 32–20 General tab, Maintains Bidirectional Relationship option



Use this table to enter data in the following fields on the tab:

Field	Description
Maintains Bidirectional Relationship	Specify if TopLink should maintain the bidirectional link for this relational mapping.
Relationship Partner	Select the relationship partner (from the list of mapped attributes of the **Reference Descriptor**) for this bidirectional relationship.

Configuring the Use of a Single Node

For the XML-based mappings that Table 32–6 summarizes, when you map a list value, you can configure whether or not the mapping unmarshalls (writes) the list to a single node, like `<item>aaa bbb ccc</item>`, or to multiple nodes, like:

```
<item>aaa</item>
<item>bbb</item>
<item>ccc</item>
```

Table 32–6 summarizes which mappings support this option.

Table 32–20 Mapping Support for Use Single Node

Mapping	Using TopLink Workbench	Using Java
EIS Mappings[1]		
EIS Direct Mapping		✔
EIS Composite Direct Collection Mapping	✔	✔
XML Mappings		
XML Direct Mapping		✔
XML Composite Direct Collection Mapping	✔	✔

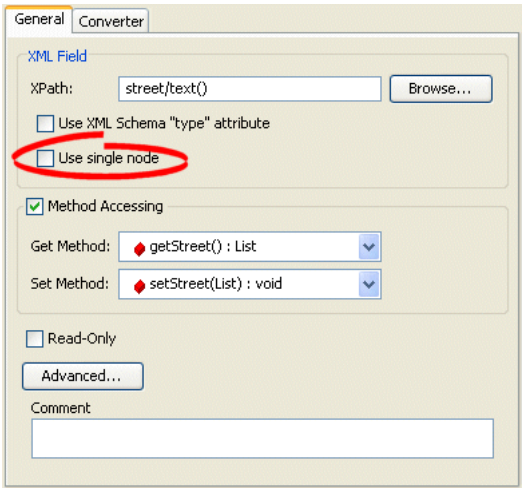
[1] When used with XML records only (see "Configuring Record Format" on page 28-5).

Using TopLink Workbench

To configure a mapping to use a single node, use this procedure.

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.

Figure 32–21 General Tab, Use Single Node Option



To configure the mapping to unmarshall (write) a list value to a single node (like `<item>aaa bbb ccc</item>`), click **Use single node**.

By default, the mapping unmarshalls a list value to separate nodes.

Using Java

Use `AbstractCompositeDirectCollectionMapping` method `setUsesSingleNode` to configure the mapping to write a list value to a single node by passing in a value of `true`. To configure the mapping to write a list value to multiple nodes, pass in a value of `false`.

For any mapping that takes an `XMLField`, use `XMLField` method `setUsesSingleNode` to configure the mapping to write a list value to a single node by passing in a value of `true`. To configure the mapping to write a list value to multiple nodes, pass in a value of `false`. Example 32–17 shows how to use this method with an `XMLDirectMapping`:

Example 32–17 Using XMLField Method setUsesSingleNode

```
XMLDirectMapping tasksMapping = new XMLDirectMapping();
tasksMapping.setAttributeName("tasks");
XMLField myField = new XMLField("tasks/text()"); // pass in the XPath
myField.setUsesSingleNode(true);
tasksMapping.setField(myField);
```

Part X

Relational Mappings

A relational mapping transforms any object data member type to a corresponding relational database (SQL) data source representation in any supported relational database. Relational mappings let you map an object model into a relational data model.

This part contains the following chapters:

- Chapter 33, "Understanding Relational Mappings"

  This chapter describes each of the different TopLink relational mapping types and important relational mapping concepts.

- Chapter 34, "Configuring a Relational Mapping"

  This chapter explains how to configure TopLink relational mapping options common to two or more relational mapping types.

- Chapter 35, "Configuring a Relational Direct-to-Field Mapping"

  This chapter explains how to configure a direct to field relational database mapping.

- Chapter 36, "Configuring a Relational Direct-to-XMLType Mapping"

  This chapter explains how to configure a direct mapping to an Oracle XDB XML type field.

- Chapter 37, "Configuring a Relational One-to-One Mapping"

  This chapter explains how to configure a one-to-one relational database mapping.

- Chapter 38, "Configuring a Relational Variable One-to-One Mapping"

  This chapter explains how to configure a variable one-to-one relational database mapping.

- Chapter 39, "Configuring a Relational One-to-Many Mapping"

  This chapter explains how to configure a one-to-many relational database mapping.

- Chapter 40, "Configuring a Relational Many-to-Many Mapping"

  This chapter explains how to configure a many-to-many relational database mapping.

- Chapter 41, "Configuring a Relational Aggregate Collection Mapping"

  This chapter explains how to configure an aggregate collection relational database mapping.

- Chapter 42, "Configuring a Relational Direct Collection Mapping"

This chapter explains how to configure a direct collection relational database mapping.

- Chapter 44, "Configuring a Relational Direct Map Mapping"

  This chapter explains how to configure a direct map relational database mapping.

- Chapter 43, "Configuring a Relational Aggregate Object Mapping"

  This chapter explains how to configure an aggregate object-relational database mapping.

- Chapter 45, "Configuring a Relational Transformation Mapping"

  This chapter explains how to configure a transformation relational database mapping.

33

Understanding Relational Mappings

A relational mapping transforms any object data member type to a corresponding relational database (SQL) data source representation in any supported relational database. Relational mappings let you map an object model into a relational data model.

Relational mappings transform object data members to relational database fields. Use them to map simple data types including primitives (such as `int`), JDK classes (such as `String`), and large object (LOB) values. You can also use them to transform object data members that reference other domain objects by way of association where data source representations require object identity maintenance (such as sequencing and back references) and possess various types of multiplicity and navigability. The appropriate mapping class is chosen primarily by the cardinality of the relationship.

Do not confuse relational mappings with object-relational mappings (see "Understanding Object-Relational Mappings" on page 46-1). An object-relational mapping transforms certain object data member types to structured data source representations optimized for storage in specialized object-relational databases such as Oracle9*i* Database Server. Object-relational mappings let you map an object model into an object-relational data model. In general, you can use relational mappings with any supported relational database. You can only use object-relational mappings with specialized object-relational databases optimized to support object-relational data source representations.

This chapter describes the following:

- Relational Mapping Types
- Relational Mapping Concepts

Relational Mapping Types

TopLink supports the relational mappings listed in Table 33–1.

Table 33–1 TopLink Relational Mapping Types

Type	Description	Type	TopLink Workbench	Java
"Direct-to-Field Mapping" on page 33-4	Map a Java attribute directly to a database field.	Basic	✔	✔
"Direct-to-XMLType Mapping" on page 33-4	Map Java attributes to an `XMLType` column in an Oracle Database (introduced in version 9.2.0.1).	Advanced	✔	✔
"One-to-One Mapping" on page 33-5	Map a reference to another persistent Java object to the database.	Basic	✔	✔

Table 33–1 (Cont.) TopLink Relational Mapping Types

Type	Description	Type	TopLink Workbench	Java
"Variable One-to-One Mapping" on page 33-6	Map a reference to an interface to the database.	Basic	✔	✔
"One-to-Many Mapping" on page 33-7	Map Java collections of persistent objects to the database.	Advanced	✔	✔
"Many-to-Many Mapping" on page 33-8	Use an association table to map Java collections of persistent objects to the database.	Advanced	✔	✔
"Aggregate Collection Mapping" on page 33-10	Map Java collections of persistent objects to the database.	Basic		✔
"Direct Collection Mapping" on page 33-11	Map Java collections of objects that do not have descriptors.	Basic	✔	✔
"Direct Map Mapping" on page 33-12	Direct map mappings store instances that implement `java.util.Map`.	Basic	✔	✔
"Aggregate Object Mapping" on page 33-12	Create strict one-to-one mappings that require both objects to exist in the same database row.	Basic	✔	
"Transformation Mapping" on page 33-15	Create custom mappings where one or more fields can be used to create the object to be stored in the attribute.	Basic	✔	✔

Relational Mapping Concepts

This section introduces direct mapping concepts unique to TopLink, including the following:

- Directionality
- Converters and Transformers
- Relational Mappings and EJB

Directionality

The direction of a relationship may be either unidirectional or bidirectional. In a unidirectional relationship, only one entity bean has a relationship field that refers to the other. All TopLink relational mappings are unidirectional, from the class being described (the *source* class) to the class with which it is associated (the *target* class). The target class does not have a reference to the source class in a unidirectional relationship.

In a bidirectional relationship, each entity bean has a relationship field that refers to the other bean. Through the relationship field, an entity bean's code can access its related object. To implement a bidirectional relationship (classes that reference each other), use two unidirectional mappings with the sources and targets reversed.

Note: Maintenance of bidirectional relationships presents a number of technical challenges. For more information, see the following:

- "Maintaining Bidirectional Relationships" on page 2-29
- "Configuring Bidirectional Relationship" on page 32-34
- "Indirection" on page 30-5

Converters and Transformers

You can store object attributes directly in a database table as follows:

- Using a Direct Mapping
- Using a Converter Mapping
- Using a Transformation Mapping

Using a Direct Mapping

If the attribute type is comparable to a database type, the information can be stored directly simply by using a direct-to-field mapping (see "Direct-to-Field Mapping" on page 33-4).

Using a Converter Mapping

If the attribute type is comparable to a database type but requires conversion, the information can be stored directly by using a direct-to-field mapping (see "Direct-to-Field Mapping" on page 33-4) and an appropriate `Converter` instance.

In the previous release, TopLink provided subclasses of `DirectToFieldMapping` for object type direct mappings, serialized object direct mappings, and type conversion direct mappings. In this release, these subclasses are deprecated. In their place, Oracle recommends that you use the `DirectToFieldMapping` method `setConverter` and the corresponding `Converter` instance. Table 33–2 summarizes these changes.

Table 33–2 Using a Converter for Direct-to-Field Mappings

Deprecated DirectToFieldMapping subclass...	Replaced by Converter instance...
`ObjectTypeMapping`	`ObjectTypeConverter` (see "Object Type Converter" on page 30-12)
`SerializedObjectMapping`	`SerializedObjectConverter` (see "Serialized Object Converter" on page 30-10)
`TypeConversionMapping`	`TypeConversionConverter` (see "Type Conversion Converter" on page 30-11

If the application's objects contain attributes that cannot be represented as direct-to-field with an existing converter, use a direct-to-field mapping with a custom converter.

Using a Transformation Mapping

If there is no database primitive type that is logically comparable to the attribute's type, or, if an attribute requires data from multiple fields, it must be transformed on its way to and from the database.

In this case, use a transformation mapping (see "Transformation Mapping" on page 33-15).

Relational Mappings and EJB

Use direct mappings to map the (non-CMR) CMF attributes of a bean.

In EJB CMP projects, the bean class does not define real variables in the class – only abstract getter and setter methods. To map the bean's attributes, you must import `ejb-jar.xml` file into TopLink Workbench (see "Configuring Persistence Type" on page 19-5).

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

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

There are some special considerations when using one-to-one mappings (see "One-to-One Mappings and EJB" on page 33-6), one-to-many mappings (see "One-to-Many Mappings and EJB" on page 33-8), and many-to-many mappings (see "Many-to-Many Mappings and EJB" on page 33-9).

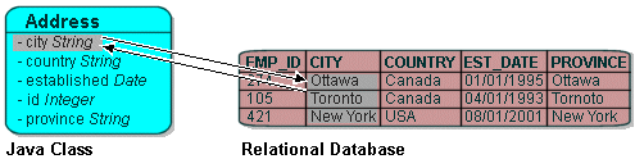
Direct-to-Field Mapping

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

Example 33–1 Direct-to-Field Mapping Example

Figure 33–1 illustrates a direct-to-field mapping between the Java attribute `city` and the relational database column `CITY`. Similarly, direct-to-field mappings could be defined from `country` to `COUNTRY`, `id` to `ADDRESS_ID`, `established` to `EST_DATE`, and `province` to `PROVINCE`.

Figure 33–1 Direct-to-Field Mapping



You can use a direct-to-field mapping with any of the following `Converter` instances:

- see "Object Type Converter" on page 30-12
- see "Serialized Object Converter" on page 30-10
- see "Type Conversion Converter" on page 30-11

You can use a direct-to-field mapping with a change policy (see "Configuring Change Policy" on page 25-73.

See Chapter 35, "Configuring a Relational Direct-to-Field Mapping" for more information.

Direct-to-XMLType Mapping

Using a direct-to-`XMLType` mapping, you can map XML data in the form of a `String` or an org`.w3c.dom.Document` object to an **XMLType** column in an Oracle Database (introduced in version 9.2.0.1).

If you plan to use direct to XML type mappings in TopLink Workbench and the TopLink runtime, you must include the Oracle Database `xdb.jar` file in the TopLink Workbench classpath (see "Configuring the TopLink Workbench Environment" on page 4-2).

The TopLink query framework provides a number of expression operators you can use to create queries based on the content of that XML data (see "XMLType Functions" on page 95-5).

See Chapter 36, "Configuring a Relational Direct-to-XMLType Mapping" for more information.

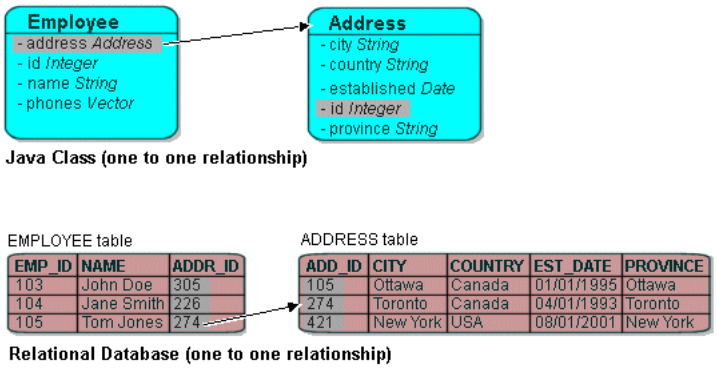
One-to-One Mapping

One-to-one mappings represent simple pointer references between two Java objects. In Java, a single pointer stored in an attribute represents the mapping between the source and target objects. Relational database tables implement these mappings using foreign keys.

Figure 33–2 illustrates a one-to-one relationship from the `address` attribute of an `Employee` object to an `Address` object. To store this relationship in the database, create a one-to-one mapping between the `address` attribute and the `Address` class. This mapping stores the `id` of the `Address` instance in the `EMPLOYEE` table when the `Employee` instance is written. It also links the `Employee` instance to the `Address` instance when the `Employee` is read from the database. Because an `Address` does not have any references to the `Employee`, it does not have to provide a mapping to `Employee`.

For one-to-one mappings, the source table normally contains a foreign key reference to a record in the target table. In Figure 33–2, the `ADDR_ID` field of the EMPLOYEE table is a foreign key.

Figure 33–2 One-to-One Mappings



You can also implement a one-to-one mapping where the target table contains a foreign key reference to the source table. In Figure 33–2, the database design would change such that the `ADDRESS` row would contain the `EMP_ID` to identify the `Employee` to which it belonged. In this case, the target must also have a relationship mapping to the source.

The update, insert and delete operations, which are normally done for the target before the source for privately owned one-to-one relationships, are performed in the opposite order when the target owns the foreign key. Target foreign keys normally occur in bidirectional one-to-one mappings (see "Directionality" on page 33-2), because one side has a foreign key and the other shares the same foreign key in the other's table.

Target foreign keys can also occur when large cascaded composite primary keys exist (that is, one object's primary key is composed of the primary key of many other

objects). In this case it is possible to have a one-to-one mapping that contains both foreign keys and target foreign keys.

In a foreign key, TopLink automatically updates the foreign key value in the object's row. In a target foreign key, it does not. In TopLink, use the **Target Foreign Key** option when a target foreign key relationship is defined.

When mapping a relationship, you must understand these differences between a foreign key and a target foreign key, to ensure that the relationship is defined correctly.

In a bidirectional relationship where the two classes in the relationship reference each other, only one of the mappings should have a foreign key. The other mapping should have a target foreign key. If one of the mappings in a bidirectional relationship is a one-to-many mapping, see "Configuring a Relational Variable One-to-One Mapping" on page 38-1 for details.

You can use a one-to-one mapping with a change policy (see "Configuring Change Policy" on page 25-73.

See "Configuring a Relational One-to-One Mapping" on page 37-1 for more information.

One-to-One Mappings and EJB

To maintain EJB compliance, the object attribute that points to the target of the relationship must be the local interface type–not the bean class.

TopLink provides variations on one-to-one mappings that lets you 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 Chapter 38, "Configuring a Relational Variable One-to-One Mapping".

Variable One-to-One Mapping

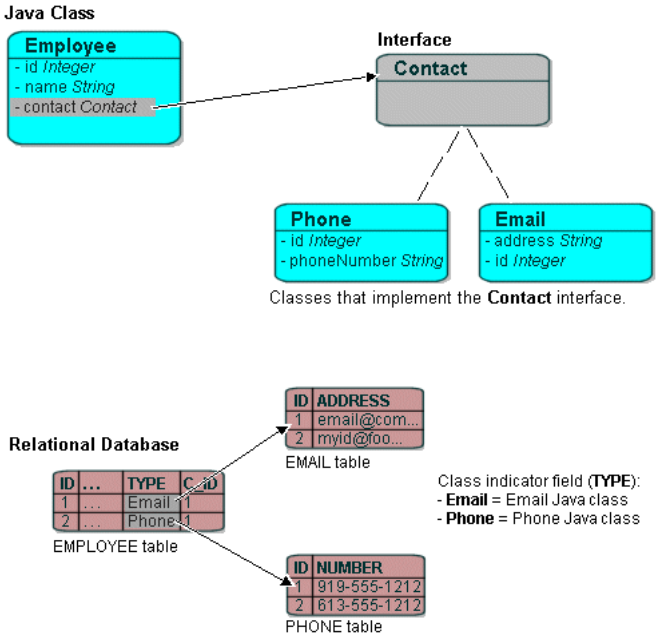
Variable class relationships are similar to polymorphic relationships, except that in this case the target classes are not related through inheritance (and thus not good candidates for an abstract table), but through an interface.

To define variable class relationships in TopLink Workbench, use the variable one-to-one mapping selection, but choose the interface as the reference class. This makes the mapping a variable one-to-one. When defining mappings in Java code, use the `VariableOneToOneMapping` class.

TopLink supports variable relationships only in one-to-one mappings. It handles this relationship in two ways:

- Through the class indicator field (see "Configuring Class Indicator" on page 38-1).
- Through unique primary key values among target classes implementing the interface (see "Configuring Unique Primary Key" on page 38-3).

Figure 33–3 Variable One-to-One Mappings with Class Indicator



See "Configuring a Relational Variable One-to-One Mapping" on page 38-1 for more information.

One-to-Many Mapping

One-to-many mappings are used to represent the relationship between a single source object and a collection of target objects. They are a good example of something that is simple to implement in Java using a `Vector` (or other collection types) of target objects, but difficult to implement using relational databases.

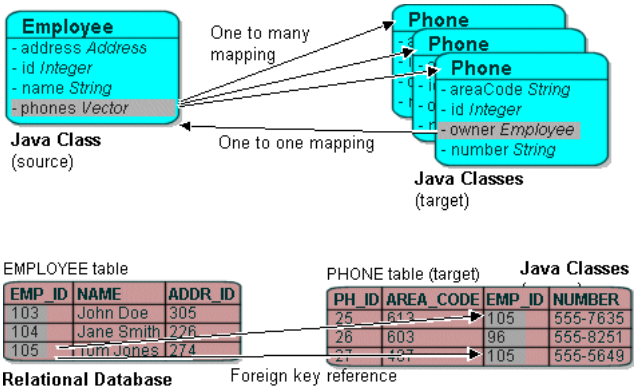
In a Java `Vector`, the owner references its parts. In a relational database, the parts reference their owner. Relational databases use this implementation to make querying more efficient.

> **Note:** See "Configuring Container Policy" on page 32-26 for information on using collection classes other than `Vector` with one-to-many mappings.

The purpose of creating this one-to-one mapping in the target is so that the foreign key information can be written when the target object is saved. Alternatives to the one-to-one mapping back reference include the following:

- Use a direct-to-field mapping to map the foreign key and maintain its value in the application. Here the object model does not require a back reference, but the data model still requires a foreign key in the target table.
- Use a many-to-many mapping to implement a logical one-to-many. This has the advantage of not requiring a back reference in the object model and not requiring a foreign key in the data model. In this model the many-to-many relation table stores the collection. It is possible to put a constraint on the join table to enforce that the relation is a logical one-to-many relationship.

Figure 33–4 One-to-Many Relationships



You can use a many-to-many mapping with a change policy (see "Configuring Change Policy" on page 25-73.

See "Configuring a Relational One-to-Many Mapping" on page 39-1 for more information.

One-to-Many Mappings and EJB

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 local interface type–not the bean class.

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

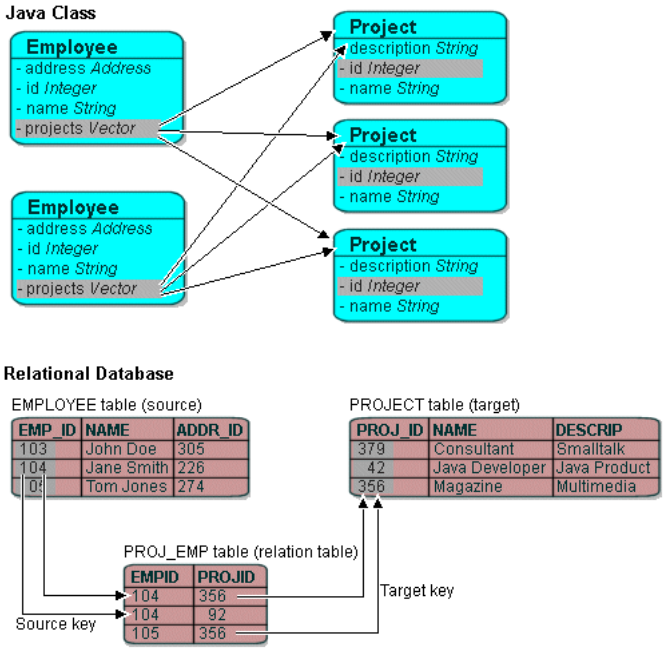
For more information, see "Configuring Bidirectional Relationship" on page 32-34.

Many-to-Many Mapping

Many-to-many mappings represent the relationships between a collection of source objects and a collection of target objects. They require the creation of an intermediate table for managing the associations between the source and target records.

Figure 33–5 illustrates a many-to-many mapping in Java and in relational database tables.

Figure 33–5 Many-to-many Relationships



Many-to-many mappings are implemented using a relation table. This table contains columns for the primary keys of the source and target tables. Composite primary keys require a column for each field of the composite key. The intermediate table must be created in the database before using the many-to-many mapping.

The target class does not have to implement any behavior for the many-to-many mappings. If the target class also creates a many-to-many mapping back to its source, then it can use the same relation table, but one of the mappings must be set to read-only. If both mappings write to the table, they can cause collisions.

Note: See "Configuring Container Policy" on page 32-26 for information on using `Collection` classes other than `Vector` with one-to-many mappings.

Indirection is enabled by default in a many-to-many mapping, which requires that the attribute have the `ValueHolderInterface` type or transparent collections.

You can use a many-to-many mapping with a change policy (see "Configuring Change Policy" on page 25-73.

See "Configuring a Relational Many-to-Many Mapping" on page 40-1 for more information.

Many-to-Many Mappings and EJB

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.

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

For more information, see "Configuring Bidirectional Relationship" on page 32-34.

Aggregate Collection Mapping

Aggregate collection mappings are used to represent the aggregate relationship between a single-source object and a collection of target objects. Unlike the TopLink one-to-many mappings, in which there should 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.

> **Note:** To use aggregate collections with TopLink Workbench, you must use an amendment method (see "Configuring Amendment Methods" on page 25-81), or manually edit the project source to add the mapping.

Although aggregate collection mappings are similar to one-to-many mappings, they are not replacements for one-to-many mappings. Use aggregate collections only in situations where the target collections are of a reasonable size and if having a one-to-one back mapping is 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 are privately owned by the source of the relationship and should not be shared or referenced by other objects.

This section describes the following:

- Aggregate Collection Mappings and Inheritance
- Aggregate Collection Mappings and EJB
- Implementing Aggregate Collection Mappings

See "Configuring a Relational Aggregate Collection Mapping" on page 41-1 for more information.

Aggregate Collection Mappings and Inheritance

Aggregate collection descriptors can use inheritance. You must also declare subclasses as aggregate collection. The subclasses can have their own mapped tables, or share the table with their parent class. See "Descriptors and Inheritance" on page 23-3 for more information on inheritance.

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.

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

To implement an aggregate collection mapping, the following must take place:

- 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.

Aggregate Collection Mappings and EJB

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.

Implementing Aggregate Collection Mappings

To implement an aggregate collection mapping, the following must take place:

- The descriptor of the target class must declare itself to be 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.

Direct Collection Mapping

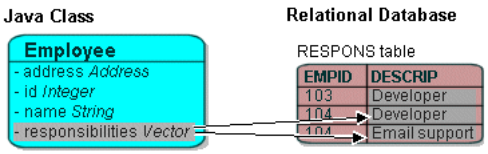
Direct collection mappings store collections of Java objects that are not TopLink-enabled. The object type stored in the direct collection is typically a Java type, such as `String`.

It is also possible to use direct collection mappings to map a collection of non-`String` objects. For example, it is possible to have an attribute that contains a collection of `Integer` or `Date` instances. The instances stored in the collection can be any type supported by the database and has a corresponding wrapper class in Java.

Support for primitive data types such as `int` is not provided, because Java `Vectors` hold only objects.

Figure 33–6 illustrates how a direct collection is stored in a separate table with two fields. The first field is the reference key field, which contains a reference to the primary key of the instance owning the collection. The second field contains an object in the collection and is called the direct field. There is one record in the table for each object in the collection.

Figure 33–6 Direct Collection Mappings



Note: The `responsibilities` attribute is of type `Vector`. When using JDK 1.2, it is possible to use a `Collection` interface (or any class that implements the Collection interface) for declaring the collection attribute. See "Configuring Container Policy" on page 32-26 for details.

Maps are not supported for direct collection because there is no key value.

You can use a direct collection mapping with any of the following `Converter` instances:

- see "Object Type Converter" on page 30-12
- see "Serialized Object Converter" on page 30-10
- see "Type Conversion Converter" on page 30-11

You can use a direct collection mapping with a change policy (see "Configuring Change Policy" on page 25-73).

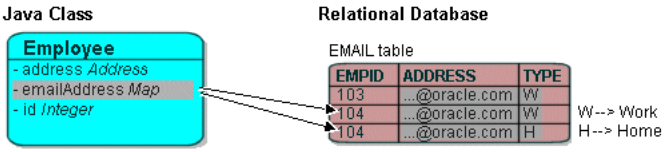
See "Configuring a Relational Direct Collection Mapping" on page 42-1 for more information.

Direct Map Mapping

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.

Figure 33–7 illustrates how a direct map is stored in a separate table with three fields. The first field (EMPID) is the reference key field, which contains a reference to the primary key of the instance owning the collection. The second field (ADDRESS) contains an object in the collection and is called the direct value field. The third field (TYPE) contains the direct key field. In this example, the direct map uses a object type converter for the direct key field, converting the single character **W** in the database to the full string **Work** in the object (and **H** to **Home**).

Figure 33–7 Direct Map Mappings



You can use a direct collection mapping with any of the following `Converter` instances:

- see "Object Type Converter" on page 30-12
- see "Serialized Object Converter" on page 30-10
- see "Type Conversion Converter" on page 30-11

You can use a direct map mapping with a change policy (see "Configuring Change Policy" on page 25-73).

See "Configuring a Relational Direct Map Mapping" on page 44-1 for more information.

Aggregate Object Mapping

Two objects–a source (parent or owning) object and a target (child or owned) object–are related by aggregation if there is a strict one-to-one relationship between them and all the attributes of the target object can be retrieved from the same table(s) as the source object. This means that if the source object exists, then the target object must also exist and if the source object is destroyed, then the target object is also destroyed.

An aggregate mapping allows you to associate data members in the target object with fields in the source object's underlying database tables.

You configure the aggregate mapping in the source object's descriptor. However, before doing so, you must designate the target object's descriptor as an aggregate (see "Configuring a Relational Descriptor as a Class or Aggregate Type" on page 26-11).

Aggregate objects are privately owned and should not be shared or referenced by other objects.

You cannot configure one-to-one, one-to-many, or many-to-many mappings from a nonaggregate object to an aggregate target object.

You can configure such mappings from an aggregate target object to another nonaggregate object. If you configure a one-to-many mapping from an aggregate target object to another nonaggregate object, you must configure a one-to-one mapping from the other object back to the source object that owns the aggregate (instead of to the aggregate target object itself). This is because the source object contains the table and primary key information of the aggregate target.

You can configure inheritance for a descriptor designated as an aggregate (see "Descriptors and Inheritance" on page 23-3), however, in this case, *all* the descriptors in the inheritance tree must be aggregates. Aggregate and class descriptors cannot exist in the same inheritance tree.

This section describes the following:

- Aggregate Object Mappings with a Single Source Object
- Aggregate Object Mappings With Multiple Source Objects
- Implementing an Aggregate Object Relationship Mapping

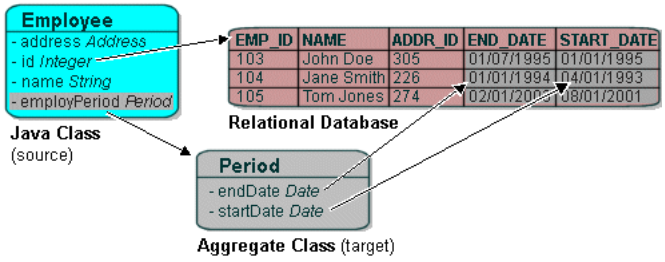
You can use an aggregate object mapping with a change policy (see "Configuring Change Policy" on page 25-73.

For more information on configuring an aggregate object relationship mapping, see "Configuring a Relational Aggregate Object Mapping" on page 43-1.

Aggregate Object Mappings with a Single Source Object

Figure 33–8 shows an example aggregate object mapping between source object `Employee` and target object `Period`. In this example, the target object is not shared by other types of source object.

Figure 33–8 Aggregate Object Mapping with a Single Source Object

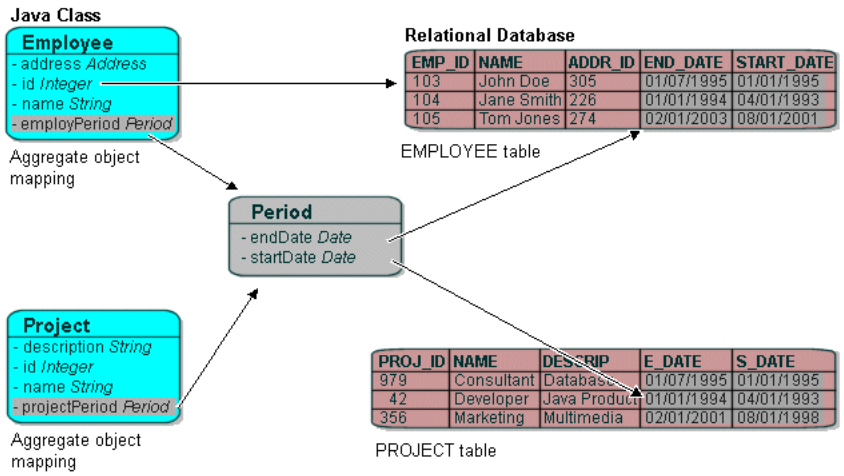


Aggregate target classes not shared among multiple source classes can have any type of mapping, including other aggregate object mappings.

Aggregate Object Mappings With Multiple Source Objects

Figure 33–9 shows an example aggregate object mapping in which different source objects–`Employee` and `Project`–map instances of the same type of target object, `Period`.

Figure 33–9 Aggregate Object Mapping with Multiple Source Objects



When you configure the aggregate object mapping in the source object, you choose the source object table for that particular mapping. This allows different source types to store the same target information within their tables. Each source object's table may use different field names. TopLink automatically manages the case where multiple source object tables use different field names.

For example, in Figure 33–9, The `Employee` attribute `employPeriod` is mapped by an aggregate object mapping to target object `Period`. This mapping associates `Period` attribute `startDate` with `EMPLOYEE` table field `START_DATE`. The `Project` attribute `projectPeriod` is also mapped by an aggregate object mapping to target object `Period`. This mapping associates `Period` attribute `startDate` with `PROJECT` table field `S_DATE`.

Aggregate target classes shared with multiple source classes cannot have one-to-many or many-to-many mappings.

Implementing an Aggregate Object Relationship Mapping

You must ensure that the following takes place:

- The descriptor of the target class declares itself to be an aggregate object. Because all its information comes from its parent's table(s), the target descriptor does not have a specific table associated with it. You must, however, choose one or more candidate table(s) from which you can use fields in mapping the target.

  In the example above, you could choose the `EMPLOYEE` table so that the `START_DATE` and `END_DATE` fields are available during mapping.

- The descriptor of the source class adds an aggregate object mapping that specifies the target class.

  In the example above, the `Employee` class has an attribute called `employPeriod` that would be mapped as an aggregate object mapping with `Period` as the reference class.

The source class must ensure that its table has fields that correspond to the field names registered with the target class.

- If a source object has a `null` target reference, TopLink writes NULLs to the aggregate database fields (see "Configuring Allowing Null Values" on page 43-2). When the source is read from the database, it can handle this `null` target in one of two ways:
  - Create an instance of the object with all its attributes equal to `null`.
  - Put a `null` reference in the source object without instantiating a target. (This is the default method of handling `null` targets.)

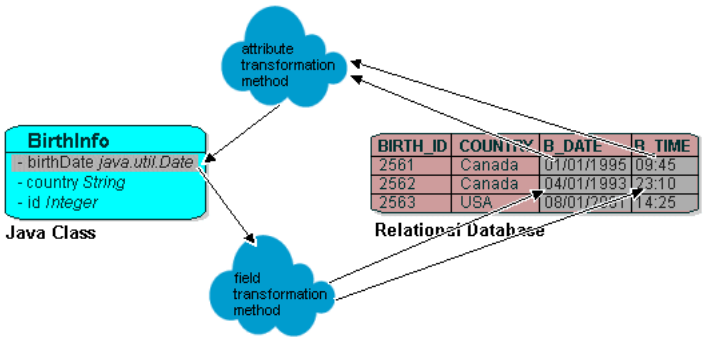
Transformation Mapping

Use transformation mappings for specialized translations for how a value is represented in Java and how it is represented in the database.

> **Tip:** Use transformation mappings only when mapping multiple fields into a single attribute. Because of the complexity of transformation mappings, it is often easier to perform the transformation with a converter or getter and setter methods of a direct-to-field mapping. See Chapter 35, "Configuring a Relational Direct-to-Field Mapping" for more information.

Figure 33–10 illustrates a transformation mapping. The values from the `B_DATE` and `B_TIME` fields are used to create a `java.util.Date` to be stored in the `birthDate` attribute.

Figure 33–10 Transformation Mappings



Often, a transformation mapping is appropriate when values from multiple fields are used to create an object. This type of mapping requires that you provide an *attribute transformation* that is invoked when reading the object from the database. This must have at least one parameter that is an instance of `Record`. In your attribute transformation, you can use `Record` method `get` to retrieve the value in a specific column. Your attribute transformation can specify a second parameter, when it is an instance of `Session`. The `Session` performs queries on the database to get additional values needed in the transformation. The transformation should *return* the value to be stored in the attribute.

Transformation mappings also require a *field transformation* for each field, to be written to the database when the object is saved. The transformation returns the value to be stored in that field.

See Chapter 45, "Configuring a Relational Transformation Mapping" for more information.

34

Configuring a Relational Mapping

This chapter describes how to configure a relational mapping.

Table 34–1 lists the types of relational mappings that you can configure and provides a cross-reference to the type-specific chapter that lists the configurable options supported by that type.

Table 34–1 Configuring Relational Mappings

If you are creating...	See...
Direct-to-Field Mapping	Chapter 35, "Configuring a Relational Direct-to-Field Mapping"
Transformation Mapping	Chapter 45, "Configuring a Relational Transformation Mapping"
Direct-to-XMLType Mapping	Chapter 36, "Configuring a Relational Direct-to-XMLType Mapping"
One-to-One Mapping	Chapter 37, "Configuring a Relational One-to-One Mapping"
Variable One-to-One Mapping	Chapter 38, "Configuring a Relational Variable One-to-One Mapping"
One-to-Many Mapping	Chapter 39, "Configuring a Relational One-to-Many Mapping"
Many-to-Many Mapping	Chapter 40, "Configuring a Relational Many-to-Many Mapping"
Aggregate Collection Mapping	Chapter 41, "Configuring a Relational Aggregate Collection Mapping"
Direct Collection Mapping	Chapter 42, "Configuring a Relational Direct Collection Mapping"
Direct Map Mapping	Chapter 44, "Configuring a Relational Direct Map Mapping"
Aggregate Object Mapping	Chapter 43, "Configuring a Relational Aggregate Object Mapping"

Table 34–2 lists the configurable options shared by two or more relational mapping types.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "Understanding Relational Mappings" on page 33-1

Configuring Common Relational Mapping Options

Table 34–2 lists the configurable options shared by two or more relational mapping types. In addition to the configurable options described here, you must also configure the options described for the specific Relational Mapping Types, as shown in Table 34–1.

Table 34–2 Common Relational Mapping Options

Option	Type	TopLink Workbench	Java
"Configuring a Database Field" on page 34-2	Basic	✔	✔
"Configuring Reference Descriptor" on page 34-5	Basic	✔	✔
"Configuring Container Policy" on page 32-26	Basic	✔	✔
"Configuring Method Accessing" on page 32-14	Advanced	✔	✔
"Configuring a Default Null Value at the Mapping Level" on page 32-12	Basic	✔	✔
"Configuring Read-Only Mappings" on page 32-2	Advanced	✔	✔
"Configuring Indirection" on page 32-3	Basic	✔	✔
"Configuring Private or Independent Relationships" on page 32-16	Advanced	✔	✔
"Configuring Mapping Comments" on page 32-18	Advanced	✔	
"Configuring a Serialized Object Converter" on page 32-18	Advanced	✔	✔
"Configuring a Type Conversion Converter" on page 32-20	Advanced	✔	✔
"Configuring an Object Type Converter" on page 32-22	Advanced	✔	✔
"Configuring Bidirectional Relationship" on page 32-34	Advanced	✔	✔
"Configuring Batch Reading" on page 34-6	Advanced	✔	✔
"Configuring Query Key Order" on page 34-8	Advanced	✔	✔
"Configuring Table and Field References (Foreign and Target Foreign Keys)" on page 34-8	Advanced	✔	✔

Configuring a Database Field

You can associate an object attribute with a database field.

Table 34–3 summarizes which relational mappings support this option.

Table 34–3 Relational Mapping Support for Database Field

Mapping	Using TopLink Workbench	Using Java
Direct-to-Field Mapping	✔	✔
Direct-to-XMLType Mapping	✔	✔

When choosing the database field, you must consider Java and database field type compatibility.

TopLink supports the following Java types:

- `java.lang`: `Boolean`, `Float`, `Integer`, `String`, `Double`, `Long`, `Short`, `Byte`, `Byte[ ]`, `Character`, `Character[ ]`; all the primitives associated with these classes
- `java.math`: `BigInteger`, `BigDecimal`
- `java.sql`: `Date`, `Time`, `Timestamp`
- `java.util`: `Date`, `Calendar`

While executing reads, the mappings in Table 34–6 perform the simple one-way data conversions that Table 34–4 describes. For two-way or more complex conversions, You must use converters (see "Converters and Transformers" on page 33-3).

The mappings in Table 34–3 also allow you to specify a null value. This may be required if primitive types are used in the object, and the database field allows `null` values. For more information, see "Configuring a Default Null Value at the Mapping Level" on page 32-12.

Table 34–4 Type Conversions Provided by Direct-to-Field Mappings

Java type	Database type
Integer, Float, Double, Byte, Short, BigDecimal, BigInteger, int, float, double, byte, short	NUMBER, NUMERIC, DECIMAL, FLOAT, DOUBLE, INT, SMALLINT, BIT, BOOLEAN
Boolean, boolean	BOOLEAN, BIT, SMALLINT, NUMBER, NUMERIC, DECIMAL, FLOAT, DOUBLE, INT
String	VARCHAR, CHAR, VARCHAR2, CLOB, TEXT, LONG, LONG VARCHAR, MEMO The following types apply only to Oracle9: NVARCHAR2, NCLOB, NCHAR
byte[ ]	BLOB, LONG RAW, IMAGE, RAW, VARBINARY, BINARY, LONG VARBINARY
Time	TIME
sql.Date	DATE (only applies to DB2)
Timestamp, util.Date, Calendar	TIMESTAMP (only applies to DB2)
sql.Date, Time, Timestamp, util.Date, Calendar To use `oracle.sql.TimeStamp`, see "Support for oracle.sql.TimeStamp" on page 34-3.	DATE, DATETIME (applies to Oracle, Sybase, SQL Server)

Support for oracle.sql.TimeStamp

TopLink provides additional support for mapping Java date and time data types to Oracle database `DATE`, `TIMESTAMP`, and `TIMESTAMPTZ` data types when you use the Oracle JDBC driver with Oracle9*i* Database Server or later and the `Oracle9Platform` in TopLink.

In a direct-to-field mapping, you are not required to specify the database type of the field value; TopLink determines the appropriate data type conversion.

Table 34–5 lists the supported direct-to-field mapping combinations.

Table 34–5 Supported Oracle Database Date and Time Direct-to-Field Mappings

Java Type	Database Type	Description
`java.sql.Time`	`TIMESTAMP`	Full bidirectional support.
	`TIMESTAMPTZ`	Full bidirectional support.
	`DATE`	Full bidirectional support.
`java.sql.Date`	`TIMESTAMP`	Full bidirectional support.
	`TIMESTAMPTZ`	Full bidirectional support.
	`DATE`	Full bidirectional support.
`java.sql.Timestamp`	`TIMESTAMP`	Full bidirectional support.
	`TIMESTAMPTZ`	Full bidirectional support.

Table 34–5 (Cont.) Supported Oracle Database Date and Time Direct-to-Field Mappings

Java Type	Database Type	Description
	`DATE`	Nanoseconds are not stored in the database.
`java.util.Date`	`TIMESTAMP`	Full bidirectional support.
	`TIMESTAMPTZ`	Full bidirectional support.
	`DATE`	Milliseconds are not stored in the database.
`java.util.Calendar`	`TIMESTAMP`	Native SQL or binding gives Calendar timezone. Note: The TIMESTAMP database value has no timezone – the Calendar object provides the local timezone by default. If the database is not in this timezone, you must obtain the database timezone by some other means and update the Calendar object accordingly. For this reason, TIMESTAMPTZ may be a better choice.
	`TIMESTAMPTZ`	Native SQL or binding stores timezone; standard SQL is based on the local timezone.
	`DATE`	Neither timezone nor milliseconds are stored in the database.

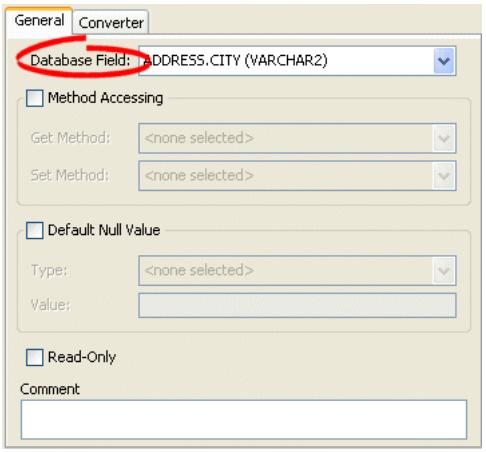
Note that some of these mappings result in a loss of precision: avoid these combinations if you require this level of precision. For example, if you create a direct-to-field mapping between a `java.sql.Date` attribute and a `TIMESTAMPTZ` database field, there is no loss of precision. However, if you create a direct-to-field mapping between a `java.sql.Timestamp` attribute and a `DATE` database field, the nanoseconds or milliseconds of the attribute are not stored in the database.

Using TopLink Workbench

Use this procedure to select a specific database field for a direct mapping.

1. Select the direct mapping attribute in the **Navigator**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.

Figure 34–1 Direct Mapping General Tab, Database Field Option



Use the **Database Field** field to select a field for this direct mapping. You must have previously associated the descriptor with a database table as described in "Configuring Associated Tables" on page 26-2.

> **Note:** For direct to field mappings of a an Aggregate descriptor (see "Configuring a Relational Descriptor as a Class or Aggregate Type" on page 26-11), this field is for display only and cannot be changed.

Configuring Reference Descriptor

In relational mappings that extend `oracle.toplink.mappings.ForeignReferenceMapping`, attributes reference other TopLink descriptors–not the data source. You can select any descriptor in the project.

Table 34–6 summarizes which relational mappings support this option.

Table 34–6 Relational Mapping Support for Reference Descriptor

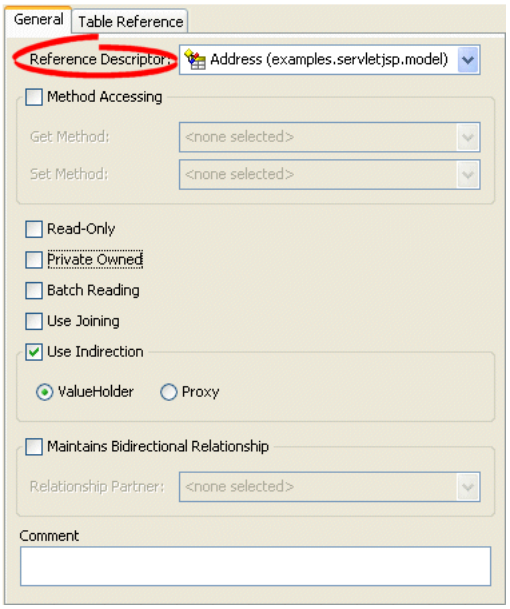
Mapping	Using TopLink Workbench	Using Java
One-to-One Mapping	✔	✔
Variable One-to-One Mapping	✔	✔
One-to-Many Mapping	✔	✔
Many-to-Many Mapping	✔	✔
Aggregate Collection Mapping		✔
Aggregate Object Mapping	✔	✔

Using TopLink Workbench

To specify a reference descriptor for a relational mapping, use this procedure.

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.

Figure 34–2 General Tab, Reference Descriptor Field



Use the **Reference Descriptor** field to select the descriptor referenced by this relationship mapping.

Note: For aggregate mappings the **Reference Descriptor** must be an *aggregate*. See "Configuring a Relational Descriptor as a Class or Aggregate Type" on page 26-11 for more information.

For variable one-to-one mappings, the Reference Descriptor must be an *interface*. See Chapter 38, "Configuring a Relational Variable One-to-One Mapping" for more information.

You can specify a reference descriptor that is not in the current TopLink Workbench project. For example, to create a mapping to an `Employee` class that does not exist in the current project, do the following:

1. Add the `Employee` class to your current project. See "Working With Projects" on page 18-10.
2. Create the relationship mapping to the `Employee` descriptor.
3. Deactivate the `Employee` descriptor. See "Active and Inactive Descriptors" on page 4-10.

When you generate the deployment XML for your project, the *mapping* to the `Employee` class will be included, but not the `Employee` class.

Configuring Batch Reading

Batch reading can be used in most of the relational mappings. This feature should be used only if it is known that the related objects are always required with the source object.

Table 34–7 summarizes which relational mappings support this option.

Table 34–7 Relational Mapping Support for Batch Reading

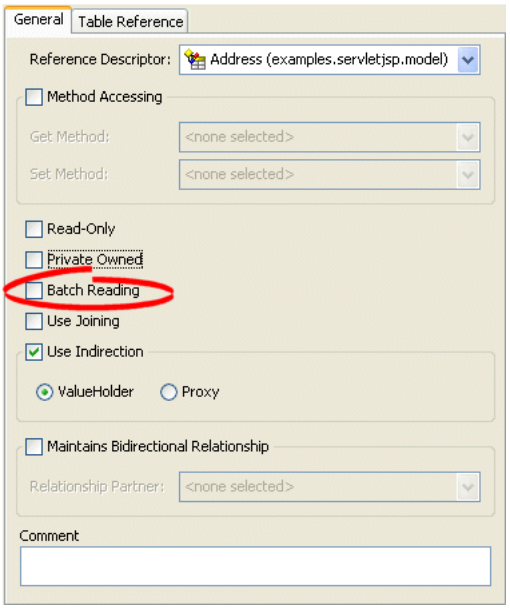
Mapping	Using TopLink Workbench	Using Java
One-to-One Mapping	✔	✔
One-to-Many Mapping	✔	✔
Many-to-Many Mapping	✔	✔
Direct Collection Mapping	✔	✔
Direct Map Mapping	✔	✔
Aggregate Object Mapping		✔

Using TopLink Workbench

To use batch reading in a relationship mapping, use this procedure:

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.

Figure 34–3 General Tab, Batch Reading Option



To specify that this mapping using batch reading, select the **Batch Reading** option.

Using Java

Example 34–1 Query Optimization Using Batching

The following code example illustrates using batch for query optimization.

```
// Queries on Employee are configured to always batch read Address
OneToManyMapping phoneNumbersMapping = new OneToManyMapping();
phoneNumbersMapping.setReferenceClass("
PhoneNumber.class")
phoneNumbersMapping.setAttributeName("phones");
```

```
phoneNumbersMapping.useBatchReading();
phoneNumbersMapping.privateOwnedRelationship();
```

Configuring Query Key Order

You can configure TopLink to maintain collections in order by query key.

Table 34–8 summarizes which relational mappings support this option.

Table 34–8 Relational Mapping Support for Query Key Order

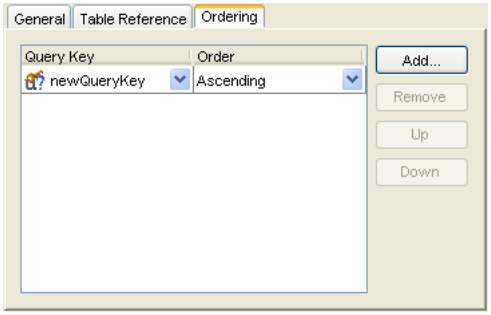
Mapping	Using TopLink Workbench	Using Java
Variable One-to-One Mapping	✔	✔
One-to-Many Mapping	✔	✔
Aggregate Collection Mapping		✔

Using TopLink Workbench

To specify the order of a mapping's query keys, use this procedure:

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.
2. Click the **Ordering** tab. The Ordering tab appears.

Figure 34–4 Ordering Tab



Field	Description
Query Key	Specify the query key to order by. Click **Add** to add query keys to, or **Remove** to remove query keys from the ordering operation. Click **Up** or **Down** to change the sort order of selected query keys.
Order	Specify if TopLink orders the selected query key in **Ascending** or **Descending** (alphabetical) order.

Configuring Table and Field References (Foreign and Target Foreign Keys)

A foreign key is a combination of one or more database columns that reference a unique key, usually the primary key, in another table. Foreign keys can be any number

of fields (similar to a primary key), all of which are treated as a unit. A foreign key and the parent key it references must have the same number and type of fields.

Mappings that extend `oracle.toplink.mappings.ForeignReferenceMapping` use foreign keys to find information in the database so that the target object(s) can be instantiated. 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 would specify foreign key information to find an `Address` for a particular `Employee`.

TopLink classifies foreign keys into two categories in mappings–**foreign keys** and **target foreign keys**:

- In a *foreign key*, the key is found in the table associated with the mapping's own descriptor. For example, an `Employee` foreign key to `ADDRESS` would be in the `EMPLOYEE` table.
- In a *target foreign key*, the reference is from the target object's table back to the key from the mapping's descriptor's table. For example, the `ADDRESS` table would have a foreign key to `EMPLOYEE`.

> **Caution:** Make sure you fully understand the distinction between *foreign key* and *target foreign key* before defining a mapping.

The table reference is the database table that contains the foreign key references.

Table 34–9 summarizes which relational mappings support this option.

Table 34–9 Relational Mapping Support for Table Reference

Mapping	Using TopLink Workbench	Using Java
One-to-One Mapping	✔	✔
One-to-Many Mapping	✔	✔
Many-to-Many Mapping	✔	✔
Aggregate Collection Mapping		✔
Direct Collection Mapping	✔	✔
Direct Map Mapping	✔	✔

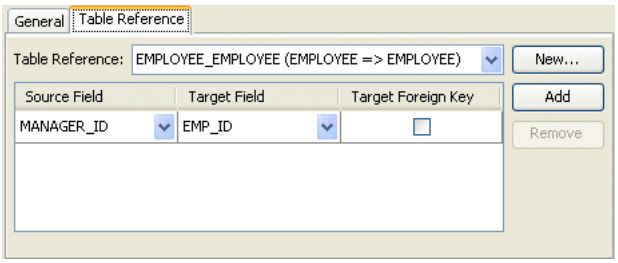
Using TopLink Workbench, you can either import this table from your database or create it. If you import tables from the database (see "Importing Tables From a Database" on page 4-22), TopLink creates references that correspond to existing database constraints (if supported by the driver). You can also define references in TopLink without creating similar constraints on the database.

Using TopLink Workbench

To specify a table for a mapping reference, use this procedure:

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.
2. Click the **Table Reference** tab. The Reference tab appears.

Figure 34–5 Table Reference Tab, Table Reference Field



Use the following information to select the field references on the tab:

Field	Description
Table Reference	Select an existing table, or click **New** to create a new table reference.
Source and Target Field	Click **Add** to create new foreign key reference. To delete an existing key pair reference, select the **Source** and **Target** fields and click **Remove**.
Source Field	Select the database field from the *source* table for this foreign key reference.
Target Field	Select the database field from the *target* table for this foreign key reference.
Target Foreign Key	Specify whether or not the reference is from the target object's table back to the key from the mapping's descriptor's table.

35

Configuring a Relational Direct-to-Field Mapping

This chapter describes the various components that you must configure in order to use a relational direct-to-field mapping.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "Direct-to-Field Mapping" on page 33-4

Relational Direct-to-Field Mapping Configuration Overview

Table 35–1 lists the configurable options for a relational direct-to-field mapping.

Table 35–1 Configurable Options for Relational Direct-to-Field Mapping

Option	Type	TopLink Workbench	Java
"Configuring a Database Field" on page 34-2	Basic	✔	✔
"Configuring Method Accessing" on page 32-14	Advanced	✔	✔
"Configuring a Default Null Value at the Mapping Level" on page 32-12	Advanced	✔	✔
"Configuring Read-Only Mappings" on page 32-2	Advanced	✔	✔
"Configuring Mapping Comments" on page 32-18	Advanced	✔	
"Configuring a Serialized Object Converter" on page 32-18	Advanced	✔	✔
"Configuring a Type Conversion Converter" on page 32-20	Advanced	✔	✔
"Configuring an Object Type Converter" on page 32-22	Advanced	✔	✔

36

Configuring a Relational Direct-to-XMLType Mapping

This chapter describes the various components that you must configure in order to use a relational direct-to-`XMLType` mapping.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "Direct-to-XMLType Mapping" on page 33-4

Relational Direct-to-XMLType Mapping Overview

Table 36–1 lists the configurable options for a relational direct-to-`XMLType` mapping.

Table 36–1 Configurable Options for Relational Direct-To-XMLType Mapping

Option	Type	TopLink Workbench	Java
"Configuring a Database Field" on page 34-2	Basic	✔	✔
"Configuring Method Accessing" on page 32-14	Advanced	✔	✔
"Configuring Read-Only Mappings" on page 32-2	Advanced	✔	✔
"Configuring Read Whole Document" on page 36-1	Advanced	✔	✔
"Configuring Mapping Comments" on page 32-18	Advanced	✔	

Configuring Read Whole Document

When mapping an XML Type to a Document Object Model (DOM), by default TopLink uses the database representation of the DOM. This allows for lazy loading of the XML data from the database.

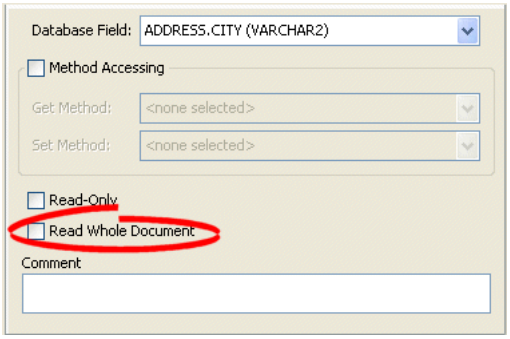
However, if you require the entire DOM, (or if you require the DOM to be available in a disconnected fashion from the database connection) use the **Read Whole** option to retrieve the entire DOM from the database.

Using TopLink Workbench

To specify that this mapping reads the whole XML document, use this procedure:

1. Select the mapping in the **Navigator**. Its properties appear in the Editor.
2. Click **General**. The General tab appears.

Figure 36–1 DIrect to XML Mapping Property Sheet, Read Whole Document Option



Choose the **Read Whole Document** option to read the whole XML document. If you do not select this option, the connection must remain open for TopLink to read the database values.

37

Configuring a Relational One-to-One Mapping

This chapter describes the various components that you must configure in order to use a relational one-to-one mapping.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "One-to-One Mapping" on page 33-5

Relational One-to-One Mapping Configuration Overview

Table 37–1 lists the configurable options for a relational one-to-one mapping.

Table 37–1 Configurable Options for Relational One-to-One Mapping

Option	Type	TopLink Workbench	Java
"Configuring Reference Descriptor" on page 34-5	Basic	✔	✔
"Configuring Method Accessing" on page 32-14	Advanced	✔	✔
"Configuring Read-Only Mappings" on page 32-2	Advanced	✔	✔
"Configuring Private or Independent Relationships" on page 32-16	Advanced	✔	✔
"Configuring Batch Reading" on page 34-6	Advanced	✔	✔
"Configuring Joining at the Mapping Level" on page 37-1	Advanced	✔	✔
"Configuring Indirection" on page 32-3	Advanced	✔	✔
"Configuring Bidirectional Relationship" on page 32-34	Advanced	✔	✔
"Configuring Mapping Comments" on page 32-18	Advanced	✔	
"Configuring Table and Field References (Foreign and Target Foreign Keys)" on page 34-8	Basic	✔	✔

Configuring Joining at the Mapping Level

TopLink supports configuring an inner join at the mapping level for one-to-one mappings only. When a class that owns the mapping is read, the TopLink runtime will always get the class and the target of the one-to-one mapping with one database hit.

Use this feature only if the target object is *always* required with the source object, or when indirection is *not* used.

You can also configure join reading (see "Join Reading and Object-Level Read Queries" on page 93-12) at the query level.

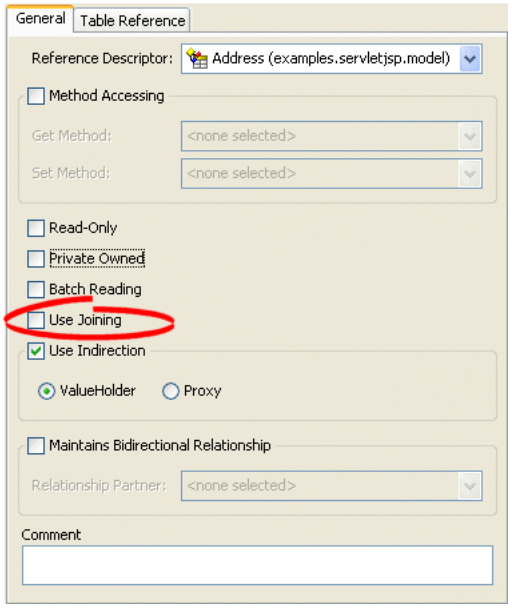
For more information about joins, see "Expressions for Joining and Complex Relationships" on page 95-6.

Using TopLink Workbench

To use joining in a relationship mapping, use this procedure:

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.

Figure 37–1 General Tab, Use Joining Option



To use joining with this relationship, select the **Use Joining** option.

38

Configuring a Relational Variable One-to-One Mapping

This chapter describes the various components that you must configure in order to use a relational variable one-to-one mapping.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "Variable One-to-One Mapping" on page 33-6

Relational Variable One-to-One Mapping Configuration Overview

Table 38–1 lists the configurable options for a relational variable one-to-one mapping.

Table 38–1 Configurable Options for Relational Variable One-to-One Mapping

Option	Type	TopLink Workbench	Java
"Configuring Reference Descriptor" on page 34-5	Basic	✔	✔
"Configuring Method Accessing" on page 32-14	Advanced	✔	✔
"Configuring Read-Only Mappings" on page 32-2	Advanced	✔	✔
"Configuring Private or Independent Relationships" on page 32-16	Advanced	✔	✔
"Configuring Indirection" on page 32-3	Advanced	✔	✔
"Configuring Mapping Comments" on page 32-18	Advanced	✔	
"Configuring Query Key Association" on page 38-4	Advanced	✔	✔
"Configuring Class Indicator" on page 38-1	Basic	✔	✔
"Configuring Unique Primary Key" on page 38-3	Basic	✔	✔

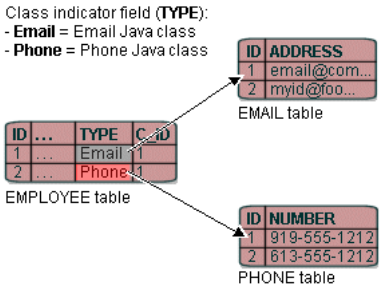
Configuring Class Indicator

In variable one-to-one mappings, you can use an indicator column in the source table to specify the correct target table, as illustrated in Figure 38–1. This section includes information on implementing class indicators.

A source table has an indicator column that specifies the target table through the class indicator field.

Figure 38–1 illustrates the EMPLOYEE table that has a TYPE column that indicates the target for the row (either PHONE or EMAIL).

Figure 38–1 Variable One-to-One Mapping using Class indicator Field



The principles of defining such a variable class relationship are similar to defining a normal one-to-one relationship, except:

- The reference class is a Java *interface*, not a Java *class*. However, the method to set the interface is identical.
- You must specify a type indicator field.
- You specify the class indicator values on the mapping so that mapping can determine the class of object to create.
- You must specify the foreign key names and the respective abstract query keys from the target interface descriptor.

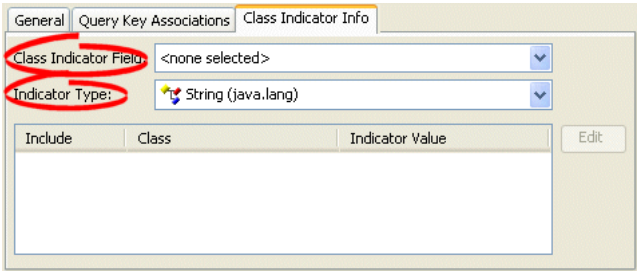
Alternatively, you can use unique primary keys to specify the correct target. See "Configuring Unique Primary Key" on page 38-3 for more information.

Using TopLink Workbench

To specify the class indicators for a variable one-to-one mapping, use this procedure:

1. Select the variable one-to-one mapping in the **Navigator**. Its attributes appear in the Editor.
2. Click the **Class Indicator Info** tab. The Class Indicator Info tab appears.

Figure 38–2 Class Indicator Info Tab



Use the **Class Indicator Field** to select the field on the database table (associated with the mapping's descriptor) to use as the indicator for the variable mapping.

Use the **Indicator Type** to specify the data type of the class indicator field by selecting the data type from the list.

To specify the specific class indicator field values for each (nonabstract) child class, click **Edit** and enter the appropriate value for each child class.

Configuring Unique Primary Key

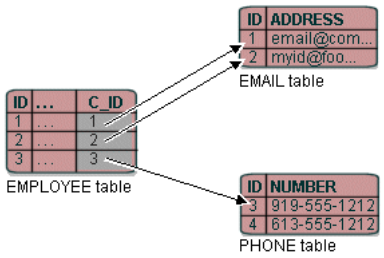
In variable one-to-one mappings, you can use a unique primary key in the source table to specify the correct target table, as illustrated in Figure 38–3. This section includes information on implementing class indicators.

Alternatively, you can use a class indicator to specify the correct target. See "Configuring Class Indicator" on page 38-3 for more information.

Understanding Unique Primary Key

As Figure 38–3 illustrates, the value of the foreign key in the source table (C_ID) mapped to the primary key of the target table is unique. No primary key values among the target tables are the same, so primary key values are not unique just in the table, but also among the tables.

Figure 38–3 Variable One-to-One Relationship with Unique Primary Key



If there is no indicator stored in the source table, TopLink cannot determine to which target table the foreign key value is mapped. Therefore, TopLink reads through all the target tables until it finds an entry in one of the target tables. This is an inefficient way of setting up a relation model. The class indicator is much more efficient as it reduces the number of reads performed on the tables to get the data. In the class indicator method, TopLink knows exactly which target table to look into for the data.

The principles of defining such a variable class relationship are similar to defining class indicator variable one-to-one relationships, except:

- A type indicator field is not specified.
- The class indicator values are not specified.

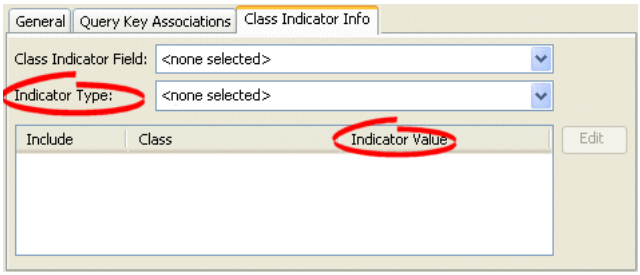
The type indicator field and its values are not needed, because TopLink goes through all the target tables until data is finally found.

Using TopLink Workbench

To specify the variable one-to-one mapping with a primary key, use this procedure:

1. Select the variable one-to-one mapping in the **Navigator**. Its attributes appear in the Editor.
2. Click the **Class Indicator Info** tab. The Class Indicator Info tab appears.

Figure 38–4 Class Indicator Info Tab, Configuring Primary Key



Use the **Class Indicator Field** to select none.

Use the **Indicator Type** to select none.

Use the **Indicator Value** column to specify none.

After choosing the reference descriptor for the mapping, deselect the **Include** check boxes.

Note: You cannot deselect the value in the Class Indicator Field, unless the foreign key values in the source table are unique.

Using Java

Example 38–1 illustrates how to define a variable one-to-one mapping using a unique primary key in Java code.

Example 38–1 Defining a variable one-to-one mapping using a unique primary key

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

Configuring Query Key Association

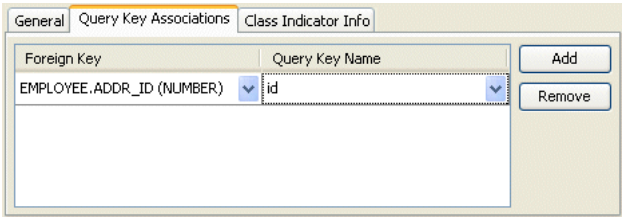
The API to configure query key associations is `oracle.toplink.mappings.VariableOneToOneMapping` method `addForeingQueryKeyName(String, String)`.

Using TopLink Workbench

To specify the query keys used for a variable one-to-one mapping, use this procedure:

1. Select the variable one-to-one mapping in the **Navigator**. Its attributes appear in the Editor.
2. Click the **Query Key Associations** tab. The Query Key Associations tab appears

Figure 38–5 Query Key Associations Tab



Use the following information to enter data in each field on the tab:

Use the **Indicator Type** to specify the data type of the class indicator field by selecting the data type from the list.

Field	Description
Foreign Key	The field from the database table to use as the foreign key in this relationship.
Query Key Name	The name of the query key (from the referenced descriptor) for this association. See "Configuring Query Keys" on page 25-30 for more information.

39

Configuring a Relational One-to-Many Mapping

This chapter describes the various components that you must configure in order to use a relational one-to-many mapping.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "One-to-Many Mapping" on page 33-7

Relational One-to-Many Mapping Configuration Overview

Table 39–1 lists the configurable options for a relational one-to-many mapping.

Table 39–1 Configurable Options for Relational One-to-Many Mapping

Option	Type	TopLink Workbench	Java
"Configuring Reference Descriptor" on page 34-5	Basic	✔	✔
"Configuring Method Accessing" on page 32-14	Advanced	✔	✔
"Configuring Read-Only Mappings" on page 32-2	Advanced	✔	✔
"Configuring Private or Independent Relationships" on page 32-16	Advanced	✔	✔
"Configuring Batch Reading" on page 34-6	Advanced	✔	✔
"Configuring Indirection" on page 32-3	Advanced	✔	✔
"Configuring Bidirectional Relationship" on page 32-34	Advanced	✔	✔
"Configuring Container Policy" on page 32-26	Basic	✔	✔
"Configuring Mapping Comments" on page 32-18	Advanced	✔	
"Configuring Table and Field References (Foreign and Target Foreign Keys)" on page 34-8	Basic	✔	✔
"Configuring Query Key Order" on page 34-8	Advanced	✔	✔

40

Configuring a Relational Many-to-Many Mapping

This chapter describes the various components that you must configure in order to use a relational many-to-many mapping.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "Many-to-Many Mapping" on page 33-8

Relational Many-to-Many Mapping Configuration Overview

Table 40–1 lists the configurable options for a relational many-to-many mapping.

Table 40–1 Configurable Options for Relational Many-to-Many Mapping

Option	Type	TopLink Workbench	Java
"Configuring Reference Descriptor" on page 34-5	Basic	✔	✔
"Configuring Method Accessing" on page 32-14	Advanced	✔	✔
"Configuring Read-Only Mappings" on page 32-2	Advanced	✔	✔
"Configuring Private or Independent Relationships" on page 32-16	Advanced	✔	✔
"Configuring Batch Reading" on page 34-6	Advanced	✔	✔
"Configuring Indirection" on page 32-3	Advanced	✔	✔
"Configuring Bidirectional Relationship" on page 32-34	Basic	✔	✔
"Configuring Container Policy" on page 32-26	Basic	✔	✔
"Configuring Mapping Comments" on page 32-18	Advanced	✔	
"Configuring a Relation Table" on page 40-1	Basic	✔	✔
"Configuring Table and Field References (Foreign and Target Foreign Keys)" on page 34-8 (Source)	Basic	✔	✔
"Configuring Table and Field References (Foreign and Target Foreign Keys)" on page 34-8 (Target)	Basic	✔	✔
"Configuring Query Key Order" on page 34-8	Advanced	✔	✔

Configuring a Relation Table

The relation table contains the columns for the primary keys of the source table and target table involved in the many-to-many mapping. You must create this table in the

database before completing the mapping. See "Working With Databases" on page 4-21 for information on creating database tables.

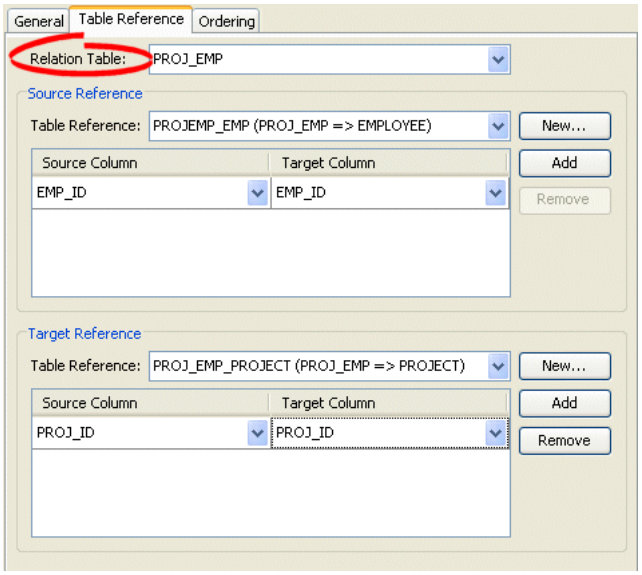
In Figure 33–5 on page 33-9, the PROJ_EMP table serves as the relation table between the PROJECT and EMPLOYEE tables.

Using TopLink Workbench

To select a relation table for a mapping, use this procedure:

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.

Figure 40–1 Table Reference Tab, Relation Table Option



Use the **Relation Table** field to select a database table to define this mapping.

41

Configuring a Relational Aggregate Collection Mapping

This chapter describes the various components that you must configure in order to use a relational aggregate collection mapping.

> **Note:** To use a relational aggregate collection mapping with TopLink Workbench, you must use an amendment method (see "Configuring Amendment Methods" on page 25-81).

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "Aggregate Collection Mapping" on page 33-10

Relational Aggregate Collection Mapping Configuration Overview

Table 41–1 lists the configurable options for a relational aggregate collection mapping.

Table 41–1 Configurable Options for Relational Aggregate Collection Mapping

Option	Type	TopLink Workbench	Java
"Configuring a Database Field" on page 34-2	Basic		✔
"Configuring Reference Descriptor" on page 34-5	Basic		✔
"Configuring Container Policy" on page 32-26	Basic		✔
"Configuring Method Accessing" on page 32-14	Advanced		✔
"Configuring Read-Only Mappings" on page 32-2	Advanced		✔
"Configuring Batch Reading" on page 34-6	Advanced		✔
"Configuring Bidirectional Relationship" on page 32-34	Advanced		✔
"Configuring Query Key Order" on page 34-8	Advanced		✔
"Configuring Table and Field References (Foreign and Target Foreign Keys)" on page 34-8	Advanced		✔

42

Configuring a Relational Direct Collection Mapping

This chapter describes the various components that you must configure in order to use a relational direct collection mapping.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "Direct Collection Mapping" on page 33-11

Relational Direct Collection Mapping Configuration Overview

Table 42–1 lists the configurable options for a relational direct collection mapping.

Table 42–1 Configurable Options for Relational Direct Collection Mapping

Option	Type	TopLink Workbench	Java
"Configuring Target Table" on page 42-1	Basic	✔	✔
"Configuring Direct Value Field" on page 42-2	Basic	✔	✔
"Configuring Method Accessing" on page 32-14	Advanced	✔	✔
"Configuring Read-Only Mappings" on page 32-2	Advanced	✔	✔
"Configuring Batch Reading" on page 34-6	Advanced	✔	✔
"Configuring Indirection" on page 32-3	Advanced	✔	✔
"Configuring Container Policy" on page 32-26	Basic	✔	✔
"Configuring Mapping Comments" on page 32-18	Advanced	✔	
"Configuring a Serialized Object Converter" on page 32-18	Advanced	✔	✔
"Configuring a Type Conversion Converter" on page 32-20	Advanced	✔	✔
"Configuring an Object Type Converter" on page 32-22	Advanced	✔	✔
"Configuring Table and Field References (Foreign and Target Foreign Keys)" on page 34-8	Basic	✔	✔

Configuring Target Table

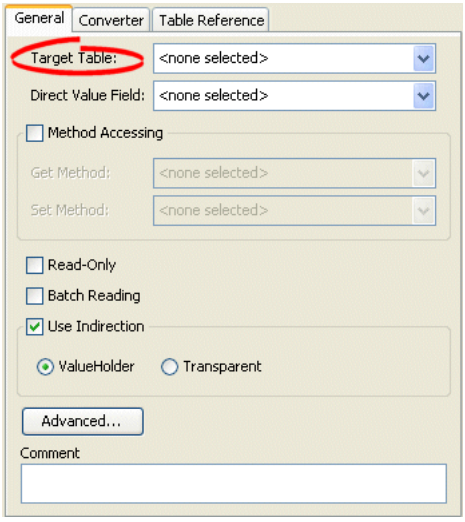
Each direct collection stores reference information in a target table. In Figure 33–6 on page 33-11, the `RESPONS` table contains the primary key and object of the instance owning the collection. You must create this table in your database.

Using TopLink Workbench

To specify the direct collection specifics, use this procedure:

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.

Figure 42–1 General Tab, Target Table Options



Use the **Target Table** list to select the table that contains the reference fields for the direct collection mapping.

Configuring Direct Value Field

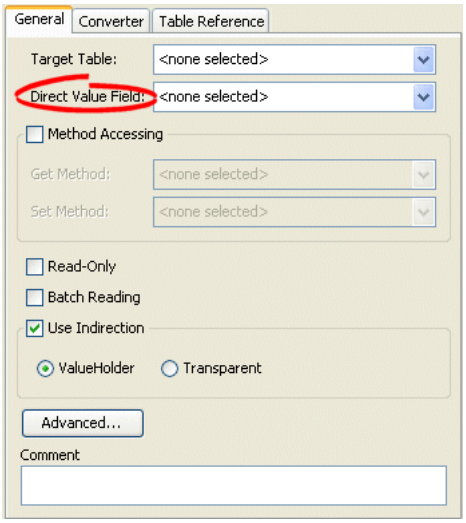
The direct value field, located in the reference table, stores the primitive data value. In Figure 33–6 on page 33-11, the `DESCRIP` field stores the collection.

Using TopLink Workbench

To specify the direct collection specifics, use this procedure:

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.

Figure 42–2 General Tab, Direct Value Field



Use the **Direct Value Field** list to select the field from the **Target Table** table that contains the object of the collection.

43

Configuring a Relational Aggregate Object Mapping

This chapter describes the various components that you must configure in order to use a relational aggregate object mapping.

> **Note:** You configure the relational aggregate object mapping in the source object's descriptor. However, before doing so, you must designate the target object's descriptor as an aggregate (see "Configuring a Relational Descriptor as a Class or Aggregate Type" on page 26-11).

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "Aggregate Object Mapping" on page 33-12

Relational Aggregate Object Mapping Configuration Overview

Table 43–1 lists the configurable options for a relational aggregate object mapping.

Table 43–1 Configurable Options for Relational Aggregate Object Mapping

Option	Type	TopLink Workbench	Java
"Configuring Reference Descriptor" on page 34-5	Basic	✔	✔
"Configuring Method Accessing" on page 32-14	Advanced	✔	✔
"Configuring Read-Only Mappings" on page 32-2	Advanced	✔	✔
"Configuring Allowing Null Values" on page 43-2	Advanced	✔	✔
"Configuring Mapping Comments" on page 32-18	Advanced	✔	
"Configuring Aggregate Fields" on page 43-1	Basic	✔	✔

Configuring Aggregate Fields

When you designate a descriptor as an aggregate, TopLink allows you to specify a mapping type for each field in the target class, but defers associating the field with a database table until you configure the aggregate object mapping in the source class descriptor. In other words, the target class descriptor defines how each target class field is mapped but the source class descriptor defines where each target class field is mapped.

This section explains how to configure the source class descriptor to define where each target class field is mapped.

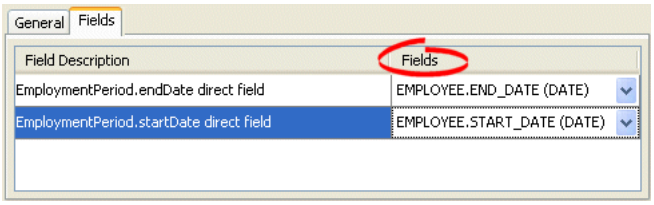
For more information on how to configure the target class descriptor to define how each target class field is mapped, see "Configuring a Relational Descriptor as a Class or Aggregate Type" on page 26-11.

Using TopLink Workbench

To specify the mapped fields of an aggregate mapping, use this procedure.

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.
2. Click the **Fields** tab. The Fields tab appears.

Figure 43–1 Fields Tab



3.

Use the following information to complete each field on the tab:

Field	Description
Field Description	This column shows the name of the fields from the target object, whose descriptor is designated as an aggregate (see "Configuring a Relational Descriptor as a Class or Aggregate Type" on page 26-11). These are for display only and cannot be changed.
Fields	Use this column to select the source object database table field that TopLink will map to the corresponding target object field.

Configuring Allowing Null Values

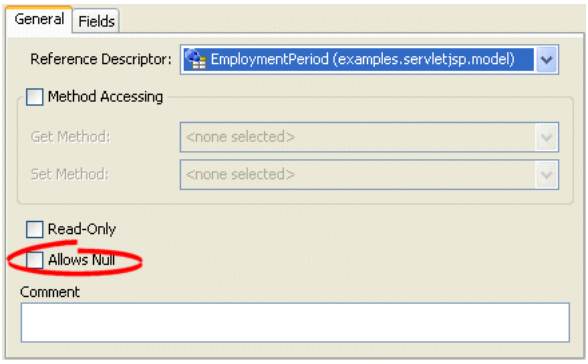
If all the fields in the database row for the aggregate object are `null`, then, by default, TopLink places `null` in the appropriate source object, as opposed to filling an aggregate object with `null` values.

Using TopLink Workbench

To allow a mapping to contain a null value, use this procedure.

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.

Figure 43–2 General Tab, Allow Null Option



Select the **Allows Null** option to allow this mapping to contain a null value.

44

Configuring a Relational Direct Map Mapping

This chapter describes the various components that you must configure in order to use a relational direct map mapping.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "Direct Map Mapping" on page 33-12

Relational Direct Map Mapping Configuration Overview

Table 44–1 lists the configurable options for a relational direct map mapping.

Table 44–1 Configurable Options for Relational Direct Map Mapping

Option	Type	TopLink Workbench	Java
"Configuring Target Table" on page 42-1	Basic	✔	✔
"Configuring Direct Value Field" on page 44-1	Basic	✔	✔
"Configuring Direct Key Field" on page 44-2	Basic	✔	✔
"Configuring Method Accessing" on page 32-14	Advanced	✔	✔
"Configuring Read-Only Mappings" on page 32-2	Advanced	✔	✔
"Configuring Batch Reading" on page 34-6	Advanced	✔	✔
"Configuring Indirection" on page 32-3	Advanced	✔	✔
"Configuring Container Policy" on page 32-26	Advanced	✔	✔
"Configuring Mapping Comments" on page 32-18	Advanced	✔	
"Configuring Key Converters" on page 44-3	Advanced	✔	✔
"Configuring Value Converters" on page 44-4	Advanced	✔	✔
"Configuring Table and Field References (Foreign and Target Foreign Keys)" on page 34-8	Basic	✔	✔

Configuring Direct Value Field

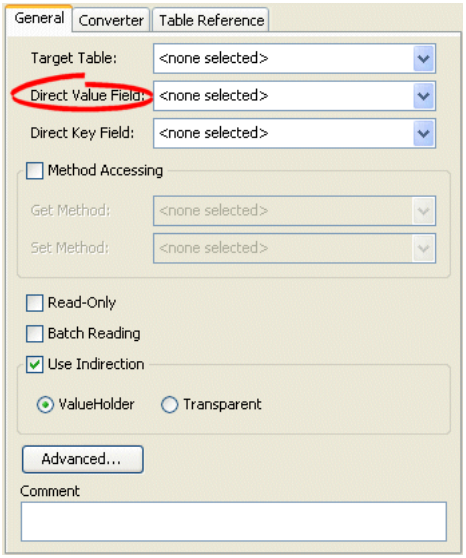
The direct value field in the reference table stores the primitive data value of the map value. If the value's object value and database value are different types, use a converter (see "Configuring Value Converters" on page 44-3).

Using TopLink Workbench

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.

2. Click the **General** tab. The General tab appears.

Figure 44–1 General Tab, Direct Value Field



Use the **Direct Value Field** list to select the field from the **Target Table** table that contains the object of the direct map mapping.

Configuring Direct Key Field

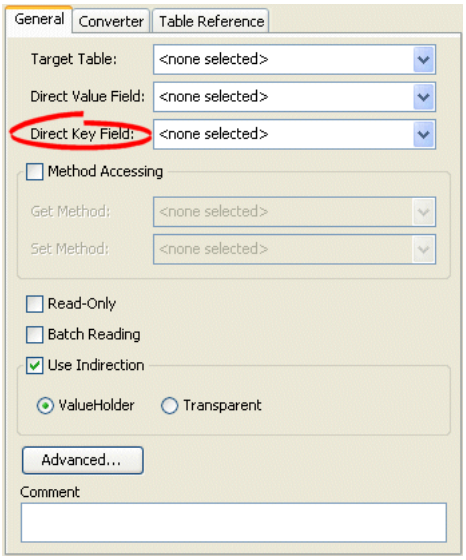
The direct key field in the reference table stores the primitive data value of the map key. If the key's object value and database value are different types, use a converter (see "Configuring Key Converters" on page 44-3).

Using TopLink Workbench

To specify the direct key field in the reference table, use this procedure.

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.

Figure 44–2 General Tab, Direct Key Field



Use the **Direct Key Field** list to select the key from the **Target Table** table that contains the object of the direct map mapping.

Configuring Key Converters

If the key's object value and database value are different types, use a converter. TopLink supports the following key converters:

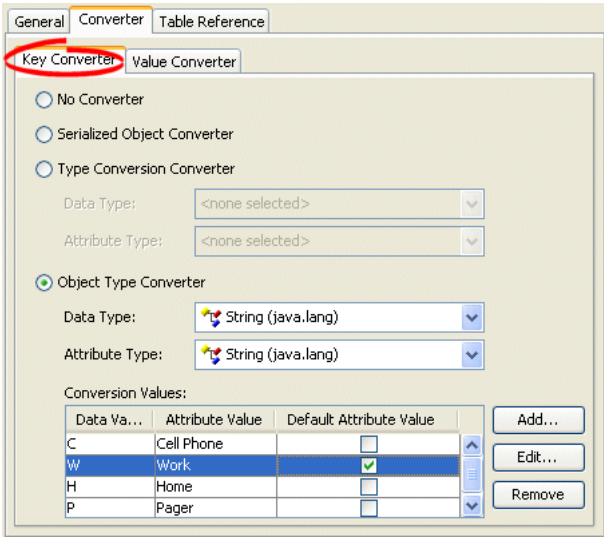
- see "Serialized Object Converter" on page 30-10
- see "Type Conversion Converter" on page 30-11
- see "Object Type Converter" on page 30-12

Using TopLink Workbench

Use this procedure to specify the converter for a direct map mapping key:

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.
2. Click the **Converter** tab. The Converter tab appears.
3. Click the **Key Converter** tab. The Key Converter tab appears.

Figure 44–3 Converter Tab, Key Converter Subtab



Converter	Description
No Converter	Do not use a Key Converter for this mapping.
Serialized Object Converter	See "Configuring a Serialized Object Converter" on page 32-18.
Type Conversion Converter	See "Configuring a Type Conversion Converter" on page 32-20.
Object Type Converter	See "Configuring an Object Type Converter" on page 32-22.

Configuring Value Converters

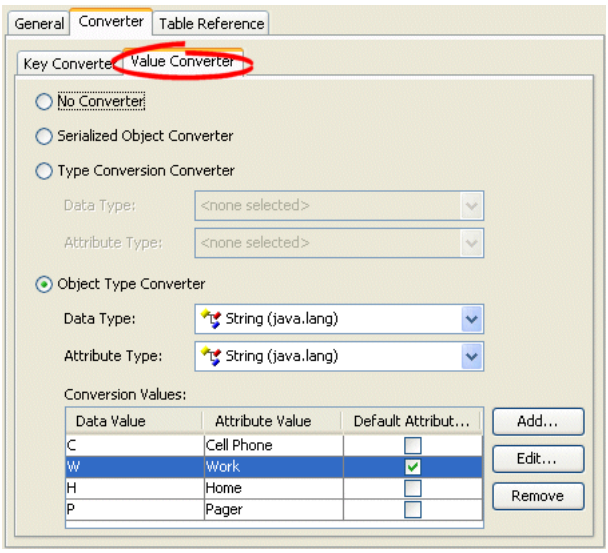
If the value's object value and database value are different types, use a converter. TopLink supports the following value converters:

- "Serialized Object Converter" on page 30-10
- "Type Conversion Converter" on page 30-11
- "Object Type Converter" on page 30-12

Using TopLink Workbench

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.
2. Click the **Converter** tab. The Converter tab appears.
3. Click the **Value Converter** tab. The Value Converter tab appears.

Figure 44–4 Converter Tab, Value Converter Subtab



Converter	Description
No Converter	Do not use a Value Converter for this mapping.
Serialized Object Converter	See "Configuring a Serialized Object Converter" on page 32-18.
Type Conversion Converter	See "Configuring a Type Conversion Converter" on page 32-20.
Object Type Converter	See "Configuring an Object Type Converter" on page 32-22.

45

Configuring a Relational Transformation Mapping

This chapter describes the various components that you must configure in order to use a relational transformation mapping.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "Transformation Mapping" on page 33-15

Relational Transformation Mapping Configuration Overview

Table 45–1 lists the configurable options for a relational transformation mapping.

Table 45–1 Configurable Options for Relational Transformation Mapping

Option	Type	TopLink Workbench	Java
"Configuring Attribute Transformer" on page 32-29	Basic	✔	✔
"Configuring Field Transformer Associations" on page 32-31	Basic	✔	✔
"Configuring Indirection" on page 32-3	Advanced	✔	✔
"Configuring Mutable Mappings" on page 32-33	Advanced	✔	✔
"Configuring Mapping Comments" on page 32-18	Advanced	✔	
"Configuring Method Accessing" on page 32-14	Advanced	✔	✔
"Configuring Read-Only Mappings" on page 32-2	Advanced	✔	✔

Part XI

Object-Relational Mappings

An object-relational mapping transforms certain object data member types to structured data source representations optimized for storage in specialized object-relational databases (such as Oracle Database). Object-relational mappings let you map an object model into an object-relational data model.

This part contains the following chapters:

- Chapter 46, "Understanding Object-Relational Mappings"

  This chapter describes each of the different TopLink object-relational mapping types and important object-relational mapping concepts.

- Chapter 47, "Configuring an Object-Relational Mapping"

  This chapter explains how to configure TopLink object-relational mapping options common to two or more object-relational mapping types.

- Chapter 49, "Configuring an Object-Relational Reference Mapping"

  This chapter explains how to configure a reference object-relational mapping.

- Chapter 52, "Configuring an Object-Relational Nested Table Mapping"

  This chapter explains how to configure a nested table object-relational mapping.

- Chapter 51, "Configuring an Object-Relational Object Array Mapping"

  This chapter explains how to configure an object array object-relational mapping.

- Chapter 48, "Configuring an Object-Relational Structure Mapping"

  This chapter explains how to configure a structure object-relational mapping.

- Chapter 50, "Configuring an Object-Relational Array Mapping"

  This chapter explains how to configure an array object-relational mapping.

46

Understanding Object-Relational Mappings

An object-relational mapping transforms certain object data member types to structured data source representations optimized for storage in specialized object-relational databases (such as Oracle Database). Object-relational mappings let you map an object model into an object-relational data model.

Do not confuse object-relational mappings with relational mappings (see "Understanding Relational Mappings" on page 33-1). A relational mapping transforms any object data member type to a corresponding relational database (SQL) data source representation in any supported relational database. Relational mappings let you map an object model into a relational data model. In general, you can use relational mappings with any supported relational database. You can only use object-relational mappings with specialized object-relational databases optimized to support object-relational data source representations.

This chapter describes:

- Object-Relational Mapping Types

Object-Relational Mapping Types

TopLink supports the object-relational mappings listed in Table 46–1.

These mappings allow for an object model to persist in an object-relational data model. Currently, TopLink Workbench does not support object-relational mappings–they must be defined in code or through amendment methods.

Table 46–1 TopLink Object-Relationship Mapping Types

Type	Description	Type	TopLink Workbench	Java
"Object-Relational Structure Mapping" on page 46-2	Map to object-relational aggregate structures (the `Struct` type in JDBC and the `OBJECT` type in Oracle Database*i*)	Advanced		✔
"Object-Relational Reference Mapping" on page 46-2	Map to object-relational references (the `Ref` type in JDBC and the `REF` type in Oracle Database)	Advanced		✔
"Object-Relational Array Mapping" on page 46-2	Map a collection of primitive data to object-relational array data types (the `Array` type in JDBC and the `VARRAY` type in Oracle Database).	Advanced		✔
"Object-Relational Object Array Mapping" on page 46-2	Map to object-relational array data types (the `Array` type in JDBC and the `VARRAY` type in Oracle Database).	Advanced		✔
"Object-Relational Nested Table Mapping" on page 46-3	Map to object-relational nested tables (the `Array` type in JDBC and the `NESTED TABLE` type in Oracle Database)	Advanced		✔

Object-Relational Structure Mapping

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 one of the following:

- A primitive data type
- Another structure
- Reference to another structure

TopLink maps nested structures with the `StructureMapping` class. The structure mapping supports null values and shared aggregates without requiring additional settings (because of the object-relational support of the database).

See "Configuring an Object-Relational Structure Mapping" on page 48-1 for more information.

Object-Relational Reference Mapping

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`. They represent an object reference in Java.

TopLink maps refs with the `ReferenceMapping` class. The reference mapping does not require foreign key information (because of the object-relational support of the database).

See "Configuring an Object-Relational Reference Mapping" on page 49-1 for more information.

Object-Relational Array Mapping

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.

TopLink maps arrays of primitive data types with the `ArrayMapping` class. An array mapping maps to object-relational array data types (the `Array` type in JDBC and the VARRAY type in Oracle Database). To map a collection of aggregate structures, use an object array mapping (see "Object-Relational Object Array Mapping" on page 46-2).

The object-relational database stores the arrays with their parent structure in the same table. To store information in a separate table from the parent structure's table, use a nested table mapping (see "Object-Relational Nested Table Mapping" on page 46-3).

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 (the VARRAY type).

See "Configuring an Object-Relational Array Mapping" on page 50-1 for more information.

Object-Relational Object Array Mapping

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.

TopLink maps arrays of structures with the `ObjectArrayMapping` class. An object array mapping defines a collection-aggregated relationship, in which the target objects share the same row as the source object.

You must associate this mapping to an attribute in the parent class.

See "Configuring an Object-Relational Object Array Mapping" on page 51-1 for more information.

Object-Relational Nested Table Mapping

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 multicolumn table (with a column for each attribute of the object type).

TopLink maps nested tables with the `NestedTableMapping` class. It represents a collection of object references in Java. Because of the object-relational support of the database, nested table mapping does not require foreign key information (as with a one-to-many mapping) or a relational table (as with a many-to-many mapping).

Typically, 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 the `VARRAY` types that are in-lined to the parent table. TopLink supports mapping a nested table of `REF` types only. TopLink does not support nested tables of basic or other structured data types–use array (see "Object-Relational Array Mapping" on page 46-2) or object array (see "Object-Relational Object Array Mapping" on page 46-2) mappings instead.

See "Configuring an Object-Relational Nested Table Mapping" on page 52-1 for more information.

47

Configuring an Object-Relational Mapping

This chapter describes how to configure an object-relational mapping.

Table 47–1 lists the types of object-relational mappings that you can configure and provides a cross-reference to the type-specific chapter that lists the configurable options supported by that type.

Table 47–1 Configuring Object-Relational Mappings

If you are creating...	See Also...
Object-Relational Structure Mapping	Chapter 48, "Configuring an Object-Relational Structure Mapping"
Object-Relational Reference Mapping	Chapter 49, "Configuring an Object-Relational Reference Mapping"
Object-Relational Array Mapping	Chapter 50, "Configuring an Object-Relational Array Mapping"
Object-Relational Object Array Mapping	Chapter 51, "Configuring an Object-Relational Object Array Mapping"
Object-Relational Nested Table Mapping	Chapter 52, "Configuring an Object-Relational Nested Table Mapping"

Table 47–2 lists the configurable options shared by two or more object-relational mapping types.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "Understanding Object-Relational Mappings" on page 46-1

Configuring Common Object-Relational Mapping Options

Table 47–2 lists the configurable options shared by two or more object-relational mapping types. In addition to the configurable options described here, you must also configure the options described for the specific Object-Relational Mapping Types, as shown in Table 47–1.

Table 47–2 Common Options for Object-Relational Mappings

Option	Type	TopLink Workbench	Java
"Configuring Reference Class" on page 47-2	Basic		✔
"Configuring Attribute Name" on page 47-2	Basic		✔
"Configuring Field Name" on page 47-3	Basic		✔

Table 47–2 (Cont.) Common Options for Object-Relational Mappings

Option	Type	TopLink Workbench	Java
"Configuring Structure Name" on page 47-4	Basic		✔
"Configuring Read-Only Mappings" on page 32-2	Basic		✔
"Configuring Method Accessing" on page 32-14	Advanced		✔
"Configuring Indirection" on page 32-3	Advanced		✔
"Configuring Container Policy" on page 32-26	Advanced		✔

Configuring Reference Class

When mapping an attribute that involves a relationship to another class, you must specify the reference class–the Java class to which the mapped attribute refers.

Table 47–3 summarizes which object-relational mappings support this option.

Table 47–3 Mapping Support for Reference Class

Mapping	Using TopLink Workbench	Using Java
Object-Relational Structure Mapping		✔
Object-Relational Reference Mapping		✔
Object-Relational Array Mapping		
Object-Relational Object Array Mapping		✔
Object-Relational Nested Table Mapping		✔

Using Java

Use `oracle.toplink.mappings.ForeignReferenceMapping` method `setReferenceClass` to specify the target class of the attribute being mapped.

Example 47–1 shows how to use this method with a `ReferenceMapping` that maps the `manager` attribute of the `Employee` class.

Example 47–1 Configuring Reference Class in Java

```
ReferenceMapping managerMapping = new new ReferenceMapping();
managerMapping.setReferenceClass("Employee.class");
managerMapping.setAttributeName("manager");
```

Configuring Attribute Name

All object-relational mappings map an attribute in a Java object to field in the database. The attribute name is the name of the attribute being mapped. The name is as specified in the reference class (see "Configuring Reference Class" on page 47-2).

Table 47–4 summarizes which object-relational mappings support this option.

Table 47–4 Mapping Support for Attribute Name

Mapping	Using TopLink Workbench	Using Java
Object-Relational Structure Mapping		✔

Table 47–4 (Cont.) Mapping Support for Attribute Name

Mapping	Using TopLink Workbench	Using Java
Object-Relational Reference Mapping		✔
Object-Relational Array Mapping		✔
Object-Relational Object Array Mapping		✔
Object-Relational Nested Table Mapping		✔

Using Java

Use `oracle.toplink.mappings.DatabaseMapping` method `setAttributeName` to specify the name of the attribute being mapped.

Example 47–2 shows how to use this method with a `ReferenceMapping` that maps the `manager` attribute of the `Employee` class.

Example 47–2 Configuring Attribute Name in Java

```
ReferenceMapping managerMapping = new new ReferenceMapping();
managerMapping.setReferenceClass("Employee.class");
managerMapping.setAttributeName("manager");
```

Configuring Field Name

All object-relational mappings require the name of database field to which their specified attribute is mapped. This field name can be the column name of a database table or the name of a field in an object type created on the database.

Table 47–5 summarizes which object-relational mappings support this option.

Table 47–5 Mapping Support for Field Name

Mapping	Using TopLink Workbench	Using Java
Object-Relational Structure Mapping		✔
Object-Relational Reference Mapping		✔
Object-Relational Array Mapping		✔
Object-Relational Object Array Mapping		✔
Object-Relational Nested Table Mapping		✔

Using Java

Use the object-relational mapping method `setFieldName` to specify the database field to which the attribute is mapped.

Example 47–3 shows how to use this method with an `ObjectArrayMapping` that maps the `Employee` class attribute `phone` to database field name PHONE_NUMBER.

Example 47–3 Configuring Field Name in Java

```
ObjectArrayMapping phonesMapping = new ObjectArrayMapping();
managerMapping.setReferenceClass("Employee.class");
phonesMapping.setAttributeName("phone");
phonesMapping.setFieldName("PHONE_NUMBER");
```

Configuring Structure Name

Certain object-relational mappings require the specification of the data type or structure name of the field being mapped. The structure name is the name of the array or table type that defines the field.

Table 47–6 summarizes which object-relational mappings support this option.

Table 47–6 Mapping Support for Structure Name

Mapping	Using TopLink Workbench	Using Java
Object-Relational Structure Mapping		
Object-Relational Reference Mapping		
Object-Relational Array Mapping		✔
Object-Relational Object Array Mapping		✔
Object-Relational Nested Table Mapping		✔

Using Java

Use the object-relational mapping method `setStructureName` to specify the structure of the attribute being mapped.

Example 47–4 shows how to use this method with an `ObjectArrayMapping` that maps the `Employee` class attribute `phones` to database field name PHONE_NUMBERS of type `PHONE_ARRAY_TYPE`.

Example 47–4 Configuring Structure Name in Java

```
ObjectArrayMapping phonesMapping = new ObjectArrayMapping();
managerMapping.setReferenceClass("Employee.class");
phonesMapping.setAttributeName("phones");
phonesMapping.setFieldName("PHONE_NUMBERS");
phonesMapping.setStructureName("PHONE_ARRAY_TYPE");
```

48

Configuring an Object-Relational Structure Mapping

This chapter describes the various components that you must configure in order to use an object-relational structure mapping.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "Object-Relational Structure Mapping" on page 46-2

Object-Relational Structure Mapping Configuration Overview

Table 48–1 lists the configurable options for an object-relational structure mapping.

Table 48–1 Configurable Options for Object-Relational Structure Mapping

Option	Type	TopLink Workbench	Java
"Configuring Reference Class" on page 47-2	Basic		✔
"Configuring Attribute Name" on page 47-2	Basic		✔
"Configuring Field Name" on page 47-3	Basic		✔
"Configuring Read-Only Mappings" on page 32-2	Advanced		✔
"Configuring Method Accessing" on page 32-14	Advanced		✔

49

Configuring an Object-Relational Reference Mapping

This chapter describes the various components that you must configure in order to use an object-relational reference mapping.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "Object-Relational Reference Mapping" on page 46-2

Object-Relational Reference Mapping Configuration Overview

Table 49–1 lists the configurable options for an object-relational reference mapping.

Table 49–1 Configurable Options for Object-Relational Reference Mapping

Option	Type	TopLink Workbench	Java
"Configuring Reference Class" on page 47-2	Basic		✔
"Configuring Attribute Name" on page 47-2	Basic		✔
"Configuring Field Name" on page 47-3	Basic		✔
"Configuring Read-Only Mappings" on page 32-2	Advanced		✔
"Configuring Method Accessing" on page 32-14	Advanced		✔
"Configuring Private or Independent Relationships" on page 32-16	Advanced		✔
"Configuring Indirection" on page 32-3	Advanced		✔

50

Configuring an Object-Relational Array Mapping

This chapter describes the various components that you must configure in order to use an object-relational array mapping.

> **Note:** To map a collection of aggregate structures, use an object-relational object array mapping (see "Object-Relational Object Array Mapping" on page 46-2). To store information in a separate table from the parent structure's table, use an object-relational nested table mapping (see "Object-Relational Nested Table Mapping" on page 46-3).

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "Object-Relational Array Mapping" on page 46-2

Object-Relational Array Mapping Configuration Overview

Table 50–1 lists the configurable options for an object-relational array mapping.

Table 50–1 Configurable Options for Object-Relational Array Mapping

Option	Type	TopLink Workbench	Java
"Configuring Attribute Name" on page 47-2	Basic		✔
"Configuring Field Name" on page 47-3	Basic		✔
"Configuring Structure Name" on page 47-4	Basic		✔
"Configuring Read-Only Mappings" on page 32-2	Basic		✔
"Configuring Method Accessing" on page 32-14	Advanced		✔
"Configuring a Serialized Object Converter" on page 32-18	Advanced		✔
"Configuring a Type Conversion Converter" on page 32-20	Advanced		✔
"Configuring an Object Type Converter" on page 32-22	Advanced		✔
"Configuring Container Policy" on page 32-26	Advanced		✔

51

Configuring an Object-Relational Object Array Mapping

This chapter describes the various components that you must configure in order to use an object-relational object array mapping.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "Object-Relational Object Array Mapping" on page 46-2

Object-Relational Object Array Mapping Configuration Overview

Table 51–1 lists the configurable options for an object-relational object array mapping.

Table 51–1 Configurable Options for Object-Relational Object Array Mapping

Option	Type	TopLink Workbench	Java
"Configuring Reference Class" on page 47-2	Basic		✔
"Configuring Attribute Name" on page 47-2	Basic		✔
"Configuring Field Name" on page 47-3	Basic		✔
"Configuring Structure Name" on page 47-4	Basic		✔
"Configuring Read-Only Mappings" on page 32-2	Basic		✔
"Configuring Method Accessing" on page 32-14	Advanced		✔
"Configuring Container Policy" on page 32-26	Advanced		✔

52

Configuring an Object-Relational Nested Table Mapping

This chapter describes the various components that you must configure in order to use an object-relational nested table mapping.

> **Note:** For an equivalent mapping for basic or other structured data types, use object-relational array (see "Object-Relational Array Mapping" on page 46-2) or object array (see "Object-Relational Object Array Mapping" on page 46-2) mappings.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "Object-Relational Nested Table Mapping" on page 46-3

Object-Relational Nested Table Mapping Configuration Overview

Table 52–1 lists the configurable options for an object-relational nested table mapping.

Table 52–1 Configurable Options for Object-Relational Nested Table Mapping

Option	Type	TopLink Workbench	Java
"Configuring Reference Class" on page 47-2	Basic		✔
"Configuring Attribute Name" on page 47-2	Basic		✔
"Configuring Field Name" on page 47-3	Basic		✔
"Configuring Structure Name" on page 47-4	Basic		✔
"Configuring Read-Only Mappings" on page 32-2	Basic		✔
"Configuring Method Accessing" on page 32-14	Advanced		✔
"Configuring Indirection" on page 32-3	Advanced		✔
"Configuring Indirection" on page 32-3	Advanced		✔

Part XII

EIS Mappings

TopLink enterprise information system (EIS) mappings provide support for accessing legacy data sources and enterprise applications through J2EE Connector architecture (J2C) adapter. TopLink EIS mappings use the J2C Common Client Interface (CCI) to access the EIS through its resource adapter. This provides the ability to directly map from an existing Java object model to any transactional data source, such as mainframes with flat file/hierarchical data. An EIS mapping transforms object data members to the EIS record format defined by the object's descriptor.

This part contains the following chapters:

- Chapter 53, "Understanding EIS Mappings"

  This chapter describes each of the different TopLink EIS mapping types and important EIS mapping concepts.

- Chapter 54, "Configuring an EIS Mapping"

  This chapter explains how to configure TopLink EIS mapping options common to two or more EIS mapping types.

- Chapter 55, "Configuring an EIS Direct Mapping"

  This chapter explains how to configure a direct EIS mapping.

- Chapter 56, "Configuring an EIS Composite Direct Collection Mapping"

  This chapter explains how to configure a direct collection EIS mapping.

- Chapter 57, "Configuring an EIS Composite Object Mapping"

  This chapter explains how to configure a composite object EIS mapping.

- Chapter 58, "Configuring an EIS Composite Collection Mapping"

  This chapter explains how to configure a composite collection EIS mapping.

- Chapter 61, "Configuring an EIS Transformation Mapping"

  This chapter explains how to configure a transformation EIS mapping.

- Chapter 59, "Configuring an EIS One-to-One Mapping"

  This chapter explains how to configure a one-to-one EIS mapping.

- Chapter 60, "Configuring an EIS One-to-Many Mapping"

  This chapter explains how to configure a one-to-many EIS mapping.

53

Understanding EIS Mappings

TopLink enterprise information system (EIS) mappings provide support for accessing legacy data sources and enterprise applications through J2EE Connector architecture (J2C) adapter. TopLink EIS mappings use the J2C Common Client Interface (CCI) to access the EIS through its resource adapter. This provides the ability to directly map from an existing Java object model to any transactional data source, such as mainframes with flat file/hierarchical data.

An EIS mapping transforms object data members to the EIS record format defined by the object's descriptor.

This chapter describes the following:

- EIS Mapping Types
- EIS Mapping Concepts

EIS Mapping Types

TopLink supports the EIS mappings listed in Table 53–1.

Table 53–1 TopLink Object EIS Mapping Types

Mapping Type	Description	Type	TopLink Workbench	Java
"EIS Direct Mapping" on page 53-5	Map a simple object attribute directly to an EIS record.	Basic	✔	✔
"EIS Composite Direct Collection Mapping" on page 53-6	Map a collection of Java attributes directly to an EIS record.	Basic	✔	✔
"EIS Composite Object Mapping" on page 53-7	Map a Java object to an EIS record in a privately owned one-to-one relationship. Composite object mappings represent a relationship between two classes.	Advanced	✔	✔
"EIS Composite Collection Mapping" on page 53-7	Map a `Map` or `Collection` of Java objects to an EIS record in a privately owned one-to-many relationship.	Advanced	✔	✔

Table 53–1 (Cont.) TopLink Object EIS Mapping Types

Mapping Type	Description	Type	TopLink Workbench	Java
"EIS One-to-One Mapping" on page 53-8	Define a reference mapping that represents the relationship between a single source object and a single mapped persistent Java object.	Basic	✔	✔
"EIS One-to-Many Mapping" on page 53-12	Define a reference mapping that represents the relationship between a single source object and a collection of mapped persistent Java objects.	Basic	✔	✔
"EIS Transformation Mapping" on page 53-17	Create custom mappings where one or more EIS record fields can be used to create the object to be stored in a Java class's attribute.	Advanced	✔	✔

EIS Mapping Concepts

This section describes concepts unique to TopLink EIS mappings, including the following:

- EIS Record Type
- XPath Support
- xsd:list and xsd:union Support
- jaxb:class Support
- Typesafe Enumeration Support
- Composite and Reference EIS Mappings
- EIS Mapping Architecture

EIS Record Type

TopLink supports the following J2C EIS record types:

- Indexed Records
- Mapped Records
- XML Records

You configure the record type at the EIS descriptor level (see "Configuring Record Format" on page 28-5). EIS mappings use the record type of their EIS descriptor to determine how to map Java attributes. That is, you use the same EIS mapping regardless of the record type, with which you configure an EIS descriptor.

Note: Not all J2C adapters support all record types. Consult your J2C adapter documentation for details.

Indexed Records

The `javax.resource.cci.IndexedRecord` represents an ordered collection of record elements based on the `java.util.List` interface.

The TopLink runtime maps Java objects to indexed record elements or subrecords of an indexed record depending on the type of EIS mapping you use (see "Composite and Reference EIS Mappings" on page 53-4).

Mapped Records

The `javax.resource.cci.MappedRecord` represents a key-value map-based collection of record elements based on the `java.util.Map` interface.

The TopLink runtime maps Java objects to mapped record elements or subrecords of a mapped record depending on the type of EIS mapping you use (see "Composite and Reference EIS Mappings" on page 53-4).

XML Records

An XML record represents a `javax.resource.cci.Record` as an XML schema (XSD)-based XML document. Not all J2C adapters support XML records.

The TopLink runtime maps Java objects to XML documents according to your XSD and the behavior defined for XML mappings.

For more information, see "Understanding XML Mappings" on page 62-1.

XPath Support

When using XML records, TopLink EIS mappings use XPath statements to efficiently map the attributes of a Java object to locations in an XML record. For more information about using XPath with XML mappings, see "Mappings and XPath" on page 30-15.

xsd:list and xsd:union Support

When using XML records, you can use EIS direct (see "EIS Direct Mapping") and composite direct collection (see "EIS Composite Direct Collection Mapping") mappings to map to `xsd:list` and `xsd:union` types in an XML record.

For more information, see "Mappings and xsd:list and xsd:union Types" on page 30-17.

jaxb:class Support

When using XML records, you can configure an EIS composite object mapping (see "EIS Composite Object Mapping" on page 53-7) to accommodate `jaxb:class` customizations with the following XSD structures:

- `all`
- `sequence`
- `choice`
- `group`

For more information, see "Mappings and the jaxb:class Customization" on page 30-20.

Typesafe Enumeration Support

You can map a Java attribute to a `typesafe` enumeration using the `JAXBTypesafeEnumConverter` with an `EISDirectMapping` or `EISCompositeDirectCollectionMapping` with XML records.

For more information, see "Mappings and JAXB Typesafe Enumerations" on page 30-24.

Composite and Reference EIS Mappings

TopLink supports composite and reference EIS mappings. Although there is a source and target object in both mapping types, the TopLink runtime handles interactions with each differently. This section explains how.

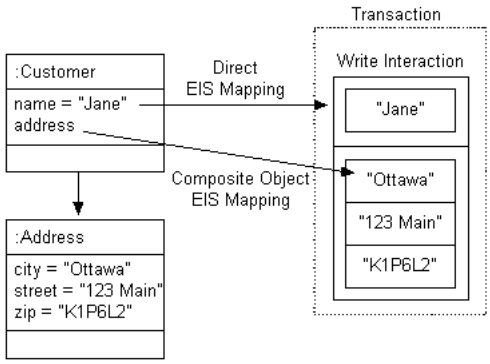
Composite EIS Mappings

In a composite EIS mapping ("EIS Composite Direct Collection Mapping", "EIS Composite Object Mapping", and "EIS Composite Collection Mapping"), the source object contains (owns) the target object.

TopLink puts the attributes of the target (owned) object (or the owned collection of objects) into the source (owning) object's record as a subrecord. The target object needs not be a root object type (see "Configuring an EIS Descriptor as a Root or Composite Type" on page 28-8): it needs not have interactions defined for it.

Figure 53–1 illustrates a read interaction on an instance of the `Customer` class using indexed records. For the composite object EIS mapping defined for the `address` attribute, TopLink creates an `Address` subrecord in the `Customer` record.

Figure 53–1 EIS Composite Mappings



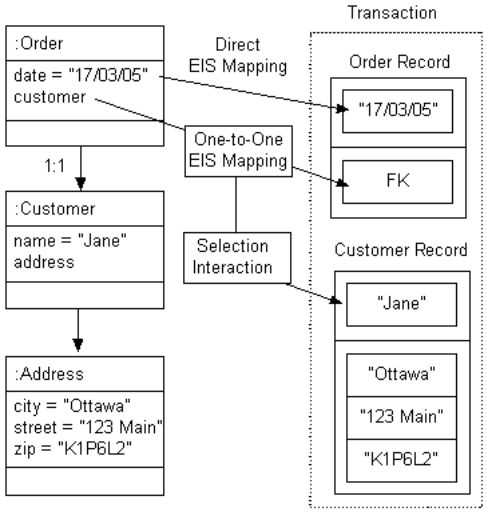
Reference EIS Mappings

In a reference EIS mapping ("EIS One-to-One Mapping" and "EIS One-to-Many Mapping"), the source object contains only a foreign key (pointer) to the target object or, alternatively, the target object contains a foreign key to the source object (key on target).

TopLink puts the foreign key of the target object into the source object's record as a simple value. When an interaction is executed on the source object, TopLink uses the selection interaction that you define on its descriptor to retrieve the appropriate target object instance and creates a record for it in the source object's transaction. By default, the selection interaction is the target object's read interaction. If the read interaction is not sufficient, you can define a separate selection interaction (see "Configuring Selection Interaction" on page 54-3). Because both the source and target object use interactions, they must both be of a root object type (see "Configuring an EIS Descriptor as a Root or Composite Type" on page 28-8).

Figure 53–2 illustrates a read interaction on an instance of the `Order` class using indexed records. For the one-to-one EIS mapping defined for the `customer` attribute, TopLink puts the target `Customer` object's foreign key into the `Order` record as a simple value. TopLink then uses the selection interaction you configure on the `Order` descriptor to retrieve the appropriate instance of `Customer` and creates a record for it in the `Order` object's transaction.

Figure 53–2 EIS Reference Mappings

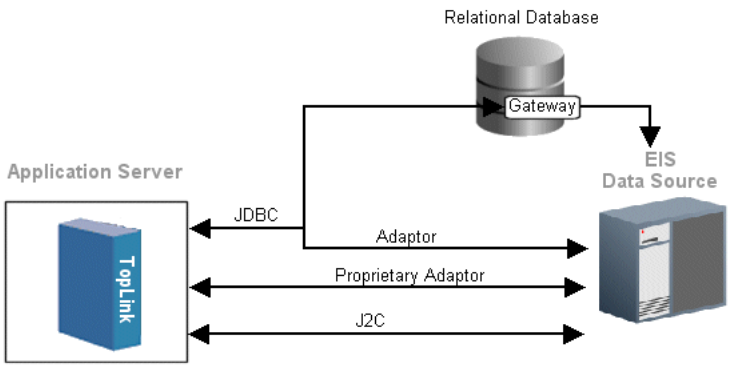


EIS Mapping Architecture

Figure 53–3 illustrates the following possible TopLink EIS mapping architectures:

- JDBC database gateway (such as Oracle Database 10*g*)
- JDBC adapter
- Proprietary adapter (such as Oracle Interconnect)
- J2C

Figure 53–3 Possible EIS Mapping Architectures



The best solution may vary, depending on your specific EIS and infrastructure.

EIS Direct Mapping

An EIS direct mapping maps a simple object attribute directly to an EIS record according to its descriptor's record type as shown in Table 53–2.

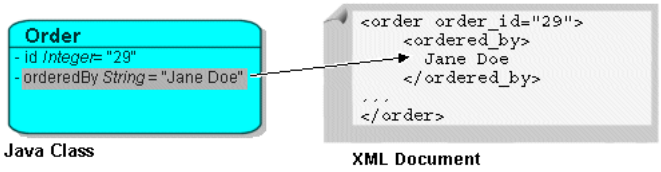
Table 53–2 EIS Direct Mapping by EIS Record Type

EIS Record Type	Mapping Behavior
Indexed	Maps directly to a field in the indexed record.
Mapped	Maps directly to a field in the mapped record.
XML	Maps directly to an attribute or text node in the XML record[1].

[1] See also "XML Direct Mapping" on page 62-5.

Figure 53–4 illustrates a direct EIS mapping between `Order` class attribute `orderedBy` and XML record attribute `ordered_by` within the `order` element.

Figure 53–4 EIS Direct Mappings



See Chapter 55, "Configuring an EIS Direct Mapping" for more information.

EIS Composite Direct Collection Mapping

An EIS composite direct collection mapping maps a collection of Java attributes directly to an EIS record according to its descriptor's record type, as shown in Table 53–3.

Table 53–3 EIS Composite Direct Collection Mapping by EIS Record Type

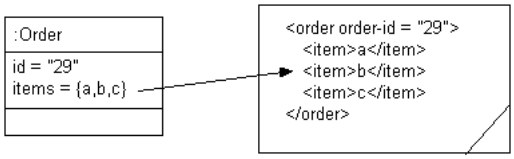
EIS Record Type	Mapping Behavior
Indexed	Maps directly to a subrecord in the indexed record[1].
Mapped	Maps directly to a subrecord in the mapped record[1].
XML	Maps directly to an attribute or text node in the XML record[2].

[1] See also "Composite EIS Mappings" on page 53-4.

[2] See also "XML Composite Direct Collection Mapping" on page 62-14.

Figure 53–5 illustrates a composite direct collection mapping between `Order` class attribute `items` and an XML record. The `Order` attribute `items` is a collection type (such as `Vector`). It is mapped to an XML record composed of an `order` element that contains a sequence of `item` elements.

Figure 53–5 EIS Composite Direct Collection Mapping



See Chapter 56, "Configuring an EIS Composite Direct Collection Mapping" for more information.

EIS Composite Object Mapping

An EIS composite object mapping maps a Java object to a privately owned one-to-one relationship in an EIS record according to its descriptor's record type, as shown in Table 53–4.

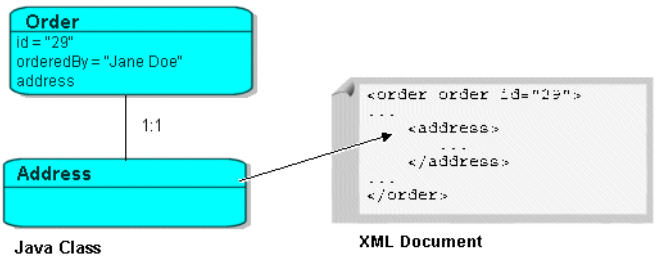
Table 53–4 EIS Composite Object Mapping by EIS Record Type

EIS Record Type	Mapping Behavior
Indexed	Maps directly to a subrecord in the indexed record[1].
Mapped	Maps directly to a subrecord in the mapped record[1].
XML	Maps directly to an attribute or text node in the XML record[2].

[1] See also "Composite EIS Mappings" on page 53-4.
[2] See also "XML Composite Object Mapping" on page 62-21.

Figure 53–6 illustrates a composite object EIS mapping between `Order` class attribute `address` and an XML record. `Order` attribute `address` is mapped to an XML record composed of an `order` element that contains an `address` element.

Figure 53–6 EIS Composite Object Mappings



You can use an EIS composite object mapping with a change policy (see "Configuring Change Policy" on page 25-73.

See Chapter 57, "Configuring an EIS Composite Object Mapping" for more information.

EIS Composite Collection Mapping

An EIS composite collection mapping maps a collection of Java objects to a privately owned one-to-many relationship in an EIS record according to its descriptor's record type as shown in Table 53–5. Composite collection mappings can reference any class that has a TopLink descriptor.

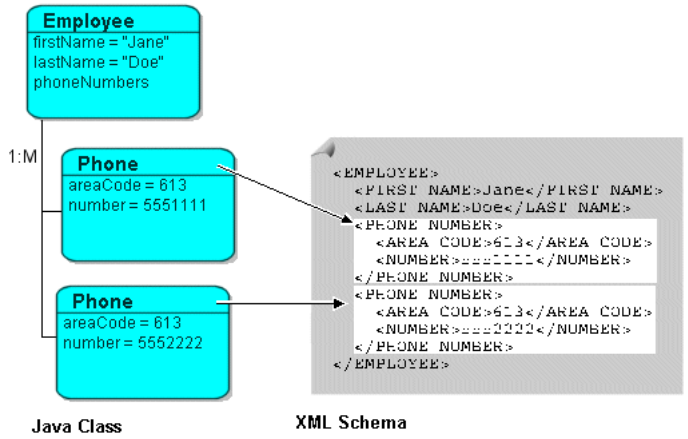
Table 53–5 EIS Composite Collection Mapping by EIS Record Type

EIS Record Type	Mapping Behavior
Indexed	Maps directly to a subrecord in the indexed record[1].
Mapped	Maps directly to a subrecord in the mapped record[1].
XML	Maps directly to an attribute or text node in the XML record[2].

[1] See also "Composite EIS Mappings" on page 53-4.
[2] See also "XML Composite Collection Mapping" on page 62-25.

Figure 53–7 illustrates a composite collection EIS mapping between `Phone` class attribute `phoneNumbers` and an XML record. `Employee` attribute `phoneNumbers` is mapped to an XML record composed of an `EMPLOYEE` element that contains a sequence of `PHONE_NUMBER` elements.

Figure 53–7 EIS Composite Collection Mappings



See Chapter 58, "Configuring an EIS Composite Collection Mapping" for more information.

EIS One-to-One Mapping

An EIS one-to-one mapping is a reference mapping that represents the relationship between a single source and target object. The source object usually contains a foreign key (pointer) to the target object (key on source). Alternatively, the target object may contain a foreign key to the source object (key on target). Because both the source and target object use interactions, they must both be of a root object type (see "Configuring an EIS Descriptor as a Root or Composite Type" on page 28-8)

Table 53–6 summarizes the behavior of this mapping depending on the EIS record type you are using.

Table 53–6 EIS One-to-One Mapping by EIS Record Type

EIS Record Type	Mapping Behavior
Indexed	A new indexed record is created for the target object[1]: ■ With the Key on Source use case, the foreign key(s) is added to the record for the source object. ■ With the Key on Target use case, the foreign key(s) is added to the record for the target object
Mapped	A new mapped record is created for the target object[1]: ■ With the Key on Source use case, the foreign key(s) is added to the record for the source object. ■ With the Key on Target use case, the foreign key(s) is added to the record for the target object

Table 53–6 (Cont.) EIS One-to-One Mapping by EIS Record Type

EIS Record Type	Mapping Behavior
XML	.A new XML record is created for the target object: ▪ With the Key on Source use case, the foreign key(s) is added to the record for the source object. ▪ With the Key on Target use case, the foreign key(s) is added to the record for the target object

[1] See also "Reference EIS Mappings" on page 53-4.

This section describes the following:

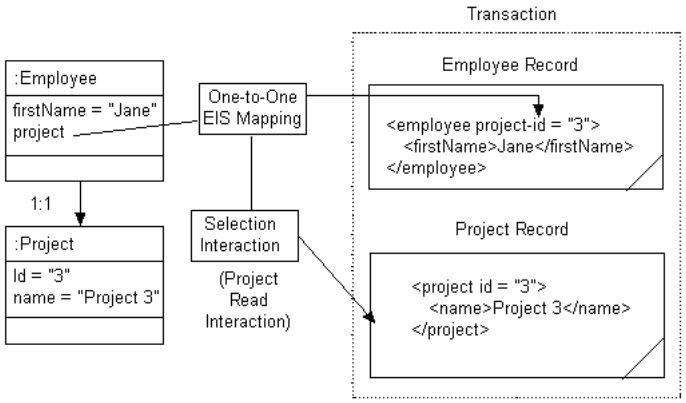
- EIS One-to-One Mappings With Key on Source
- EIS One-to-One Mappings With Key on Target

See Chapter 59, "Configuring an EIS One-to-One Mapping" for more information.

EIS One-to-One Mappings With Key on Source

Figure 53–8 illustrates a EIS one-to-one mapping between the `Employee` class attribute `project` and the `Project` class using XML records in a **key on source** design.

Figure 53–8 EIS One-to-One Mapping with Key on Source



When a read interaction is executed on the `Employee` object, TopLink puts the target `Project` object's primary key into the `Employee` record as a simple value. TopLink then uses the selection interaction you configure on the `Employee` descriptor to retrieve the appropriate instance of `Project` and creates a record for it in the `Employee` object's transaction. In this example, you can designate the `Project` class's read interaction as the selection interaction.

The general procedure for creating and configuring this mapping is as follows:

1. Create a one-to-one EIS mapping on `Employee` attribute `project`.
2. Configure the reference descriptor as `Project` (see "Configuring Reference Descriptors" on page 54-2).
3. Configure the source and target foreign keys (see "Configuring Foreign Key Pairs" on page 59-1).

   In this example:

- Source XML Field: `@project-id`
- Target XML Field: `@id`

4. Configure the selection interaction (see "Configuring Selection Interaction" on page 54-3).

   In this example, you can designate the `Project` class's read interaction as the selection interaction.

Given the XSD shown in Example 53–1, you can configure an EIS one-to-one mapping with key on source, as Example 53–2 shows. In this case, the source object contains a foreign key reference to the target object. In the following example, the source object is `Employee` and the target object is `Project`. Here, the `Employee` object has a `Project` that is referenced using the project's id.

Example 53–1 XML Schema for EIS One-to-One Mapping with Key on Source

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
    <xsd:element name="employee" type="employee-type"/>
    <xsd:element name="project" type="project-type"/>
    <xsd:complexType name="employee-type">
        <xsd:sequence>
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="project">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="project-id" type="xsd:integer"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="project-type">
        <xsd:sequence>
            <xsd:element name="id" type="xsd:integer"/>
            <xsd:element name="leader" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

Example 53–2 EIS One-to-One Mapping with Key On Source

```
// Employee descriptor
EISDescriptor descriptor = new EISDescriptor();
descriptor.setJavaClass(Employee.class);
descriptor.setDataTypeName("employee");
descriptor.setPrimaryKeyFieldName("name/text()");

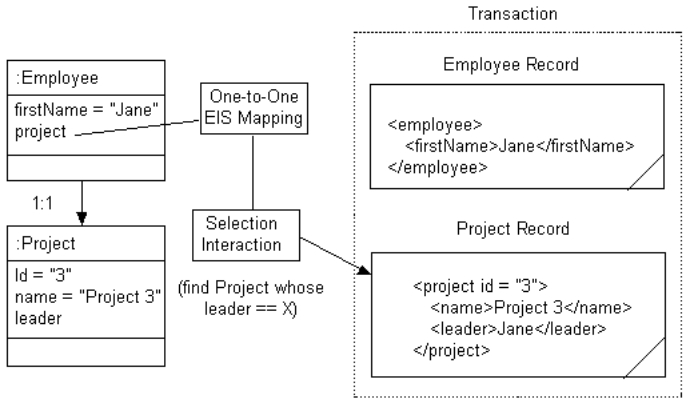
EISOneToOneMapping projectMapping = new EISOneToOneMapping();
projectMapping.setReferenceClass(Project.class);
projectMapping.setAttributeName("project");
projectMapping.dontUseIndirection();
projectMapping.addForeignKeyFieldName("project/project-id/text()", "id/text()");
```

EIS One-to-One Mappings With Key on Target

Figure 53–9 illustrates EIS one-to-one mapping between the `Employee` class attribute `project` and the `Project` class using XML records in a **key on target** design. You still configure a one-to-one EIS mapping between `Employee` and `Project`, but in this

design, the `Project` attribute `leader` contains the foreign key of the `Employee` object.

Figure 53–9 EIS One-to-One Mapping with Key on Target



When a read interaction is executed on the `Employee` object, TopLink uses the selection interaction you configure on the `Employee` descriptor to retrieve the appropriate instance of `Project` and creates a record for it in the `Employee` object's transaction. In this example, the `Project` class's read interaction is unlikely to be sufficient: it is likely implemented to read based on `Project` attribute `Id`, not on `leader`. If this is the case, you must define a separate selection interaction on the `Employee` descriptor that does the following: finds the `Project`, whose `leader` equals X, where X is the value of `Employee` attribute `firstName`.

Note that in this configuration, `Project` attribute `leader` is not persisted. If you want this attribute persisted, you must configure a one-to-one EIS mapping from it to `Employee` attribute `firstName`.

The general procedure for creating and configuring this mapping is as follows:

1. Create a one-to-one EIS mapping on `Employee` attribute `project`.
2. Configure the reference descriptor as `Project` (see "Configuring Reference Descriptors" on page 54-2).
3. Configure the source and target foreign keys (see "Configuring Foreign Key Pairs" on page 59-1).

   In this example:

   - Source XML Field: `firstName/text()`
   - Target XML Field: `leader/text()`
4. Configure the selection interaction (see "Configuring Selection Interaction" on page 54-3).

   In this example, you must define a separate selection interaction on the `Employee` descriptor.

Given the XSD shown in Example 53–3, you can configure an EIS one-to-one mapping with key on target, as Example 53–4 shows. In this case, the target object contains a foreign key reference to the source object. In the following example, the source object is `Employee`, and the target object is `Project`. Here, a `Project` references its `leader` using the employee's name.

Example 53–3 XML Schema for EIS One-to-One Mapping with Key on Target

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
    <xsd:element name="employee" type="employee-type"/>
    <xsd:element name="project" type="project-type"/>
    <xsd:complexType name="employee-type">
        <xsd:sequence>
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="project">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="project-id" type="xsd:integer"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="project-type">
        <xsd:sequence>
            <xsd:element name="id" type="xsd:integer"/>
            <xsd:element name="leader" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
    </xsd:schema>
```

Example 53–4 EIS One-to-One Mapping with Key on Target

```
// Project descriptor
EISDescriptor descriptor = new EISDescriptor();
descriptor.setJavaClass(Project.class);
descriptor.setDataTypeName("project");
descriptor.setPrimaryKeyFieldName("id/text()");

EISOneToOneMapping leaderMapping = new EISOneToOneMapping();
leaderMapping.setReferenceClass(Employee.class);
leaderMapping.setAttributeName("leader");
leaderMapping.dontUseIndirection();
leaderMapping.addForeignKeyFieldName("leader/text()", "name/text()");
```

EIS One-to-Many Mapping

An EIS one-to-many mapping is a reference mapping that represents the relationship between a single source object and a collection of target objects. The source object usually contains a foreign key (pointer) to the target objects (key on source); alternatively, the target objects may contain a foreign key to the source object (key on target). Because both the source and target objects use interactions, they must all be of a root object type (see "Configuring an EIS Descriptor as a Root or Composite Type" on page 28-8).

Table 53–7 summarizes the behavior of this mapping depending on the EIS record type you are using.

Table 53–7 EIS One-to-Many Mapping by EIS Record Type

EIS Record Type	Mapping Behavior
Indexed	A new indexed record is created for each target object[1]: ■ With the Key on Source use case, the foreign key(s) is added to the record for the source object for each target object. ■ With the Key on Target use case, the foreign key(s) is added to the record for the target object
Mapped	A new mapped record is created for each target object[1]: ■ With the Key on Source use case, the foreign key(s) is added to the record for the source object. ■ With the Key on Target use case, the foreign key(s) is added to the record for the target object
XML	.A new XML record is created for each target object: ■ With the Key on Source use case, the foreign key(s) is added to the record for the source object for each target object. ■ With the Key on Target use case, the foreign key(s) is added to the record for the target object

[1] See also "Reference EIS Mappings" on page 53-4.

This section describes the following:

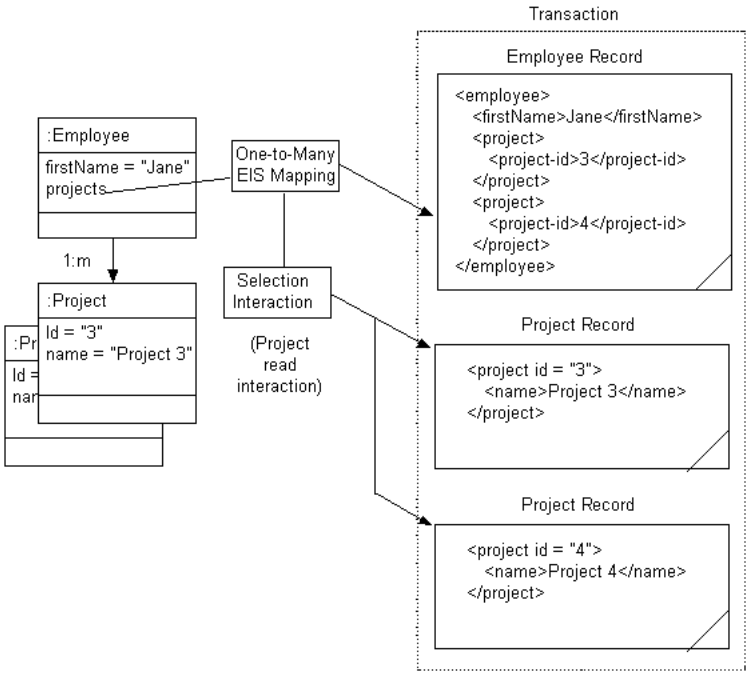
- EIS One-to-Many Mappings With Key on Source
- EIS One-to-Many Mappings With Key on Target

See Chapter 60, "Configuring an EIS One-to-Many Mapping" for more information.

EIS One-to-Many Mappings With Key on Source

Figure 53–10 illustrates an EIS one-to-many mapping between the `Employee` class attribute `projects` and multiple `Project` class instances using XML records in a key on source design.

Figure 53–10 EIS One-to-Many Mapping with Key on Source



When a read interaction is executed on the `Employee` object, TopLink puts each target `Project` object's foreign key into the `Employee` record as a subelement. If you specify only one pair of source and target XML fields, by default, the foreign keys are not grouped in the `Employee` record. If you specify more than one pair of source and target XML fields, you must choose a grouping element (see "Configuring Reference Descriptors" on page 54-2). Figure 53–10 shows an `Employee` record with grouping element `Project`. TopLink then uses the selection interaction you configure on the `Employee` descriptor to retrieve the appropriate instances of `Project` and creates a record for each in the `Employee` object's transaction. In this example, you can designate the `Project` class's read interaction as the selection interaction.

The general procedure for creating and configuring this mapping is as follows:

1. Create a one-to-many EIS mapping on `Employee` attribute `project`.
2. Configure the reference descriptor as `Project` (see "Configuring Reference Descriptors" on page 54-2).
3. Configure the source and target foreign keys (see "Configuring Foreign Key Pairs" on page 60-1).

   In this example:

   - Source XML Field: `PROJECT`
   - Target XML Field: `@ID`
4. Configure the selection interaction (see "Configuring Selection Interaction" on page 54-3).

   In this example, you can designate the `Project` class's read interaction as the selection interaction.

Given the XSD shown in Example 53–3, you can configure an EIS one-to-many mapping with key on source, as Example 53–4 shows. In this case, the source object contains a foreign key reference to the target object. In the following example, the

source object is `Employee`, and the target object is `Project`. Here, the `Employee` object has one or more `Project` instances that are referenced by `Project` id.

Example 53–5 XML Schema for EIS One-to-Many Mapping with Key on Source

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
    <xsd:element name="employee" type="employee-type"/>
    <xsd:element name="project" type="project-type"/>
    <xsd:complexType name="employee-type">
        <xsd:sequence>
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="projects">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="project-id"
                            type="xsd:integer" maxOccurs="unbounded"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="project-type">
        <xsd:sequence>
            <xsd:element name="id" type="xsd:integer"/>
            <xsd:element name="leader" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

Example 53–6 EIS One-to-Many Mapping with Key on Source

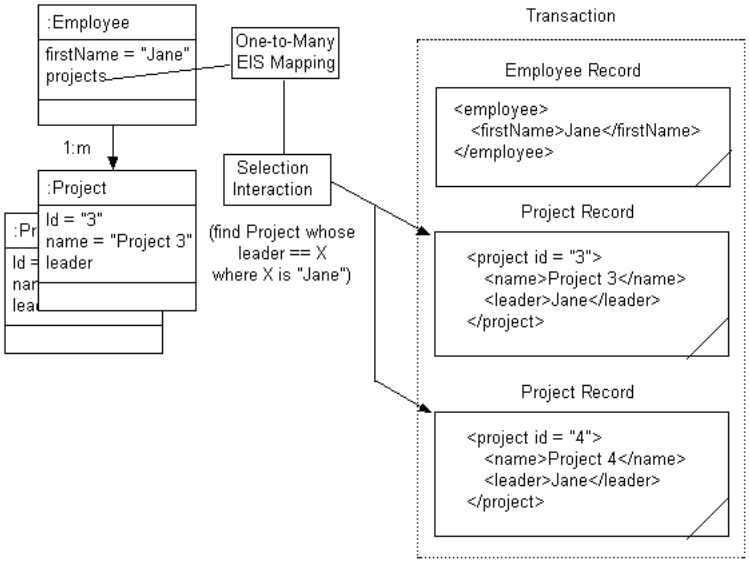
```
// Employee descriptor
EISDescriptor descriptor = new EISDescriptor();
descriptor.setJavaClass(Employee.class);
descriptor.setDataTypeName("employee");
descriptor.setPrimaryKeyFieldName("name/text()");

EISOneToManyMapping projectMapping = new EISOneToManyMapping();
projectMapping.setReferenceClass(Project.class);
projectMapping.setAttributeName("projects");
projectMapping.setForeignKeyGroupingElement("projects");
projectMapping.setIsForeignKeyRelationship(true);
projectMapping.dontUseIndirection();
projectMapping.addForeignKeyFieldName("project-id/text()", "id/text()");
```

EIS One-to-Many Mappings With Key on Target

Figure 53–9 illustrates an EIS one-to-many mapping between the `Employee` class attribute `projects` and multiple `Project` class instances using XML records in a key on target design. You still configure a one-to-one EIS mapping between `Employee` and `Project` but in this design, the `Project` attribute `leader` contains the foreign key of the `Employee` object.

Figure 53–11 EIS One-to-Many Mapping with Key on Target



When a read interaction is executed on the `Employee` object, TopLink uses the selection interaction you configure on the `Employee` descriptor to retrieve the appropriate instances of `Project` and creates a record for each in the `Employee` object's transaction. In this example, the `Project` class's read interaction is unlikely to be sufficient: it is likely implemented to read based on `Project` attribute `Id`, not on `leader`. If this is the case, you must define a separate selection interaction on the `Employee` descriptor that does the following: finds the `Project`, whose `leader` equals X, where X is "Jane".

Note that in this configuration, `Project` attribute `leader` is not persisted. If you want this attribute persisted, you must configure a one-to-one EIS mapping from it to `Employee` attribute `firstName`.

The general procedure for creating and configuring this mapping is as follows:

1. Create a one-to-one EIS mapping on `Employee` attribute `project`.
2. Configure the reference descriptor as `Project` (see "Configuring Reference Descriptors" on page 54-2).
3. Configure the source and target foreign keys (see "Configuring Foreign Key Pairs" on page 59-1).

   In this example, you select **Foreign Keys Located On Source** and specify one pair of source and target XML fields:

   - Source XML Field:
   - Target XML Field:
4. Configure the selection interaction (see "Configuring Selection Interaction" on page 54-3).

   In this example, you must define a separate selection interaction on the `Employee` descriptor.

Given the XSD shown in Example 53–3, you can configure an EIS one-to-many mapping with key on target as Example 53–4 shows. In this case, the target object contains a foreign key reference to the source object. In the following example, the

source object is `Employee`, and the target object is `Project`. Here, each `Project` references its `leader` using the employee's name.

Example 53–7 XML Schema for EIS One-to-Many Mapping with Key on Target

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
    <xsd:element name="employee" type="employee-type"/>
    <xsd:element name="project" type="project-type"/>
    <xsd:complexType name="employee-type">
        <xsd:sequence>
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="projects">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="project-id"
                            type="xsd:integer" maxOccurs="unbounded"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="project-type">
        <xsd:sequence>
            <xsd:element name="id" type="xsd:integer"/>
            <xsd:element name="leader" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

Example 53–8 EIS One-to-Many Mapping with Key on Target

```
// Project descriptor
EISDescriptor descriptor = new EISDescriptor();
descriptor.setJavaClass(Project.class);
descriptor.setDataTypeName("project");
descriptor.setPrimaryKeyFieldName("id/text()");

EISOneToManyMapping leaderMapping = new EISOneToOneMapping();
leaderMapping.setReferenceClass(Employee.class);
leaderMapping.setAttributeName("leader");
leaderMapping.dontUseIndirection();
leaderMapping.addForeignKeyFieldName("leader/text()", "name/text()");
```

EIS Transformation Mapping

A transformation EIS mapping lets you create a custom mapping, where one or more fields in an EIS record can be used to create the object to be stored in a Java class's attribute.

Table 53–8 summarizes the behavior of this mapping depending on the EIS record type you are using.

Table 53–8 EIS Transformation Mapping by EIS Record Type

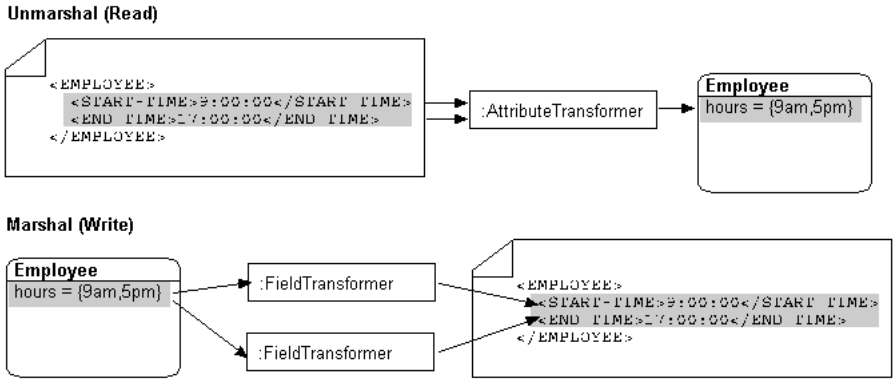
EIS Record Type	Mapping Behavior
Indexed	.The field transformer adds data to the indexed record (you have access to the indexed record in the attribute transformer).
Mapped	.The field transformer adds data to the mapped record (you have access to the mapped record in the attribute transformer).

Table 53–8 (Cont.) EIS Transformation Mapping by EIS Record Type

EIS Record Type	Mapping Behavior
XML	.The field transformer adds data to the XML record (you have access to the XML record in the attribute transformer).

As Figure 53–12 illustrates, you configure the transformation mapping with an `oracle.toplink.mappings.transformers.AttributeTransformer` instance to perform the XML instance-to-Java attribute transformation at unmarshall time. In this example, the `AttributeTransformer` combines two XML text nodes into a single Java object.

Figure 53–12 EIS Transformation Mappings



See Chapter 61, "Configuring an EIS Transformation Mapping" for more information.

54

Configuring an EIS Mapping

This chapter describes how to configure an enterprise information service (EIS) mapping.

Table 54–2 lists the types of EIS mappings that you can configure and provides a cross-reference to the type-specific chapter that lists the configurable options supported by that type.

Table 54–1 Configuring EIS Mappings

If you are creating...	See Also...
EIS Direct Mapping	Chapter 55, "Configuring an EIS Direct Mapping"
EIS Composite Direct Collection Mapping	Chapter 56, "Configuring an EIS Composite Direct Collection Mapping"
EIS Composite Object Mapping	Chapter 57, "Configuring an EIS Composite Object Mapping"
EIS Composite Collection Mapping	Chapter 58, "Configuring an EIS Composite Collection Mapping"
EIS One-to-One Mapping	Chapter 59, "Configuring an EIS One-to-One Mapping"
EIS One-to-Many Mapping	Chapter 60, "Configuring an EIS One-to-Many Mapping"
EIS Transformation Mapping	Chapter 61, "Configuring an EIS Transformation Mapping"

Table 54–2 lists the configurable options shared by two or more EIS mapping types.

Fore more information, see:

- "Mapping Creation Overview" on page 31-1
- "Understanding EIS Mappings" on page 53-1

Configuring Common EIS Mapping Options

Table 54–2 lists the configurable options shared by two or more EIS mapping types. In addition to the configurable options described here, you must also configure the options described for the specific EIS Mapping Types, as shown in Table 54–1.

Table 54–2 Common Options for EIS Mapping

Option	Type	TopLink Workbench	Java
"Configuring Read-Only Mappings" on page 32-2	Basic	✔	✔
"Configuring Indirection" on page 32-3	Basic	✔	✔
"Configuring XPath" on page 32-10	Basic	✔	✔

Table 54–2 (Cont.) Common Options for EIS Mapping

Option	Type	TopLink Workbench	Java
"Configuring a Default Null Value at the Mapping Level" on page 32-12	Basic	✔	✔
"Configuring Reference Descriptors" on page 54-2	Basic	✔	✔
"Configuring Method Accessing" on page 32-14	Advanced	✔	✔
"Configuring Read-Only Mappings" on page 32-2	Advanced	✔	✔
"Configuring Private or Independent Relationships" on page 32-16	Advanced	✔	✔
"Configuring Mapping Comments" on page 32-18	Advanced	✔	
"Configuring a Serialized Object Converter" on page 32-18	Advanced	✔	✔
"Configuring a Type Conversion Converter" on page 32-20	Advanced	✔	✔
"Configuring an Object Type Converter" on page 32-22	Advanced	✔	✔
"Configuring a Simple Type Translator" on page 32-23	Advanced	✔	✔
"Configuring Container Policy" on page 32-26	Advanced	✔	✔
"Configuring Selection Interaction" on page 54-3	Advanced	✔	✔
"Configuring the Use of a Single Node" on page 32-36	Advanced	✔	✔
"Configuring a JAXB Typesafe Enumeration Converter" on page 32-25	Advanced	✔	

Configuring Reference Descriptors

In EIS mappings that extend `oracle.toplink.mappings.ForeignReferenceMapping` or `oracle.toplink.mappings.AggregateMapping` class, attributes reference other TopLink descriptors–not the data source. You can select a descriptor in the current project, or a descriptor from some other project.

Table 54–3 summarizes which EIS mappings support this option.

Table 54–3 Mapping Support for Reference Descriptor

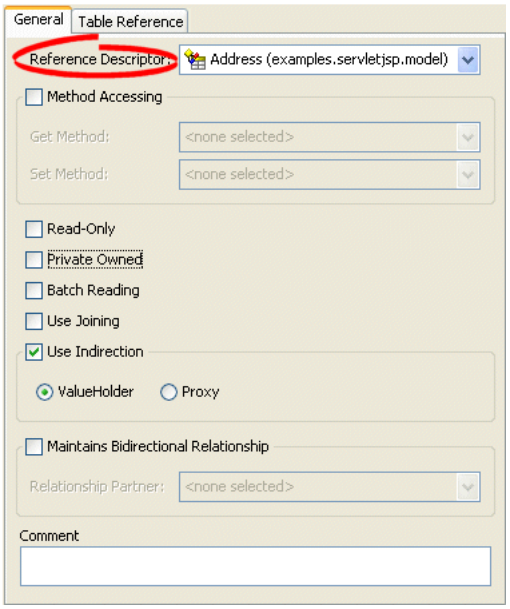
Mapping	Using TopLink Workbench	Using Java
EIS Direct Mapping		
EIS Composite Direct Collection Mapping		
EIS One-to-One Mapping	✔	✔
EIS One-to-Many Mapping	✔	✔
EIS Composite Object Mapping	✔	✔
EIS Composite Collection Mapping	✔	✔
EIS Transformation Mapping		

Using TopLink Workbench

To specify a reference descriptor for an EIS mapping, use this procedure.

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.

Figure 54–1 General Tab, Reference Descriptor Field



Use the **Reference Descriptor** field to select the descriptor referenced by this relationship mapping.

> **Note:** For one-to-one and one-to-many EIS mappings, the reference descriptor must be a root descriptor. See "Configuring an EIS Descriptor as a Root or Composite Type" on page 28-8.

You can specify a reference descriptor that is not in the current TopLink Workbench project. For example, to create a mapping to an `Employee` class that does not exist in the current project, do the following:

1. Add the `Employee` class to your current project. See "Working With Projects" on page 18-10.
2. Create the relationship mapping to the `Employee` descriptor.
3. Deactivate the `Employee` descriptor. See "Active and Inactive Descriptors" on page 4-10.

When you generate the deployment XML for your project, the mapping to the `Employee` class will be included, but not the `Employee` class itself.

Configuring Selection Interaction

In EIS mappings that extend `oracle.toplink.mappings.ForeignReferenceMapping` class, TopLink uses a selection interaction to acquire the instance of the target object to which the mapping refers.

By default, TopLink uses the read interaction you define for the mapping's reference descriptor (see "Configuring Reference Descriptors" on page 54-2). In most cases, this interaction is sufficient. If the reference descriptor's read interaction is not sufficient, you can define a separate interaction.

Table 54–4 summarizes which EIS mappings support this option.

Table 54–4 Mapping Support for Selection Interaction

Mapping	Using TopLink Workbench	Using Java
EIS Direct Mapping		
EIS Composite Direct Collection Mapping		
EIS One-to-One Mapping	✔	✔
EIS One-to-Many Mapping	✔	✔
EIS Composite Object Mapping		
EIS Composite Collection Mapping		
EIS Transformation Mapping		

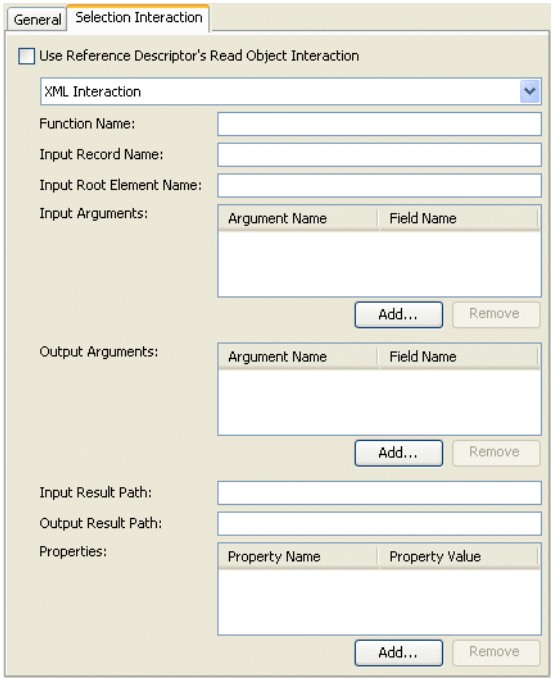
For more information about how TopLink uses the selection criteria, see "Reference EIS Mappings" on page 53-4.

Using TopLink Workbench

To specify the selection interaction (such as Read Object) for the EIS mapping, use this procedure:

1. Select the one-to-many EIS mapping in the **Navigator**. Its properties appear in the Editor.
2. Click the **Selection Interaction** tab. The Selection Interaction tab appears.

Figure 54–2 Selection Interaction Tab



Use the following information to enter data in each field on the tab:

Field	Description
Function Name	The name of the EIS function that this call type (Read Object or Read All) invokes on the EIS.
Input Record Name	The name passed to the J2C adapter when creating the input record.
Input Root Element Name	The root element name to use for the input DOM.
Input Arguments	The query argument name to map to the interaction field or XPath nodes in the argument record. For example, if you are using XML records, use this option to map input argument `name` to the XPath `name/first-name`.
Output Arguments	The result record field or XPath nodes to map to the correct nodes in the record used by the descriptor's mappings. For example, if you are using XML records, use this option to map the output `fname` to `name/first-name`. Output arguments are not required if the interaction returns an XML result that matches the descriptor's mappings.
Input Result Path	Use this option if the EIS interaction expects the interaction arguments to be nested in the XML record. For example, specify `arguments`, if the arguments were to be nested under the root element `exec-find-order`, then under an `arguments` element.
Output Result Path	The name of the EIS function that this call type (Read Object or Read All) invokes on the EIS.
Properties	Any properties required by your EIS platform. For example, property name `operation` (from `AQPlatform.QUEUE_OPERATION`) and property value `enqueue` (from `AQPlatform.ENQUEUE`).

55

Configuring an EIS Direct Mapping

This chapter describes the various components that you must configure in order to use an EIS direct mapping.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "EIS Direct Mapping" on page 53-5

EIS Direct Mapping Configuration Overview

Table 55–1 lists the configurable options for an EIS direct mapping.

Table 55–1 Configurable Options for EIS Direct Mapping

Option	Type	TopLink Workbench	Java
"Configuring XPath" on page 32-10	Basic	✔	✔
"Configuring the Use of a Single Node" on page 32-36	Advanced		✔
"Configuring a Simple Type Translator" on page 32-23	Advanced	✔	✔
"Configuring Method Accessing" on page 32-14	Advanced	✔	✔
"Configuring a Default Null Value at the Mapping Level" on page 32-12	Advanced	✔	✔
"Configuring Read-Only Mappings" on page 32-2	Advanced	✔	✔
"Configuring Mapping Comments" on page 32-18	Advanced	✔	
"Configuring a Serialized Object Converter" on page 32-18	Advanced	✔	✔
"Configuring a Type Conversion Converter" on page 32-20	Advanced	✔	✔
"Configuring an Object Type Converter" on page 32-22	Advanced	✔	✔
"Configuring a JAXB Typesafe Enumeration Converter" on page 32-25	Advanced		✔

56

Configuring an EIS Composite Direct Collection Mapping

This chapter describes the various components that you must configure in order to use an EIS composite direct collection mapping.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "EIS Composite Direct Collection Mapping" on page 53-6

EIS Composite Direct Collection Mapping Configuration Overview

Table 56–1 lists the configurable options for an EIS composite direct collection mapping.

Table 56–1 Configurable Options for EIS Composite Direct Collection Mapping

Option	Type	TopLink Workbench	Java
"Configuring XPath" on page 32-10	Basic	✔	✔
"Configuring a Simple Type Translator" on page 32-23	Advanced	✔	✔
"Configuring the Use of a Single Node" on page 32-36	Advanced	✔	✔
"Configuring Method Accessing" on page 32-14	Advanced	✔	✔
"Configuring Read-Only Mappings" on page 32-2	Advanced	✔	✔
"Configuring Container Policy" on page 32-26	Advanced	✔	✔
"Configuring Mapping Comments" on page 32-18	Advanced	✔	
"Configuring a Serialized Object Converter" on page 32-18	Advanced	✔	✔
"Configuring a Type Conversion Converter" on page 32-20	Advanced	✔	✔
"Configuring an Object Type Converter" on page 32-22	Advanced	✔	✔
"Configuring a JAXB Typesafe Enumeration Converter" on page 32-25	Advanced	✔	

57

Configuring an EIS Composite Object Mapping

This chapter describes the various components that you must configure in order to use an EIS composite object mapping.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "EIS Composite Object Mapping" on page 53-7

EIS Composite Object Mapping Configuration Overview

Table 57–1 lists the configurable options for an EIS composite object mapping.

Table 57–1 Configurable Options for EIS Composite Object Mapping

Option	Type	TopLink Workbench	Java
"Configuring XPath" on page 32-10	Basic	✔	✔
"Configuring Reference Descriptors" on page 54-2	Basic	✔	✔
"Configuring Method Accessing" on page 32-14	Advanced	✔	✔
"Configuring Read-Only Mappings" on page 32-2	Advanced	✔	✔
"Configuring Mapping Comments" on page 32-18	Advanced	✔	

58

Configuring an EIS Composite Collection Mapping

This chapter describes the various components that you must configure in order to use an EIS composite collection mapping.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "EIS Composite Collection Mapping" on page 53-7

EIS Composite Collection Mapping Configuration Overview

Table 58–1 lists the configurable options for an EIS composite collection mapping.

Table 58–1 Configurable Options for EIS Composite Collection Mapping

Option	Type	TopLink Workbench	Java
"Configuring XPath" on page 32-10	Basic	✔	✔
"Configuring Reference Descriptors" on page 54-2	Basic	✔	✔
"Configuring Method Accessing" on page 32-14	Advanced	✔	✔
"Configuring Read-Only Mappings" on page 32-2	Advanced	✔	✔
"Configuring Container Policy" on page 32-26	Advanced	✔	✔
"Configuring Mapping Comments" on page 32-18	Advanced	✔	

59

Configuring an EIS One-to-One Mapping

This chapter describes the various components that you must configure in order to use an EIS one-to-one mapping.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "EIS One-to-One Mapping" on page 53-5

EIS One-to-One Mapping Configuration Overview

Table 59–1 lists the configurable options for an EIS one-to-one mapping.

Table 59–1 Configurable Options for EIS One-to-One Mappings

Option	Type	TopLink Workbench	Java
"Configuring Reference Descriptors" on page 54-2	Basic	✔	✔
"Configuring Foreign Key Pairs" on page 59-1	Basic	✔	✔
"Configuring Bidirectional Relationship" on page 32-34	Advanced	✔	✔
"Configuring Method Accessing" on page 32-14	Advanced	✔	✔
"Configuring Read-Only Mappings" on page 32-2	Advanced	✔	✔
"Configuring Private or Independent Relationships" on page 32-16	Advanced	✔	✔
"Configuring Indirection" on page 32-3	Advanced	✔	✔
"Configuring Mapping Comments" on page 32-18	Advanced	✔	
"Configuring Selection Interaction" on page 54-3	Basic	✔	✔

Configuring Foreign Key Pairs

In a one-to-one EIS mapping, you relate a source object attribute to a target object attribute by specifying one or more pairs of source and target object fields.

In a one-to-one EIS mapping with key on source (see "EIS One-to-One Mappings With Key on Source" on page 53-9) using XML records, TopLink puts the target XML field value into the source object's record as a simple value.

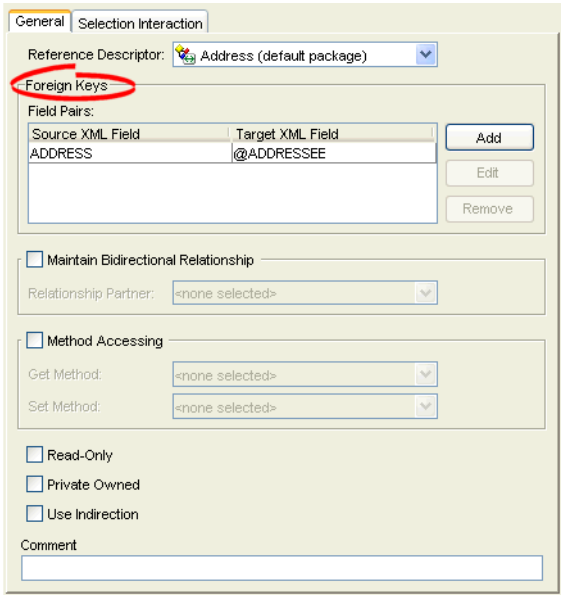
In a one-to-one EIS mapping with key on target (see "EIS One-to-One Mappings With Key on Target" on page 53-10) using XML records, TopLink uses the source XML field value in the selection interaction to acquire the appropriate instance of target object.

Using TopLink Workbench

To specify the source and target XML field pairs for a one-to-one EIS mapping, use this procedure:

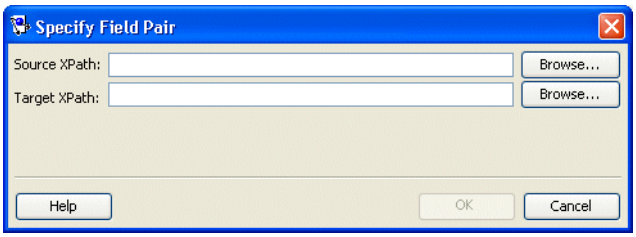
1. Select the one-to-one EIS mapping in the **Navigator**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab opens.

Figure 59–1 General Tab, Foreign Keys Field



3. Click **Add** in the Foreign Keys area to add a key pair. The Specify Field Pair dialog box appears.

Figure 59–2 Specify Field Pair Dialog Box



Click **Browse** to add a foreign key for the **Source XPath** and **Target XPath** fields.

60

Configuring an EIS One-to-Many Mapping

This chapter describes the various components that you must configure in order to use an EIS one-to-many mapping.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "EIS One-to-Many Mapping" on page 53-12

EIS One-to-Many Mapping Configuration Overview

Table 60–1 lists the configurable options for an EIS one-to-many mapping.

Table 60–1 Configurable Options for EIS One-to-Many Mappings

Option	Type	TopLink Workbench	Java
"Configuring Reference Descriptors" on page 54-2	Basic	✔	✔
"Configuring Foreign Key Pairs" on page 60-1	Advanced	✔	✔
"Configuring Bidirectional Relationship" on page 32-34	Advanced	✔	✔
"Configuring Method Accessing" on page 32-14	Advanced	✔	✔
"Configuring Read-Only Mappings" on page 32-2	Advanced	✔	✔
"Configuring Private or Independent Relationships" on page 32-16	Advanced	✔	✔
"Configuring Indirection" on page 32-3	Advanced	✔	✔
"Configuring Container Policy" on page 32-26	Advanced	✔	✔
"Configuring Mapping Comments" on page 32-18	Advanced	✔	
"Configuring Selection Interaction" on page 54-3	Advanced	✔	✔
"Configuring Delete All Interactions" on page 60-3	Advanced	✔	✔

Configuring Foreign Key Pairs

In a one-to-many EIS mapping, you relate a source object attribute to a target object attribute by specifying one or more pairs of source and target object fields.

In a one-to-many EIS mapping with key on source (see "EIS One-to-Many Mappings With Key on Source" on page 53-13) using XML records, TopLink puts the target XML field value into the source object's record as a simple value. By default, these values are not grouped, as Example 60–1 shows.

Example 60–1 Source Object XML Record without Grouping

```
<employee>
    <name>Jane</name>
    <project-id>3</project-id>
    <project-id>4</project-id>
</employee>
```

If you specify more than one source and target XML field pair, you must specify a grouping element, as Example 60–2 shows.

Example 60–2 Source Object XML Record with Grouping

```
<employee>
    <name>Jane</name>
    <project>
        <project-id>3</project-id>
        <project-name>Project 3</project-name>
    </project>
    <project>
        <project-id>4</project-id>
        <project-name>Project 4</project-name>
    </project>
</employee>
```

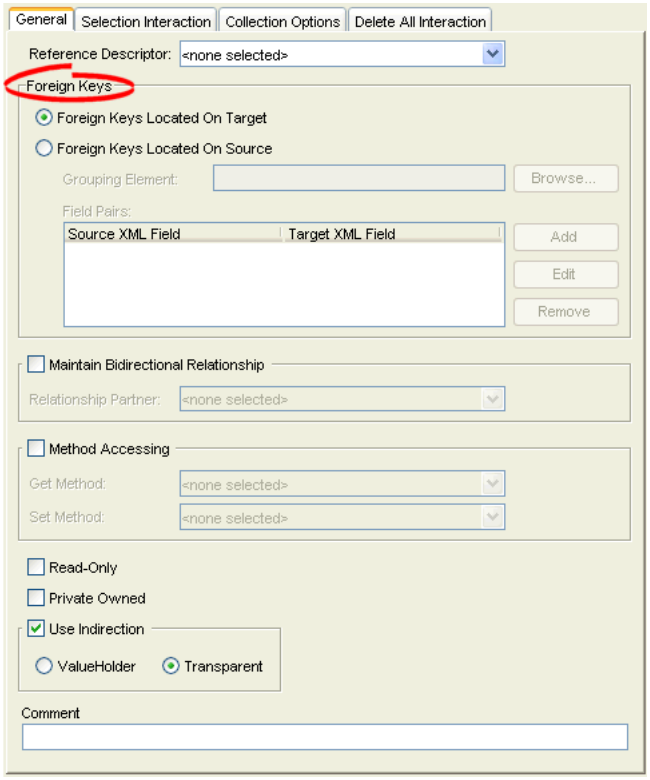
In a one-to-one EIS mapping with key on target (see "EIS One-to-Many Mappings With Key on Target" on page 53-15) using XML records, TopLink uses the source XML field value in the selection interaction to acquire the appropriate instances of target object.

Using TopLink Workbench

To specify the source and target XML field pairs for a one-to-many EIS mapping, use this procedure:

1. Select the one-to-one EIS mapping in the **Navigator**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.

Figure 60–1 Foreign Keys Field on General Tab



Use the following information to complete the Foreign Keys fields on the **General** tab:

Field	Description
Foreign Keys Located On Target	Select if you are creating a one-to-many EIS mapping with key on target (see "EIS One-to-Many Mappings With Key on Target" on page 53-15).
Foreign Keys Located On Source	Select if you are creating a one-to-many EIS mapping with key on source (see "EIS One-to-Many Mappings With Key on Source" on page 53-13).
Grouping Element	Specify the element in which foreign key pairs are grouped in the source object's EIS record. If you specify only one pair of source and target XML fields, this is optional. If you specify more than one pair of source and target XML fields, this is required.
Field Pairs	Click **Add** to add a pair of source and target XML fields. Specify Field Pair dialog box opens. Click **Browse** to add a foreign key for the **Source XPath** and **Target XPath** fields.

Configuring Delete All Interactions

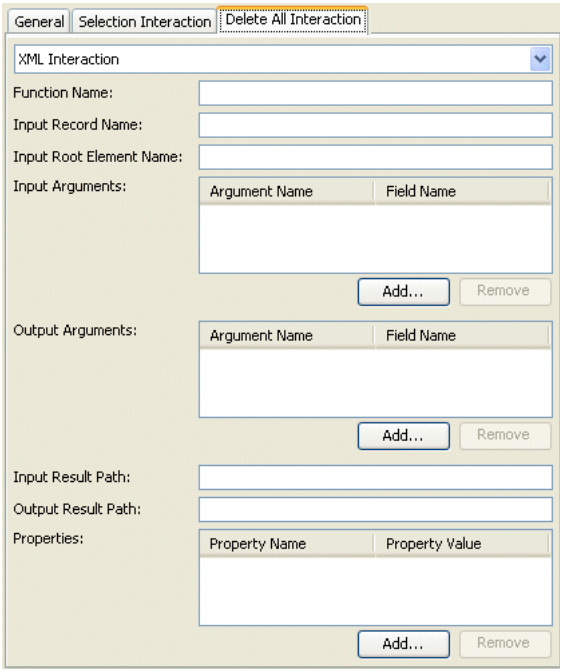
The TopLink query and expression framework supports delete all queries. If your J2C adapter provides access to an EIS Delete All function, you can configure a delete all interaction to support TopLink delete all queries.

Using TopLink Workbench

To specify the DeleteAll interaction for an EIS one-to-many mapping, use this procedure:

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.
2. Click the **Delete All Interaction** tab. The Delete All Interaction tab appears.

Figure 60–2 Delete All Interaction Tab



Use the following information to enter data in each field on the Delete All Interaction tab:

Field	Description
Function Name	The name of the EIS function that this call type (Read Object or Read All) invokes on the EIS.
Input Record Name	The name passed to the J2C adapter when creating the input record.
Input Root Element Name	The root element name to use for the input DOM.
Input Arguments	The query argument name to map to the interaction field or XPath nodes in the argument record. For example, if you are using XML records, use this option to map input argument `name` to the XPath `name/first-name`.
Output Arguments	The result record field or XPath nodes to map to the correct nodes in the record used by the descriptor's mappings. For example, if you are using XML records, use this option to map the output `fname` to `name/first-name`. Output arguments are not required if the interaction returns an XML result that matches the descriptor's mappings.

Field	Description
Input Result Path	Use this option if the EIS interaction expects the interaction arguments to be nested in the XML record. For example, specify `arguments`, if the arguments were to be nested under the root element `exec-find-order`, then under an `arguments` element.
Output Result Path	The name of the EIS function that this call type (Read Object or Read All) invokes on the EIS.
Properties	Any properties required by your EIS platform. For example, property name `operation` (from `AQPlatform.QUEUE_OPERATION`) and property value `enqueue` (from `AQPlatform.ENQUEUE`).

61

Configuring an EIS Transformation Mapping

This chapter describes the various components that you must configure in order to use an EIS transformation mapping.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "EIS Transformation Mapping" on page 53-17

EIS Transformation Mapping Configuration Overview

Table 61–1 lists the configurable options for an EIS transformation mapping.

Table 61–1 Configurable Options for EIS Transformation Mapping

Option	Type	TopLink Workbench	Java
"Configuring Attribute Transformer" on page 32-29	Basic	✔	✔
"Configuring Field Transformer Associations" on page 32-31	Basic	✔	✔
"Configuring Method Accessing" on page 32-14	Advanced	✔	✔
"Configuring Read-Only Mappings" on page 32-2	Advanced	✔	✔
"Configuring Mutable Mappings" on page 32-33	Advanced	✔	✔
"Configuring Mapping Comments" on page 32-18	Advanced	✔	

Part XIII

XML Mappings

An XML mapping transforms object data members to the XML nodes of an XML document, whose structure is defined by an XML schema document (XSD).

This part contains the following chapters:

- Chapter 62, "Understanding XML Mappings"

  This chapter describes each of the different TopLink XML mapping types and important XML mapping concepts.

- Chapter 63, "Configuring an XML Mapping"

  This chapter explains how to configure TopLink XML mapping options common to two or more XML mapping types.

- Chapter 64, "Configuring an XML Direct Mapping"

  This chapter explains how to configure a direct XML mapping.

- Chapter 65, "Configuring an XML Composite Direct Collection Mapping"

  This chapter explains how to configure a composite direct collection XML mapping.

- Chapter 66, "Configuring an XML Composite Object Mapping"

  This chapter explains how to configure a composite object XML mapping including an XML mapping to a single named complex type of type `xs:anyType`.

- Chapter 67, "Configuring an XML Composite Collection Mapping"

  This chapter explains how to configure a composite collection XML mapping.

- Chapter 68, "Configuring an XML Any Object Mapping"

  This chapter explains how to configure an XML mapping to a single unnamed complex type specified as `xs:any`.

- Chapter 69, "Configuring an XML Any Collection Mapping"

  This chapter explains how to configure an XML mapping to an unnamed sequence of complex types specified as `xs:any`, an unnamed sequence of complex types of type `xs:anyType`, or a root element of type `xs:anyType`.

- Chapter 70, "Configuring an XML Transformation Mapping"

  This chapter explains how to configure a transformation XML mapping.

62

Understanding XML Mappings

An XML mapping transforms object data members to the XML nodes of an XML document whose structure is defined by an XML schema document (XSD).

This chapter describes the following:

- XML Mapping Types
- XML Mapping Concepts

XML Mapping Types

TopLink supports the XML mappings listed in Table 62–1.

Table 62–1 TopLink XML Mapping Types

Mapping Type	Description	Type	TopLink Workbench	Java
"XML Direct Mapping" on page 62-5	Map a simple object attribute to an XML attribute or text node.	Basic	✔	✔
"XML Composite Direct Collection Mapping" on page 62-14	Map a collection of simple object attributes to XML attributes or text nodes.	Basic	✔	✔
"XML Composite Object Mapping" on page 62-21	Map any attribute that contains a single object to an XML element. The TopLink runtime uses the descriptor for the referenced object to populate the contents of that element.	Basic	✔	✔
"XML Composite Collection Mapping" on page 62-25	Map an attribute that contains a homogenous collection of objects to multiple XML elements. The TopLink runtime uses the descriptor for the referenced object to populate the contents of those elements.	Basic	✔	✔

Table 62–1 (Cont.) TopLink XML Mapping Types

Mapping Type	Description	Type	TopLink Workbench	Java
"XML Any Object Mapping" on page 62-27	The any object XML mapping is similar to the composite object XML mapping (see "XML Composite Object Mapping" on page 62-21), except that the reference object may be of different types (including `String`), not necessarily related to each other through inheritance or a common interface.	Advanced	✔	✔
"XML Any Collection Mapping" on page 62-29	The any collection XML mapping is similar to the composite collection XML mapping (see "XML Composite Collection Mapping" on page 62-25) except that the referenced objects may be of different types (including `String`), not necessarily related to each other through inheritance or a common interface.	Advanced	✔	✔
"XML Transformation Mapping" on page 62-31	Create custom mappings where one or more XML nodes can be used to create the object to be stored in a Java class's attribute.	Advanced	✔	✔

XML Mapping Concepts

You can map the attributes of a Java object to a combination of XML simple and complex types using a wide variety of XML mapping types.

TopLink stores XML mappings for each class in the class descriptor. TopLink uses the descriptor to instantiate objects mapped from an XML document and to store new or modified objects as an XML document.

To configure XML mappings, Oracle recommends that you use TopLink Workbench and its rich graphical user interface (GUI) environment to set the descriptor properties and configure the mappings.

This section describes concepts unique to TopLink XML mappings, including the following:

- Mapping to Simple and Complex Types
- Mapping Order
- XPath Support
- xsd:list and xsd:union Support
- xs:any and xs:anyType Support
- jaxb:class Support
- Typesafe Enumeration Support
- Mapping Extensions

Mapping to Simple and Complex Types

Consider the XML document shown in Example 62–1.

Example 62–1 XML Document

```
<EMPLOYEE ID="123">
    <NAME>Jane Doe</NAME>
```

```
    <ADDRESS>
        <STREET>123 Any St.</STREET>
        <CITY>MyCity</CITY>
    </ADDRESS>
</EMPLOYEE>
```

In general, using TopLink XML mappings, you can map a Java class to a simple type (such as `NAME`) or to a complex type (such as `ADDRESS`).

Specifically, you can map a Java object's simple attributes to XML attributes (such as `ID`) and text nodes (such as `NAME`). You can also map a Java object's relationships to XML elements (such as `ADDRESS`).

Table 62–2 summarizes the XML simple and complex types supported by each TopLink XML mapping.

Table 62–2 XML Mapping Support for XML Simple and Complex Types

Mapping	XML Attribute	XML Text Node	XML Element
XML Direct Mapping	✔	✔	
XML Composite Direct Collection Mapping	✔	✔	
XML Composite Object Mapping			✔
XML Composite Collection Mapping			✔
XML Any Object Mapping			✔
XML Any Collection Mapping			✔
XML Transformation Mapping	✔	✔	✔

Mapping Order

Unlike relational database mappings, the order in which mappings are persisted in XML is significant.

The order in which you define XML mappings in TopLink (whether in TopLink Workbench or in Java code) including the order in which you define mapping components such as `Transformers` (see "XML Transformation Mapping" on page 62-31) is reflected in the order, in which TopLink persists data in an XML document.

XPath Support

TopLink uses XPath statements to efficiently map the attributes of a Java object to locations in an XML document. For more information about using XPath with XML mappings, see "Mappings and XPath" on page 30-15.

xsd:list and xsd:union Support

You can use XML direct (see "XML Direct Mapping" on page 62-5) and composite direct collection (see "XML Composite Direct Collection Mapping" on page 62-14) mappings to map to `xsd:list` and `xsd:union` types in an XML document.

For more information, see "Mappings and xsd:list and xsd:union Types" on page 30-17.

xs:any and xs:anyType Support

In an XML schema, you can define elements and complex types that correspond to any data type using `xs:any` and `xs:anyType`. You can map objects to such elements and complex types using XML mappings `XMLAnyObjectMapping` and `XMLAnyCollectionMapping`.

Table 62–3 lists the XML mappings to use with common applications of `xs:any` and `xs:anyType`. For more details, see the specified XML mapping type.

Table 62–3 XML Mappings and XML Schema xs:any and xs:anyType

Use XML Mapping ...	To Map XML Schema Definition ...
See "XML Any Object Mapping" on page 62-27	Element with a single[1] unnamed complex type specified as `xs:any`.
See "XML Any Collection Mapping" on page 62-29	Element with an unnamed sequence[2] of complex types specified as `xs:any`. Element with a named sequence[2] of complex types of type `xs:anyType`. Root element of type `xs:anyType`.

[1] `minOccurs` and `maxOccurs` are both equal to 1.
[2] `maxOccurs` is greater than 1.

jaxb:class Support

You can configure an XML composite object mapping (see "XML Composite Object Mapping" on page 62-21) to accommodate `jaxb:class` customizations with the following XSD structures:

- `all`
- `sequence`
- `choice`
- `group`

For more information, see "Mappings and the jaxb:class Customization" on page 30-20.

Typesafe Enumeration Support

You can map a Java attribute to such a typesafe enumeration using the `JAXBTypesafeEnumConverter` with an `XMLDirectMapping` or `XMLCompositeDirectCollectionMapping` with XML documents.

For more information, see "Mappings and JAXB Typesafe Enumerations" on page 30-24

Mapping Extensions

If existing TopLink XML mappings do not meet your needs, you can create custom XML mappings using XML mapping extensions, including object type, serialized object, type conversion converters, and a simple type translator. For more information, see "Mapping Converters and Transformers" on page 30-10.

XML Direct Mapping

XML direct mappings map a Java attribute directly to XML text nodes. You can use an XML direct mapping in the following scenarios:

- Mapping to a Text Node
- Mapping to an Attribute
- Mapping to a Specified Schema Type
- Mapping to a List Field With an XML Direct Mapping
- Mapping to a Union Field With an XML Direct Mapping
- Mapping to a Union of Lists With an XML Direct Mapping
- Mapping to a Union of Unions With an XML Direct Mapping
- Mapping With a Simple Type Translator

See Chapter 64, "Configuring an XML Direct Mapping" for more information.

> **Note:** Do not confuse an XML direct mapping with a relational direct-to-`XMLType` mapping (see "Direct-to-XMLType Mapping" on page 33-4).

Mapping to a Text Node

This section describes using an XML direct mapping when:

- Mapping to a Simple Text Node
- Mapping to a Text Node in a Simple Sequence
- Mapping to a Text Node in a Subelement
- Mapping to a Text Node by Position

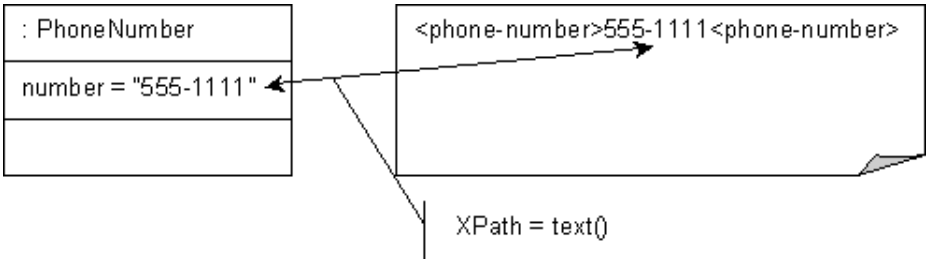
Mapping to a Simple Text Node

Given the XML schema in Example 62–2, Figure 62–1 illustrates an XML direct mapping to a simple text node in a corresponding XML document. Example 62–3 shows how to configure this mapping in Java.

Example 62–2 Schema for XML Direct Mapping to Simple Text Node

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="phone-number" type="xsd:string"/>
</xsd:schema>
```

Figure 62–1 XML Direct Mapping to Simple Text Node



Example 62–3 Java for XML Direct Mapping to Simple Text Node

```
XMLDirectMapping numberMapping = new XMLDirectMapping();
numberMapping.setAttributeName("number");
numberMapping.setXPath("text()");
```

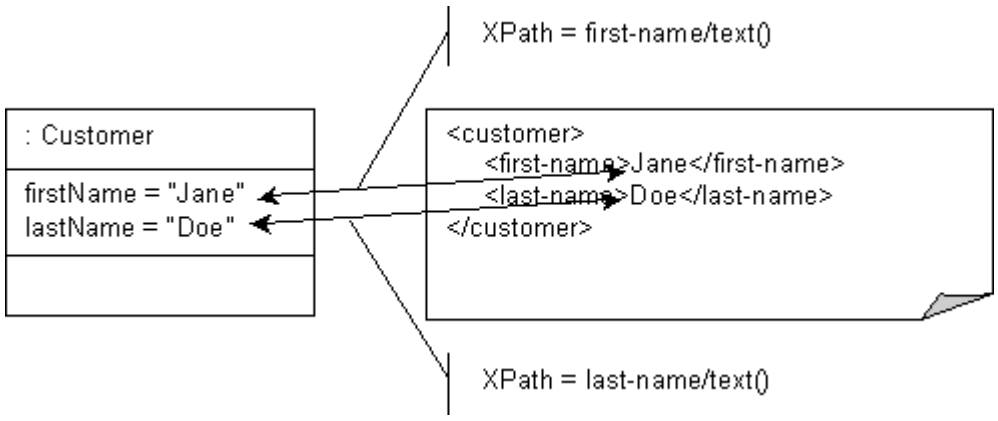
Mapping to a Text Node in a Simple Sequence

Given the XML schema in Example 62–4, Figure 62–2 illustrates an XML direct mapping to individual text nodes in a sequence in a corresponding XML document. Example 62–5 shows how to configure this mapping in Java.

Example 62–4 Schema for XML Direct Mapping to a Text Node in a Simple Sequence

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="customer" type="customer-type"/>
    <xsd:complexType name="customer-type">
        <xsd:sequence>
            <xsd:element name="first-name" type="xsd:string"/>
            <xsd:element name="last-name" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

Figure 62–2 XML Direct Mapping to a Text Node in a Simple Sequence



Example 62–5 Java for XML Direct Mapping to a Text Node in a Simple Sequence

```
XMLDirectMapping firstNameMapping = new XMLDirectMapping();
firstNameMapping.setAttributeName("firstName");
firstNameMapping.setXPath("first-name/text()");

XMLDirectMapping lastNameMapping = new XMLDirectMapping();
lastNameMapping.setAttributeName("lastName");
lastNameMapping.setXPath("last-name/text()");
```

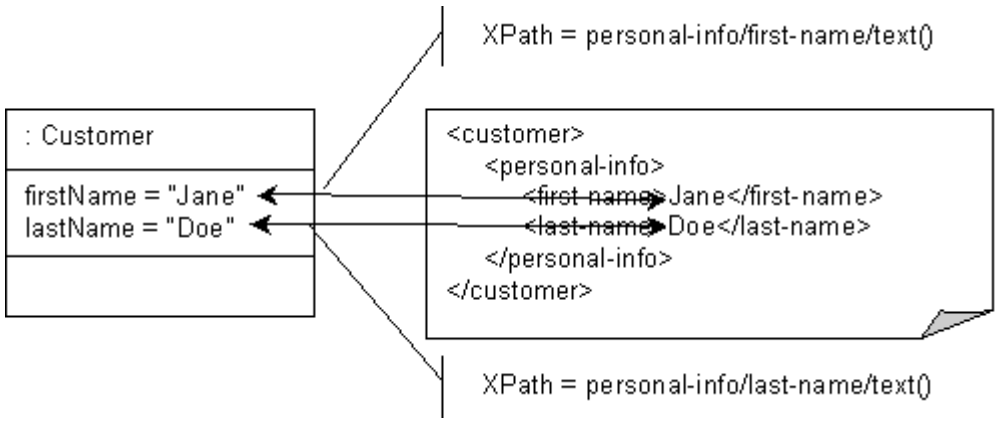
Mapping to a Text Node in a Subelement

Given the XML schema in Example 62–6, Figure 62–3 illustrates an XML direct mapping to a text node in a subelement in a corresponding XML document. Example 62–7 shows how to configure this mapping in Java.

Example 62–6 Schema for XML Direct Mapping to a Text Node in a Subelement

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="customer" type="customer-type"/>
    <xsd:complexType name="customer-type">
        <xsd:sequence>
            <xsd:element name="personal-info">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="first-name" type="xsd:string"/>
                        <xsd:element name="last-name" type="xsd:string"/>
                    <xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

Figure 62–3 XML Direct Mapping to a Text Node in a Subelement



Example 62–7 Java for XML Direct Mapping to a Text Node in a Subelement

```
XMLDirectMapping firstNameMapping = new XMLDirectMapping();
firstNameMapping.setAttributeName("firstName");
firstNameMapping.setXPath("personal-info/first-name/text()");

XMLDirectMapping lastNameMapping = new XMLDirectMapping();
lastNameMapping.setAttributeName("lastName");
lastNameMapping.setXPath("personal-info/last-name/text()");
```

Mapping to a Text Node by Position

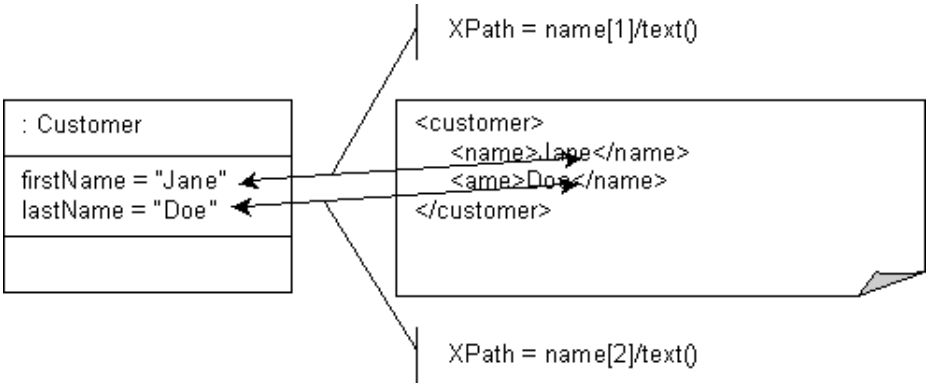
Given the XML schema in Example 62–8, Figure 62–4 illustrates an XML direct mapping to a text node by position in a corresponding XML document. Example 62–9 shows how to configure this mapping in Java.

Example 62–8 Schema for XML Direct Mapping to Text Node by Position

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="customer" type="customer-type"/>
    <xsd:complexType name="customer-type">
        <xsd:sequence>
            <xsd:element name="name" type="xsd:string" maxOccurs="2"/>
        </xsd:sequence>
    </xsd:complexType>
```

```
</xsd:schema>
```

Figure 62–4 XML Direct Mapping to Text Node by Position



Example 62–9 Java for XML Direct Mapping to Text Node by Position

```
XMLDirectMapping firstNameMapping = new XMLDirectMapping();
firstNameMapping.setAttributeName("firstName");
firstNameMapping.setXPath("name[1]/text()");

XMLDirectMapping lastNameMapping = new XMLDirectMapping();
lastNameMapping.setAttributeName("lastName");
lastNameMapping.setXPath("name[2]/text()");
```

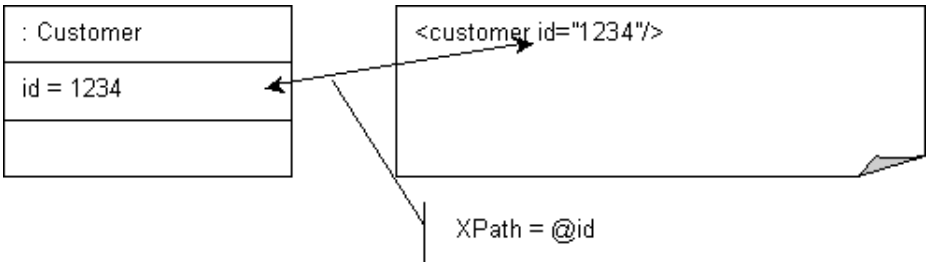
Mapping to an Attribute

Given the XML schema in Example 62–8, Figure 62–4 illustrates an XML direct mapping to a text node by position in a corresponding XML document. Example 62–9 shows how to configure this mapping in Java.

Example 62–10 Schema for XML Direct Mapping to an Attribute

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="customer" type="customer-type"/>
    <xsd:complexType name="customer-type">
        <xsd:attribute name="id" type="xsd:integer"/>
    </xsd:complexType>
</xsd:schema>
```

Figure 62–5 XML Direct Mapping to an Attribute



Example 62–11 Java for XML Direct Mapping to an Attribute

```
XMLDirectMapping idMapping = new XMLDirectMapping();
```

```
idMapping.setAttributeName("id");
idMapping.setXPath("@id");
```

Mapping to a Specified Schema Type

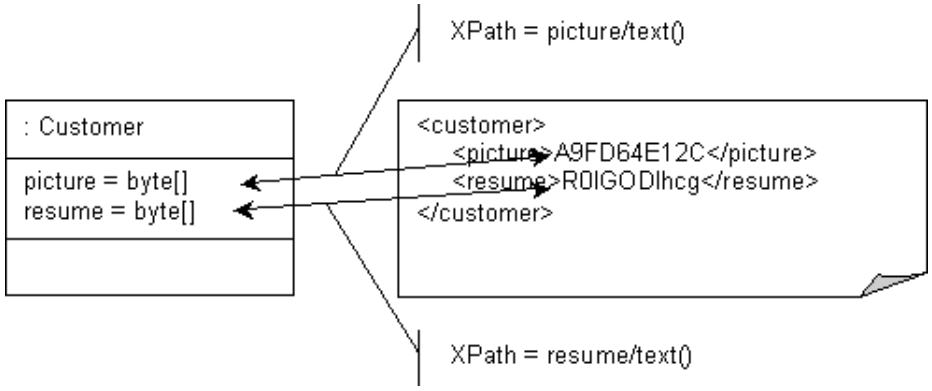
In most cases, TopLink can determine the target format in the XML document. However, there are cases where you must specify which one of a number of possible targets TopLink should use. For example, a `java.util.Calendar` could be marshalled to a schema `date`, `time`, or `dateTime` node, or a `byte[]` could be marshalled to a schema `hexBinary` or `base64Binary` node.

Given the XML schema in Example 62–8, Figure 62–4 illustrates an XML direct mapping to a text node by position in a corresponding XML document. Example 62–9 shows how to configure this mapping in Java.

Example 62–12 Schema for XML Direct Mapping to a Specified Schema Type

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="customer" type="customer-type"/>
    <xsd:complexType name="customer-type">
        <xsd:sequence>
            <xsd:element name="picture" type="xsd:hexBinary"/>
            <xsd:element name="resume" type="xsd:base64Binary"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

Figure 62–6 XML Direct Mapping to a Specified Schema Type



Example 62–13 Java for XML Direct Mapping to a Specified Schema Type

```
XMLDirectMapping pictureMapping = new XMLDirectMapping();
pictureMapping.setAttributeName("picture");
pictureMapping.setXPath("picture/text()");
XMLField pictureField = (XMLField) pictureMapping.getField();
pictureField.setSchemaType(XMLConstants.HEX_BINARY_QNAME);

XMLDirectMapping resumeMapping = new XMLDirectMapping();
resumeMapping.setAttributeName("resume");
resumeMapping.setXPath("resume/text()");
XMLField resumeField = (XMLField) resumeMapping.getField();
resumeField.setSchemaType(XMLConstants.BASE_64_BINARY_QNAME);
```

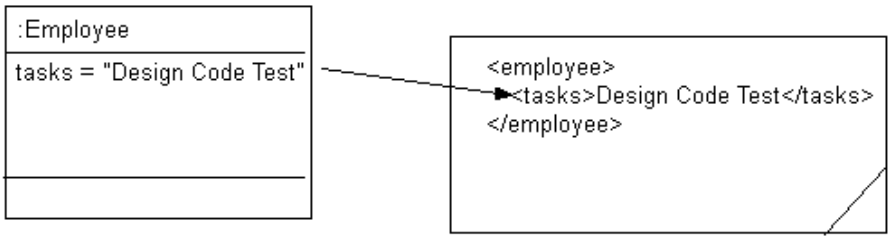
Mapping to a List Field With an XML Direct Mapping

Given the XML schema in Example 62–14, Figure 62–7 illustrates an XML direct mapping to an `xsd:list` type in a corresponding XML document when you represent the list in your object model as a `String` of white space delimited tokens. Example 62–15 shows how to configure this mapping in Java.

Example 62–14 Schema for XML Direct Mapping to a List Field

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="employee" type="employee-type"/>
    <xsd:complexType name="employee-type">
        <xsd:sequence>
            <xsd:element name="tasks" type="tasks-type"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:simpleType name="tasks-type">
        <xsd:list itemType="xsd:string"/>
    </xsd:simpleType>
</xsd:schema>
```

Figure 62–7 XMLDirect Mapping to a List Field



Example 62–15 Java for XML Direct Mapping to a List Field Node

```
XMLDirectMapping tasksMapping = new XMLDirectMapping();
tasksMapping.setAttributeName("tasks");
XMLField myField = new XMLField("tasks/text()"); // pass in the XPath
myField.setUsesSingleNode(true);
tasksMapping.setField(myField);
```

Mapping to a Union Field With an XML Direct Mapping

Given the XML schema in Example 62–16, Figure 62–8 illustrates a Java class that can be mapped to a corresponding XML document. Note the `shoeSize` attribute in this class: when using a union field, the corresponding attribute must be able to store all possible values.

Example 62–16 Schema for XML Direct Mapping to a Union Field

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="customer" type="customer-type"/>
    <xsd:complexType name="customer-type">
        <xsd:sequence>
            <xsd:element name="shoe-size" type="size-type"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:simpleType name="size-type">
        <xsd:union memberTypes="xsd:decimal xsd:string"/>
```

```
    </xsd:simpleType>
</xsd:schema>
```

Figure 62–8 Java Class for XML Direct Mapping to a Union Field

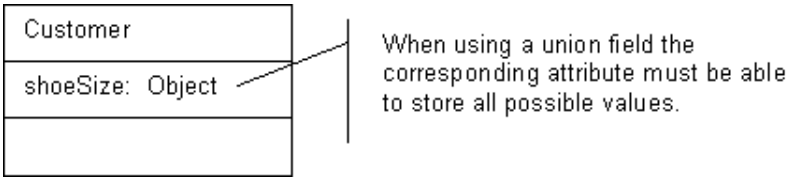


Figure 62–9 illustrates an XML direct mapping to a union field in an XML document that conforms to the schema in Example 62–16. When TopLink unmarshalls the XML document, it tries each of the union types until it can make a successful conversion. The first schema type in the union is `xsd:decimal`. Because "10.5" is a valid decimal, TopLink converts the value to the appropriate type. If the `Object` attribute is specific enough to trigger an appropriate value, TopLink will use that type instead. Otherwise, TopLink uses a default (in this case `BigDecimal`). You can override this behavior in Java code.

Figure 62–9 XML Direct Mapping to the First Valid Union Type

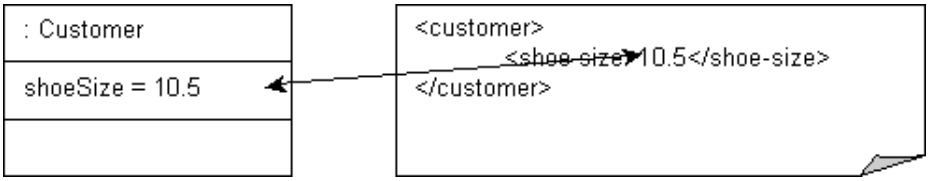
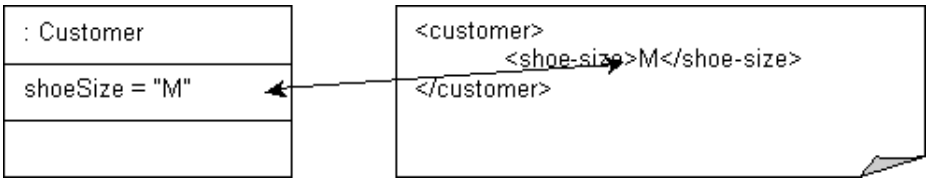


Figure 62–10 illustrates an XML direct mapping to union field in another XML document that conforms to the schema in Example 62–16. In this document, the value "M" is not a valid `xsd:decimal` type so the next union type is tried. The next union type is `xsd:string` and a conversion can be done.

Figure 62–10 XML Direct Mapping to Another Valid Union Type



Example 62–17 shows how to configure this mapping in Java.

Example 62–17 Java for XML Direct Mapping to a Union Type

```
XMLDirectMapping shoeSizeMapping = new XMLDirectMapping();
shoeSizeMapping.setAttributeName("shoeSize");
XMLUnionField shoeSizeField = new XMLUnionField();
shoeSizeField.setXPath("shoe-size/text()");
shoeSizeField.addSchemaType(XMLConstants.DECIMAL_QNAME);
shoeSizeField.addSchemaType(XMLConstants.STRING_QNAME);
shoeSizeMapping.setField(shoeSizeField);
```

To override the default conversion, use the `XMLUnionField` method `addConversion`:

```
shoeSizeField.addConversion(XMLConstants.DECIMAL_QNAME, Float.class);
```

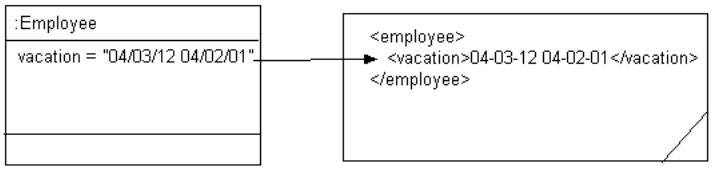
Mapping to a Union of Lists With an XML Direct Mapping

Given the XML schema in Example 62–18, Figure 62–11 illustrates an XML direct mapping to a union of lists in a corresponding XML document. Example 62–19 shows how to configure this mapping in Java.

Example 62–18 Schema for XML Direct Mapping to Union of Lists

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="vacation" type="unionOfLists"/>
    <xsd:simpleType name="unionOfLists">
        <xsd:union memberTypes="xsd:double">
            <xsd:simpleType>
                <xsd:list itemType="xsd:date"/>
            </xsd:simpleType>
            <xsd:simpleType>
                <xsd:list itemType="xsd:integer"/>
            </xsd:simpleType>
        </xsd:union>
    </xsd:simpleType>
</xsd:schema>
```

Figure 62–11 XML Direct Mapping to Union of Lists



Note that in this example, valid XML documents contain either all `xsd:double`, all `xsd:date`, or all `xsd:integer` values.

Example 62–19 Java for XML Direct Mapping to Union of Lists

```
XMLDirectMapping mapping = new XMLDirectMapping();
mapping.setAttributeName("vacation");
mapping.setXPath("UnionOfLists/text()");
```

Mapping to a Union of Unions With an XML Direct Mapping

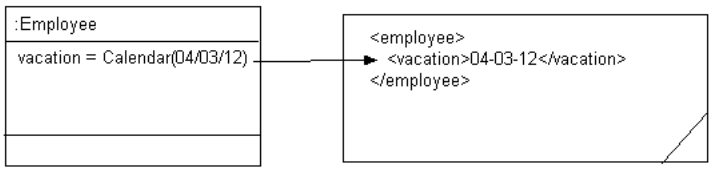
Given the XML schema in Example 62–20, Figure 62–12 illustrates a Java class that can be mapped to a corresponding XML document. Example 62–27 shows how to configure this mapping in Java.

Example 62–20 Schema for XML Direct Mapping to a Union of Unions

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="vacation" type="unionOfUnions"/>
    <xsd:simpleType name="unionOfUnions">
        <xsd:union>
            <xsd:simpleType>
                <xsd:union>
                    <xsd:simpleType>
```

```
                        <xsd:list itemType="xsd:date"/>
                    </xsd:simpleType>
                    <xsd:simpleType>
                        <xsd:list itemType="xsd:integer"/>
                    </xsd:simpleType>
                </xsd:union>
            </xsd:simpleType>
            <xsd:simpleType>
                <xsd:union>
                    <xsd:simpleType>
                        <xsd:list itemType="xsd:string"/>
                    </xsd:simpleType>
                    <xsd:simpleType>
                        <xsd:list itemType="xsd:float"/>
                    </xsd:simpleType>
                </xsd:union>
            </xsd:simpleType>
        </xsd:union>
    </xsd:simpleType>
</xsd:schema>
```

Figure 62–12 Java Class for XML Direct Mapping to a Union of Unions



Example 62–21 Java for XML Direct Mapping to a Union of Unions

```
XMLDirectMapping vacationMapping = new XMLDirectMapping();
vacationMapping.setAttributeName("vacation");
XMLUnionField vacationField = new XMLUnionField();
vacationField.setXPath("vacation/text()");
vacationField.addSchemaType(XMLConstants.DATE_QNAME);
vacationField.addSchemaType(XMLConstants.INTEGER_QNAME);
vacationField.addSchemaType(XMLConstants.STRING_QNAME);
vacationField.addSchemaType(XMLConstants.FLOAT_QNAME);
vacationMapping.setField(vacationField);
```

Mapping With a Simple Type Translator

If the type of a node is not defined in your XML schema, you can configure an XML direct mapping to use the `xsi:type` attribute to provide type information.

Given the XML schema fragment in Example 62–22, Figure 62–13 illustrates a Java class that can be mapped to a corresponding XML document.

Example 62–22 Schema for XML Direct Mapping with Simple Type Translator

```
...
    <xs:element name="area-code" type="anySimpleType"/>
    <xs:element name="number" type="anySimpleType"/>
...
```

Figure 62–13 Java Class for XML Direct Mapping with Simple Type Translator

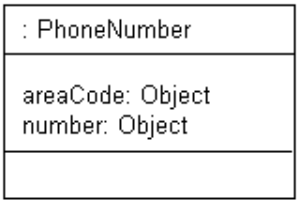
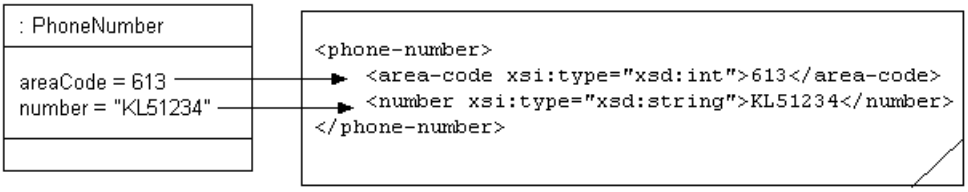


Figure 62–14 illustrates an XML direct mapping with a simple type translator in an XML document that conforms to the schema in Example 62–22.

Figure 62–14 XML Direct Mapping with a Simple Type Translator



Example 62–23 shows how to configure this mapping in Java.

Example 62–23 Java for XML Direct Mapping with Simple Type Translator

```
XMLDirectMapping numberMapping = new XMLDirectMapping();
numberMapping.setAttributeName("number");
numberMapping.setXPath("number/text()");
XMLField numberField = (XMLField) numberMapping.getField();
numberField.setIsTypedTextField(true);
```

For more information, see "Simple Type Translator" on page 30-12.

XML Composite Direct Collection Mapping

XML composite direct collection mappings map a Java collection of simple object attributes to XML attributes and text nodes. Use multiplicity settings to specify an element as a collection. The XML schema allows you to define minimum and maximum occurrences. You can use a composite direct collection XML mapping in the following scenarios:

- Mapping to Multiple Text Nodes
- Mapping to Multiple Attributes
- Mapping to a Single Text Node With an XML Composite Direct Collection Mapping
- Mapping to a Single Attribute With an XML Composite Direct Collection Mapping
- Mapping to a List of Unions With an XML Composite Direct Collection Mapping
- Mapping to a Union of Lists With an XML Composite Direct Collection Mapping
- Specifying the Content Type of a Collection With an XML Composite Direct Collection Mapping

See Chapter 65, "Configuring an XML Composite Direct Collection Mapping" for more information.

Mapping to Multiple Text Nodes

This section describes using a composite direct collection XML mapping when:

- Mapping to a Simple Sequence
- Mapping to a Sequence in a Subelement

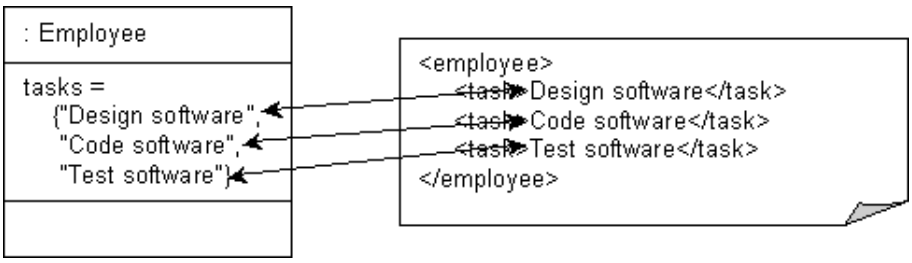
Mapping to a Simple Sequence

Given the XML schema in Example 62–24, Figure 62–15 illustrates a composite direct collection XML mapping to a simple sequence of text nodes in a corresponding XML document. Example 62–25 shows how to configure this mapping in Java.

Example 62–24 Schema for Composite Direct Collection XML Mapping to a Simple Sequence

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="employee" type="employee-type"/>
    <xsd:complexType name="employee-type">
        <xsd:sequence>
            <xsd:element name="task" type="xsd:string" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

Figure 62–15 Composite Direct Collection XML Mapping to a Simple Sequence



Example 62–25 Java for Composite Direct Collection XML Mapping to a Simple Sequence

```
XMLCompositeDirectCollectionMapping tasksMapping = new XMLCompositeDirectCollectionMapping();
tasksMapping.setAttributeName("tasks");
tasksMapping.setXPath("task/text()");
```

Mapping to a Sequence in a Subelement

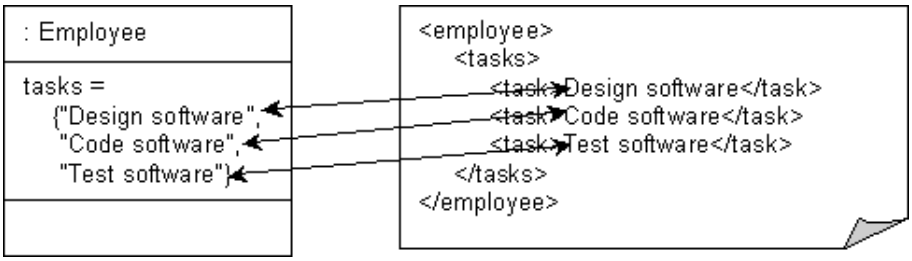
Given the XML schema in Example 62–26, Figure 62–16 illustrates a composite direct collection XML mapping to a sequence of text nodes in a subelement in a corresponding XML document. Example 62–27 shows how to configure this mapping in Java.

Example 62–26 Schema for Composite Direct Collection XML Mapping to a Subelement Sequence

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="employee" type="employee-type"/>
    <xsd:complexType name="employee-type">
        <xsd:sequence>
```

```
            <xsd:element name="tasks">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="task" type="xsd:string" maxOccurs="unbounded"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

Figure 62–16 Composite Direct Collection XML Mapping to a Subelement Sequence



Example 62–27 Java for Composite Direct Collection XML Mapping to a Subelement Sequence

```
XMLCompositeDirectCollectionMapping tasksMapping = new XMLCompositeDirectCollectionMapping();
tasksMapping.setAttributeName("tasks");
tasksMapping.setXPath("tasks/task/text()");
```

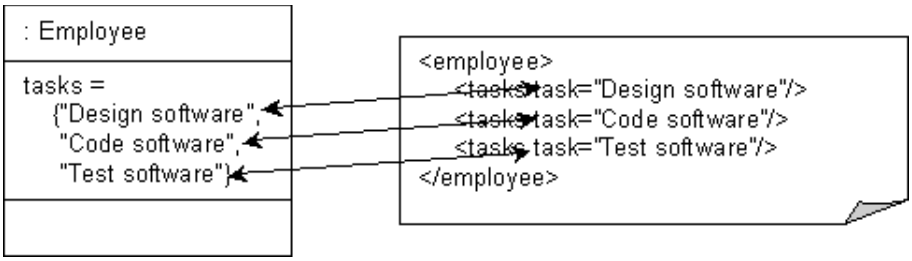
Mapping to Multiple Attributes

Given the XML schema in Example 62–28, Figure 62–17 illustrates a composite direct collection XML mapping to a sequence of text nodes in a subelement in a corresponding XML document. Example 62–29 shows how to configure this mapping in Java.

Example 62–28 Schema for Composite Direct Collection XML Mapping to Multiple Attributes

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="employee" type="employee-type"/>
    <xsd:complexType name="employee-type">
        <xsd:sequence>
            <xsd:element name="tasks" maxOccurs="unbounded">
                <xsd:complexType>
                    <xsd:attribute name="task" type="xsd:string"/>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

Figure 62–17 Composite Direct Collection XML Mapping to Multiple Attributes



Example 62–29 Java for Composite Direct Collection XML Mapping to Multiple Attributes

```
XMLCompositeDirectCollectionMapping tasksMapping = new XMLCompositeDirectCollectionMapping();
tasksMapping.setAttributeName("tasks/@task");
tasksMapping.setXPath("task/text()");
```

Mapping to a Single Text Node With an XML Composite Direct Collection Mapping

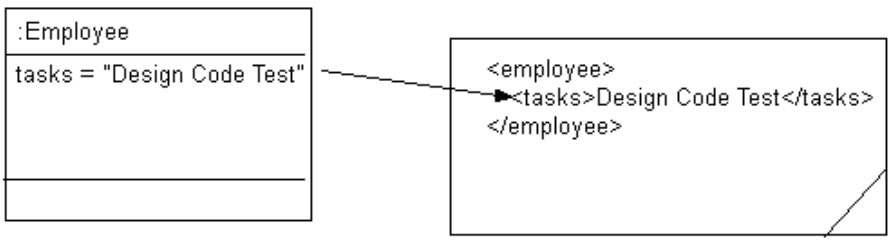
When you map a collection to a single node, the contents of the node is treated as a space-separated list.

Given the XML schema in Example 62–30, Figure 62–18 illustrates a composite direct collection XML mapping to a single text node in a corresponding XML document. Example 62–31 shows how to configure this mapping in Java.

Example 62–30 Schema for XML Composite Direct Collection Mapping to a Single Text Node

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="employee" type="employee-type"/>
    <xsd:complexType name="employee-type">
        <xsd:sequence>
            <xsd:element name="tasks" type="tasks-type"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:simpleType name="tasks-type">
        <xsd:list itemType="xsd:string"/>
    </xsd:simpleType>
</xsd:schema>
```

Figure 62–18 XML Composite Direct Collection Mapping to a Single Text Node



Example 62–31 Java for XML Composite Direct Collection Mapping to a Single Text Node

```
XMLCompositeDirectCollectionMapping tasksMapping = new XMLCompositeDirectCollectionMapping();
tasksMapping.setAttributeName("tasks");
```

```
tasksMapping.setXPath("tasks/text()");
tasksMapping.setUsesSingleNode(true);
```

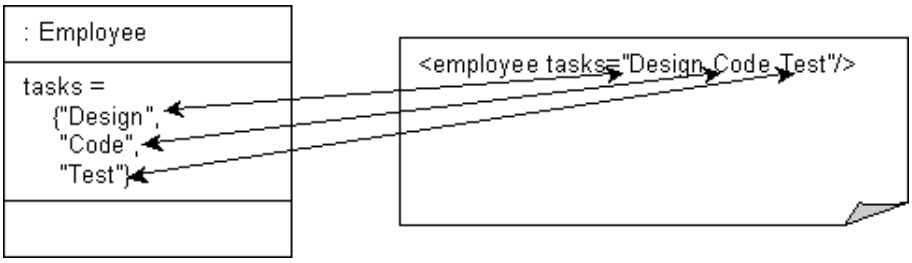
Mapping to a Single Attribute With an XML Composite Direct Collection Mapping

Given the XML schema in Example 62–32, Figure 62–19 illustrates a composite direct collection XML mapping to a single attribute in a corresponding XML document. Example 62–33 shows how to configure this mapping in Java.

Example 62–32 Schema for XML Composite Direct Collection Mapping to a Single Attribute

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="employee" type="employee-type"/>
    <xsd:complexType name="employee-type">
        <xsd:attribute name="tasks" type="tasks-type"/>
    </xsd:complexType>
    <xsd:simpleType name="tasks-type">
        <xsd:list itemType="xsd:string"/>
    </xsd:simpleType>
</xsd:schema>
```

Figure 62–19 XML Composite Direct Collection Mapping to a Single Attribute



Example 62–33 Java for XML Composite Direct Collection Mapping to a Single Attribute

```
XMLCompositeDirectCollectionMapping tasksMapping = new XMLCompositeDirectCollectionMapping();
tasksMapping.setAttributeName("tasks");
tasksMapping.setXPath("@tasks");
tasksMapping.setUsesSingleNode(true);
```

Mapping to a List of Unions With an XML Composite Direct Collection Mapping

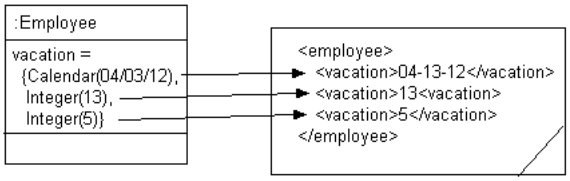
Given the XML schema in Example 62–34, Figure 62–20 illustrates a composite direct collection XML mapping to a list of unions in a corresponding XML document. Example 62–35 shows how to configure this mapping in Java.

Example 62–34 Schema for XML Composite Direct Collection Mapping to List of Unions

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="vacation" type="listOfUnions"/>
    <xsd:simpleType name="listOfUnions">
        <xsd:list>
            <xsd:simpleType>
                <xsd:union memberTypes="xsd:date xsd:integer"/>
            </xsd:simpleType>
        </xsd:list>
```

```
    </xsd:simpleType>
</xsd:schema>
```

Figure 62–20 Composite XML Direct Collection Mapping to List of Unions



Example 62–35 Java for XML Composite Direct Collection Mapping to List of Unions

```
XMLCompositeDirectCollectionMapping mapping = new XMLCompositeDirectCollectionMapping();
mapping.setAttributeName("myattribute");
XMLUnionField field = new XMLUnionField("listOfUnions/text()");
mapping.addSchemaType(new Qname(url,"int"));
mapping.addSchemaType(new Qname(url,"date"));
mapping.setField(field);
mapping.useSingleElement(false);
```

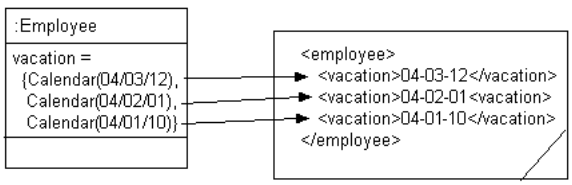
Mapping to a Union of Lists With an XML Composite Direct Collection Mapping

Given the XML schema in Example 62–34, Figure 62–20 illustrates an XML composite direct collection mapping to a list of unions in a corresponding XML document. Example 62–35 shows how to configure this mapping in Java.

Example 62–36 Schema for XML Composite Direct Collection Mapping to a Union of Lists

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="vacation" type="unionOfLists"/>
    <xsd:simpleType name="unionOfLists">
        <xsd:union memberTypes="xsd:double">
            <xsd:simpleType>
                <xsd:list itemType="xsd:date"/>
            </xsd:simpleType>
            <xsd:simpleType>
                <xsd:list itemType="xsd:integer"/>
            </xsd:simpleType>
        </xsd:union>
    </xsd:simpleType>
</xsd:schema>
```

Figure 62–21 XML Composite Direct Collection Mapping to a Union of Lists



Note that in this example, valid XML documents contain either all `xsd:double`, all `xsd:date`, or all `xsd:integer` values.

Example 62–37 Java for XML Composite Direct Collection Mapping to a Union of Lists

```
XMLCompositeDirectCollectionMapping mapping = new XMLCompositeDirectCollectionMapping();
mapping.setAttributeName("myattribute");
mapping.useSingleElement(false);
XMLUnionField unionField = new XMLUnionField("UnionOfLists/text()");
field.addSchemaType(new Qname(url," integer"))
field.addSchemaType (new Qname(url," date"))
field.addSchemaType (new Qname(url," double"))
field.setUsesSingleNode(false);
```

Specifying the Content Type of a Collection With an XML Composite Direct Collection Mapping

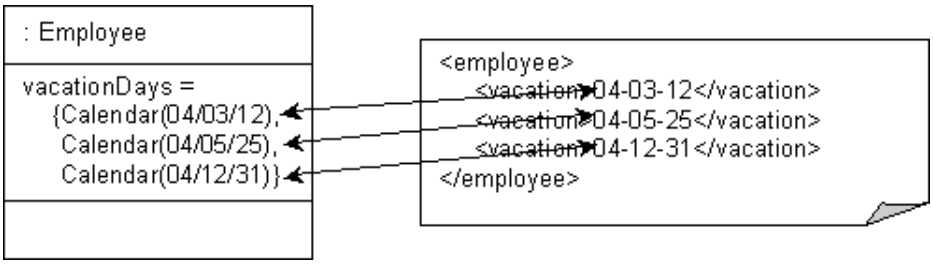
By default, TopLink will treat the node values read by a composite direct collection XML mapping as objects of type `String`. You can override this behavior by specifying the type of the collection's contents.

Given the XML schema in Example 62–38, Figure 62–22 illustrates an XML composite direct collection mapping to a simple sequence in a corresponding XML document. The mapping is configured to specify the content type of the collection as `Calendar`. Example 62–39 shows how to configure this mapping in Java.

Example 62–38 Schema for XML Composite Direct Collection Mapping with Specified Content Type

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="employee" type="employee-type"/>
    <xsd:complexType name="employee-type">
        <xsd:sequence>
            <xsd:element name="vacation" type="xsd:string" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

Figure 62–22 XML Composite Direct Collection Mapping with Specified Content Type



Example 62–39 Java for XML Composite Direct Collection Mapping with Specified Content Type

```
XMLCompositeDirectCollectionMapping tasksMapping = new XMLCompositeDirectCollectionMapping();
tasksMapping.setAttributeName("vacationDays");
tasksMapping.setXPath("vacation/text()");
tasksMapping.setAttributeElementClass(Calendar.class);
```

XML Composite Object Mapping

XML composite object mappings represent a relationship between two classes. In XML, the "owned" class may be nested with the element tag representing the "owning" class. You can use a composite object XML mapping in the following scenarios:

- Mapping Into the Parent Record
- Mapping to an Element
- Mapping to Different Elements by Element Name
- Mapping to Different Elements by Element Position

See Chapter 66, "Configuring an XML Composite Object Mapping" for more information.

Mapping Into the Parent Record

The composite object may be mapped to the same record as the parent.

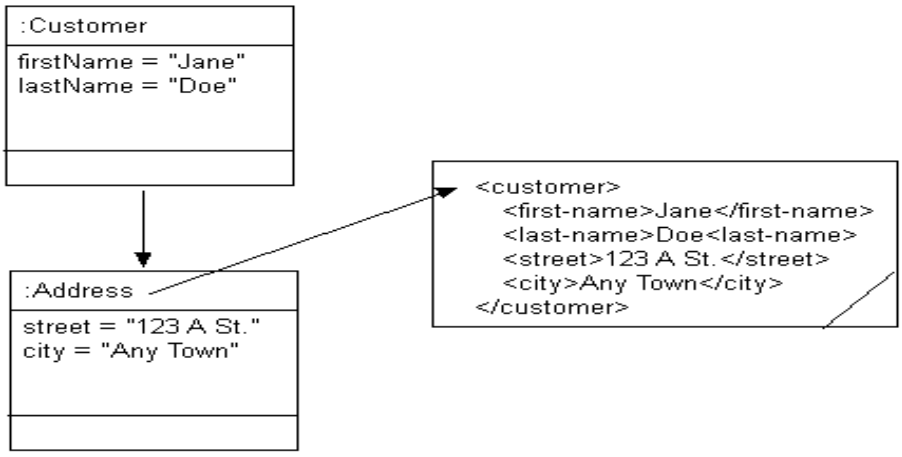
> **Note:** The nodes mapped to by the composite object must be sequential.

Given the XML schema in Example 62–40, Figure 62–23 illustrates an XML composite object mapping into the parent record in a corresponding XML document. Example 62–41 shows how to configure this mapping in Java.

Example 62–40 Schema for XML Composite Object Mapping into the Parent Record

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="customer" type="customer-type"/>
    <xsd:complexType name="customer-type">
        <xsd:sequence>
            <xsd:element name="first-name" type="xsd:string"/>
            <xsd:element name="last-name" type="xsd:string"/>
            <xsd:element name="street" type="xsd:string"/>
            <xsd:element name="city" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

Figure 62–23 XML Composite Object Mapping into the Parent Record



Example 62–41 Java for XML Composite Object Mapping into the Parent Record

```
XMLCompositeObjectMapping addressMapping = new XMLCompositeObjectMapping();
addressMapping.setAttributeName("address");
addressMapping.setXPath(".");
addressMapping.setReferenceClass(Address.class);
```

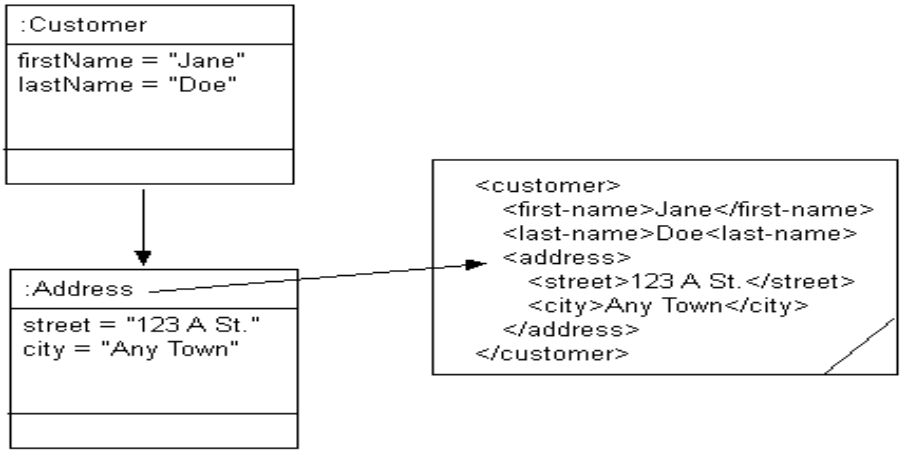
Mapping to an Element

Given the XML schema in Example 62–42, Figure 62–24 illustrates an XML composite object mapping to an element in a corresponding XML document. Example 62–43 shows how to configure this mapping in Java.

Example 62–42 Schema for XML Composite Object Mapping to an Element

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="customer" type="customer-type"/>
    <xsd:complexType name="customer-type">
        <xsd:sequence>
            <xsd:element name="first-name" type="xsd:string"/>
            <xsd:element name="last-name" type="xsd:string"/>
            <xsd:element name="address">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="street" type="xsd:string"/>
                        <xsd:element name="city" type="xsd:string"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

Figure 62–24 XML Composite Object Mapping to an Element



Example 62–43 Java for XML Composite Object Mapping to an Element

```
XMLCompositeObjectMapping addressMapping = new XMLCompositeObjectMapping();
addressMapping.setAttributeName("address");
addressMapping.setXPath("address");
addressMapping.setReferenceClass(Address.class);
```

Mapping to Different Elements by Element Name

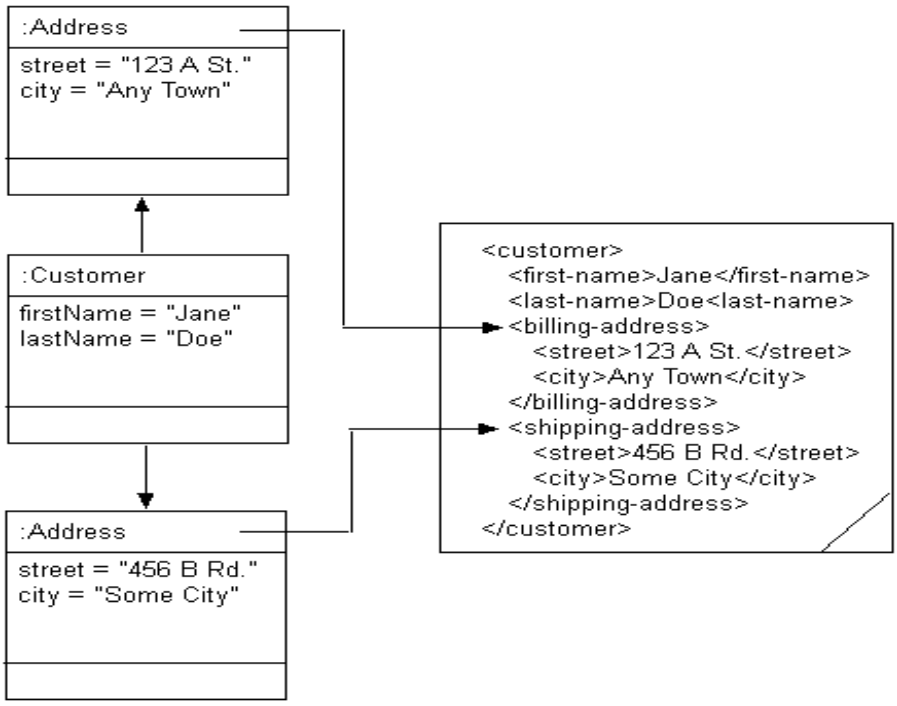
An object may have multiple composite object mappings to the same reference class. Each composite object mapping must have a unique XPath. This example uses unique XPaths by name .

Given the XML schema in Example 62–44, Figure 62–25 illustrates an XML composite object mapping to different elements by name in a corresponding XML document. Example 62–45 shows how to configure this mapping in Java.

Example 62–44 Schema for XML Composite Object Mapping to Elements by Name

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="customer" type="customer-type"/>
    <xsd:complexType name="customer-type">
        <xsd:sequence>
            <xsd:element name="first-name" type="xsd:string"/>
            <xsd:element name="last-name" type="xsd:string"/>
            <xsd:element name="billing-address" type="address-type"/>
            <xsd:element name="shipping-address" type="address-type"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="address-type">
        <xsd:sequence>
            <xsd:element name="street" type="xsd:string"/>
            <xsd:element name="city" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

Figure 62–25 XML Composite Object Mapping to Elements by Name



Example 62–45 Java for XML Composite Object Mapping to Elements by Name

```
XMLCompositeObjectMapping billingAddressMapping = new XMLCompositeObjectMapping();
billingAddressMapping.setAttributeName("billingAddress");
billingAddressMapping.setXPath("billing-address");
billingAddressMapping.setReferenceClass(Address.class);

XMLCompositeObjectMapping shippingAddressMapping = new XMLCompositeObjectMapping();
shippingAddressMapping.setAttributeName("shippingAddress");
shippingAddressMapping.setXPath("shipping-address");
shippingAddressMapping.setReferenceClass(Address.class);
```

Mapping to Different Elements by Element Position

An object may have multiple composite object mappings to the same reference class. Each composite object mapping must have a unique XPath. This example uses unique XPaths by position.

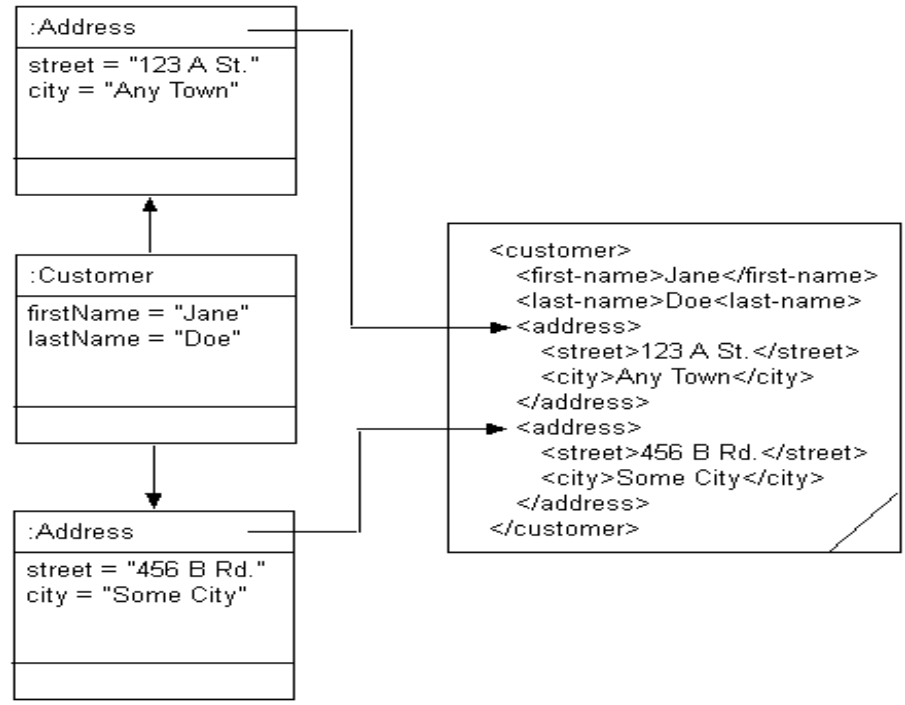
Given the XML schema in Example 62–44, Figure 62–25 illustrates an XML composite object mapping to different elements by position in a corresponding XML document. Example 62–45 shows how to configure this mapping in Java.

Example 62–46 Schema for XML Composite Object Mapping to Elements by Position

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="customer" type="customer-type"/>
    <xsd:complexType name="customer-type">
        <xsd:sequence>
            <xsd:element name="first-name" type="xsd:string"/>
            <xsd:element name="last-name" type="xsd:string"/>
            <xsd:element name="address" maxOccurs="2">
                <xsd:complexType>
                    <xsd:sequence>
```

```
                    <xsd:element name="street" type="xsd:string"/>
                    <xsd:element name="city" type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

Figure 62–26 XML Composite Object Mapping to Elements by Position



Example 62–47 Java for XML Composite Object Mapping to Elements by Position

```
XMLCompositeObjectMapping billingAddressMapping = new XMLCompositeObjectMapping();
billinAddressMapping.setAttributeName("billingAddress");
billinAddressMapping.setXPath("address[1]");
billinAddressMapping.setReferenceClass(Address.class);

XMLCompositeObjectMapping shippingAddressMapping = new XMLCompositeObjectMapping();
shippingAddressMapping.setAttributeName("shippingAddress");
shippingAddressMapping.setXPath("address[2]");
shippingAddressMapping.setReferenceClass(Address.class);
```

XML Composite Collection Mapping

Use XML composite collection mappings to represent one-to-many relationships. Composite collection XML mappings can reference any class that has a TopLink descriptor. The attribute in the object mapped must implement either the Java `Collection` interface (for example, `Vector` or `HashSet`) or `Map` interface (for example, `Hashtable` or `TreeMap`). The `CompositeCollectionMapping` class allows a reference to the mapped class and the indexing type for that class.

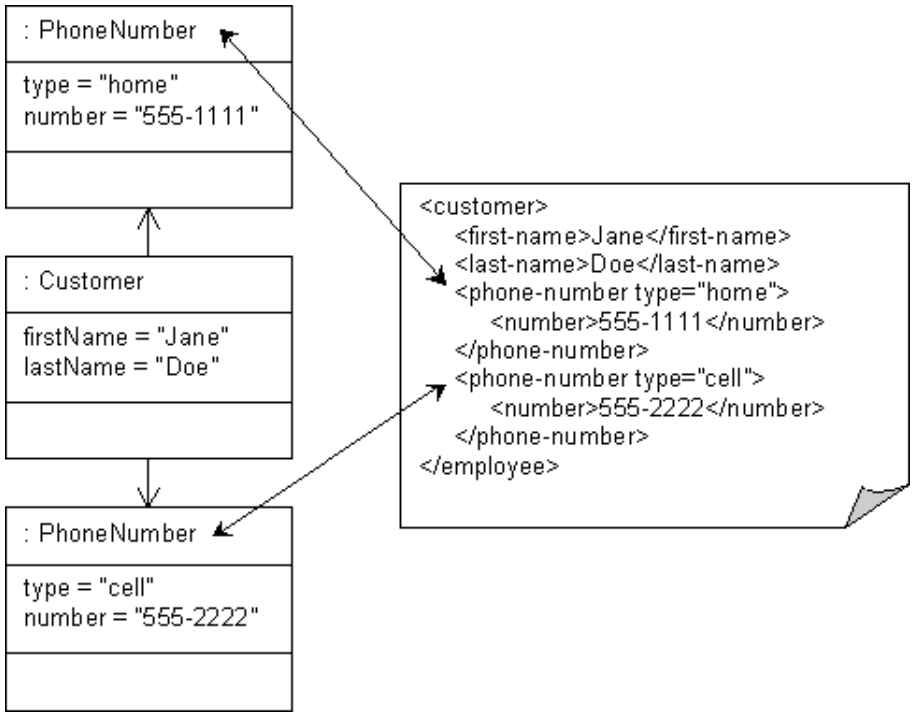
Given the XML schema in Example 62–48, Figure 62–27 illustrates an XML composite collection mapping to different elements by position in a corresponding XML

document. Example 62–49 shows how to configure this mapping in Java for a `Collection` attribute and Example 62–50 shows how to configure this mapping in Java for a `Map` attribute.

Example 62–48 Schema for XML Composite Collection Mapping

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="customer" type="customer-type"/>
    <xsd:complexType name="customer-type">
        <xsd:sequence>
            <xsd:element name="first-name" type="xsd:string"/>
            <xsd:element name="last-name" type="xsd:string"/>
            <xsd:element name="phone-number">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="number" type="xsd:string"/>
                    </xsd:sequence>
                    <xsd:attribute name="type" type="xsd:string"/>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

Figure 62–27 XML Composite Collection Mapping



Example 62–49 Java for XML Composite Collection Mapping for a Collection Attribute

```
XMLCompositeCollectionMapping phoneNumbersMapping = new XMLCompositeCollectionMapping();
phoneNumbersMapping.setAttributeName("phoneNumbers");
phoneNumbersMapping.setXPath("phone-number");
phoneNumbersMapping.setReferenceClass(PhoneNumber.class);
```

Example 62–50 Java for XML Composite Collection Mapping for a Map Attribute

```
XMLCompositeCollectionMapping phoneNumbersMapping = new XMLCompositeCollectionMapping();
phoneNumbersMapping.setAttributeName("phoneNumbers");
phoneNumbersMapping.setXPath("phone-number");
phoneNumbersMapping.setReferenceClass(PhoneNumber.class);
phoneNumbersMapping.useMapClass(HashMap.class, "getType");
```

See Chapter 67, "Configuring an XML Composite Collection Mapping" for more information.

XML Any Object Mapping

The XML any object mapping is similar to the composite object XML mapping (see "XML Composite Object Mapping" on page 62-21) except that the reference object may be of any type (including `String`). This type does not need to be related to any other particular type through inheritance or a common interface.

The corresponding object attribute value can be an instance of any object with a `Descriptor`, a `java.lang.Object`, a `java.lang.String`, a primitive object (such as `java.lang.Integer`), or a user defined type generic enough for all possible application values.

This mapping is useful with the following XML schema constructs:

- any
- choice
- substitution groups

Referenced objects can specify a default root element on their descriptor (see "Default Root Element" on page 23-9).

> **Note:** The undefined document root element of a referenced object is ignored during marshalling with an any collection mapping and object mapping.

Given the XML schema in Example 62–51, Figure 62–28 illustrates the Java classes used in this example. A single XML any object mapping is used to map `Customer` attribute `contactMethod`. This attribute must be generic enough to reference all possible values: in this example, instances of `Address`, `PhoneNumber`, and `String`.

Example 62–51 Schema for XML Any Object Mapping

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="customer" type="customer-type"/>
    <xsd:complexType name="customer-type">
        <xsd:sequence>
            <xsd:element name="contact-method" type="xsd:anyType"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="address">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="street" type="xsd:string"/>
                <xsd:element name="city" type="xsd:string"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="phone-number" type="xsd:string"/>
```

```
</xsd:schema>
```

Figure 62–28 Java Classes for XML Any Object Mapping

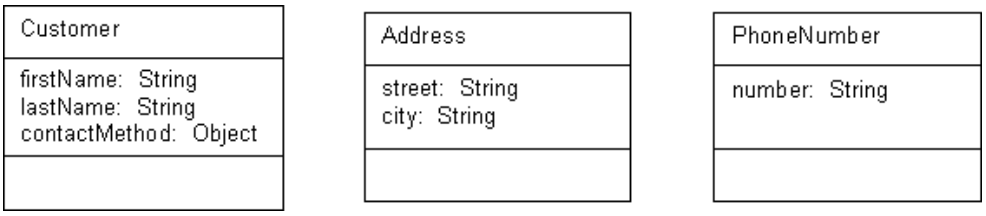


Figure 62–29, Figure 62–30, and Figure 62–31 illustrate how the XML any object mapping maps to an `Address`, `PhoneNumber`, and `String` (respectively) in XML documents that conform to the schema in Example 62–51.

Figure 62–29 XML Any Object Mapping to Address Type

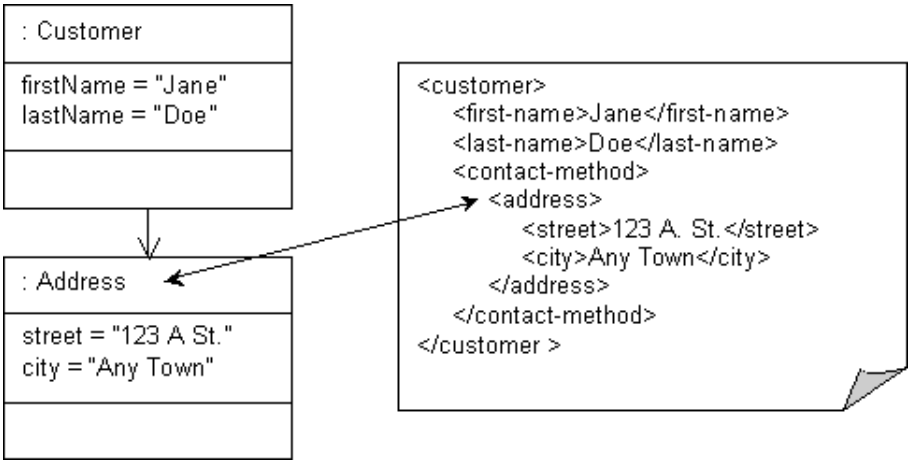


Figure 62–30 XML Any Object Mapping to PhoneNumber Type

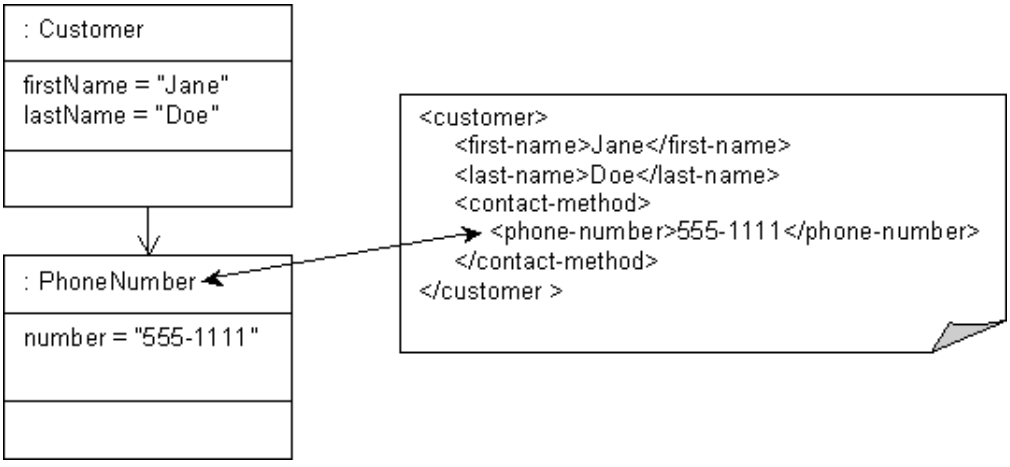
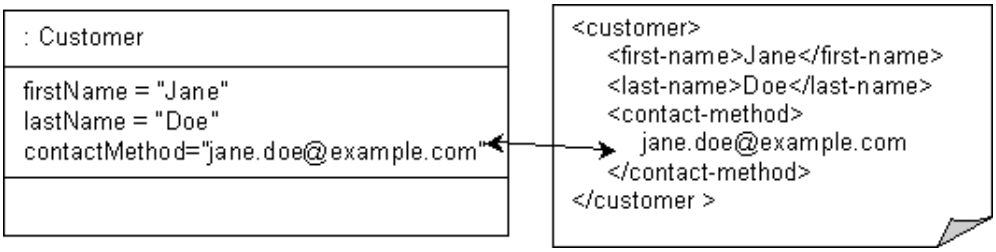


Figure 62–31 XML Any Object Mapping to String Type



Example 62–49 shows how to configure this mapping in Java.

Example 62–52 Java for XML Any Object Mapping

```
XMLAnyObjectMapping contactMethodMapping = new XMLAnyObjectMapping();
contactMethodMapping.setAttributeName("contactMethod");
contactMethodMapping.setXPath("contact-method");
```

For more information about TopLink XML mapping support for `xs:any` and `xs:anyType`, see "xs:any and xs:anyType Support" on page 62-4.

See Chapter 68, "Configuring an XML Any Object Mapping" for more information.

XML Any Collection Mapping

The XML any collection mapping is similar to the composite collection XML mapping (see "XML Composite Collection Mapping" on page 62-25), except that the referenced objects may be of different types (including `String`). These types need not be related to each other through inheritance or a common interface.

The corresponding object attribute value can be an instance of any object with a `Descriptor`, a `java.lang.Object`, a `java.lang.String`, a primitive object (such as `java.lang.Integer`), or a user defined type generic enough for all possible application values.

This mapping is useful with the following XML schema constructs:

- any
- choice
- substitution groups

Each of the referenced objects (except `String`) must specify a default root element on their descriptor (see "Default Root Element" on page 23-9).

Given the XML schema in Example 62–53, Figure 62–32 illustrates the Java classes used in this example. A single XML any collection mapping is used to map `Customer` attribute `contactMethods`. This attribute must be generic enough to reference all possible values: in this example, instances of `Address`, `PhoneNumber`, and `String`.

Example 62–53 Schema for XML Any Collection Mapping

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="customer" type="customer-type"/>
    <xsd:complexType name="customer-type">
        <xsd:sequence>
            <xsd:element name="contact-methods" type="xsd:anyType"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="address">
```

```
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="street" type="xsd:string"/>
                <xsd:element name="city" type="xsd:string"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="phone-number" type="xsd:string"/>
</xsd:schema>
```

Figure 62–32 Java Classes for XML Any Collection Mapping

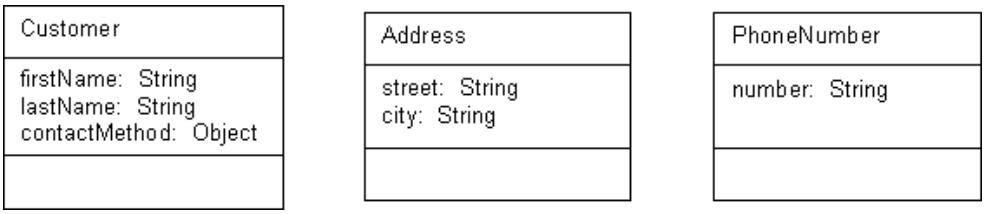
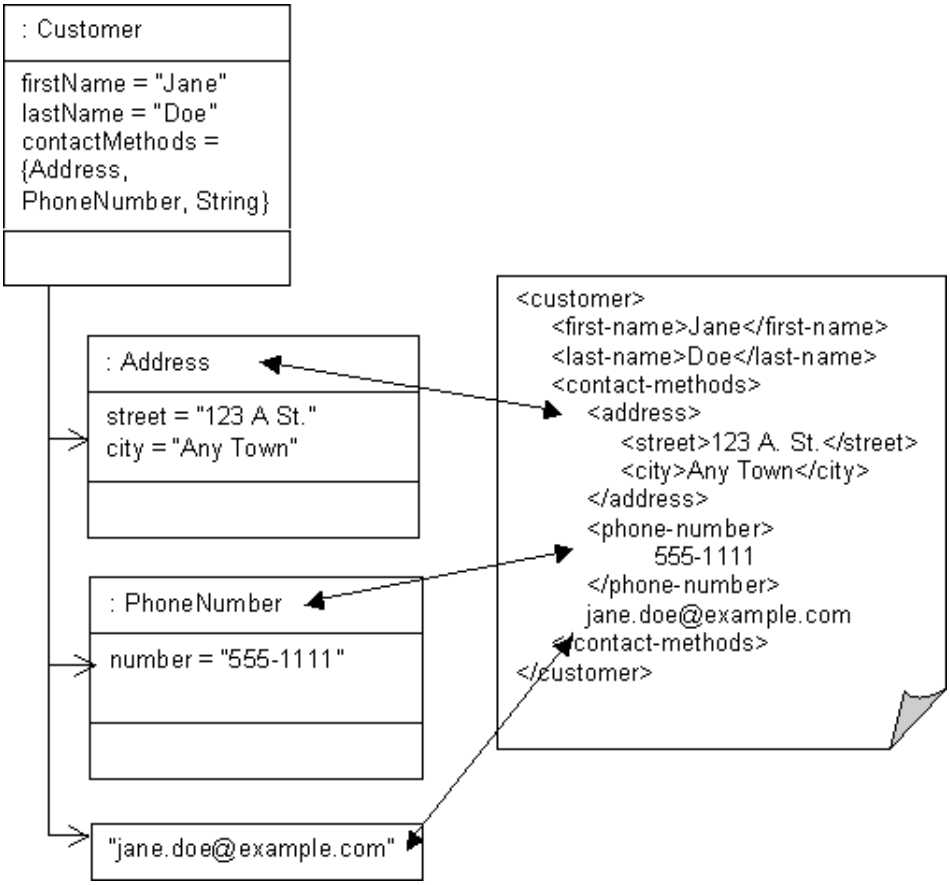


Figure 62–33 illustrate how the XML any collection mapping maps to a collection of `Address`, `PhoneNumber`, and `String` objects in an XML document that conforms to the schema in Example 62–53.

Figure 62–33 XML Any Collection Mapping



Example 62–54 shows how to configure this mapping in Java.

Example 62–54 Java for XML Any Collection Mapping

```
XMLAnyCollectionMapping contactMethodsMapping = new XMLAnyCollectionMapping();
contactMethodsMapping.setAttributeName("contactMethods");
contactMethodsMapping.setXPath("contact-methods");
```

For more information about TopLink XML mapping support for `xs:any` and `xs:anyType`, see "xs:any and xs:anyType Support" on page 62-4.

See Chapter 69, "Configuring an XML Any Collection Mapping" for more information.

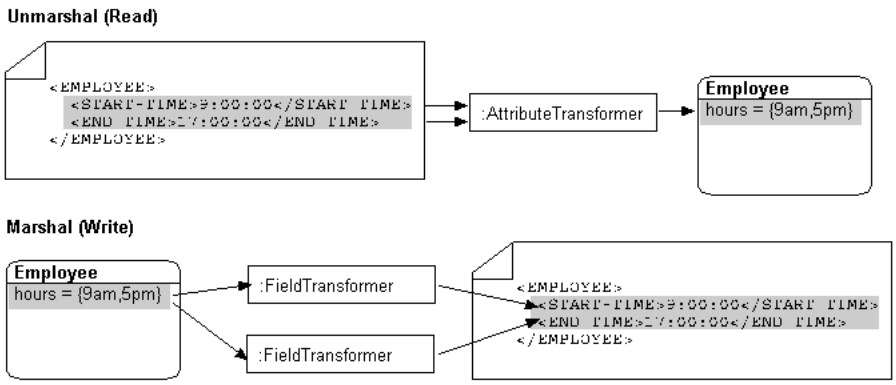
XML Transformation Mapping

You can use an XML transformation mapping to create a custom mapping where one or more XML nodes can be used to create the object to be stored in a Java class's attribute. To handle the custom requirements at marshall (write) and unmarshall (read) time, a transformation mapping takes instances of `oracle.toplink.mappings.transformers` (such as `AttributeTransformer` and `FieldTransformer`) that you provide. This provides a nonintrusive solution that avoids the need for your domain objects to implement special interfaces for this purpose.

As Figure 62–34 illustrates, you configure the transformation mapping with an `oracle.toplink.mappings.transformers.AttributeTransformer` instance to perform the XML instance-to-Java attribute transformation at unmarshall time. In this example, the `AttributeTransformer` combines two XML text nodes into a single Java object.

Similarly, you also configure the transformation mapping with one or more `oracle.toplink.mappings.transformers.FieldTransformer` instances to perform the Java attribute-to-XML instance transformation at marshall time. In this example, each `FieldTransformer` is responsible for mapping one of the Java object values to an XML text node.

Figure 62–34 XML Transformation Mappings



See Chapter 70, "Configuring an XML Transformation Mapping" for more information.

63

Configuring an XML Mapping

This chapter describes how to configure an XML mapping.

Table 63–1 lists the types of XML mappings that you can configure and provides a cross-reference to the type-specific chapter that lists the configurable options supported by that type.

Table 63–1 Configuring XML Mappings

If you are creating...	See Also...
XML Direct Mapping	Chapter 64, "Configuring an XML Direct Mapping"
XML Composite Direct Collection Mapping	Chapter 65, "Configuring an XML Composite Direct Collection Mapping"
XML Composite Object Mapping	Chapter 66, "Configuring an XML Composite Object Mapping"
XML Composite Collection Mapping	Chapter 67, "Configuring an XML Composite Collection Mapping"
XML Any Object Mapping	Chapter 68, "Configuring an XML Any Object Mapping"
XML Any Collection Mapping	Chapter 69, "Configuring an XML Any Collection Mapping"
XML Transformation Mapping	Chapter 70, "Configuring an XML Transformation Mapping"

Table 63–2 lists the configurable options shared by two or more XML mapping types.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "Understanding XML Mappings" on page 62-1
- "Understanding Descriptors" on page 23-1
- "Configuring an XML Descriptor" on page 29-1

Configuring Common XML Mapping Options

Table 63–2 lists the configurable options shared by two or more XML mapping types. In addition to the configurable options described here, you must also configure the options described for the specific XML Mapping Types, as shown in Table 63–1.

Table 63–2 Common Options for XML Mappings

Option	Type	TopLink Workbench	Java
"Configuring XPath" on page 32-10	Basic	✔	✔
"Configuring Reference Descriptor" on page 63-2	Basic	✔	✔

Table 63–2 (Cont.) Common Options for XML Mappings

Option	Type	TopLink Workbench	Java
"Configuring Container Policy" on page 32-26	Advanced	✔	✔
"Configuring Method Accessing" on page 32-14	Advanced	✔	✔
"Configuring Read-Only Mappings" on page 32-2	Advanced	✔	✔
"Configuring Maps to Wildcard" on page 63-3	Advanced	✔	
"Configuring a Serialized Object Converter" on page 32-18	Advanced	✔	✔
"Configuring a Type Conversion Converter" on page 32-20	Advanced	✔	✔
"Configuring an Object Type Converter" on page 32-22	Advanced	✔	✔
"Configuring a Simple Type Translator" on page 32-23	Advanced	✔	✔
"Configuring the Use of a Single Node" on page 32-36	Advanced	✔	✔

Configuring Reference Descriptor

For XML attributes that reference other descriptors (instead of a schema element), you must select a specific reference descriptor.

Table 63–3 summarizes which XML mappings support reference descriptor configuration.

Table 63–3 XML Mapping Support for Reference Descriptor Configuration

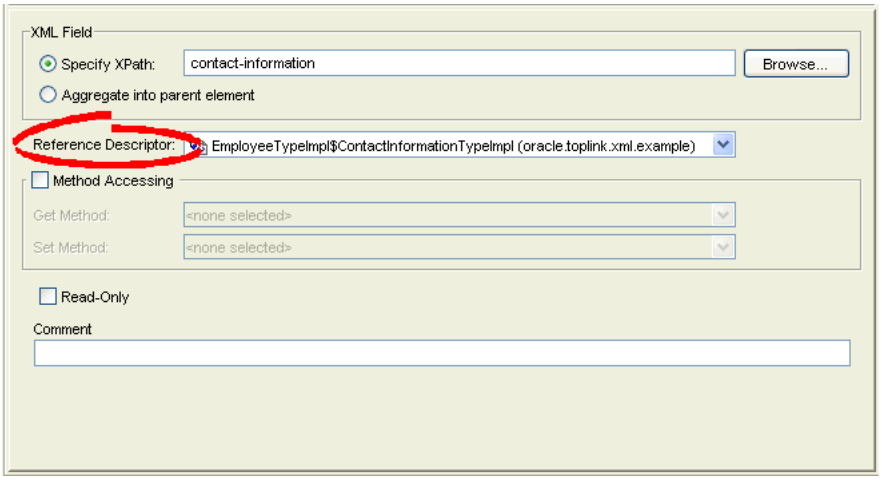
XML Mapping	Using TopLink Workbench	Using Java
XML Direct Mapping		
XML Composite Direct Collection Mapping		
XML Composite Object Mapping	✔	✔
XML Composite Collection Mapping	✔	✔
XML Any Object Mapping		
XML Any Collection Mapping		
XML Transformation Mapping		

Using TopLink Workbench

To specify a reference descriptor for an XML mapping that references another descriptor (instead of a schema element), use this procedure.

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.

Figure 63–1 General Tab, Reference Descriptor Field



If this XML attribute refers to another descriptor (instead of a schema element), use the **Reference Descriptor** field to select a descriptor in the project.

Configuring Maps to Wildcard

This attribute applies only to TopLink Workbench. Use this option to solve "No XPath specified" problems (see "Using the Problems Window" on page 4-11) for an XML mapping that does not need an XPath (see "Configuring XPath" on page 32-10) for it maps to a wildcard.

If the XML mapping is owned by an `anyType` descriptor (see "Configuring for Complex Type of anyType" on page 29-3), it cannot map to a wildcard, and you must specify an XPath.

Table 63–4 summarizes which XML mappings support maps to wildcard configuration.

Table 63–4 XML Mapping Support for Maps to Wildcard Configuration

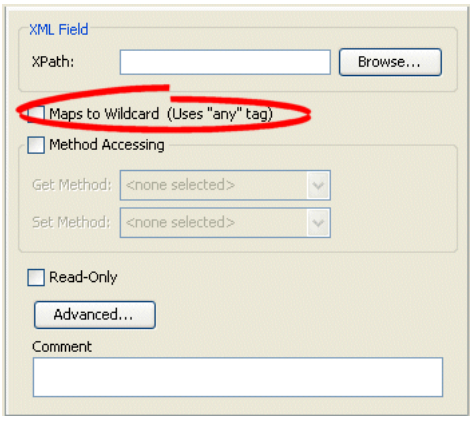
XML Mapping	Using TopLink Workbench	Using Java
XML Direct Mapping		
XML Composite Direct Collection Mapping		
XML Composite Object Mapping		
XML Composite Collection Mapping		
XML Any Object Mapping	✔	
XML Any Collection Mapping	✔	
XML Transformation Mapping		

Using TopLink Workbench

To specify a map a schema element using the `xs:any` declaration, use this procedure.

1. Select the mapped attribute in the **Navigator**. Its properties appear in the Editor.

Figure 63–2 Mapping Tab, Maps to Wildcard Option



If the XML mapping is not owned by an `anyType` descriptor (see "Configuring for Complex Type of anyType" on page 29-3) and maps to a wildcard, then you do not need to specify an XPath (see "Configuring XPath" on page 32-10). Select the **Maps to Wildcard (uses "any" tag)** option to clear the missing XPath neediness message.

If the XML mapping is owned by an `anyType` descriptor, it cannot map to a wildcard and you must specify an XPath. Deselect the **Maps to Wildcard (Uses "any" tag)** option and ensure that you specify an XPath.

64

Configuring an XML Direct Mapping

This chapter describes the various components that you must configure in order to use an XML direct mapping.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "XML Direct Mapping" on page 62-5

XML Direct Mapping Configuration Overview

Table 64–1 lists the configurable options for an XML direct mapping.

Table 64–1 Configurable Options for XML Direct Mapping

Option	Type	TopLink Workbench	Java
"Configuring XPath" on page 32-10	Basic	✔	✔
"Configuring a Simple Type Translator" on page 32-23	Advanced	✔	✔
"Configuring the Use of a Single Node" on page 32-36	Advanced		✔
"Configuring Method Accessing" on page 32-14	Advanced	✔	✔
"Configuring a Default Null Value at the Mapping Level" on page 32-12	Advanced	✔	✔
"Configuring Read-Only Mappings" on page 32-2	Advanced	✔	✔
"Configuring Mapping Comments" on page 32-18	Advanced	✔	✔
"Configuring a Serialized Object Converter" on page 32-18	Advanced	✔	✔
"Configuring a Type Conversion Converter" on page 32-20	Advanced	✔	✔
"Configuring an Object Type Converter" on page 32-22	Advanced	✔	
"Configuring a JAXB Typesafe Enumeration Converter" on page 32-25	Advanced		✔

65

Configuring an XML Composite Direct Collection Mapping

This chapter describes the various components that you must configure in order to use an XML composite direct collection mapping.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "XML Composite Direct Collection Mapping" on page 62-14

XML Composite Direct Collection Mapping Configuration Overview

Table 65–1 lists the configurable options for an XML direct collection mapping.

Table 65–1 Configurable Options for XML Direct Collection Mapping

Option	Type	TopLink Workbench	Java
"Configuring XPath" on page 32-10	Basic	✔	✔
"Configuring a Simple Type Translator" on page 32-23	Advanced	✔	✔
"Configuring the Use of a Single Node" on page 32-36	Advanced	✔	✔
"Configuring Method Accessing" on page 32-14	Basic	✔	✔
"Configuring Read-Only Mappings" on page 32-2	Basic	✔	✔
"Configuring Container Policy" on page 32-26	Advanced	✔	✔
"Configuring Mapping Comments" on page 32-18	Advanced	✔	
"Configuring a Serialized Object Converter" on page 32-18	Advanced	✔	✔
"Configuring a Type Conversion Converter" on page 32-20	Advanced	✔	✔
"Configuring an Object Type Converter" on page 32-22	Advanced	✔	✔
"Configuring a JAXB Typesafe Enumeration Converter" on page 32-25	Advanced		✔

66

Configuring an XML Composite Object Mapping

This chapter describes the various components that you must configure in order to use an XML composite object mapping.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "XML Composite Object Mapping" on page 62-21

XML Composite Object Mapping Configuration Overview

Table 66–1 lists the configurable options for an XML composite object mapping.

Table 66–1 Configurable Options for XML Composite Object Mapping

Option	Type	TopLink Workbench	Java
"Configuring XPath" on page 32-10	Basic	✔	✔
"Configuring Reference Descriptor" on page 63-2	Basic	✔	✔
"Configuring Method Accessing" on page 32-14	Advanced	✔	✔
"Configuring Read-Only Mappings" on page 32-2	Advanced	✔	✔
"Configuring Mapping Comments" on page 32-18	Advanced	✔	

67

Configuring an XML Composite Collection Mapping

This chapter describes the various components that you must configure in order to use an XML composite collection mapping.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "XML Composite Collection Mapping" on page 62-21

XML Composite Collection Mapping Configuration Overview

Table 67–1 lists the configurable options for an XML composite collection mapping.

Table 67–1 Configurable Options for XML Composite Collection Mapping

Option	Type	TopLink Workbench	Java
"Configuring XPath" on page 32-10	Basic	✔	✔
"Configuring Reference Descriptor" on page 63-2	Basic	✔	✔
"Configuring Method Accessing" on page 32-14	Advanced	✔	✔
"Configuring Read-Only Mappings" on page 32-2	Advanced	✔	✔
"Configuring Container Policy" on page 32-26	Advanced	✔	✔
"Configuring Mapping Comments" on page 32-18	Advanced	✔	

68

Configuring an XML Any Object Mapping

This chapter describes the various components that you must configure in order to use an XML any object mapping.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "XML Any Object Mapping" on page 62-27

XML Any Object Mapping Configuration Overview

Table 68–1 lists the configurable options for an XML any object mapping.

Table 68–1 Configurable Options for XML Any Object Mapping

Option	Type	TopLink Workbench	Java
"Configuring XPath" on page 32-10	Basic	✔	✔
"Configuring Maps to Wildcard" on page 63-3	Advanced	✔	
"Configuring Method Accessing" on page 32-14	Advanced	✔	✔
"Configuring Read-Only Mappings" on page 32-2	Advanced	✔	✔
"Configuring Mapping Comments" on page 32-18	Advanced	✔	

69

Configuring an XML Any Collection Mapping

This chapter describes the various components that you must configure in order to use an XML any collection mapping.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "XML Any Collection Mapping" on page 62-29

XML Any Collection Mapping Configuration Overview

Table 69–1 lists the configurable options for an XML any collection mapping.

Table 69–1 Configurable Options for XML Any Collection Mapping

Option	Advanced	TopLink Workbench	Java
"Configuring XPath" on page 32-10	Basic	✔	✔
"Configuring Maps to Wildcard" on page 63-3	Advanced	✔	
"Configuring Method Accessing" on page 32-14	Advanced	✔	✔
"Configuring Read-Only Mappings" on page 32-2	Advanced	✔	✔
"Configuring Container Policy" on page 32-26	Advanced	✔	✔
"Configuring Mapping Comments" on page 32-18	Advanced	✔	

70

Configuring an XML Transformation Mapping

This chapter describes the various components that you must configure in order to use an XML transformation mapping.

For more information, see the following:

- "Mapping Creation Overview" on page 31-1
- "XML Transformation Mapping" on page 62-31

XML Transformation Mapping Configuration Overview

Table 70–1 lists the configurable options for a XML transformation mapping.

Table 70–1 Configurable Options for XML Transformation Mapping

Option	Type	TopLink Workbench	Java
"Configuring Attribute Transformer" on page 32-29	Basic	✔	✔
"Configuring Field Transformer Associations" on page 32-31	Basic	✔	✔
"Configuring Method Accessing" on page 32-14	Advanced	✔	✔
"Configuring Read-Only Mappings" on page 32-2	Advanced	✔	✔
"Configuring Mutable Mappings" on page 32-33	Advanced	✔	✔
"Configuring Mapping Comments" on page 32-18	Advanced	✔	
"Configuring Indirection" on page 32-3	Advanced		✔

Part XIV

Using TopLink Overview

This part describes how to associate a TopLink project with a particular instance of a data source and use it to manage persistence in your application. It contains the following chapters:

- Chapter 71, "Understanding the Persistence Layer"

  This chapter provides an overview of how to use sessions, queries, and transactions in your application.

71

Understanding the Persistence Layer

This chapter describes the following:

- Overview of the Persistence Layer
- Sessions
- Data Access
- Cache
- Queries and Expressions
- Transactions

Overview of the Persistence Layer

The purpose of your application's persistence layer is to use a session (see "Sessions" on page 71-1) at run time to associate mapping metadata (see "Mapping Metadata" on page 2-13) and a data source (see "Data Access" on page 71-1) in order to create, read, update, and delete persistent objects using the TopLink cache (see "Cache" on page 71-2), queries and expressions (see "Queries and Expressions" on page 71-2), as well as transactions (see "Transactions" on page 71-3).

Sessions

A session is the primary interface between the client application and the TopLink runtime, and represents the connection to the underlying data source.

For non-CMP projects, TopLink offers several different session types (see "Understanding TopLink Sessions" on page 72-1), each optimized for different design requirements and architectures. The most commonly used session is the server session–a session that clients access on the server through a client session. The server session provides a shared cache and shared connection resources.

For CMP projects, the TopLink runtime creates and uses a session internally, but your application does not acquire or use this session directly. Depending on the application server you use, you can specify some of the parameters for this internal session (see "<J2EE-Container>-ejb-jar.xml File" on page 8-5).

Data Access

The login (if any) associated with a session determines how the TopLink runtime connects to the project's data source.

A login includes details of data source access, such as authentication, use of connection pools, and use of external transaction controllers. A login (an instance of `Login` interface) owns a data source platform.

A platform includes options specific to a particular data source, including such as binding, use of native SQL, use of batch writing, and sequencing. For more information about platforms, see "Data Source Platform Types" on page 81-3.

For projects that do not persist to a data source, a login is not required. For projects that do persist to a data source, a login is always required.

For relational and For more information, see "Understanding Data Access" on page 81-1

Cache

By default, a TopLink session provides an object level cache that guarantees object identity and enhances performance by reducing the number of times the application needs to access the data source. TopLink provides a variety of cache options, including locking, refresh, invalidation, isolation, and coordination. Using cache coordination, you can configure TopLink to synchronize changes with other instances of the deployed application. You configure most cache options at the session level. You can also configure cache options on a per-query basis, or on a descriptor to apply to all queries on the reference class.

For more information, see "Understanding the Cache" on page 87-1

Queries and Expressions

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

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

With these options, developers can build any type of query. Oracle recommends using 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.

When using EJB entity beans, you can code finders using only EJB QL (in addition to any of the other TopLink query options), enabling the application to comply with the J2EE specification.

Regardless of the architecture or persistent entity type, you are free to use any of the query options. TopLink Workbench provides the simplest way to define queries. Alternatively, you can build queries in code, using the TopLink API.

For more information, see the following:

- "Understanding TopLink Queries" on page 93-1
- "Understanding TopLink Expressions" on page 95-1

Transactions

TopLink provides the ability to write transactional code isolated from the underlying database and schema by using a **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 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 coordinated cache.

If an application uses EJB entity beans, you do not access the unit of work API directly, but you still benefit from its features: the integration between the TopLink runtime and the J2EE container automatically uses the unit of work.

For more information, see "Understanding TopLink Transactions" on page 97-1.

Part XV

TopLink Sessions

This part describes the TopLink artifact used to associate a TopLink project with a particular instance of a data source. It contains the following chapters:

- Chapter 72, "Understanding TopLink Sessions"

  This chapter describes each of the different TopLink session types and important session concepts.

- Chapter 73, "Creating Sessions"

  This chapter contains procedures for creating TopLink sessions.

- Chapter 74, "Configuring a Session"

  This chapter explains how to configure TopLink session options common to two or more session types.

- Chapter 75, "Acquiring and Using Sessions at Run Time"

  This chapter explains how to acquire and use a TopLink session at runtime.

- Chapter 76, "Configuring Server Sessions"

  This chapter explains how to configure TopLink server and client sessions.

- Chapter 77, "Configuring Exclusive Isolated Client Sessions for Virtual Private Database"

  This chapter explains how to configure a TopLink isolated client session.

- Chapter 78, "Configuring Historical Sessions"

  This chapter explains how to configure a TopLink historical session.

- Chapter 79, "Configuring Session Broker and Client Sessions"

  This chapter explains how to configure TopLink session broker and client sessions.

- Chapter 80, "Configuring Database Sessions"

  This chapter explains how to configure a TopLink database session suitable for simple single-user, single-data source and prototyping applications.

72

Understanding TopLink Sessions

A TopLink session provides the primary access to the TopLink runtime. It is the means by which your application performs all persistence operations with the data source that contains persistent objects.

A session associates data source platform information, data source login information, and mapping metadata for a particular application. You can reuse mapping metadata in different applications by defining different sessions.

TopLink provides different session types, each optimized for different design requirements and data access strategies. You can combine different session types in the same application.

This chapter explains the following:

- Session Types
- Session Concepts
- Sessions and the Cache
- Understanding the Session API

Session Types

Table 72–1 lists the session types that you can use in a non-CMP TopLink application and classifies them as basic or advanced. See "Sessions and CMP" on page 72-13 for information on using Oracle TopLink with CMP.

Table 72–1 TopLink Session Types

Session Type	Description	Type	TopLink Workbench	Java
Server and Client Sessions	Server sessions provide session management to a single data source (including shared object cache and connection pools) for multiple clients in a three-tier architecture using database or EIS platforms. This is the most flexible, scalable, and commonly used session. You acquire a client session from a server session at run time to provide access to a single data source for each client.	Basic	✔	✔
Unit of Work Sessions	Acquired from any session type (directly, or by way of an external transaction controller) to transactionally modify objects.	Basic		✔
Isolated Client Sessions	A special type of client session that uses a session cache isolated from the shared object cache of its parent server session.	Advanced		✔
Historical Sessions	A special type of client session that provides a read-only snapshot of object versions as of a specified time and uses a session cache isolated from the shared object cache of its parent server session.	Advanced		✔

Table 72–1 (Cont.) TopLink Session Types

Session Type	Description	Type	TopLink Workbench	Java
Session Broker and Client Sessions	Provides session management to multiple data sources for multiple clients by aggregating two or more server sessions (can also be used with database sessions). You acquire a client session from a session broker at run-time to provide access to all the data sources managed by the session broker for each client.	Advanced	✔	✔
Database Sessions	Provides session management to a single database for a single client suitable for simple or two-tiered applications. Oracle does not recommend this session type in three-tiered applications because it does not offer the same flexibility and scalability as the server session.	Basic	✔	✔
Remote Sessions	A client-side session that communicates over RMI with a corresponding dedicated client session and shared server session. Remote sessions handle object identity and marshalling and unmarshalling between client-side and server-side.	Advanced		✔

For more information, see the following:

- "Creating Sessions" on page 73-1
- "Configuring a Session" on page 74-1
- "Acquiring and Using Sessions at Run Time" on page 75-1

Session Concepts

This section describes concepts unique to TopLink sessions, including the following:

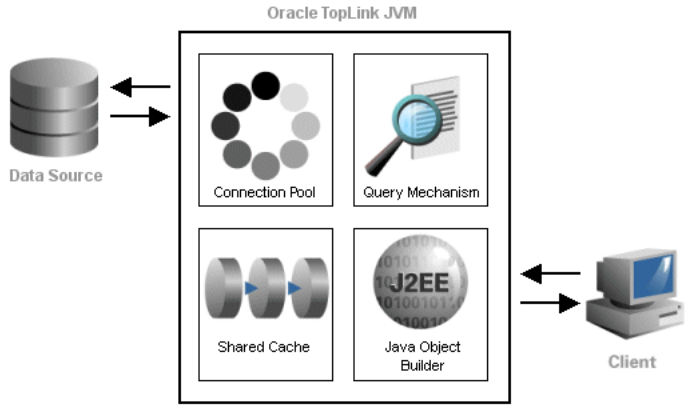
- Session Architecture
- Session Configuration and the sessions.xml File
- Session Customization
- Acquiring a Session at Run Time With the Session Manager
- Managing Session Events With the Session Event Manager
- Logging
- Profiler
- Integrity Checker
- Exception Handlers
- Registering Descriptors
- Sessions and CMP
- Sessions and Sequencing

Session Architecture

As Figure 72–1 illustrates, a session instance is composed of the following components:

- Object Cache
- Connection Pools
- Query Mechanism
- Java Object Builder

Figure 72–1 Simple TopLink Session Architecture



How these session components are implemented and how they interact depends on the type of session. For example, for server and client sessions, the server session provides a connection pool and shared object cache on behalf of all client sessions acquired from it.

Object Cache

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 a TopLink application.

Typically, a server session's object cache is shared by all client sessions acquired from it. That is, for a Server session `myServerSession`, each client session acquired by calling server session method `acquireClientSession` shares the same object cache as `myServerSession`.

Isolated and historical sessions provide their own session cache isolated from the shared object cache of their parent server session. For more information, see "Isolated Client Sessions" on page 72-19 and "Historical Sessions" on page 72-25.

You can easily manage concurrent access to this shared cache by using a unit of work session acquired from any session. For more information, see "Unit of Work Sessions" on page 72-19.

For more information, see "Sessions and the Cache" on page 72-33.

Connection Pools

A **connection pool** is a collection of reusable connections to a single data source.

> **Note:** To simultaneously access multiple databases from within a single session, use a session broker. For more information, see "Session Broker and Client Sessions" on page 72-26.

Because creating a data source connection is usually expensive, a properly configured connection pool significantly improves performance.

You can configure your session to use internal connection pools provided by TopLink or external connection pools provided by a JDBC driver or J2EE container. By default, TopLink uses internal connection pools.

Internal connection pools are usually used in non-EJB applications, or when an external transaction controller (JTA) is not used. If you configure your session to use internal connection pools, you can configure its default read and write connection pools. You can create special purpose connection pools for application-specific purposes (named connection pools) or exclusively for sequencing (sequence connection pool). For more information, see "Internal Connection Pools" on page 81-7.

External connection pools are usually used in EJB applications and when an external transaction controller (JTA) is used. For more information, see "External Connection Pools" on page 81-8.

For more information about data access configuration in general, see "Understanding Data Access" on page 81-1.

Query Mechanism

At run time, your application uses a session to perform all persistence operations: creating, reading, updating, and deleting objects. You perform these operations using TopLink queries and expressions with the session query API.

For more information, see "Understanding TopLink Queries" on page 93-1.

Java Object Builder

When you use object-level read queries, TopLink automatically builds Java objects from the data retrieved. When you use object-level write queries, TopLink automatically converts the affected Java objects into the appropriate data native to your data source.

Session Configuration and the sessions.xml File

TopLink provides two ways to configure your sessions: through Java code using the `Session` API, or using TopLink Workbench to build a session configuration file, the `sessions.xml` file.

In most cases, you configure sessions for the application using the `sessions.xml` file. This 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.

Oracle recommends that you use the `sessions.xml` file to deploy a TopLink application, because it provides the following advantages:

- It is easy to create and maintain in TopLink Workbench.
- It is easy to troubleshoot.
- It provides access to most session configuration options.
- It offers excellent flexibility, including the ability to modify deployed applications without recompiling.

For more information on creating a session in the `sessions.xml` file, see "Session Creation Overview" on page 73-1.

Session Customization

You can customize a session at run time by specifying a session customizer–a Java class that implements the `oracle.toplink.tools.sessionconfiguration.SessionCustomizer` interface and provides a default (zero-argument) constructor.

Since the `sessions.xml` file is not used for CMP projects, use `oracle.toplink.ejb.cmp.DeploymentCustomization` interface as a customizer when creating a CMP project to specify your database login information.

You use a session customizer to customize a session at run time through code API similar to how you use an amendment method to customize a descriptor (see "Amendment and After-Load Methods" on page 23-5).

For more information, see "Configuring Customizer Class" on page 74-13.

Acquiring a Session at Run Time With the Session Manager

The TopLink session manager enables developers to build a series of sessions that are maintained under a singleton object called the session manager.

The session manager is a static utility class that loads TopLink sessions from the `sessions.xml` file (see "Session Configuration and the sessions.xml File" on page 72-4), caches the sessions by name in memory, and provides a single access point for TopLink sessions.

At run time, TopLink will attempt to load the `sessions.xml` file from the two following default resource names: `sessions.xml` and `META-INF/sessions.xml`. Refer to Chapter 9, "Packaging a TopLink Application" for additional information.

The session manager supports the following session types:

- `ServerSession` (see "Server and Client Sessions" on page 72-14)
- `SessionBroker` (see "Session Broker and Client Sessions" on page 72-26)
- `DatabaseSession` (see "Database Sessions" on page 72-29)

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.

The session manager instantiates sessions as follows:

1. The client application requests 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.

Once you have a session instance, you can use it to acquire additional types of sessions for special tasks. For example, you can acquire a unit of work from any session to perform transactional operations. You can acquire a client session from a server session to perform client operations in a three-tier architecture.

For more information, see "Acquiring and Using Sessions at Run Time" on page 75-1.

Managing Session Events With the Session Event Manager

Sessions raise **session events** for most session operations. Session events help you debug or coordinate the actions of multiple sessions.

The session event manager handles information about session events. Applications register session event listeners with the session event manager to receive session events.

For example, session event listeners play an important role in the configuration of isolated sessions (see "Configuring Exclusive Isolated Client Sessions for Virtual

Private Database" on page 77-1). In an isolated session, if the TopLink runtime raises a `SessionEvent.NoRowsModified` event, it is handled by your `SessionEventListener` (see "NoRowsModifiedSessionEvent Event Handler" on page 77-3). This event listener is your opportunity to determine whether the update failure was due to a security violation (in which case you should not retry the operation) or due to an optimistic lock issue (in which case a retry may be appropriate). See "Logging" on page 72-7 for information on adding logging to your event listeners.

Another example is the use of session event listeners to configure proxy authentication in an Oracle Database. (see "Configuring Oracle Database Proxy Authentication" on page 83-12).

This section explains how to use session events, including the following:

- Session Event Manager Events
- Session Event Listeners

Session Event Manager Events

The session event manager supports the session events listed in the following tables:

- Table 72–2, " Session Events"
- Table 72–3, " Unit of Work Events"

Table 72–2 Session Events

Event	Description
MissingDescriptor	Raised if a descriptor is missing for a class being persisted. You can use this event to lazy register the descriptor or set of descriptors.
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.
NoRowsModified	Raised after update or delete SQL has been sent to the database and a row count of zero is returned.
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.
PostAcquireClientSession	Raised after a client Session is acquired
PostAcquireConnection	Raised after acquiring a connection
PostAcquireExclusiveConnection	Raised when a client Session, with isolated data, acquires an exclusive connection.
PostBeginTransaction	Raised after a database transaction starts
PostCommitTransaction	Raised after a database transaction commits
PostConnect	Raised after connecting to the database
PostExecuteQuery	Raised after the execution of every query on the session
PostLogin	Raised after the Session initializes and acquires connections
PostReleaseClientSession	Raised after releasing a client Session
PostRollbackTransaction	Raised after a database transaction rolls back
PreBeginTransaction	Raised before a database transaction starts
PreCommitTransaction	Raised before a database transaction commits
PreExecuteQuery	Raised before the execution of every query on the session
PreLogin	Raised before the Session initializes and acquires connections
PreReleaseClientSession	Raised before releasing a client Session

Table 72–2 (Cont.) Session Events

Event	Description
PreReleaseConnection	Raised before releasing a connection
PreReleaseExclusiveConnection	Raised before a client Session, with isolated data, releases its exclusive connection.
PreRollbackTransaction	Raised before a database transaction rolls back

Table 72–3 Unit of Work Events

Event	Description
PostAcquireUnitOfWork	Raised after a `UnitOfWork` is acquired
PostCalculateUnitOfWorkChangeSet	Raised after the commit has begun on the `UnitOfWork` and after the changes are calculated. The `UnitOfWorkChangeSet`, at this point, will contain change sets without the version fields updated and without identity field type primary keys. These will be updated after the insert, or update, of the object.
PostCommitUnitOfWork	Raised after a `UnitOfWork` commits
PostDistributedMergeUnitOfWorkChangeSet	Raised after a `UnitOfWork` change set has been merged when that change set has been received from a distributed session.
PostMergeUnitOfWorkChangeSet	Raised after a `UnitOfWork` change set has been merged.
PostReleaseUnitOfWork	Raised on a `UnitOfWork` after it is released.
PostResumeUnitOfWork	Raised on a `UnitOfWork` after it resumes.
PreCalculateUnitOfWorkChangeSet	Raised after the commit has begun on the `UnitOfWork` but before the changes are calculated.
PreCommitUnitOfWork	Raised before a `UnitOfWork` commits.
PreDistributedMergeUnitOfWorkChangeSet	Raised before a `UnitOfWork` change set has been merged when that change set has been received from a distributed session.
PreMergeUnitOfWorkChangeSet	Raised before a `UnitOfWork` change set has been merged.
PrepareUnitOfWork	Raised after the a `UnitOfWork` flushes its SQL, but before it commits its transaction.
PreReleaseUnitOfWork	Raised on a `UnitOfWork` before it is released.

Session Event Listeners

You can create session event listeners in two ways: either by implementing the `SessionEventListener` interface, or by extending the `SessionEventAdapter` class.

To register a `SessionEventListener` for session events, register it with a session using the `SessionEventManager` method `addListener`.

For more information, see "Configuring Session Event Listeners" on page 74-17.

Logging

You can configure a session to write run-time information to a TopLink log. This information includes status, diagnostic, SQL, and, when profiling is enabled, performance data (see "Measuring TopLink Performance With the TopLink Profiler" on page 11-2 or "Measuring TopLink Performance With the Oracle Dynamic Monitoring System (DMS)" on page 11-4).

Logging options are configurable at the session level (see "Configuring Logging" on page 74-4).

Note: To facilitate the debugging, you can add logging to your listeners to only log the events that are of the interest to your application. Within the session context, use the following logging utility:

```
Session.getSessionLog().log(int level, String message)
```

Without the session context, use the following logging utility:

```
AbstractSessionLog.getLog().log(int level, String message)
```

Both the `getSessionLog` and `getLog` methods return a session log (an instance of a `SessionLog` interface) loaded with an accessor's log messages and SQL. Then the session log performs logging at the level that you specify.

For more information on session event listeners, see "Session Event Listeners" on page 72-7.

This section describes session log options, including the following:

- Log Types
- Log Output
- Log Level
- Logging SQL
- Logging Chained Exceptions
- Viewing TopLink Log Messages From the Application Server Control Console

Log Types

TopLink supports the following types of logging:

- TopLink Native Logging
- java.util Logging
- Server Logging

For a non-CMP application, you can configure the log type using TopLink Workbench (see "Using TopLink Workbench" on page 74-5). For a CMP application, see "Configuring Logging in a CMP Application" on page 74-9.

TopLink Native Logging

TopLink native logging is the default session log type. It is provided by `oracle.toplink.logging.DefaultSessionLog`. Example 72–1 shows a typical TopLink native log message.

You can configure TopLink native logging options using TopLink Workbench (see "Using TopLink Workbench" on page 74-5).

Example 72–1 Sample TopLink Log Message

```
[TopLink Info]: DATE
TIME-DatabaseSession(12345)-Thread(12345)-TopLink, version: Oracle TopLink - 10g (Build
031203)
[TopLink Config]: DATE
TIME-DatabaseSession(12345)-Thread(12345)-Connection(12345)-
```

```
  connecting(DatabaseLogin(
              platform=>Oracle9Platform
              user name=> "username"
              datasource URL=> "jdbc:oracle:thin:@144.23.214.115:1521:toplink"
))
[TopLink Config]: DATE
TIME-DatabaseSession(12345)-Thread(12345)-Connection(12345)-
  Connected: jdbc:oracle:thin:@144.23.214.115:1521:toplink
  User: USERNAME
  Database: Oracle Version: Oracle9i Enterprise Edition - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.3.0 - Production
  Driver: Oracle JDBC driver Version: 9.2.0.3.0
[TopLink Info]: DATE
TIME-DatabaseSession(12345)-Thread(12345)-loggingTestSession login
successful
```

java.util Logging

This type of logging makes TopLink conform to the `java.util.logging` package. It is provided by `oracle.toplink.logging.JavaLog`. Logging options are configured in the `<JRE_HOME>/lib/logging.properties` file. Messages are written to any number of destinations based on this configuration. Example 72–2 shows a typical `java.util.logging` log message.

For more information on using `java.util.logging` package, see "Configuring a Session to use java.util.logging Package" on page 74-8.

Example 72–2 Sample java.util.logging Log Messages

```
Dec 9, 2003 2:05:05 PM oracle.toplink.loggingTestSession DatabaseSession(32603767) Thread(10)
INFO: TopLink, version: Oracle TopLink - 10g (10.0.3) Developer Preview (Build 031203)
Dec 9, 2003 2:05:07 PM oracle.toplink.loggingTestSession.connection DatabaseSession(32603767)
Connection(927929) Thread(10)
CONFIG: connecting(DatabaseLogin(
    platform=>Oracle9Platform
    user name=> "coredev8"
    datasource URL=> "jdbc:oracle:thin:@144.23.214.115:1521:toplink"
))
Dec 9, 2003 2:05:08 PM oracle.toplink.loggingTestSession.connection DatabaseSession(32603767)
Connection(927929) Thread(10)
CONFIG: Connected: jdbc:oracle:thin:@144.23.214.115:1521:toplink
    User: COREDEV8
    Database: Oracle  Version: Oracle9i Enterprise Edition Release 9.2.0.3.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.3.0 - Production
    Driver: Oracle JDBC driver  Version: 9.2.0.3.0
Dec 9, 2003 2:05:08 PM oracle.toplink.loggingTestSession DatabaseSession(32603767) Thread(10)
INFO: loggingTestSession login successful
```

Server Logging

Server logging is used to integrate TopLink logging with an application server log.

The TopLink runtime determines the server log type to use given the server platform you configure when you create your project ("Project Creation Overview" on page 18-1).

For example, if your project uses the OC4J platform, TopLink uses the `oracle.toplink.platform.server.oc4j.OjdlLog`; if your project uses the BEA WebLogic platform, TopLink uses the `oracle.toplink.platform.server.wls.WlsLog`.

Log Output

If you are using TopLink native logging, you can configure TopLink to write log messages to a file or to the console (see "Configuring Logging" on page 74-4).

If you are using `java.util.logging`, TopLink writes log messages to the destinations you configure in the `<JRE_HOME>/lib/logging.properties` file (see "Configuring a Session to use java.util.logging Package" on page 74-8).

If you are using server logging, TopLink writes log messages to the application server's log file (there is no separate TopLink log file in this case).

Log Level

You can control the amount and detail of log output by configuring the log level (in ascending order of information) in the following way:

- `SEVERE`–Logs exceptions indicating TopLink cannot continue, as well as any exceptions generated during login. This includes a stack trace.
- `WARNING`–Logs exceptions that do not force TopLink to stop, including all exceptions not logged with severe level. This does not include a stack trace.
- `INFO` (default)–Logs the login/logout per server session, including the user name. After acquiring the session, detailed information is logged.
- `CONFIG`–Logs only login, JDBC connection, and database information.
- `FINE`–Logs SQL (including thread information).
- `FINER`–Similar to warning. Includes stack trace.
- `FINEST`–Includes additional low level information
- `ALL`–Logs everything.

By default, TopLink logs at the `oracle.toplink.logging.SessionLog.INFO` level so that some information is logged by default.

At run time, set the log level using `Session` method `setLogLevel`, passing in one of the log level constants provided by `oracle.toplink.logging.SessionLog`.

Logging SQL

In a relational project, 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, set the session log level to `oracle.toplink.logging.SessionLog.FINE`–the session will log all executed SQL to the session log.

Example 72–3 shows how to configure the log destination using the `setLog()` method on the session.

Example 72–3 Configuring the Log Destination

```
private static SessionEventListener buildListener() {
    return new SessionEventAdapter() {
        public void preLogin(SessionEvent event) {
            File file = new
            File("C:\\oracle\\904\\toplink\\examples\\jdev\\2-TierEmployee\\toplin
k.log");
        try {
```

```
            System.out.println("FILE: " + file.getAbsolutePath());
            FileWriter writer = new FileWriter(file);
            event.getSession().setLog(writer);
            } catch (IOException ioe) {
            ioe.printStackTrace();
            throw new RuntimeException("Failed to setup logging to: " +
file.getAbsolutePath());
            }
        }
    };
}
```

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.

Viewing TopLink Log Messages From the Application Server Control Console

You can view TopLink log messages using Oracle Enterprise Manager 10*g*.

If you are using TopLink native logging (to a file) or `java.util.logging` package, edit the `<OC4J_HOME>\toplink\config\TOPLINK.xml` file `log` element attribute `path` to match the log output file you specified in your session (see "Configuring Logging" on page 74-4), or in the `<JRE_HOME>/lib/logging.properties` file (see "Configuring a Session to use java.util.logging Package" on page 74-8). Example 72–4 shows the default `TOPLINK.xml` file:

Example 72–4 Default TOPLINK.xml File

```
<?xml version="1.0" encoding="UTF-8" ?>
<logs xmlns="http://www.oracle.com/iAS/EMComponent/ojdl">
    <log path="toplink/config/toplink.log" componentId="TOPLINK">
        <logviewer LogType="ERROR" />
    </log>
</logs>
```

If you are using server logging, you do not need to configure the `<OC4J_HOME>\toplink\config\TOPLINK.xml` file: in this case, it is not used. Instead, all TopLink log messages are written to the OC4J log file (which you can also view from the application server control console).

For more information, see the *Oracle Application Server Administrator's Guide*.

Profiler

The TopLink session provides profiling API that lets you identify performance bottlenecks in your application (see "Configuring a Performance Profiler" on page 74-10). When enabled, the profiler logs a summary of the performance statistics for every query that the application executes.

TopLink allows you to measure application performance using the following tools:

- TopLink Profiler
- Oracle Dynamic Monitoring System (DMS)

TopLink Profiler

The 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 lets you 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 TopLink Web client, or create your own application or applet to view the profiler logs. For more information, see "Accessing the TopLink Profiler Results" on page 11-3.

For more information, see "Measuring TopLink Performance With the TopLink Profiler" on page 11-2.

Oracle Dynamic Monitoring System (DMS)

Oracle DMS is a library that enables application and system developers to use a variety of DMS sensors to measure and export customized performance metrics for specific software components (called nouns).

TopLink includes DMS instrumentation in essential objects to provide efficient monitoring of runtime data in TopLink enabled applications, including both J2EE and non-J2EE applications.

By enabling DMS profiling in a TopLink application, you can collect and easily access run-time data that can help you with application administration tasks and performance tuning.

You can easily access DMS data at run time using a management application that supports the Java Management Extensions (JMX) API (see "Accessing Oracle DMS Profiler Data Using JMX" on page 11-7), or using any Web browser and the DMS Spy servlet (see "Accessing Oracle DMS Profiler Data Using the DMS Spy Servlet" on page 11-7).

For more information, see "Measuring TopLink Performance With the Oracle Dynamic Monitoring System (DMS)" on page 11-4.

Integrity Checker

When you log into a session, TopLink initializes and validates the descriptors you registered with it. By configuring the integrity checker, you can customize this validation process.

For more information, see "Configuring the Integrity Checker" on page 74-18.

Exception Handlers

Exception handlers allow any exception that occurs in a session to be caught and processed. Exception handlers can be used for debugging purposes, or to resolve database timeouts or failures.

To use exception handlers, register an implementor of the `oracle.toplink.exceptions.ExceptionHandler` interface with the session (see "Configuring an Exception Handler" on page 74-12).

If an exception occurs during a session operation, such as executing a query, the exception is passed to the exception handler. The exception handler can either rethrow the exception, or handle the exception and retry the operation. When handling exceptions, ensure that the following conditions are met:

- If you are performing a write query and you are within a transaction, you should not retry the operation.
- If you are performing a read query, you may retry the operation, and, if successful, return the query result.

If your exception handler cannot proceed, you should throw an appropriate application-specific exception.

For more information on the types of exceptions that TopLink can throw, see Part V, "Troubleshooting a TopLink Application".

Registering Descriptors

You use a session to perform persistence operations on the objects described by TopLink mapping metadata represented as a TopLink project (see "Understanding Projects" on page 17-1). Each session must therefor be associated with the descriptors of at least one TopLink project. You associate descriptors with a session by registering them with the session.

The preferred way to register descriptors with a session is to use TopLink Workbench to configure the session with a mapping project (see "Configuring a Primary Mapping Project" on page 74-2 and "Configuring Multiple Mapping Projects" on page 74-9).

Sessions and CMP

Although TopLink is an integral part of a J2EE application, in most cases the client does not interact with TopLink directly. Instead, TopLink features are invoked indirectly by way of EJB container callbacks.

In a CMP TopLink project, you do not explicitly create, configure, or acquire a session. The TopLink runtime creates, configures, acquires and uses a session itself internally. For more information, see "Creating Session Metadata" on page 2-21. Similarly, in a CMP TopLink project, how metadata is deployed depends on the EJB container and application server you use (see "Deploying Metadata" on page 2-22).

Sessions and Sequencing

An essential part of maintaining object identity is managing the assignment of unique values to distinguish one instance from another. For more information, see "Projects and Sequencing" on page 17-4.

Sequencing options you configure in a `sessions.xml` (or `project.xml`) file determine the type of sequencing that TopLink uses.

In a CMP project, you do not configure a `sessions.xml` file directly: in this case you must configure the sequence type in the `project.xml` file (see "Configuring Sequencing at the Project Level" on page 20-3).

In a non-CMP project, you can use session-level sequence configuration to override project-level sequence configuration, on a session-by-session basis, if required (see "Configuring Sequencing at the Session Level" on page 83-4).

After configuring the sequence type at the session (or project) level, for each descriptor you must also configure sequencing options for that descriptor to use sequencing (see "Descriptors and Sequencing" on page 23-9).

Server and Client Sessions

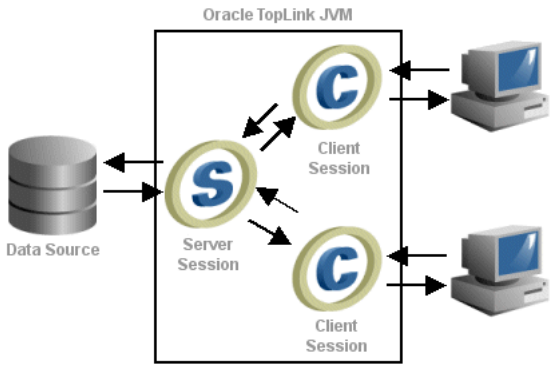
A server session manages the server side of client/server communications, providing shared resources, including a shared object cache and connection pools to a single data source.

A client session is a client-side communications mechanism that works together with the server session to provide the client/server connection. You acquire client sessions from a server session at run time as required. By default, a client session shares the session cache of its parent server session. Each client session serves one client. A client session communicates with the server session on behalf of the client application.

Each client session can have only one associated server session, but a server session can support any number of client sessions.

As Figure 72–2 illustrates, together, the client session and server session provide a three-tier architecture that you can scale easily, by adding more client sessions. A server session is the most common TopLink session type because it supports this three-tier architecture that is common in enterprise applications. Because of this scalability, Oracle recommends that you use the three-tier architecture to build your TopLink applications.

Figure 72–2 Typical TopLink Server Session with Client Session Architecture



This section explains the advantages of using server sessions and client sessions in your TopLink application, including the following:

- Three-Tier Architecture Overview
- Advantages of the TopLink Three-Tier Architecture

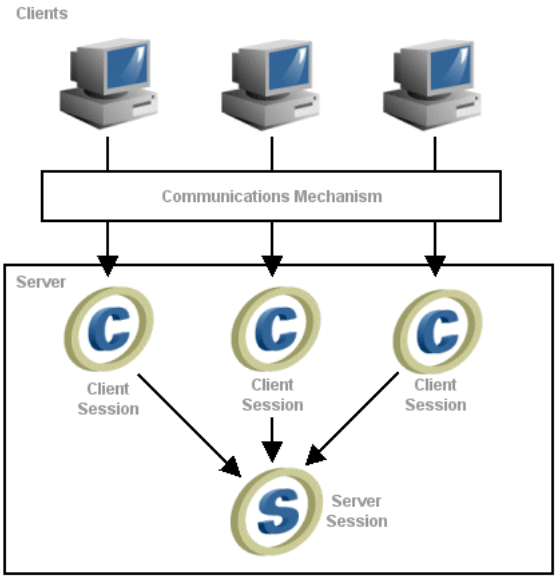
For more information, see the following:

- "Creating a Server Session" on page 73-4
- "Configuring Server Sessions" on page 76-1
- "Acquiring a Session From the Session Manager" on page 75-3
- "Acquiring a Client Session" on page 75-6

Three-Tier Architecture Overview

In a TopLink three-tier architecture, client sessions and server sessions both reside on the server. Client applications access the TopLink application through a client session, and the client session communicates with the database using the server session.

Figure 72–3 Server Session and Client Session Usage



Advantages of the TopLink Three-Tier Architecture

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 majority of the session functionality.

This section discusses some of the advantages and general concepts associated with the TopLink three-tier design, including the following:

- Shared Resources
- Providing Read Access
- Providing Write Access
- Security and User Privileges
- Concurrency
- Connection Allocation

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 HyperText Markup Language (HTML), Servlet, JavaServer Pages (JSP), Remote Method Invocation (RMI), Common Object Request Broker Architecture (CORBA), Web services, and EJB.

To support a shared object cache, client sessions must do the following:

- Implement any changes to the database with the TopLink unit of work.

- Share a common database login for reading (you can implement separate logins for writing).

Providing Read Access

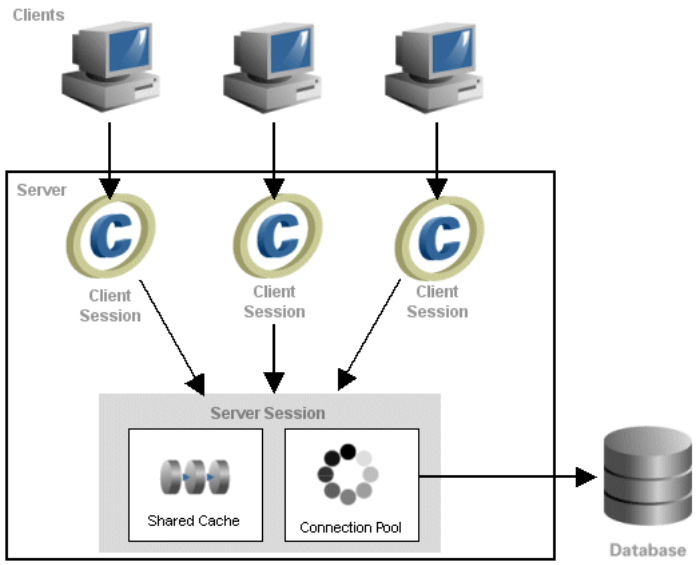
To read objects from the database, the client must first acquire a client session from the server session. Acquiring a client session gives the client access to the session cache and the database through the server session. The server session behaves as follows:

- 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 72–4 illustrates how multiple clients read from the database using the server session.

Figure 72–4 Multiple Client Sessions Reading the Database Using the Server Session



To read objects from the database using a client session, do the following:

1. Acquire a `Session` from the `Server`:

```
Server server =
    (Server) SessionManager.getManager().getSession(
        sessionName, MyServerSession.class.getClassLoader()
    );
Session clientSession = (Session) server.acquireClientSession();
```

   For more information, see "Acquiring and Using Sessions at Run Time" on page 75-1.

2. Use the `Session` object to perform read operations (for more information, see "Understanding TopLink Queries" on page 93-1 and "Understanding TopLink Expressions" on page 95-1).

Note: Oracle recommends that you do not use the Server session object directly to read objects from the database.

Providing Write Access

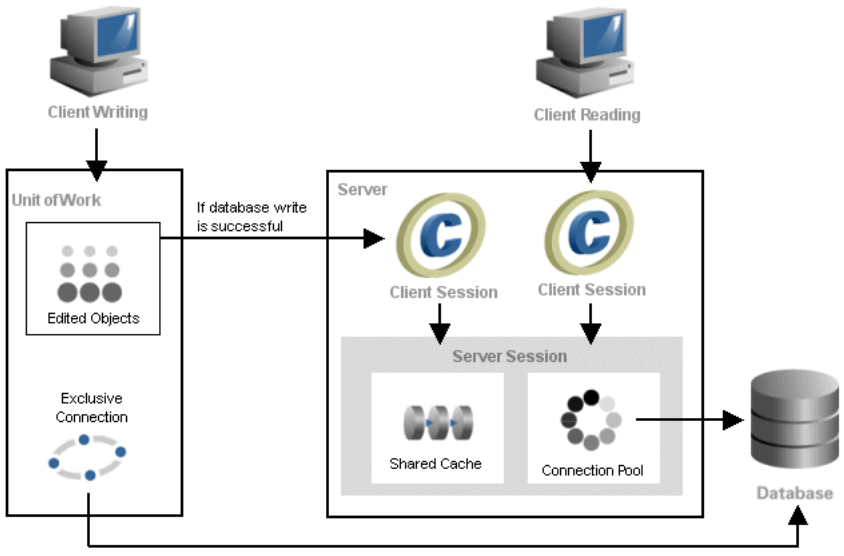
Because the client session disables all database modification methods, a client session cannot create, change, or delete objects directly. Instead, the client must obtain a unit of work from the client session to perform database modification methods.

To write to the database, the client acquires a client session from the server session and then acquires a unit of work 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.

Note: Although client sessions are thread-safe, do not use them to write across multiple threads. Multithread write operations from the same client session can result in errors and a loss of data. For more information, see "Concurrency" on page 72-18.

Figure 72–5 illustrates how to write to the database using a client session acquired from a server session.

Figure 72–5 Writing with Client Sessions and Server Sessions



To write to the database using a unit of work, use this procedure:

1. Acquire a session from the server session:

```
Server server =
    (Server) SessionManager.getManager().getSession(
        sessionName, MyServerSession.class.getClassLoader()
    );
Session clientSession = (Session) server.acquireClientSession();
```

For more information, see "Acquiring and Using Sessions at Run Time" on page 75-1.

2. Acquire a `UnitOfWork` object from the `Session` object.

   ```
   UnitOfWork uow = clientSession.acquireUnitOfWork();
   ```

   For more information, see "Unit of Work Sessions" on page 72-19.

3. Use the unit of work to perform the required updates and then commit the `UnitOfWork`.

   For more information, see the following:

   - "Understanding TopLink Queries" on page 93-1
   - "Understanding TopLink Expressions" on page 95-1)
   - "Understanding TopLink Transactions" on page 97-1

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.

When you use internal TopLink connection pools (see "Connection Pools" on page 81-7), each server session provides a read connection pool and a write connection pool. All read queries use connections from the read connection pool and all queries that write changes to the data store use connections from the write connection pool. This ensures that connections for one session are kept separate from the connections used in another.

To further isolate users from one another, you can use an isolated session: a special type of client session that provides its own session cache isolated from the shared object cache of its parent server session to provide improved user-based security, or to avoid caching highly volatile data. For more information, see "Isolated Client Sessions" on page 72-19.

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.

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 value holders.

For more information about handling concurrency issues, see "Handling Stale Data" on page 87-6.

Connection Allocation

When you instantiate the server session, it creates a pool of data source connections. The session then manages the connection pool based on your session configuration, and shares the connections among its client sessions. 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.

The server session provides connections to client sessions as needed. By default, the server session does not allocate a data source connection for a client session until a transaction starts (a lazy data source connection). Alternatively, you can acquire a client session that allocates a connection immediately (see "Acquiring a Client Session That Does Not Use Lazy Connection Allocation" on page 75-9).

The server session allocates read connections from its read connection pool to all client sessions. If your application requires multiple read security levels then you must use multiple server sessions or TopLink isolated sessions (see "Isolated Client Sessions" on page 72-19).

The server session also supports multiple write connection pools and nonpooled connections. Be default, all client sessions use the default write connection pool. However, if your application requires multiple security levels or user logins for write access, then you can use multiple write connection pools. You can configure a client session to use a specific write connection pool or nonpooled connection when it is acquired (see "Acquiring a Client Session That Uses a Named Connection Pool" on page 75-8). This connection is only used for writes, not reads (reads still go through the server session read connection pool).

For more information, see the following:

- "Internal Connection Pools" on page 81-7
- "External Connection Pools" on page 81-8

Unit of Work Sessions

The unit of work ensures that the client edits objects in a separate object transaction space. This feature lets clients perform object transactions in parallel. When transactions are committed, the unit of work makes any required changes in the database, and then merges the changes into the shared TopLink session cache. The modified objects are then available to all other users.

For information on creating, configuring, and using a unit of work, see "Understanding TopLink Transactions" on page 97-1.

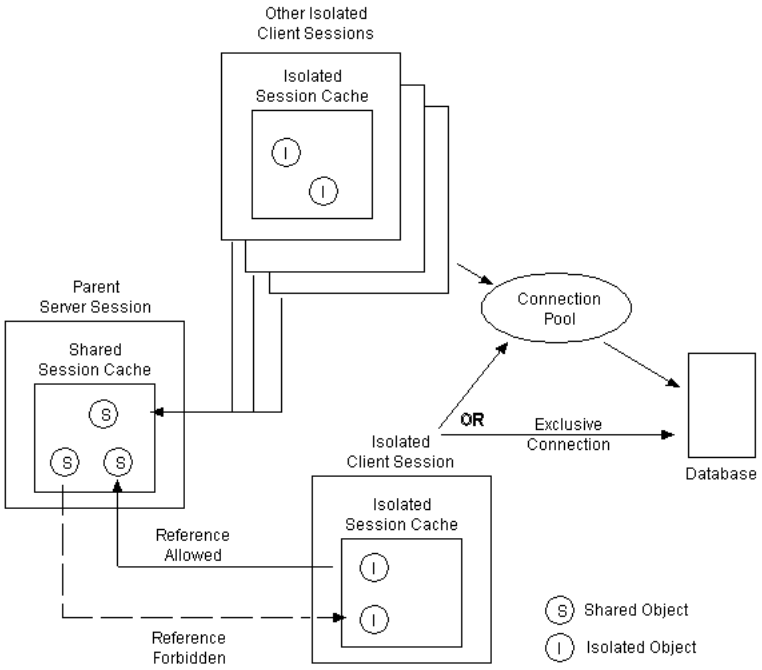
Isolated Client Sessions

An isolated client session is a special type of client session that provides its own session cache. This session cache is isolated from the shared session cache of its parent server session.

If in your TopLink project you configure all classes as isolated (see "Configuring Cache Isolation at the Project Level" on page 19-16), or one or more classes as isolated (see "Configuring Cache Isolation at the Descriptor Level" on page 25-37), then all client sessions that you acquire from a parent server session will be isolated client sessions.

Figure 72–6 illustrates the relationship between a parent server session's shared session cache and its child isolated client sessions.

Figure 72–6 Isolated Client Sessions



Each isolated client session owns an initially empty cache and identity maps used exclusively for isolated objects that the isolated client session accesses while it is active. The isolated client session's isolated session cache is discarded when the isolated client session is released.

When you use an isolated client session to read an isolated class, the client session reads the isolated object directly from the database and stores it in that client session's isolated session cache. When you use the client session to read a shared class, the client session reads the shared object from the parent server session's shared session cache. If the shared object is not in the parent server session's shared session cache, it will read it from the database and store it in the parent server session's shared session cache.

Isolated objects in an isolated client session's isolated session cache may reference shared objects in the parent server session's shared session cache, but shared objects in the parent server session's shared session cache cannot reference isolated objects in an isolated client session's isolated session cache.

Note: You cannot define mappings from shared classes to isolated classes. If using CMP, you also cannot define references from isolated enterprise beans to shared EJB.

Client sessions can access the data source using a connection pool, or an exclusive connection. To use an exclusive connection, acquire the isolated client session using a `ConnectionPolicy` (see "Acquiring a Client Session That Uses Exclusive Connections" on page 75-7). Using an exclusive connection provides improved user-based security for reads and writes. Named queries can also use an exclusive connection (see "Configuring Named Query Advanced Options" on page 25-24).

Note: If an isolated session contains an exclusive connection, you must release the session when you are finished using it. Relying on the finalizer to release the connection when the session is garbage collected may cause errors when dealing with Java Transaction API (JTA) transactions.

Use isolated client sessions to do the following:

- avoid caching highly volatile data in the shared session cache
- achieve serializable transaction isolation (see "Isolated Client Session Cache" on page 99-29)
- use the Oracle Virtual Private Database (VPD) feature in your TopLink-enabled application (see "Isolated Client Sessions and Oracle Virtual Private Database (VPD)" on page 72-21)

For more information, see the following:

- "Isolated Client Session Limitations" on page 72-24
- "Acquiring an Isolated Client Session" on page 75-7
- "Configuring Exclusive Isolated Client Sessions for Virtual Private Database" on page 77-1

Isolated Client Sessions and Oracle Virtual Private Database (VPD)

Oracle9*i* Database Server (and later) provides a server-enforced, fine-grained access control mechanism called Virtual Private Database (VPD). VPD ties a security policy to a table by dynamically appending SQL statements with a predicate to limit data access at the row level. You can create your own security policies, or use Oracle's custom implementation of VPD called Oracle Label Security (OLS). For more information on VPD and OLS, see:

`http://www.oracle.com/technology/deploy/security/index.html`.

To use the Oracle Database VPD feature in your TopLink-enabled application, use isolated client sessions.

Any class that maps to a table that uses VPD must have the descriptor configured as isolated (see "Configuring Cache Isolation at the Descriptor Level" on page 25-37).

When you use isolated client sessions with VPD, you typically use exclusive connections (see "Acquiring a Client Session That Uses Exclusive Connections" on page 75-7).

To support VPD, you are responsible for implementing session event handlers that the TopLink runtime invokes during the isolated client session life cycle (see "Isolated Client Session Life Cycle" on page 72-22). The session event handler you must implement depends on whether or not you are using Oracle Database proxy authentication (see "VPD With Oracle Database Proxy Authentication" on page 72-22 and "VPD Without Oracle Database Proxy Authentication" on page 72-22).

For information, see "Configuring Exclusive Isolated Client Sessions for Virtual Private Database" on page 77-1.

VPD With Oracle Database Proxy Authentication

If you are using Oracle Database proxy authentication ("Oracle Database Proxy Authentication" on page 81-5), you must implement a session event handler for the following session events:

- `noRowsModifiedSessionEvent` (see "NoRowsModifiedSessionEvent Event Handler" on page 77-3)

By using Oracle Database proxy authentication, you can set up VPD support entirely in the database. That is, rather than making the isolated client session execute SQL (see "PostAcquireExclusiveConnection Event Handler" on page 77-1 and "PreReleaseExclusiveConnection Event Handler" on page 77-2), the database performs the required setup in an after login trigger using the proxy `session_user`.

VPD Without Oracle Database Proxy Authentication

If you are not using Oracle Database proxy authentication, you must implement session event handlers for the following session events:

- `postAcquireExclusiveConnection` (see "PostAcquireExclusiveConnection Event Handler" on page 77-1): used to perform VPD setup at the time TopLink allocates a dedicated connection to an isolated session and before the isolated session user uses the connection to interact with the database.
- `preReleaseExclusiveConnection` (see "PreReleaseExclusiveConnection Event Handler" on page 77-2): used to perform VPD cleanup at the time the isolated session is released and after the user is finished interacting with the database.
- `noRowsModifiedSessionEvent` (see "NoRowsModifiedSessionEvent Event Handler" on page 77-3)

In your implementation of these handlers, you obtain the required user credentials from the `ConnectionPolicy` associated with the session (see "Acquiring a Client Session That Uses Connection Properties" on page 75-8).

Isolated Client Session Life Cycle

This section provides an overview of the key phases in the life cycle of an isolated session, including the following:

- Setup required before using an isolated session
- Interaction among isolated session objects
- Clean-up required after using an isolated session

To enable the life cycle of an isolated session, use this procedure:

1. Prepare VPD configuration in the database.
2. Configure your project and session:
   - Designate descriptors as isolated (see "Configuring Cache Isolation at the Descriptor Level" on page 25-37).
   - Configure your server session to allocate exclusive connections (see "Configuring Connection Policy" on page 74-19).
   - Implement session event listeners for the required connection events
     - If you are using Oracle Database proxy authentication (see "Oracle Database Proxy Authentication" on page 81-5), see "NoRowsModifiedSessionEvent Event Handler" on page 77-3.

– If you are not using Oracle Database proxy authentication, see "PostAcquireExclusiveConnection Event Handler" on page 77-1, "PreReleaseExclusiveConnection Event Handler" on page 77-2, and "NoRowsModifiedSessionEvent Event Handler" on page 77-3

Note: You must add these session event listeners to the server session from which you acquire your isolated client session. You cannot add them to the isolated client session itself. For more information, see "Configuring Session Event Listeners" on page 74-17

- Implement exception handlers for the appropriate exceptions (see "ValidationException Handler" on page 77-3).

3. Acquire an isolated session:

   - If you are using Oracle Database proxy authentication (see "Oracle Database Proxy Authentication" on page 81-5):

     ```
     Session myIsolatedClientSession = server.acquireClientSession();
     ```

     Because you configured one or more descriptors as isolated, `myIsolatedClientSession` is an isolated session with an exclusive connection.

   - If you are not using Oracle Database proxy authentication:

     ```
     ConnectionPolicy myConnPolicy =
         (ConnectionPolicy)server.getDefaultConnectionPolicy().clone();
     myConnectionPolicy.setProperty("credentials", myUserCredentials);
     Session myIsolatedClientSession =
         server.acquireClientSession(myConnectionPolicy);
     ```

     Set the user's credentials as appropriate properties on `myConnectionPolicy`. Because you configured one or more descriptors as isolated, `myIsolatedClientSession` is an isolated session with an exclusive connection.

   The TopLink runtime raises a `SessionEvent.PostAcquireExclusiveConnection` event handled by your `SessionEventListener` (see "PostAcquireExclusiveConnection Event Handler" on page 77-1).

4. Use `myIsolatedClientSession` to interact with the database.

   If the TopLink runtime raises a `SessionEvent.NoRowsModified` event, it is handled by your `SessionEventListener` (see "NoRowsModifiedSessionEvent Event Handler" on page 77-3).

5. When you are finished using `myIsolatedClientSession`, release the isolated session:

   ```
   myIsolatedClientSession.release();
   ```

   The TopLink runtime prepares to destroy the isolated cache and to close the exclusive connection associated with this isolated session.

   The TopLink runtime raises a `SessionEvent.PreReleaseExclusiveConnection` event handled by your `SessionEventListener` (see "PreReleaseExclusiveConnection Event Handler" on page 77-2).

6. Repeat steps 3 to 5 (as required) until the application exits.

Isolated Client Session Limitations

For the purposes of security as well as efficiency, observe the limitations described in the following section, when you use isolated client sessions in your TopLink three-tier application:

- Mapping
- Inheritance
- Caching and Cache Coordination
- Sequencing
- CMP
- Transactions and JTA

Mapping

Consider the following mapping and relationship restrictions when using isolated sessions with your relational model:

- Isolated objects may be related to shared objects, but shared objects cannot have any relationships with isolated objects.
- If a table has a VPD security policy associated with it, then the class mapped to that table must be isolated.
- If one of the tables in a multiple table mapping is isolated, then the main class must also be isolated.

The TopLink runtime enforces these restrictions during descriptor initialization.

Inheritance

Aggregates and aggregate mappings inherit the isolated configuration of their parents.

If a class is isolated, then all inheriting classes should be isolated. Otherwise, if you relate a shared class to a shared superclass with isolated subclasses, it is possible that some of the isolated subclasses will lose object identity when the isolated session is released.

To give you the flexibility to mix shared and isolated classes, the TopLink runtime does not enforce these restrictions during descriptor initialization. If you wish to mix shared and isolated classes in your inheritance hierarchy, then you must be prepared to deal with this possible loss of object identity.

Caching and Cache Coordination

Isolated classes are never loaded into the shared cache of a parent server session. Isolated classes cannot be used with cache coordination.

Sequencing

Oracle recommends that you do not configure a sequencing object as isolated. TopLink does not access sequencing objects using the isolated session's dedicated connection, and so the sequence values are not available to the isolated session.

CMP

For CMP, relationships between isolated and shared data is not allowed. This is because of the relationship maintenance requirements of having bidirectional references for all relationships.

Transactions and JTA

Oracle recommends that you explicitly release an isolated session when you are finished using it, rather than wait for the Java garbage collector to invoke the finalizer. The finalizer is provided as a last resort: waiting for the garbage collector may cause errors when dealing with a JTA transaction.

Historical Sessions

By default, a session represents a view of the most current version of objects, and when you execute a query in that session, it returns the most current version of selected objects.

If your data source maintains past versions of objects, you can configure TopLink to access this historical data so that you can express read queries conditional on how your objects are changing over time. You can also do the following:

- Make series of queries relative to any point in time–not just the time of the first query.
- Provide read consistency so that a series of read operations or report queries all execute as if at the same time.
- Use the `mergeClone` method to provide deep recovery of an object by passing in a past version of it.

In addition, you can express query selection criteria as either of the following:

- A condition at a past time: for example, "employees who used to...".
- A change over time: for example, "employees who recently...".

For more information, see the following:

- "Historical Session Limitations" on page 72-25
- "Configuring Historical Sessions" on page 78-1
- "Acquiring a Historical Session" on page 75-9
- "Historical Queries" on page 93-21.

Historical Session Limitations

The `HistoryPolicy` provides a very flexible means of accommodating a wide variety of historical schemas. However, be aware of the following restrictions:

- You cannot use the `HistoryPolicy`, if your design combines both current and historical data in a single schema.
- You cannot use historical sessions, nor historical queries, with EJB entity beans.
- TopLink assumes that the current version of an object corresponds to the row in the historical table whose row end field is `NULL`.
- You cannot directly map the start and end fields of a history table because they do not exist in the regular schema.
- You cannot query on ranges of historical objects, only as of a specific point in time.

Session Broker and Client Sessions

The **TopLink session broker** is a mechanism that enables client applications to transparently access multiple databases through a single TopLink session.

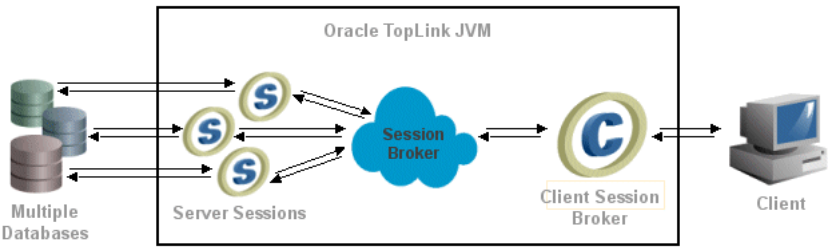
The TopLink session broker enables client applications to access two or more databases through a single session. If your application stores objects in multiple databases, the session broker, which provides seamless communication for client applications, enables the client to view multiple databases as if they were a single database.

When a three-tier session broker application uses server sessions to communicate with the database, clients require a client session to access the database. Similarly, when you implement a session broker, the client requires a *client session broker* to access 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 to be a single client session to the client application.

As Figure 72–7 illustrates, a session broker connects to the databases through two or more server sessions or database sessions.

Figure 72–7 TopLink Session Broker with Server Session Architecture



This section explains the following:

- Session Broker Architecture
- Committing a Transaction with a Session Broker
- Session Broker Session Limitations
- Session Broker Alternatives

For information, see:

- "Creating Session Broker and Client Sessions" on page 73-6
- "Configuring Session Broker and Client Sessions" on page 79-1
- "Acquiring a Session From the Session Manager" on page 75-3
- "Acquiring a Client Session" on page 75-6

Session Broker Architecture

As Figure 72–7 illustrates, a session broker contains a broker object that acts as an intermediary between the application and the multiple sessions added to the session broker.

To construct a session broker, use TopLink Workbench to modify your `sessions.xml` file as follows:

1. Define two or more sessions (of the same type, either server sessions or database sessions).
2. Define a session broker.
3. Add the sessions to the session broker.

When you use `SessionManager` method `getSession(sessionBrokerName)` where `sessionBrokerName` is the name of the session broker you defined, the session manager returns the corresponding session broker session (call it `mySessionBroker`) that contains an instance of each of the sessions you added to it. When you use `mySessionBroker` method `login`, it logs into each defined session. Thereafter, you use `mySessionBroker` as you would any other session: TopLink transparently handles access to the multiple databases.

In the case of a three-tier architecture where the session broker contains two or more server sessions, you use session broker method `acquireClientSessionBroker` to acquire a single client session that lets you query across all the data sources managed by the various server sessions. You use this client session as you would any other client session.

Committing a Transaction with a Session Broker

By default, when you commit a transaction with a session broker session, a two-stage commit is performed.

Ideally, you should incorporate a JTA external transaction controller in order to benefit from its two-phase commit.

Committing a Session with a JTA Driver: Two-Phase Commits

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 operation to any one database fails, then all database transactions roll back. The two-phase commit operation is the safest method available to commit a transaction to the database.

Two-phase commit support requires integration with a compliant JTA driver.

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 guarantees data integrity only up to the point of the final commit of the transaction. If the SQL script executes successfully on all databases, but the commit operation 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.

Session Broker Session Limitations

Although the session broker is a powerful tool that lets you use data that is distributed across multiple databases from a single application, it has some limitations:

- It may not meet the needs of your particular distributed data application (see "Session Broker Alternatives" on page 72-28).
- 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 must reside on the same database as the target object (See "Many-to-Many Join Tables and Direct Collection Tables" on page 72-28 for a work-around for this limitation).

Many-to-Many Join Tables and Direct Collection Tables

By default, TopLink assumes that many-to-many and direct collection tables are on the same database as the source object. If they are on a different database, then you must configure the mapping's session name using `ManyToManyMapping` or `DirectCollectionMapping` method `setSessionName` as Example 72–5 illustrates.

Note that a many-to-many join table must still reside on the same database as the target object.

Example 72–5 Using Mapping setSessionName in a Descriptor Amendment Method

```
public void addToDescriptor(ClassDescriptor descriptor) {
    descriptor.getMappingForAttributeName("projects").setSessionName("branch-database");
}
```

To work around this problem for data-level queries, use the `DatabaseQuery` method `setSessionName`.

Session Broker Alternatives

When evaluating whether or not to use a session broker in your application, consider the following alternatives:

- Database Linking
- Multiple Sessions

Database Linking

Most enterprise databases, such as the Oracle Database, support linking other databases on the database server. This allows querying and two-phase commit across linked databases. Using the session broker is not the same as linking databases. If your database allows linking, Oracle recommends that you use that functionality to provide multiple database access instead of using a session broker.

Multiple Sessions

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 department.

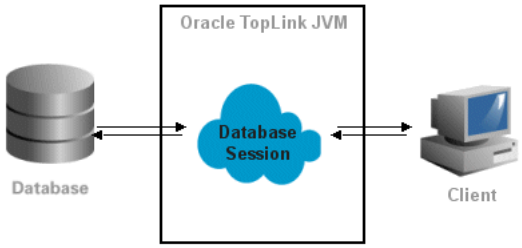
This arrangement requires that you to manage each session manually and ensure that the class descriptors for your project reside in the correct session.

- You can use additional sessions to house a standard 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 lets you maintain the client cache.

Database Sessions

A **database session** provides a client application with a single data source connection, for simple, standalone applications in which a single connection services all data source requests for one user.

Figure 72–8 TopLink Database Session Architecture



A database session is the simplest session TopLink offers. It provides both client and server communications and supports only a single client and a single database connection. It is suitable for simple applications or 2-tier applications.

Note: Oracle does not recommend using this session type in a 3-tier application because it is not as flexible or scalable as a server and client session. Oracle recommends that you use server sessions and client sessions (see "Server and Client Sessions" on page 72-14). 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 store 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

For more information, see the following:

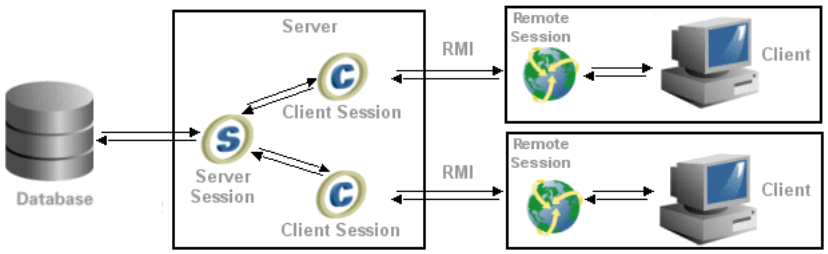
- "Creating Database Sessions" on page 73-4
- "Configuring Database Sessions" on page 73-8
- "Acquiring a Session From the Session Manager" on page 75-3

Remote Sessions

A **remote session** is a client-side session that communicates over RMI with a corresponding client session and server session on the server-side. Remote sessions handle object identity and marshalling and unmarshalling between client-side and server-side.

A remote session resides on the client rather than the TopLink server. The remote session does not replace the client session; rather, a remote session requires a client session to communicate with the server session.

Figure 72–9 Typical TopLink Server Session with Remote Session Architecture



The remote session provides a full TopLink session, complete with a session cache, on the client system. TopLink manages the remote session cache and enables client applications to execute operations on the server.

A remote session offers database access to clients that do not reside on the server. The remote session resides on the client and connects by way of RMI to a corresponding client session, which, in turn, connects to its server session on the server.

This section describes the following:

- Architectural Overview
- Remote Session Concepts

For more information, see "Creating Remote Sessions" on page 73-10.

Architectural Overview

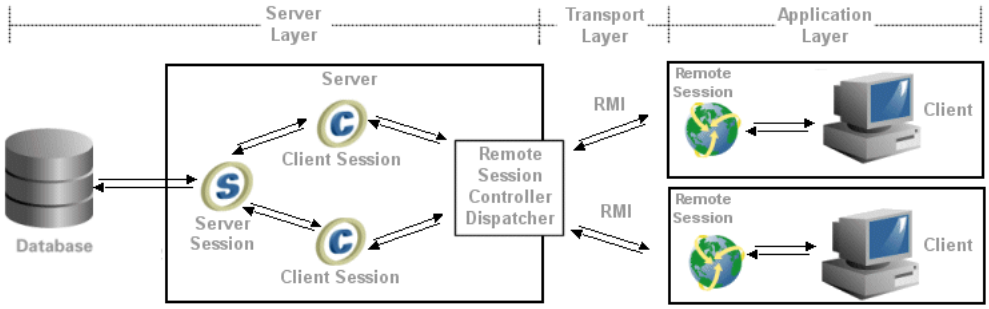
As Figure 72–10 illustrates, the remote session model consists of the following layers:

- The application layer–a client-side application talking to a remote session
- The transport layer–a communication layer, RMI or RMI-IIOP
- The server layer–a 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 run time, 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 72–10 An Architectural Overview of the Remote Session



Application Layer

The application layer includes the application client and the remote session. The remote session is a subclass of `Session` and maintains all the public protocols of the session, giving the appearance of working with the corresponding client session.

The remote session maintains its own identity map and a project 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 layers.

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.

Server Layer

The server layer includes a remote session controller dispatcher and a TopLink sessions: Figure 72–10 illustrates a three-tier server and its client sessions. The remote session controller dispatcher is an interface between the session and transport layers: it marshals and unmarshals all responses and requests between the sessions on the server and their corresponding remote sessions on the client.

Remote Session Concepts

When using remote sessions, consider the following:

- Securing Remote Session Access
- Queries
- Refreshing
- Indirection
- Cursored Streams
- Unit of Work

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 nonprivileged access.

To reduce this threat, run a server manager as a service to hold the remote session controller 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 a 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 read operations, 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. To avoid this, disable the remote session cache hits on read object queries based on the primary key.

For more information, see "Configuring Cache Refreshing" on page 25-27.

Indirection

The remote session supports indirection objects. An indirection object is a value holder that can be invoked remotely on the client side. When invoked, the value holder first checks to see if the requested object exists on the remote session. If not, then the associated value holder on the server is instantiated to get the value that is then passed back to the client. Remote value holders are used automatically; the application's code does not change.

Cursored Streams

A remote session supports both cursored streams and scrollable cursors.

For more information, see "Stream and Cursor Query Results" on page 93-8.

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.

Sessions and the Cache

Server, database, isolated, and historical sessions include an identity map that maintains object identity, and acts as a cache.

This section explains how the cache differs between the following sessions:

- Server and Database Session Cache
- Isolated Session Cache
- Historical Session Cache

For more information, see "Understanding the Cache" on page 87-1.

Server and Database Session Cache

When a server or database session reads objects from the database, it instantiates them and stores them in its identity map (cache). When the application subsequently queries for the same object, TopLink returns the object in the cache, rather than read the object from the database again.

This cache plays an important role in the performance of your application.

In the case of a server session, all client sessions acquired from it share the server session's cache.

To define how the cache manages objects, specify a strategy for cache management in TopLink Workbench.

Isolated Session Cache

When an isolated session reads an object, whose descriptor is configured as isolated, that object is instantiated and stored in the isolated session's cache only–it is not stored in the parent server session's shared object cache. Objects in the isolated session's cache may reference objects in the parent server session's shared object cache, but objects in the parent server session's shared object cache can never reference objects in the isolated session's cache.

Historical Session Cache

When a historical session reads objects, it does so only from its static, read-only cache, which is populated with all objects as of a specified time.

Understanding the Session API

The session API is defined by the following interfaces:

- `oracle.toplink.sessions.Session`
- `oracle.toplink.sessions.DatabaseSession`
- `oracle.toplink.sessions.UnitOfWork`
- `oracle.toplink.threetier.Server`

These APIs are used at run time to access objects and the data source. Always use the session public interfaces, not the corresponding implementation classes.

You should use the `Session` interface when reading and querying with any of client sessions, session brokers, isolated client sessions, historical sessions, remote sessions, and database sessions.

You should use the `UnitOfWork` interface for all units of work acquired from any type of session.

You should use the `Server` interface to configure and acquire a client session from a `Server` session.

The `DatabaseSession` interface can be used for a database session.

Typically, you define server sessions, database sessions, and session broker sessions in a `sessions.xml` file and acquire them at run time using the `SessionManager`. You can also acquire a server session or database session from a `Project`. The only session that should ever be instantiated directly is the `SessionBroker`, and only when not using the `SessionManager`.

You acquire a client session from a server session.

You can also acquire a client session broker from a session broker composed of server sessions.

You acquire a unit of work from any session instance, client session broker, or session broker which contains `DatabaseSession` instances.

Example 72–6 illustrates the session interfaces that derive from `oracle.toplink.sessions.Session` interface.

Example 72–6 Session Interface Inheritance Hierarchy

```
oracle.toplink.sessions.Session
    oracle.toplink.sessions.DatabaseSession
        oracle.toplink.threetier.Server
    oracle.toplink.sessions.UnitOfWork
```

73

Creating Sessions

This chapter explains how to create TopLink sessions, including the following:

- Session Creation Overview
- Creating a Sessions Configuration
- Configuring a Sessions Configuration
- Creating a Server Session
- Creating Session Broker and Client Sessions
- Creating Database Sessions
- Creating Remote Sessions

For information on the various types of session available, see "Session Types" on page 72-1.

Session Creation Overview

Each TopLink session is contained within a sessions configuration (`sessions.xml`) file. You can create a sessions configuration using TopLink Workbench or Java code. Oracle recommends that you use TopLink Workbench to create and manage your sessions (see "Creating a Sessions Configuration" on page 73-1).

Alternatively, you can create sessions in Java. For more information on creating sessions in Java, see Oracle TopLink API Reference.

After you create a session, you must configure its various options (see "Configuring a Session" on page 74-1). After configuring the session, you can use it in your application to manage persistence (see "Acquiring and Using Sessions at Run Time" on page 75-1).

Creating a Sessions Configuration

TopLink Workbench lets you create session instances and save them in the `sessions.xml` file. It represents the `sessions.xml` file as a sessions configuration. Individual session instances are contained within the sessions configuration. You can create multiple sessions configurations, each corresponding to its own uniquely named `sessions.xml` file.

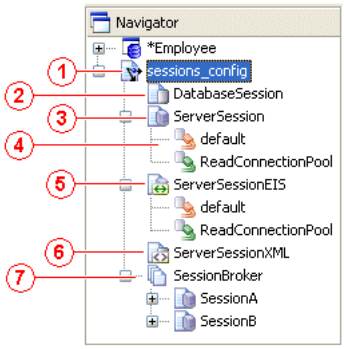
Oracle recommends that you use TopLink Workbench to create and manage sessions. It is the most efficient and flexible approach to session management. For more information about the advantages of this approach, see "Session Configuration and the sessions.xml File" on page 72-4.

TopLink Workbench displays sessions configurations and their contents in the Navigator window. When you select a session configuration, its attributes are displayed in the Editor window.

Figure 73–1 calls out the following user interface elements:

1. Sessions Configuration
2. Database Session
3. Relational Server Session
4. Connection Pool
5. EIS Server Session
6. XML Session
7. Session Broker

Figure 73–1 Sessions Configurations in Navigator Window



Using TopLink Workbench

To create a TopLink sessions configuration (`sessions.xml` file), use this procedure:




1. Click **New** on the toolbar and select **Sessions Configuration**.

   You can also create a new sessions configuration by selecting **File > New > Session Configuration** from the menu, or by clicking **Create New Sessions Configuration** in the standard toolbar.

2. The new sessions configuration element appears in the Navigator window; the Sessions Configuration property sheet appears in the Editor window.

   Enter data in each field on the Sessions Configuration property sheet as "Configuring a Sessions Configuration" on page 73-2 describes.

Configuring a Sessions Configuration

Each TopLink sessions configuration (`sessions.xml` file) can contain multiple sessions and session brokers. In addition, you can specify a classpath for each sessions configuration that applies to all the sessions it contains.

Using TopLink Workbench

To configure a session configuration, use this procedure:

1. Select the session configuration in the **Navigator**. Its properties appear in the Editor.

Figure 73–2 Sessions Configuration Property Sheet



Use the following information to enter data in each field of the Sessions configuration property sheet:

Field	Description
Project Save Location	Click **Change** and select the directory in which to save the sessions configuration.
Classpath	Lists the JAR or ZIP files that contain the compiled Java classes on which this sessions configuration depends for features that require an external Java class (for example, session event listeners). ■ To add a JAR or ZIP file, click **Add Entries** or **Browse** add the file. ■ To remove a JAR or ZIP file, select the file and click **Remove**. ■ To change the order in which TopLink searches these JAR or ZIP files, select a file and click **Up** to move it forward or click **Down** to move it back in the list.
Sessions for *<sessions configuration name>*	Lists the available sessions defined in this sessions configuration: ■ To add a session, click **Add Session**. ■ To remove a session, select the session and click **Remove**. ■ To rename a session, select the session and click **Rename**. For more information on creating sessions using TopLink Workbench, seen the following: ■ "Creating a Server Session" on page 73-4 ■ "Creating Session Broker and Client Sessions" on page 73-6 ■ "Creating Database Sessions" on page 73-8

Creating a Server Session

Oracle recommends that you create server sessions using TopLink Workbench (see "Using TopLink Workbench" on page 73-4).

After you create a server session, you create a client session by acquiring it from the server session (see "Acquiring a Client Session" on page 75-6).

Using TopLink Workbench

Before you create a server session, you must first create a sessions configuration (see "Creating a Sessions Configuration" on page 73-1).

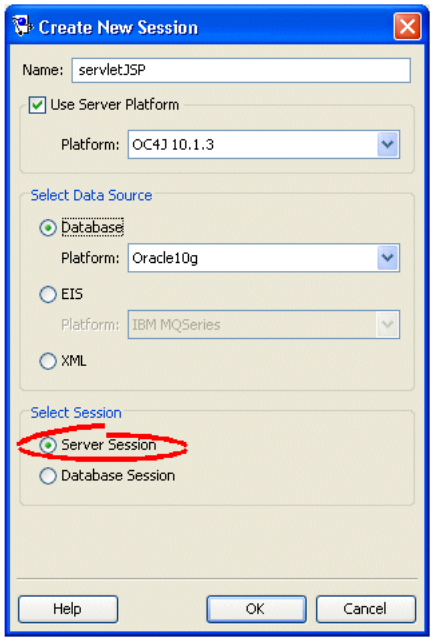
Creating a Session

To create a new TopLink server session, use this procedure:

1. Select the sessions configuration in the **Navigator** window in which you want to create a session.
2. Click **Add Session** on the toolbar. The Create New Session dialog box appears.

   You can also create a new server session by right-clicking the sessions configuration in the **Navigator** and selecting **New > Session** from the context menu, or by clicking **Add Session** on the Sessions Configuration property sheet.

Figure 73–3 Create New Session Dialog Box, Server Session Option



Use the following information to enter data in each field of the dialog box:

Field	Description
Name	Specify the name of the new session.
Use Server Platform	Check this field if you intend to deploy your application to a J2EE application server. If you check this field, you must configure the target application server by selecting a **Platform**.

Field	Description
Platform	This option is only available if you check **Use Server Platform**. Select the J2EE application server to which you will deploy your application. TopLink supports the following J2EE application servers: ■ OC4J 10.1.3 ■ OC4J 10.1.2 ■ OC4J 9.0.4 ■ OC4J 9.0.3 ■ WebLogic 8.1 ■ WebLogic 7.0 ■ WebLogic 6.1 ■ WebSphere 6.0 ■ WebSphere 5.1 ■ WebSphere 5.0 ■ WebSphere 4.0 ■ Custom The server platform you select is the default server platform for all sessions you create in this sessions configuration. At the session level, you can override this selection or specify a custom server platform class (see "Configuring the Server Platform" on page 74-14).
Select Data Source	Select the data source for this session configuration. Each session configuration must contain one data source. Choose one of the following: ■ **Database** to create a session for a relational project. ■ **EIS** to create a session for an EIS project. ■ **XML** to create a session for an XML project[1]. See "TopLink Project Types" on page 17-1 for more information.
Select Session	Select **Server Session** to create a session for a single data source (including shared object cache and connection pools) for multiple clients in a three-tier application.

[1] You cannot create a server session for an XML project.

Using Java

You can create a server session in Java code using a project. You can create a project in Java code, or read a project from a `project.xml` file.

Example 73–1 illustrates creating an instance (called `serverSession`) of a `Server` class using a `Project` class.

Example 73–1 Creating a Server Session from a Project Class

```
Project myProject = new Project();
Server serverSession = myProject.createServerSession();
```

Example 73–2 illustrates creating an instance (called `serverSession`) of a `Server` class using a `Project` read in from a `project.xml` file.

Example 73–2 Creating a Server Session from a project.xml File Project

```
Project myProject = XMLProjectReader.read("myproject.xml");
Server serverSession = myProject.createServerSession();
```

Example 73–3 illustrates creating a server session with a specified write connection pool minimum and maximum size (when using TopLink internal connection pooling). The default write connection pool minimum size is 5 and maximum size is 10.

Example 73–3 Creating a Server Session with Custom Write Connection Pool Size

```
XMLProjectReader.read("myproject.xml");
Server serverSession = myProject.createServerSession(32, 32);
```

Creating Session Broker and Client Sessions

A session broker may contain both server sessions and database sessions. Oracle recommends that you use the session broker with server sessions because server sessions are the most scalable session type.

Oracle recommends that you create server sessions using TopLink Workbench (see "Using TopLink Workbench" on page 73-6).

After you create and configure a session broker with server sessions, you can acquire a client session from the session broker at run time to provide a dedicated connection to all the data sources managed by the session broker for each client. For more information, see "Acquiring a Client Session" on page 75-6.

Using TopLink Workbench

Before you create a session broker session, you must first create a sessions configuration (see "Creating a Sessions Configuration" on page 73-1) and one or more server sessions ("Creating a Server Session" on page 73-4), or one or more database sessions ("Creating Database Sessions" on page 73-8).

To create a new TopLink session broker, use this procedure:

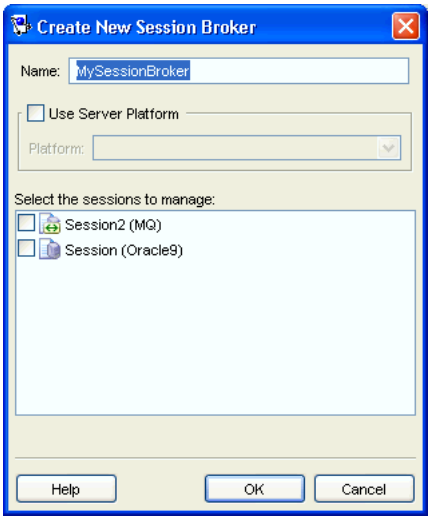
1. Select the sessions configuration in the **Navigator** window in which you want to create a session broker.



2. Click **Add Session Broker** on the toolbar. The Create New Session Broker dialog box appears.

   You can also create a new session broker by right-clicking the sessions configuration in the **Navigator** window and selecting **Add Session Broker** from the context menu or by clicking **Add Session Broker** on the Sessions Configuration property sheet.

Figure 73–4 Create New Session Broker Dialog Box



Use the following information to enter data in each field of the dialog box:

Field	Description
Name	Specify the name of the new session broker.
Use Server Platform	Check this field if you intend to deploy your application to a J2EE application server. If you check this field, you must configure the target application server by selecting a **Platform**.
Platform	This option is available only if you check **Use Server Platform**. Select the J2EE application server to which you will deploy your application. TopLink supports the following J2EE application servers: ▪ OC4J 10.1.3 ▪ OC4J 9.0.3[1] ▪ WebLogic 8.1 ▪ WebLogic 7.0 ▪ WebLogic 6.1 ▪ websphere 5.1 ▪ websphere 5.0 ▪ websphere 4.0 ▪ Custom The server platform you select is the default server platform for all sessions you create in this sessions configuration. At the session level, you can override this selection or specify a custom server platform class (see "Configuring the Server Platform" on page 74-14).
Select the sessions to Manage	Select sessions to be managed by this new session broker (from the list of available sessions) and click **OK**. **Note**: This field appears only if the configuration contains valid sessions.

[1] Includes support for both 9.0.3 and 9.0.4.

Continue with Chapter 74, "Configuring a Session".

Using Java

Example 73–4 illustrates how you can create a session broker in Java code by instantiating a `SessionBroker` and registering the brokered sessions with it.

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. The session broker manages these issues when you instantiate it.

Example 73–4 Creating a Session Broker

```
Project databaseProject = new MyDatabaseProject();
Server databaseSession = databaseProject.createServerSession();

Project eisProject = new MyEISProject();
Server eisSession = eisProject.createServerSession();

SessionBroker sessionBroker = new SessionBroker();
sessionBroker.registerSession("myDatabase", databaseSession);
sessionBroker.registerSession("myEIS", eisSession);

sessionBroker.login();
```

Creating Database Sessions

Oracle recommends that you create database sessions using TopLink Workbench (see "Using TopLink Workbench" on page 73-8).

After you create a database session, you can acquire and use it at run time. For more information on acquiring a database session, see "Acquiring a Session From the Session Manager" on page 75-3.

Using TopLink Workbench

Before you create a database session, you must first create a sessions configuration (see "Creating a Sessions Configuration" on page 73-1).

To create a new TopLink database session, use this procedure:

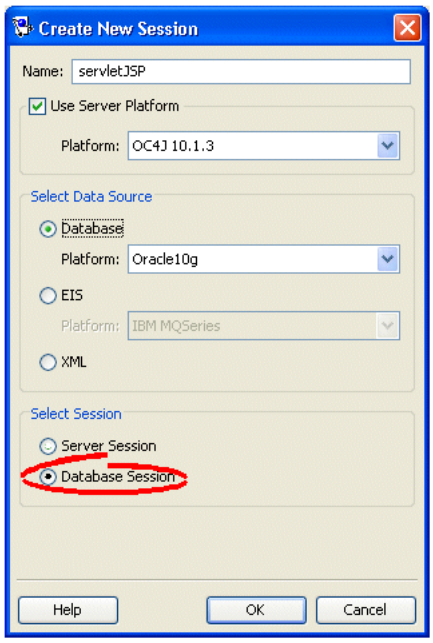
1. Select the session configuration in the **Navigator** window in which you want to create a session.



2. Click **Add Session** on the toolbar. The Create New Session dialog box appears.

   You can also create a new configuration by right-clicking the sessions configuration in the **Navigator** window and selecting **New > Session** from the context menu.

Figure 73–5 Create New Session Dialog Box, Database Session Option



Use the following information to enter data in each field of the dialog box:

Field	Description
Name	Specify the name of the new session.
Use Server Platform	Check this field if you intend to deploy your application to a J2EE application server. If you check this field, you must configure the target application server by selecting a **Platform**.
Platform	This option is available only if you check **Use Server Platform**. Select the J2EE application server to which you will deploy your application. TopLink supports the following J2EE application servers: ▪ OC4J 10.1.3 ▪ OC4J 9.0.3[1] ▪ WebLogic 8.1 ▪ WebLogic 7.0 ▪ WebLogic 6.1 ▪ websphere 5.1 ▪ websphere 5.0 ▪ websphere 4.0 ▪ Custom The server platform you select is the default server platform for all sessions you create in this sessions configuration. At the session level, you can override this selection or specify a custom server platform class (see "Configuring the Server Platform" on page 74-14).

Field	Description
Select Data Source	Select the data source for this session configuration. Each session configuration must contain one data source. Choose one of the following: ■ **Database** to create a session for a relational project. ■ **EIS** to create a session for an EIS project. ■ **XML** to create a session for an XML project. See "TopLink Project Types" on page 17-1 for more information.
Select Session	Select **Database Session** to create a session for a single database (including shared object cache and connection pools) for a single client suitable for simple applications or prototyping.

[1] Includes support for both 9.0.3 and 9.0.4.

Enter the necessary information and click **OK**.

TopLink Workbench window appears, showing the database session in the Navigator window.

Continue with Chapter 74, "Configuring a Session".

Using Java

You can create an instance of the `DatabaseSession` class in Java code using a `Project`. You can create a project in Java code or read a project from a `project.xml` file.

Example 73–5 illustrates creating a `DatabaseSession` using a `Project` class.

Example 73–5 Creating a Database Session from a Project Class

```
Project myProject = new Project();
DatabaseSession databaseSession = myProject.createDatabaseSession();
```

Example 73–6 illustrates creating a `DatabaseSession` using a `Project` read in from a `project.xml` file.

Example 73–6 Creating a Database Session from a project.xml File Project

```
Project myProject = XMLProjectReader.read("myproject.xml");
DatabaseSession databaseSession = myProject.createDatabaseSession();
```

Creating Remote Sessions

Remote sessions are acquired through a remote connection to their server-side session. Remote sessions are acquired through Java code on the remote client. The server-side session must also be registered with an `oracle.toplink.remote.ejb.RemoteSessionController` and accessible from the RMI naming service.

You create remote sessions entirely in Java code (see "Using Java" on page 73-10).

Using Java

Example 73–7 and Example 73–8 demonstrate how to create a remote 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 TopLink session.

Server

Example 73–7 shows the code you add to your application's RMI service (`MyRMIServerManagerImpl`) to create and return an instance of an `RMIRemoteSessionController` to the client. The controller sits between the remote client and the local TopLink session.

The `RMIRemoteSessionController` you create on the server is based on a TopLink server session. You create and configure this server session as described in "Creating a Server Session" on page 73-4 and "Configuring Server Sessions" on page 76-1.

Example 73–7 Server Creating RMIRemoteSessionController for Client

```
RMIRemoteSessionController controller = null;
try {
    // Create instance of RMIRemoteSessionControllerDispatcher which implements
    // RMIRemoteSessionController. The constructor takes a TopLink session as a parameter
    controller = new RMIRemoteSessionControllerDispatcher (localTopLinkSession);
} catch (RemoteException exception) {
    System.out.println("Error in invocation " + exception.toString());
}
return controller;
```

Client

The client-side code gets a reference to the application's RMI service (in this example it is called `MyRMIServerManager`) 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.

Example 73–8 Client Acquiring RMIRemoteSessionController from Server

```
MyRMIServerManager serverManager = null;
// Set the client security manager
try {
    System.setSecurityManager(new MyRMISecurityManager());
} catch(Exception exception) {
    System.out.println("Security violation " + exception.toString());
}
// Get the remote factory object from the Registry
try {
    serverManager = (MyRMIServerManager) 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 use as a normal TopLink session
Session session = rmiConnection.createRemoteSession();
```

74

Configuring a Session

This chapter describes how to configure TopLink sessions.

Table 74–1 lists the types of TopLink sessions that you can configure and provides a cross-reference to the type-specific chapter that lists the configurable options supported by that type.

Table 74–1 Configuring TopLink Sessions

If you are creating...	See...
Server and Client Sessions	Chapter 76, "Configuring Server Sessions"
Unit of Work Sessions	Chapter 97, "Understanding TopLink Transactions"
Isolated Client Sessions	Chapter 77, "Configuring Exclusive Isolated Client Sessions for Virtual Private Database"
Historical Sessions	Chapter 78, "Configuring Historical Sessions"
Session Broker and Client Sessions	Chapter 79, "Configuring Session Broker and Client Sessions"
Database Sessions	Chapter 80, "Configuring Database Sessions"

Table 74–2 lists the configurable options shared by two or more TopLink sessions types.

For more information, see the following:

- "Understanding TopLink Sessions" on page 72-1
- "Creating Sessions" on page 73-1

Configuring Common Session Options

Table 74–2 lists the configurable options shared by two or more TopLink session types. In addition to the configurable options described here, you must also configure the options described for the specific Session Types, as shown in Table 74–1

Table 74–2 Configurable Options for Session

Option	Type	TopLink Workbench	Java
"Configuring a Primary Mapping Project" on page 74-2	Basic	✔	✔
"Configuring a Session Login" on page 74-4	Basic	✔	✔
"Configuring Logging" on page 74-4	Basic	✔	✔
"Configuring Multiple Mapping Projects" on page 74-9	Advanced	✔	✔

Table 74–2 (Cont.) Configurable Options for Session

Option	Type	TopLink Workbench	Java
"Configuring a Performance Profiler" on page 74-10	Advanced	✔	✔
"Configuring an Exception Handler" on page 74-12	Advanced	✔	✔
"Configuring Customizer Class" on page 74-13	Advanced	✔	✔
"Configuring the Server Platform" on page 74-14	Advanced	✔	✔
"Configuring Session Event Listeners" on page 74-17	Advanced	✔	✔
"Configuring a Coordinated Cache" on page 88-1	Advanced	✔	✔
"Configuring the Integrity Checker" on page 74-18	Advanced		✔
"Configuring Connection Policy" on page 74-19	Advanced	✔	✔
"Configuring Named Queries at the Session Level" on page 74-21	Basic		✔

Configuring a Primary Mapping Project

The mapping project contains your TopLink mapping metadata (see "Understanding Projects" on page 17-1), including descriptors and mappings. Each session is associated with at least one project so that the session can register the descriptors.

Table 74–3 summarizes which sessions support a primary mapping project configuration.

Table 74–3 Session Support for Primary Mapping Project

Session	Using TopLink Workbench	Using Java
Server and Client Sessions	✔	✔
Session Broker and Client Sessions	✔	✔
Database Sessions	✔	✔

Using TopLink Workbench, you can export your mapping metadata as either a deployment XML file or as a Java class. Consequently, in a session, you can specify the mapping project as an XML file or as a Java class.

Oracle recommends that you export your mapping metadata from TopLink Workbench as a deployment XML file (see "Exporting Project Information" on page 18-13).

If you export your mapping metadata as a Java class, you must compile it and add it to the session configuration classpath (see "Configuring a Sessions Configuration" on page 73-2) before adding it to a session.

Note: When specifying the mapping project using XML, you can specify the Java resource path. In most applications, the `sessions.xml` and `project.xml` files are deployed inside the JAR file, and the project XML path is specified as a Java resource path.

When specifying the Java resource path, ensure that you are using the forward slash character (`/`) for directories, not the back slash (`\`). For example, `com/myapp/mypersistence/my-project.xml`, or `META-INF/my-project.xml`.

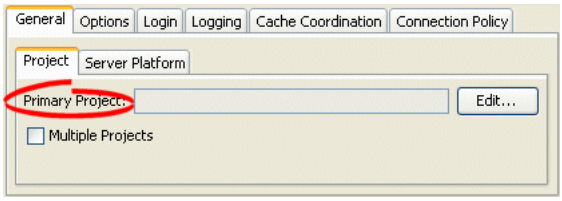
See "Configuring Multiple Mapping Projects" on page 74-9 for information on configuring additional TopLink projects for the session.

Using TopLink Workbench

To specify the primary TopLink project metadata for your session, use this procedure:

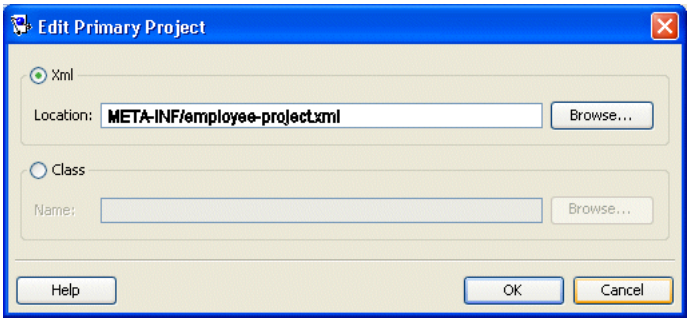
1. Select a server or database session in the **Navigator**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.
3. Click the **Project** subtab. The Project subtab appears.

Figure 74–1 General Tab, Project Subtab, Primary Project Option



4. Select the following options:
   - Click **Edit** to define the primary project. The Edit PrimaryProject dialog box appears.
   - Select the **Multiple Projects** option to add additional projects to the session. See "Configuring Multiple Mapping Projects" for more information.

Figure 74–2 Edit Primary Project Dialog Box



Use this information to enter date in each field of the Edit Primary Project dialog box:

Field	Description
XML	Select **XML** to add a mapping project as a deployment XML file. Click **Browse** to select the file.
Class	Select **Class** to add a mapping project as a *compiled* Java class file. Click **Browse** to select the file.

Using Java

Using Java, you can register descriptors with a session using the following API:

- `Project` API–Read your `project.xml` file (or instantiate your project class) and create your session using `Project` method `createServerSession` or `createDatabaseSession`.
- `Session` API–Add a descriptor or set of descriptors to a session using the `DatabaseSession` API that Table 74–4 lists. Descriptors should be registered before login, but independent sets of descriptors can be added after login.

Table 74–4 DatabaseSession API for Registering Descriptors

Session Method	Description
`addDescriptors(Project)`	Add to the session all the descriptors owned by the passed in `Project`.
`addDescriptors(Vector)`	Add to the session all the descriptors in the passed in `Vector`.
`addDescriptor(Descriptor)`	Add an individual descriptor to the session.

Configuring a Session Login

A session login encapsulates details of data source access for any session that persists to a data source. The session login overrides any other login configuration.

Table 74–5 summarizes which sessions support session login configuration.

Table 74–5 Session Support for Session Login

Session	Session Login
Server and Client Sessions	✔
Session Broker and Client Sessions	✔
Database Sessions	✔

The session login provides access to a variety of features, including the following:

- Connection configuration such as whether or not to use external connection pooling
- Sequencing configuration (that overrides sequencing configuration made at the project level, if any)
- Miscellaneous options specific to your chosen data source
- Properties (arbitrary, application-specific named values)

For more information, see the following:

- "Data Source Login Types" on page 81-2
- "Configuring a Data Source Login" on page 82-1

Configuring Logging

Use the TopLink logging framework to record TopLink behavior to a log file or session console.

Table 74–6 summarizes which sessions support logging configuration.

Table 74–6 Session Support for Logging

Session	Using TopLink Workbench	Using Java
Server and Client Sessions	✔	✔
Unit of Work Sessions	✔	✔
Session Broker and Client Sessions	✔	✔
Database Sessions	✔	✔

Note: If the session belongs to a session broker, you must specify the logging information in the session broker–not in the session itself.

For a non-CMP application, you can configure logging using TopLink Workbench (see "Using TopLink Workbench" on page 74-5). For a CMP application, see "Configuring Logging in a CMP Application" on page 74-9.

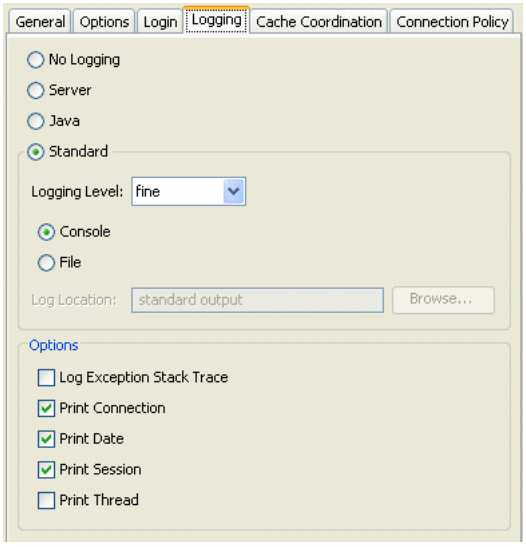
For more information, see "Logging" on page 72-7.

Using TopLink Workbench

To specify the logging information for a session, use this procedure:

1. Select a database session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Logging** tab. The Logging tab appears.

Figure 74–3 Logging Tab



Use the following information to enter data in each field of the Logging tab to select the profiler option to use with this session:

Option	Description
No Logging	Select this option to specify that nothing is logged for this session.

Option	Description
Server	Select this option to use logging capabilities of the application server to which you are deploying this application.
Java	Select this option to use `java.util.logging` package.
Standard	Select this option to use the TopLink logging framework. When selected, you can optionally configure the following options.
Logging Level	Define the amount of logging information to record (in ascending order of information): ■ **Config**–Log only login, JDBC connection, and database information. ■ **Info** (default)–Log the login/logout per sever session, with user name. After acquiring the session, detailed information is logged. ■ **Warning**–Log exceptions that do not force TopLink to stop, including all exceptions not logged with **Severe** level. This does not include a stack trace. ■ **Severe** –Log exceptions indicating TopLink cannot continue, and any exceptions generated during login. This includes a stack trace. ■ **Fine**–Log SQL (including thread information). ■ **Finer**–Similar to warning. Includes stack trace. ■ **Finest**–Includes additional low-level information. ■ **All**–Log everything.
Console	Select this option to display logging information to the standard console output.
File	Select this option to record logging information in a file. Click **Browse** to specify the name and location of the log file.
Options	Select this option to override additional logging option defaults for **Java** and **Standard** logging only.
Log Exception Stack Trace	Select this option to include the stack trace with any exception written to the log. Default: For `SEVERE` messages, log stack trace. For `WARNING` messages, only log stack trace at log level `FINER` or lower.
Print Connection	Select this option to include the connection identifier in any connection related log messages. Default: Enabled for all message and log levels.
Print Date	Select this option to include the date and time at which the log message was generated. Default: Enabled for all message and log levels.
Print Session	Select this option to include the session name in any session related log messages. Default: Enabled for all message and log levels.
Print Thread	Select this option to include the thread name in any thread related log messages. Default: Log only at log level `FINER` or lower.

Using Java

This section describes the following:

- Using Session Logging API
- Configuring a Session to use java.util.logging Package
- Configuring Logging in a CMP Application

Using Session Logging API

If you use TopLink native logging (the default), then at run time, you can configure logging options using `oracle.toplink.sessions.Session` logging API.

The `Session` interface defines the following logging methods:

- `setSessionLog`–specify the type of logging to use (any implementor of `oracle.toplink.logging.SessionLog`)
- `dontLogMessages`–disable logging
- `setLog`–specify the `java.io.Writer` to which the session logs messages
- `setLogLevel`– specify the level at which the session logs using `oracle.toplink.logging.SessionLog` constants:
  - `OFF`
  - `SEVERE`
  - `WARNING`
  - `INFO`
  - `CONFIG`
  - `FINE`
  - `FINER`
  - `FINEST`
  - `ALL`

Example 74–1 illustrates how to configure a session to use `java.util.logging` package.

Example 74–1 Configuring a Session to Use java.util.logging

```
session.setSessionLog(new JavaLog());
```

Example 74–2 illustrates how to configure a session to use the server log that OC4J provides. For more information about server logging, see "Server Logging" on page 72-9.

Example 74–2 Configuring a Session to Use Application Server Logging

```
session.setSessionLog(new OjdlLog());
```

Example 74–3 illustrates how to configure a session to log to a `java.io.Writer`:

Example 74–3 Configuring a Session to Log to a java.io.Writer

```
session.setLog(myWriter);
```

Configuring a Session to use java.util.logging Package

If you use `java.util.logging` package, then you configure logging options in the `<JRE_HOME>/lib/logging.properties` file. Messages are written to zero or multiple destinations based on this configuration file.

If you configure a session to use `java.util.logging` package, consider the following:

- logging.properties
- Formatters
- Namespace

logging.properties Configure the `logging.properties` file as Example 74–4 illustrates:

Example 74–4 java.util.logging Configuration in logging.properties

```
handlers = java.util.logging.ConsoleHandler
java.util.logging.ConsoleHandler.level = CONFIG
java.util.logging.ConsoleHandler.formatter = oracle.toplink.logging.TopLinkSimpleFormatter
oracle.toplink.LoggingSession.connection.level = CONFIG
```

For information about the types of formatters available, see "Formatters" on page 74-8.

Formatters TopLink provides two formatters: `TopLinkSimpleFormatter` and `TopLinkXMLFormatter`. They override the corresponding `java.util.logging` formatters and always log session and connection info when available. They also log thread and exception stack trace information at certain levels as specified by the logging level.

Namespace Namespace is supported for `java.util.logging`. Table 74–7 lists the static constants defined in `oracle.toplink.logging.SessionLog` for TopLink components and the corresponding strings in `logging.properties`.

Table 74–7 Logging Property FIle Names

SessionLog	logging.properites
Not Applicable	`oracle.toplink`
Not Applicable	`oracle.toplink.<sessionname>`
`SQL`	`oracle.toplink.<sessionname>.sql`
`TRANSACTION`	`oracle.toplink.<sessionname>.transaction`
`EVENT`	`oracle.toplink.<sessionname>.event`
`CONNECTION`	`oracle.toplink.<sessionname>.connection`
`QUERY`	`oracle.toplink.<sessionname>.query`
`CACHE`	`oracle.toplink.<sessionname>.cache`
`PROPAGATION`	`oracle.toplink.<sessionname>.propagation`
`SEQUENCING`	`oracle.toplink.<sessionname>.sequencing`
`EJB`	`oracle.toplink.<sessionname>.ejb`

In the `logging.properties` names listed in Table 74–7, note that `<sessionname>` is the name of the session that the application is running in. For example, if the name

of the session is `MyApplication`, then you would use `oracle.toplink.MyApplication.sql` for the SQL logging property.

An application can also define its own namespace and write to it through the logging API, as long as the logger for that namespace is defined in the logging configuration. Otherwise messages are written to the parent logger, `oracle.toplink.<sessionname>`.

Configuring Logging in a CMP Application

For a CMP project, you do not configure a session directly. In this case, you specify the type of logging by configuring system property `toplink.log.destination` with one of the following values:

- fully qualified file specification (for example, `C:\logs\toplink.log`)–use TopLink native logging to write log messages to the specified file.
- `JAVA`–use `java.util.logging` package to write log messages to any destination you configure in the *<JRE_HOME>*`/lib/logging.properties` file.
- `SERVER`–use server logging to write log messages to the application server's log file (there is no separate TopLink log file in this case).
- `SYSOUT`–write log messages to `System.out`.

The log level for TopLink standard (default) logging can be set through the `toplink.log.level` system property.

To configure other logging options, use a `customization-class` (see "Configuring pm-properties" on page 8-10).

Configuring Multiple Mapping Projects

Each session is associated with at least one mapping project (see "Configuring a Primary Mapping Project" on page 74-2). You can include additional TopLink mapping projects for a session.

Table 74–8 summarizes which sessions support additional mapping project configuration.

Table 74–8 Session Support for Additional Mapping Project

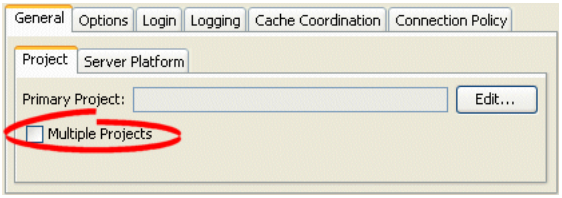
Session	Using TopLink Workbench	Using Java
Server and Client Sessions	✔	✔
Session Broker and Client Sessions	✔	✔
Database Sessions	✔	✔

Using TopLink Workbench

To specify additional TopLink projects for your session, use this procedure:

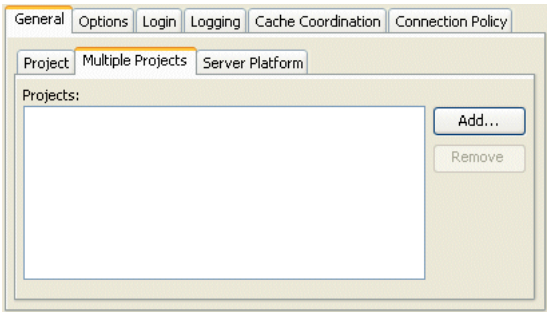
1. Select a server or database session in the **Navigator**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.
3. Click the **Project** subtab. The Project subtab appears.

Figure 74–4 General Tab, Project Subtab, Multiple Projects Options



4. Select **Multiple Projects** option. The Multiple Projects subtab appears.
5. Click the **Multiple Projects** subtab.

Figure 74–5 General Tab, Multiple Projects Subtab



To add an additional mapping project to this session, click **Add**. For more information, see "Configuring a Primary Mapping Project" on page 74-2.

To remove TopLink mapping projects, select the project file and click **Remove**.

Using Java

Using Java, you can register descriptors from more than one project with a session using the `DatabaseSession` API that Table 74–9 lists. You can register descriptors before login, but you can add independent sets of descriptors after login.

Table 74–9 DatabaseSession API for Registering Descriptors

Session Method	Description
`addDescriptors(Project)`	Add additional descriptor to the session in the form of a project.
`addDescriptors(Vector)`	Add a vector of individual descriptor files to the session in the form of a project.
`addDescriptor(Descriptor)`	Add individual descriptor to the session.

Configuring a Performance Profiler

To successfully improve the performance of a TopLink application, you must measure performance before and after each optimization. TopLink provides a variety of built-in performance measuring features (known as profilers) that you can configure at the session level.

Table 74–10 summarizes which sessions support performance profiler configuration.

Table 74–10 Session Support for Performance Profiler Configuration

Session	Using TopLink Workbench	Using Java
Server and Client Sessions	✔	✔
Session Broker and Client Sessions	✔	✔
Database Sessions	✔	✔

TopLink provides the following profilers:

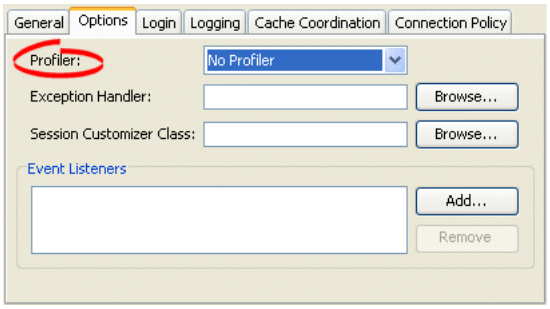
- TopLink profiler: logs performance statistics for every executed query in a given session (see "Measuring TopLink Performance With the TopLink Profiler" on page 11-2)
- Oracle Dynamic Monitoring System (DMS): includes DMS instrumentation in essential objects to provide efficient Web browser based monitoring of run-time data in TopLink-enabled applications (see "Measuring TopLink Performance With the Oracle Dynamic Monitoring System (DMS)" on page 11-4)

Using TopLink Workbench

To specify the type of profiler in a session, use this procedure:

1. Select a session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Options** tab. The Options tab appears.

Figure 74–6 Options Tab, Profiler Options



Use the following information to select the profiler option to use with this session:

Option	Description
No Profiler	Disable all profiling.
DMS	Enable Oracle Dynamic Monitoring (DMS) profiling. For more information, see the following: ■ "Configuring the Oracle DMS Profiler" on page 11-6 ■ "Measuring TopLink Performance With the Oracle Dynamic Monitoring System (DMS)" on page 11-4.
Standard (TopLink)	Enable TopLink profiling. For more information, see the following: ■ "Configuring the TopLink Performance Profiler" on page 11-3 ■ "Measuring TopLink Performance With the TopLink Profiler" on page 11-2

Using Java

You can use Java to configure a session with a profiler using `Session` method `setProfiler`, as Example 74–5 shows.

Example 74–5 Configuring a Session with a TopLink Profiler

```
session.setProfiler(new PerformanceProfiler());
```

To end a profiling session, use `Session` method `clearProfiler`.

Configuring an Exception Handler

You can associate a single exception handling class with each session. This class must implement the `oracle.toplink.exceptions.ExceptionHandler` interface.

Table 74–11 summarizes which sessions support exception handler configuration.

Table 74–11 Session Support for Exception Handler Configuration

Session	Using TopLink Workbench	Using Java
Server and Client Sessions	✔	✔
Session Broker and Client Sessions	✔	✔
Database Sessions	✔	✔

For an example exception handler implementation, see "Using Java" on page 74-12.

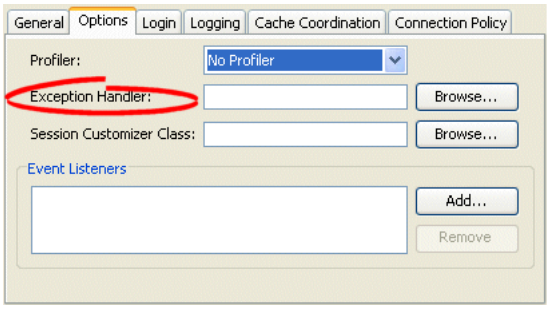
For more information, see "Exception Handlers" on page 72-12.

Using TopLink Workbench

To specify the exception handler class in a session, use this procedure:

1. Select a session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Options** tab. The Options tab appears.

Figure 74–7 Options Tab, Exception Handler Field



Click **Browse** and select the exception handler class for this session.

Using Java

Example 74–6 shows an example exception handler implementation. In this implementation, the exception handler always tries to reestablish the connection if it has been reset by peer, but only retries a query if it is an instance of `ReadQuery`. Note

that this exception handler either returns the result of the reexecuted `ReadQuery` or throws an exception.

Example 74–6 Implementing an Exception Handler

```
session.setExceptionHandler(
  new ExceptionHandler()
  {
    public Object handleException(RuntimeException exception)
    {
      if (exception instanceof DatabaseException)
      {
        DatabaseException dbex = (DatabaseException) exception;
        if ((dbex.getInternalException() instanceof SQLException) &&
           (((SQLException) dbex.getInternalException()).getErrorCode() == MyDriver.CONNECTION_RESET_BY_PEER))
        {
           dbex.getAccessor().reestablishConnection(dbex.getSession());
           if (dbex.getQuery() instanceof ReadQuery)
           {
             return dbex.getSession().executeQuery(dbex.getQuery());
           }
           throw exception;
        }
      }
      throw exception;
    }
  }
);
```

Note: Unhandled exceptions must be rethrown by the exception handler code.

Configuring Customizer Class

A session customizer class is a Java class that implements the `oracle.toplink.tools.sessionconfiguration.SessionCustomizer` interface and provides a default (zero-argument) constructor. You can use a session customizer to customize a session at run time on a loaded session before login occurs, similar to how you can use an amendment method to customize a descriptor (see "Configuring Amendment Methods" on page 25-81). For example, you can use a session customizer class to define and register session event listeners with the session event manager (see "Configuring Session Event Listeners" on page 74-17).

Table 74–12 summarizes which sessions support customizer class configuration.

Table 74–12 Session Support for Customizer Class Configuration

Session	Using TopLink Workbench	Using Java
Server and Client Sessions	✔	✔
Session Broker and Client Sessions	✔	✔
Database Sessions	✔	✔

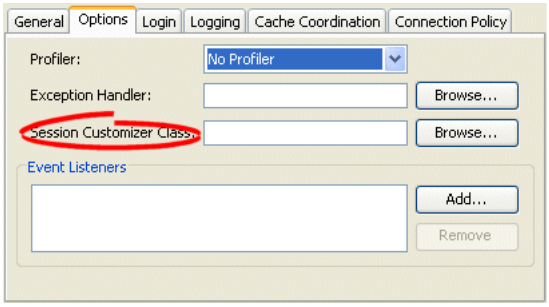
For more information, see "Session Customization" on page 72-4.

Using TopLink Workbench

To specify the session customizer class in a session, use this procedure:

1. Select a session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Options** tab. The Options tab appears.

Figure 74–8 Options Tab, Session Customizer Class Field



Click **Browse** and select the customizer class for this session.

Configuring the Server Platform

The TopLink server platform defines how a session integrates with a J2EE server including the following:

- Run-time services: Enables the deployment of a Java Management Extensions (JMX) MBean that allows monitoring of the TopLink session. Currently, this is only supported for OC4J.
- External transaction controller: Integrates the TopLink session with the server's Java Transaction API (JTA) service. This should always be used when using EJB or JTA transactions. You configure TopLink to integrate with the container's external transaction service by specifying a TopLink external transaction controller. For more information on external transaction services, see "Unit of Work Transaction Demarcation" on page 97-2.

Table 74–8 summarizes which sessions support a server platform.

Table 74–13 Session Support for Server Platform

Session	Using TopLink Workbench	Using Java
Server and Client Sessions	✔	✔
Session Broker and Client Sessions	✔	✔
Database Sessions	✔	✔

If the primary mapping project that you associate with a session has a persistence type of bean-managed persistence (BMP) or Java objects, you may configure a server platform using TopLink Workbench. For more information on primary mapping project, see "Configuring a Primary Mapping Project" on page 74-2.

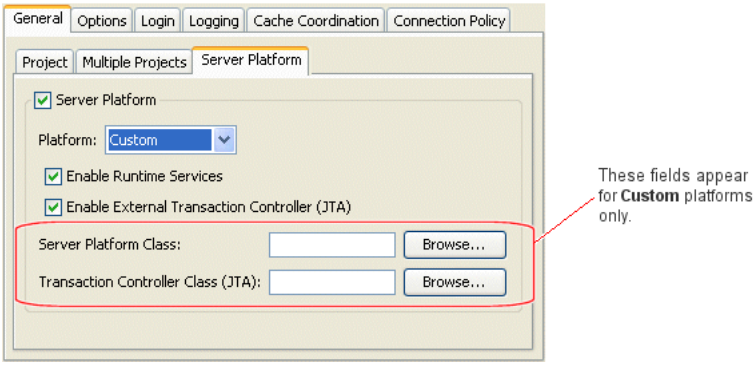
If the primary mapping project you associate with a session has a persistence type of container-managed persistence (CMP), by default, the TopLink runtime automatically configures a server platform to accommodate the application server on which it is deployed.

Using TopLink Workbench

To specify the server platform options for a session, use this procedure:

1. Select a session in the **Navigator**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.
3. Click the **Server Platform** subtab. The Server Platform subtab appears.

Figure 74–9 General Tab, Server Platform Subtab



Use the following information to enter data in each field of the Server Platform subtab:

Field	Description
Server Platform	Check this field if you intend to deploy your application to a J2EE application server. If you check this field, you must configure the target application server by selecting a **Platform**.

Field	Description
Platform	Select the J2EE application server to which you will deploy your application. TopLink supports the following J2EE application servers: ■ OC4J 10.1.3 ■ OC4J 10.1.2 ■ OC4J 9.0.4 ■ OC4J 9.0.3 ■ WebLogic 8.1 ■ WebLogic 7.0 ■ WebLogic 6.1 ■ WebSphere 6.0 ■ WebSphere 5.1 ■ WebSphere 5.0 ■ WebSphere 4.0 ■ Custom For detailed information about supported application server versions and configuration requirements, see "Integrating TopLink With an Application Server" on page 7-1. Select **Custom** if you have created your own `oracle.toplink.platform.server.ServerPlatform` class to use an application server not currently supported by TopLink or to override an existing `ServerPlatform`. If you select **Custom**, you must specify your custom `ServerPlatform` class by selecting a **Server Platform Class**. The server platform you select overrides the default server platform set at the sessions configuration level (see "Creating a Sessions Configuration" on page 73-1).
Enable Runtime Services	Check this field to configure the TopLink runtime to enable the deployment of a JMX MBean that allows monitoring of the TopLink session. Currently, this is only supported for OC4J. To use this feature, you must enable DMS data collection. For more information, see "Configuring the Oracle DMS Profiler" on page 11-6.
Enable External Transaction Controller (JTA)	Check this field if you intend to integrate your application with an external transaction controller. For more information, see "Unit of Work Transaction Demarcation" on page 97-2. If you configure **Platform** for a J2EE application server that TopLink supports, the TopLink runtime will automatically select the appropriate external transaction controller class. If you configure **Platform** as **Custom**, you must specify an external transaction controller class by selecting an **External Transaction Controller**.
Server Platform Class	This option is only available if you configure **Platform** as **Custom**. Click Browse to select your custom `ServerPlatform` class.
Transaction Controller Class (JTA)	This option is only available if you configure **Platform** as **Custom**. If you checked **Enable External Transaction Controller (JTA)**, click **Browse** to select the transaction controller class that corresponds with your custom `ServerPlatform` class.

Using Java

When using Java, you must pass the session in a server platform constructor. Example 74–7 illustrates using a session event listener (see "Configuring Session Event Listeners" on page 74-17) to configure a session with a server platform from the `oracle.toplink.platform.server` package.

Example 74–7 Configuring a Session with a Server Platform

```
public void preLogin(SessionEvent event) {
  Server server = (Server) event.getSession();
  server.setServerPlatform(new Oc4j_10_1_3_Platform((DatabaseSession) server));
  }
```

Configuring Session Event Listeners

As you perform persistence operations with a session, the session produces various events (see "Session Event Manager Events" on page 72-6) that the TopLink runtime uses to coordinate its various components. You can configure a session with one or more session event listeners (see "Session Event Listeners" on page 72-7) to customize session behavior and debug session operations. For example, session event listeners play an important role in the configuration of isolated sessions (see "Configuring Exclusive Isolated Client Sessions for Virtual Private Database" on page 77-1).

Table 74–14 summarizes which sessions support event listeners.

Table 74–14 Session Support for Event LIsteners

Session	Using TopLink Workbench	Using Java
Server and Client Sessions	✔	✔
Session Broker and Client Sessions	✔	✔
Database Sessions	✔	✔

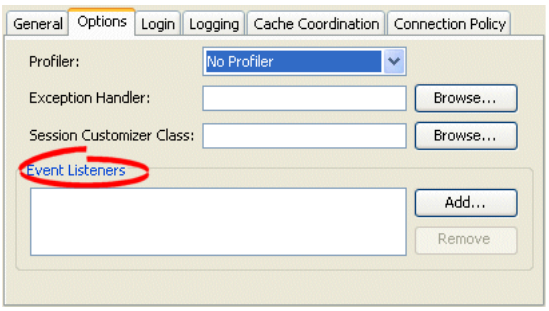
Using TopLink Workbench

Session Event Listeners

To specify the event listener class in a session, use this procedure:

1. Select a session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Options** tab. The Options tab appears.

Figure 74–10 Options Tab, Event Listeners field



To add a new event listener, click **Add**, then select the event listener class for this session.

To remove an existing event listener, select the **Event Listener** and click **Remove**.

Using Java

Example 74–8 illustrates how to use Java to register a session event listener with a session. TopLink provides a `SessionEventAdapter` to simplify creating a `SessionEventListener`. The `SessionEventAdapter` provides a default implementation of all the methods of the `SessionEventListener` interface. You need only override the specific methods of interest. Typically, you would define session event listeners in a session customizer class (see "Configuring Customizer Class" on page 74-13).

Example 74–8 Using the Session Event Adapter to Create a Session Event Listener

```
...
SessionEventAdapter myEventListener = new SessionEventAdapter() {
    // Listen for PostCommitUnitOfWork events
    public void postCommitUnitOfWork(SessionEvent event) {
        // Call the handler routine
        unitOfWorkCommitted();
    }
};
mySession.getEventManager().addListener(myEventListener);
...
```

For information on how to add logging to your listeners, see "Logging" on page 72-7.

Configuring the Integrity Checker

When you log into a session, TopLink initializes and validates the descriptors you registered with it. By configuring the integrity checker, you can customize this validation process to do the following:

- Check Database
- Catch All Exceptions
- Catch Instantiation Policy Exceptions

Table 74–15 summarizes which sessions support descriptor integrity checking configuration.

Table 74–15 Session Support for Checking Descriptor Integrity

Session	Using TopLink Workbench	Using Java
Server and Client Sessions		✔
Session Broker and Client Sessions		✔
Database Sessions		✔

Check Database

The `IntegrityChecker` method `setShouldCheckDatabase` specifies whether or not the integrity checker should verify the descriptor's metadata against the database metadata. This will report any errors due to missing or incorrect table or fields

specified in the descriptors. This is turned off by default as it adds a significant overhead to connecting a session.

Catch All Exceptions

By default, the integrity checker catches all exceptions that occur during initialization, and throws a single exception at the end of initialization reporting all of the errors detected. If you only want the first exception encountered, you can disable this feature using `IntegrityChecker` method `setShouldCatchExceptions(false)`.

Catch Instantiation Policy Exceptions

By default, the integrity checker tests the default or configured constructor for each descriptor initialized in the session. To disable this feature, use `IntegrityChecker` method `setShouldCheckInstantiationPolicy(false)`.

Using Java

As Example 74–9 shows, you can configure the integrity checker validation process.

Example 74–9 Configuring the Integrity Checker

```
session.getIntegrityChecker().setShouldCheckDatabase(true);
session.getIntegrityChecker().setShouldCatchExceptions(false);
session.getIntegrityChecker().setShouldCheckInstantiationPolicy(false);
session.login();
```

Configuring Connection Policy

Using a connection policy, you can control how a TopLink session acquires and uses read and write connections, including the following:

- Exclusive Write Connections
- Lazy Connection Acquisition

Table 74–15 summarizes which sessions support connection policy configuration.

Table 74–16 Session Support for Connection Policy

Session	Using TopLink Workbench	Using Java
Server and Client Sessions	✔	✔
Session Broker and Client Sessions		
Database Sessions		

Exclusive Write Connections

An exclusive connection is one that TopLink allocates to a client session for reading (of isolated data) and writing for the duration of the client session's life cycle.

By default, exclusive connections are not used and a client session uses the server session's read connection pool for all non-pessimistic read queries. A connection is obtained from the read connection pool for each read query execution and released back to the pool after the query is executed. A connection is only obtained from the write connection pool for the unit of work commit operation, or, potentially, earlier if data modify queries, or read queries using pessimistic locking are used. The connection will be release back to the write connection pool after the unit of work is committed or released.

Exclusive connections are provided for use with database read security or Virtual Private Database (VPD) support. When using an exclusive connection, you will obtain it from the server session's write connection pool. When you acquire the client, the exclusive connection will be used for read queries to isolated classes (see "Isolated Client Sessions" on page 72-19), exclusive read queries, pessimistic read queries, and for the unit of work commit operation. The exclusive connection will only be released when the client session is released. TopLink still acquires a shared connection from the read connection pool for reading nonisolated data. If you use a JTA-managed external connection pool with exclusive connections, do not reuse a client session across JTA transaction boundaries, as the physical JTA database connection is released and acquired from the connection pool relative to the JTA transaction life cycle. A new client session, or the active unit of work, should be used for each JTA transaction. For more information, see "Configuring Exclusive Read Connections" on page 86-6.

You can also configure exclusive connections on a client-session-by-client-session basis (see "Acquiring a Client Session That Uses Exclusive Connections" on page 75-7) and for named queries (see "Configuring Named Query Advanced Options" on page 25-24).

Note: If any client session contains an exclusive connection, you must release the session (see "Logging Out of a Session" on page 75-10) when you are finished using it. Relying on the finalizer to release the connection when the session is garbage collected may cause errors when dealing with JTA transactions.

Lazy Connection Acquisition

By default, TopLink acquires write connections lazily, when you perform the first unit of work commit operation, exclusive read query, or pessimistic read query with your client session. The write connection will also be released after each unit of work it committed or released.

Alternatively, you can configure TopLink to acquire the write connection at the time you acquire a client session, and release the connection when you release the client session.

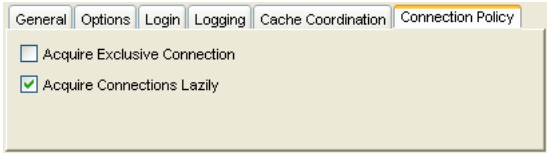
You can also configure lazy connection acquisition on a client-session-by-client-session basis (see "Acquiring a Client Session That Does Not Use Lazy Connection Allocation" on page 75-9).

Using TopLink Workbench

To specify the connection policy in a session, use this procedure:

1. Select a session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Connection Policy** tab. The Connection Policy tab appears.

Figure 74–11 Connection Policy Tab



Using Java

To configure whether or not an exclusive connection is allocated to a particular isolated session, use `ConnectionPolicy` method `setShouldUseExclusiveConnection`.

To define a map of properties used to support an isolated session, use the following `ConnectionPolicy` methods:

- `setProperty(Object key, Object value)`: Adds the property `value` to the `Map` under `key`, overwriting the existing value if `key` already exists in the `Map`.
- `Object getProperty(Object key)`: Returns the value associated with `key` as an `Object`.
- `boolean hasProperties`: Returns `true` if one or more properties exist in the `Map`; otherwise returns false.

The TopLink runtime passes this `Map` into `SessionEvent` events `PostAcquireExclusiveConnection` and `PreReleaseExclusiveConnection` so that your implementation can make the appropriate PL/SQL calls to the underlying database platform (see "PostAcquireExclusiveConnection Event Handler" on page 77-1 and "PreReleaseExclusiveConnection Event Handler" on page 77-2).

To configure the session to use a named connection pool, use the `ConnectionPool` constructor that takes a `String` connection pool name as an argument:

```
Session clientSession = server.acquireClientSession(
    new ConnectionPolicy("myConnectionPool")
);
```

Configuring Named Queries at the Session Level

A **named query** is a TopLink query that you create and store, by name, in a session for later retrieval and execution. Named queries improve application performance, because they are prepared once and they (and all their associated supporting objects) can be efficiently reused thereafter making them well-suited for frequently executed operations.

If a named query is global to a project, configure it at the session level. Alternatively, you can configure a named query at the descriptor level (see "Configuring Named Queries at the Descriptor Level" on page 25-10).

Use named queries to specify SQL, EJB QL, or TopLink `Expression` queries to access your data source.

Table 74–17 summarizes which sessions support named query configuration.

Table 74–17 Session Support for Named Queries

Session	Using TopLink Workbench	Using Java
Server and Client Sessions		✔
Session Broker and Client Sessions		✔
Database Sessions		✔

After you create a named query, you can execute it by name on the TopLink session (see "Using Named Queries" on page 94-18).

For more information about named queries, see "Named Queries" on page 93-16.

Using Java

You can store a query by name in a `Session` using `Session` method `addQuery(String name, DatabaseQuery query)`.

75

Acquiring and Using Sessions at Run Time

After you create and configure sessions, you can use the session manager to acquire a session instance at run time.

This section explains the following:

- Session Acquisition Overview
- Acquiring the Session Manager
- Acquiring a Session From the Session Manager
- Acquiring a Client Session
- Logging In to a Session
- Using Session API
- Logging Out of a Session
- Storing Sessions in the Session Manager Instance
- Destroying Sessions in the Session Manager Instance

Session Acquisition Overview

Oracle recommends that you export session instances from TopLink Workbench to one or more uniquely named `sessions.xml` files and then use the session manager to load sessions from these `sessions.xml` files.

The 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 TopLink sessions from the `sessions.xml` file, caches the sessions by name in memory, and provides a single access point for TopLink sessions.

The session manager supports the following session types:

- Server Session
- Database Session
- SessionBroker

See Chapter 72, "Understanding TopLink Sessions" for detailed information on these available sessions.

The session manager has two main functions: it creates instances of the sessions and it ensures that only a single instance of each named session exists for any instance of a session manager.

This is particularly useful for EJB applications in that an enterprise bean can acquire the session manager and acquire the desired session from it.

Understanding the Session Manager

When a client application requires a session, it requests the session from the TopLink session manager. The two main functions of the session manager are to instantiate 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:

- The client application requests a session by name.
- 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 throws an exception.
- After instantiation, the session remains viable until the application is shut down.

Multiple Sessions

Oracle does not recommend instantiating a session and passing it around as a global entity.

Oracle recommends that you acquire sessions from the session manager and perform all persistence operations using the unit of work acquired from the session.

Note that in the case of a server session or a session broker that contains server sessions, after you acquire the session you will acquire a client session from it. From a given server session (or session broker that contains server sessions), you can acquire as many client sessions as you have clients.

Each client can easily manage concurrent access and referential constraints by acquiring a unit of work from its client session and performing all persistence operations using the unit of work.

Acquiring the Session Manager

TopLink maintains only one instance of the session manager class. The singleton session manager maintains all the named TopLink sessions at run time. When an application requests a session by name, the session manager retrieves the specified session from the appropriate configuration file.

As Example 75–1 illustrates, to access the session manager instance, use the `oracle.toplink.tools.sessionmanagement.SessionManager` method `getManager`. You can then use the session manager instance to load TopLink sessions.

Example 75–1 Acquiring a Session Manager Instance

```
import oracle.toplink.tools.sessionmanagement.SessionManager;
SessionManager sessionManager = SessionManager.getManager();
```

Acquiring a Session From the Session Manager

When the session manager loads a session that is not yet in its cache, the session manager creates an instance of the appropriate session type and configures it according to the `sessions.xml` file configuration.

> **Note:** To best use 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.

This section explains the following:

- Loading a Session From sessions.xml Using Defaults
- Loading a Session From sessions.xml With an Alternative Class Loader
- Loading a Session From an Alternative Session Configuration File
- Loading a Session Without Logging In
- Reloading and Refreshing Session Configuration
- Refreshing a Session When the Class Loader Changes

Loading a Session From sessions.xml Using Defaults

If you have a single sessions configuration file (`sessions.xml`) that contains all the session instances created from by TopLink Workbench, then you can load a session by name, as Example 75–2 illustrates.

Example 75–2 Acquiring a Named Session from Session Manager Using Defaults

```
/* Load a named session (mysession) defined in the sessions.xml file. */
SessionManager manager = SessionManager.getManager();
Session session = manager.getSession("mysession");
```

In this example, the following session manager default configuration applies:

- Class loader–The thread-based class loader is used to find and load the `sessions.xml` resource and resolve any classes referenced in the `sessions.xml` and `project.xml` files.

  If you acquire the session in an application class, this will typically be the application's class loader, which is correct. In a J2EE application, it is best to specify this as the class loader from a class in the same JAR file that the `sessions.xml` file is deployed in.

- File–By default, the file named `sessions.xml` in the root directory relative to the class loader is used.

  If the file is named differently, or not in the root directory, the relative path must be specified. Relative resource paths in Java must use `/`, not `\`.

- Session name–The name passed into the `getSession` call.

  This name must be unique for the entire application server, not just unique within the application.

- Login–`true`. The session will be connected by default.

If you must manually configure the session before login, set this option to `false` (see "Loading a Session Without Logging In" on page 75-5).

- Refresh–`false`. If already loaded, the same session will be returned.

  Refresh should only be used, if it is known that the existing session is not being used, and the configuration has changed, such as in a J2EE environment redeployment scenario.

- Verify class loader–false. The session manager will not refresh the session if the class loader changes.

  This should normally be set to `true`. It must be set to `true` in a J2EE environment, if hot deployment or redeployment to a running application server is required (see "Refreshing a Session When the Class Loader Changes" on page 75-6).

Loading a Session From sessions.xml With an Alternative Class Loader

You can use an alternative class loader to load sessions. This is common when your TopLink application integrates with a J2EE container. The session manager uses the class loader to find and load the `sessions.xml` resource and resolve any classes referenced in the `sessions.xml` and `project.xml` files.

In most cases, you use the class loader from the current thread context, as Example 75–3 illustrates. In this example, the session named `mysession` is loaded from the first file in the application classpath named `sessions.xml` using the class loader associated with the current thread context.

Example 75–3 Loading a Session Using the Current Thread Context Class Loader

```
/* Use the specified ClassLoader to load a session (mysession) defined in the
sessions.xml file */
SessionManager manager = SessionManager.getManager();
Session session = manager.getSession(
    "mysession",  // session name to load
    Thread.current().getContextClassLoader() // ClassLoader instance to use
);
```

However, if your J2EE container does not support using the current thread context class loader, you can use the class loader from the current class, as Example 75–4 illustrates.

Example 75–4 Loading a Session Using the Current Class's Class Loader

```
/* Use the specified ClassLoader to load a session (mysession) defined in the
sessions.xml file */
SessionManager manager = SessionManager.getManager();
Session session = manager.getSession(
    "mysession",  // session name to load
    this.getClass().getClassLoader() // ClassLoader instance to use
);
```

Note: Oracle Containers for J2EE supports the use of the class loader from the current thread.

Loading a Session From an Alternative Session Configuration File

If your session instances are contained in multiple, uniquely named session configuration files (`sessions.xml` files), then you must explicitly create an

`XMLSessionConfigLoader` object initialized with the name of the `sessions.xml` file and pass that `XMLSessionConfigLoader` into the `SessionManager` method `getSession`, as Example 75–5 illustrates.

The file path you specify is relative to the class loader root directory. Relative resource paths in Java must use the forward slash (`/`), not back slash (`\`).

In this example, the session named `mysession` is loaded by the specified class loader from the first file in the application classpath named `toplink-sessions.xml`.

Example 75–5 Loading a Session from an Alternative Configuration File

```
/* XMLSessionConfigLoader loads the toplink-sessions.xml file */
SessionManager manager = SessionManager.getManager();
manager.getSession(
    new XMLSessionConfigLoader("toplink-sessions.xml"),
    "mysession",
    this.class.getClassLoader()
);
```

Loading a Session Without Logging In

The `XMLSessionConfigLoader` (see "Loading a Session From an Alternative Session Configuration File" on page 75-4) lets you call a session using the `SessionManager` method `getSession`, without invoking the `Session` method `login`, as Example 75–6 shows. This lets you prepare a session for use and leave login to the application.

Example 75–6 Open Session with No Login

```
SessionManager manager = SessionManager.getManager();
Session session = manager.getSession(
    new XMLSessionConfigLoader(), // XMLSessionConfigLoader (sessions.xml file)
    "mysession", // session name
    YourApplicationClass.getClassLoader(), // class loader
    false, // do not log in session
    false); // do not refresh session
```

Reloading and Refreshing Session Configuration

You can tell the session manager to refresh an existing session from the `sessions.xml` file. Typically, this would only ever be used in a J2EE environment at redeployment time, or after a reset of a running server. You should only use this option when you know that the existing session is not being used.

Example 75–7 Forcing a Reparse of the sessions.xml File

```
//In this example, XMLSessionConfigLoader loads sessions.xml from the classpath
SessionManager manager = SessionManager.getManager();
Session session = manager.getSession(
    new XMLSessionConfigLoader(), // XMLSessionConfigLoader (sessions.xml file)
    "mysession", // session name
    YourApplicationClass.getClassLoader(), // class loader
    true, // log in session
    true  // refresh session
);
```

Refreshing a Session When the Class Loader Changes

In a non-CMP J2EE environment, if you require hot deployment or redeployment to a running application server, you must tell the session manager to refresh your session if the class loader changes, as Example 75–8 shows. This option makes the session manager refresh the session if the class loader changes, which occurs when the application is redeployed. When this option is set to `true`, the same class loader must always be used to retrieve the session.

Example 75–8 Forcing a Reparse of the sessions.xml File

```
//In this example, XMLSessionConfigLoader loads sessions.xml from the classpath
SessionManager manager = SessionManager.getManager();
Session session = manager.getSession(
    new XMLSessionConfigLoader(), // XMLSessionConfigLoader (sessions.xml file)
    "mysession", // session name
    YourApplicationClass.getClassLoader(), // class loader
    true,   // log in session
    false,  // do not refresh session when loaded
    true    // do refresh session if class loader changes
);
```

In a CMP J2EE environment, the TopLink runtime and CMP integration handles this for you automatically.

Acquiring a Client Session

Before you can acquire a client session, you must first use the session manager to acquire a server session or a session broker that contains server sessions (see "Acquiring a Session From the Session Manager" on page 75-3).

Table 75–1 summarizes the methods used to acquire various types of client sessions from a server session and a session broker session that contains server sessions.

Table 75–1 Method Used to Acquire a Client Session

Client Session	Server Session Method	Session Broker Session Method
Regular or Isolated	`acquireClientSession()`	`acquireClientSessionBroker()`
Regular or Isolated	`acquireClientSession(ConnectionPolicy)`	*not applicable*
Historical	`acquireHistoricalSession(AsOfClause)`	*not applicable*

The `acquireClientSession` method returns a session of type `ClientSession`.

The `acquireClientSessionBroker` method returns a session of type `SessionBroker`.

In both cases, you should cast the returned object to type `Session` and use it as you would any other session.

For more information, see the following:

- "Acquiring an Isolated Client Session" on page 75-7
- "Acquiring a Historical Session" on page 75-9
- "Acquiring a Client Session That Uses Exclusive Connections" on page 75-7
- "Acquiring a Client Session That Uses Connection Properties" on page 75-8

- "Acquiring a Client Session That Uses a Named Connection Pool" on page 75-8
- "Acquiring a Client Session That Does Not Use Lazy Connection Allocation" on page 75-9

Acquiring an Isolated Client Session

If in your TopLink project you configure all classes as isolated (see "Configuring Cache Isolation at the Project Level" on page 19-16), or one or more classes as isolated (see "Configuring Cache Isolation at the Descriptor Level" on page 25-37), then all client sessions that you acquire from a parent server session will be isolated client sessions (see "Isolated Client Sessions" on page 72-19).

Using a `ConnectionPolicy`, you can acquire an isolated client session that uses exclusive connections (see "Acquiring a Client Session That Uses Exclusive Connections" on page 75-7). This isolated client session can be configured with connection properties for use with the Oracle Virtual Private Database (VPD) feature (see "Acquiring a Client Session That Uses Connection Properties" on page 75-8).

For more information about VPD, see "Isolated Client Sessions and Oracle Virtual Private Database (VPD)" on page 72-21.

Acquiring a Client Session That Uses Exclusive Connections

Example 75–9 illustrates how to configure a `ConnectionPolicy` and use it to acquire a client session that uses exclusive connections.

Example 75–9 Acquiring a Client Session that Uses Connection Properties

```
ConnectionPolicy connectionPolicy = new ConnectionPolicy();
// Use an exclusive connection for the session
connectionPolicy.setShouldUseExclusiveConnection(true);

Session clientSession = server.acquireClientSession(connectionPolicy);
// By default, an exclusive connection will be acquired lazily
```

An exclusive connection is allocated from a shared connection pool. The connection is dedicated to the client session that acquires it.

Note: Typically, the life cycle of a client session is the duration of a server request. However, if you are using JTA, it is the life cycle of a JTA transaction.

You cannot hold the client session across the JTA transaction boundaries. If you are not using a unit of work in your transaction and you are configuring a client session to use an exclusive connection (see Chapter 77, "Configuring Exclusive Isolated Client Sessions for Virtual Private Database"), you must explicitly acquire and release the session when you are finished using it. Although client sessions have a finalizer that would release the session when it is garbage-collected, you must not rely on the finalizer and release the exclusive client session (or a non-lazy session) in the application to release the data source connection. Note that the requirement to release the session in not JTA-specific.

If you are using a unit of work (see Chapter 99, "Using Advanced Unit of Work API"), you do not have to worry about releasing its client session as the unit of work always automatically releases it at the end of the JTA transaction.

A named query can also use an exclusive connection (see "Configuring Named Query Advanced Options" on page 25-24).

For more information, see the following:

- "Acquiring a Client Session That Does Not Use Lazy Connection Allocation" on page 75-9
- "Configuring Connection Policy" on page 74-19.

Acquiring a Client Session That Uses Connection Properties

Example 75–10 illustrates how to configure a `ConnectionPolicy` and use it to acquire a client session that uses connection properties. In this example, the properties are used by the Oracle VPD feature (see "Isolated Client Sessions and Oracle Virtual Private Database (VPD)" on page 72-21). You can use connection properties for other application purposes.

Example 75–10 Acquiring an Isolated Session Using Connection Properties

```
ConnectionPolicy connectionPolicy = new ConnectionPolicy();
// Set VPD specific properties to be used in the events
connectionPolicy.setProperty("userLevel", new Integer(5));

Session clientSession = server.acquireClientSession(connectionPolicy);
```

For more information, see "Configuring Connection Policy" on page 74-19.

Acquiring a Client Session That Uses a Named Connection Pool

Before you can acquire a client session that uses a named connection pool, you must configure your session with a named connection pool. For more information on named connection pools, see "Application-Specific Connection Pools" on page 81-9. For more information on creating a named connection pool, see "Internal Connection Pool Creation Overview" on page 85-1.

To acquire a client session that uses a named connection pool, use `Server` method `acquireClientSession`, passing in a `ConnectionPolicy` configured with the desired connection pool. The acquired `ClientSession` uses connections from the specified pool for writes (reads still go through the `Server` read connection pool).

Example 75–11 illustrates how to configure a `ConnectionPolicy` to specify a named connection pool named `myConnectionPool`.

Example 75–11 Acquiring a Client Session that Uses a Named Connection Pool

```
// Assuming you created a connection pool named "myConnectionPool"
Session clientSession = server.acquireClientSession(
    new ConnectionPolicy("myConnectionPool")
);
```

For more information, see "Configuring Connection Policy" on page 74-19.

Acquiring a Client Session That Does Not Use Lazy Connection Allocation

By default, the server session does not allocate a data source connection for a client session until a transaction starts (a lazy data source connection). Alternatively, you can acquire a client session that allocates a connection immediately.

Example 75–12 illustrates how to configure a `ConnectionPolicy` to specify that lazy connection allocation is not used.

Example 75–12 Acquiring a Client Session that Does Not Use Lazy Connections

```
ConnectionPolicy connectionPolicy = new ConnectionPolicy();
connectionPolicy.setIsLazy(false);
Session clientSession = server.acquireClientSession(connectionPolicy);
```

For more information, see "Configuring Connection Policy" on page 74-19.

Acquiring a Historical Session

After you configure TopLink to access historical data (see "Historical Session Configuration Overview" on page 78-1), you can query historical data using any session type.

When you query historical data using a regular client session or database session, you must always set `ObjectLevelReadQuery` method `maintainCache` to `false` in order to prevent old (historical) data from corrupting the session cache. However, you can query both current and historical object versions.

As a convenience, TopLink provides a historical session to simplify this process. When you query historical data using a historical session, you do not need to set `ObjectLevelReadQuery` method `maintainCache` to `false`. However, you can query objects only as of the specified time.

Before you can acquire a historical session, you must first use the session manager to acquire a server session.

To acquire a historical session, use `Server` method `acquireHistoricalSession` passing in an `AsOfClause`.

The `AsOfClause` specifies a point in time that applies to all queries and expressions subsequently executed on the historical session. The historical session's cache is a read-only snapshot of object versions as of the specified time. Its cache is isolated from its parent server session's shared object cache.

Logging In to a Session

Before you can use a session, you must first log in to the session using `Session` method `login`.

By default, when you load a session using the session manager, TopLink automatically logs in to the session using the zero-argument `login` method. For information on loading a session without automatically logging into the session, see "Loading a Session Without Logging In" on page 75-5.

If you load a session without logging in, you can choose from the following signatures of the `login` method:

- `login()`: Use the Login, user name, and password defined in the corresponding `sessions.xml` file.
- `login(Login login)`: Override the `Login` defined in the corresponding `sessions.xml` file with the specified `Login`.
- `login(String username, String password)`: Override the user name and password defined in the corresponding `sessions.xml` file with the specified user name and password.

When you log in to a session broker, the 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. TopLink handles the multiple database access transparently.

Using Session API

For more information on using session API, for caching, see "Understanding the Cache" on page 87-1.

For more information on using session API for queries, see "Understanding TopLink Queries" on page 93-1.

For more information on using session API for transactions, see "Understanding TopLink Transactions" on page 97-1.

Logging Out of a Session

When you are finished using a database or server session, you must log out of the session using `Session` method `logout`.

When logging out of a session broker session logs out of all sessions registered with the session broker.

When you are finished using a client session, you must release the session and any connections allocated to it using `Session` method `release`.

Although TopLink provides a finalizer to release sessions, this is a last resort. Oracle recommends that you always log out of your sessions.

Storing Sessions in the Session Manager Instance

Although Oracle recommends that you export all session instances from TopLink Workbench to one or more `sessions.xml` files, alternatively, you can manually create a session in your application and, as Example 75–13 illustrates, manually store it in the session manager using `SessionManager` method `addSession`. Then, you can acquire a session by name using the `SessionManager` method `getSession`.

Note: The `addSession` method is not necessary if you are loading sessions from a session configuration file.

Example 75–13 Storing Sessions Manually in the Session Manager

```
// create and log in to the 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

You can destroy sessions individually by name or destroy all sessions.

Note: You should only do this when a J2EE application is un-deployed, or when the entire application is shut down and only when it is known that the session is no longer in use. You should log out of a session before destroying it (see "Logging Out of a Session" on page 75-10). If you do not log out of a session, the session manager will at the time you use it to destroy a session.

To destroy one session instance by name, use `SessionManager` method `destroySession`, as Example 75–14 illustrates. If the specified session is not in the session manager cache, a `ValidationException` is thrown.

Example 75–14 Destroying a Session in the Session Manager

```
SessionManager manager = SessionManager.getManager();
Server server = (Server) manager.getSession("myserversession");
…
// Destroy session by name. If the session named myserversession is not in the
// session manager cache, throw a ValidationException
manager.destroySession("myserversession");
```

To destroy all session instances, use the `SessionManager` method `destoryAllSessions`, as Example 75–15 illustrates.

Example 75–15 Destroying All Sessions in the Session Manager

```
SessionManager manager = SessionManager.getManager();
Server server = (Server) manager.getSession("myserversession");
SessionBroker broker = (SessionBroker) manager.getSession("mysessionbroker");
…
// Destroy all sessions stored in the session manager
manager.destroyAllSessions();
```

76

Configuring Server Sessions

This chapter describes the various components that you must configure to use server and client sessions.

Server Session Configuration Overview

Table 76–1 lists the configurable options for server sessions.

Table 76–1 Configurable Options for Server Sessions

Option	Type	TopLink Workbench	Java
"Configuring Internal Connection Pools" on page 76-1	Basic	✔	✔
"Configuring a Primary Mapping Project" on page 74-2	Basic	✔	✔
"Configuring a Session Login" on page 74-4	Basic	✔	✔
"Configuring Logging" on page 74-4	Basic	✔	✔
"Configuring External Connection Pools" on page 76-2	Advanced	✔	✔
"Configuring Multiple Mapping Projects" on page 74-9	Advanced	✔	✔
"Configuring a Performance Profiler" on page 74-10	Advanced	✔	✔
"Configuring an Exception Handler" on page 74-12	Advanced	✔	✔
"Configuring Customizer Class" on page 74-13	Advanced	✔	✔
"Configuring the Server Platform" on page 74-14	Advanced	✔	✔
"Configuring Session Event Listeners" on page 74-17	Advanced	✔	✔
"Configuring a Coordinated Cache" on page 88-1	Advanced	✔	✔
"Configuring the Integrity Checker" on page 74-18	Advanced	✔	✔
"Configuring Named Queries at the Session Level" on page 74-21	Basic		✔

Configuring Internal Connection Pools

An internal connection pool is a collection of reusable connections to a single data source provided by any session that persists to a data source. By default, such a session provides both an internal read and write connection pool.

In this case, you can do the following:

- Configure read and write connection pool options such as minimum and maximum number of connections, alternate connection configuration, and properties (arbitrary, application-specific named values).

- Create named connection pools for whatever application-specific purpose you choose.
- Create sequence connection pools that TopLink uses exclusively for obtaining object identifiers.

For more information about creating and configuring internal connection pools, see the following:

- "Creating an Internal Connection Pool" on page 85-1
- "Configuring an Internal Connection Pool" on page 86-1

For more information about configuring the type of connection pool your session uses, see "Configuring External Connection Pooling" on page 82-2.

Configuring External Connection Pools

An external connection pool is a collection of reusable connections to a single data source provided by a JDBC driver or J2EE container.

By default, a session uses internal connection pools (see "Configuring Internal Connection Pools" on page 76-1). For more information about configuring a session to use an external connection pool, see "Configuring External Connection Pooling" on page 82-2.

77

Configuring Exclusive Isolated Client Sessions for Virtual Private Database

This chapter describes the various components that you must configure before you can acquire an exclusive isolated client session from a server session.

Exclusive Isolated Client Session Configuration Overview

Table 77–1 lists the configurable options for isolated sessions.

Table 77–1 Configurable Options for Isolated Client Sessions

Option	Type	TopLink Workbench	Java
"Configuring Cache Isolation at the Descriptor Level" on page 25-37	Basic		✔
"Configuring Connection Policy" on page 74-19	Basic		✔
"PostAcquireExclusiveConnection Event Handler" on page 77-1	Basic		✔
"PreReleaseExclusiveConnection Event Handler" on page 77-2	Basic		✔
"NoRowsModifiedSessionEvent Event Handler" on page 77-3	Basic		✔
"ValidationException Handler" on page 77-3	Basic		✔

These options are used throughout the isolated session life cycle (see "Isolated Client Session Life Cycle" on page 72-22).

PostAcquireExclusiveConnection Event Handler

TopLink raises this event after an exclusive connection is allocated to an isolated session after the user has logged in to the database with it.

If you are using Oracle Database proxy authentication (see "Oracle Database Proxy Authentication" on page 81-5), then you do not need to implement this session event handler.

If you are not using Oracle Database proxy authentication, then, as part of the isolated session life cycle, you must implement a `SessionEventListener` for `SessionEvent.PostAcquireExclusiveConnection`.

Note: You must add this session event listener to the server session from which you acquire your isolated client session. You cannot add them to the isolated client session itself. For more information, see "Configuring Session Event Listeners" on page 74-17

Using Java

The `SessionEvent.PostAcquireExclusiveConnection` event listener is your opportunity to authenticate your user and interact with the underlying database platform: for example, to execute PL/SQL to create VPD packages and set VPD context information.

Example 77–1 illustrates a typical session event listener used to handle `postAcquireExclusiveConnection` events for an isolated session.

Example 77–1 Session Event Listener for an Isolated Session

```
class VPDEventListener extends SessionEventAdaptor{
    public void postAcquireExclusiveConnection(SessionEvent event){
        ClientSession session = (ClientSession)event.getSession();
        // Get property set on the ConnectionPolicy prior to acquiring the connection
        String userLevel = session.getConnectionPolicy().getProperty("userLevel");
        // Make the Stored Procedure call for VPD to set up the Context Information
        session.executeNonSelectingSQL("StoreProcSetContextUser("+ userLevel + ")");
    }
}
```

To get the required user credentials, use `ClientSession` method `getConnectionPolicy` to get the associated `ConnectionPolicy`, and then use `ConnectionPolicy` method `getProperty`. The `ConnectionPolicy` associated with the `ClientSession` should contain all required user credentials (see "Configuring Connection Policy" on page 74-19).

After you implement the required `SessionEventListener`, add it to the parent server session from which you acquire your isolated client session. For more information, see "Configuring Session Event Listeners" on page 74-17.

PreReleaseExclusiveConnection Event Handler

TopLink raises a `SessionEvent.PreReleaseExclusiveConnection` event after you call the isolated session method `release`.

If you are using Oracle Database proxy authentication (see "Oracle Database Proxy Authentication" on page 81-5), then you do not need to implement this session event handler.

If you are not using Oracle Database proxy authentication, then as part of the isolated session life cycle, you must implement a `SessionEventListener` for `SessionEvent.PreReleaseExclusiveConnection`.

> **Note:** You must add this session event listener to the server session from which you acquire your isolated client session. You cannot add them to the isolated client session itself. For more information, see "Configuring Session Event Listeners" on page 74-17

Using Java

The `SessionEvent.PreReleaseExclusiveConnection` event listener gives you an opportunity to interact with the underlying database platform: for example, to perform any VPD-specific cleanup such as executing PL/SQL to delete VPD packages or context information.

Example 77–1 illustrates a typical session event listener used to handle `preReleaseExclusiveConnection` events for an isolated session.

Example 77–2 Session Event Listener for an Isolated Session

```
class VPDEventListener extends SessionEventAdaptor{
    public void preReleaseExclusiveConnection(SessionEvent event){
        Session session event.getSession();
        // Make the Stored Procedure call for VPD to reset the Context Information
        session.executeNonSelectingSQL("StoreProcResetContext()");
    }
}
```

To get the required user credentials, use `ClientSession` method `getConnectionPolicy` to get the associated `ConnectionPolicy`, and then use `ConnectionPolicy` method `getProperty`. The `ConnectionPolicy` associated with the `ClientSession` should contain all required user credentials (see "Configuring Connection Policy" on page 74-19).

After you implement the required `SessionEventListener`, add it to the parent server session, from which you acquire your isolated client session. For more information, see "Configuring Session Event Listeners" on page 74-17.

NoRowsModifiedSessionEvent Event Handler

As part of your general error handling strategy, you should implement a `SessionEventListener` for `SessionEvent.NoRowsModifiedSessionEvent`.

TopLink raises this event when an update or delete query is executed against the database, but no rows are updated, that is, a zero row count is returned.

If optimistic locking is not enabled and you query the database and violate your VPD security configuration, no exception is thrown: the query simply returns zero rows updated.

If optimistic locking is enabled and you query the database and violate your VPD security configuration, an `OptimisticLockException` is thrown even though the root cause of the failure was a security violation, not an optimistic locking issue.

> **Note:** You must add this session event listener to the server session from which you acquire your isolated client session. You cannot add them to the isolated client session itself. For more information, see "Configuring Session Event Listeners" on page 74-17

Using Java

This event listener gives you an opportunity to determine whether the update failure was due to a security violation (in which case you should not retry the operation), or due to an optimistic lock issue (in which case a retry may be appropriate).

You can use the existing session event API, such as `getQuery().getResult()`, to get the affected object, if any.

After you implement the required `SessionEventListener`, add it to the parent server session, from which you acquire your isolated client session. For more information, see "Configuring Session Event Listeners" on page 74-17.

ValidationException Handler

As part of your general error handling strategy, your application should be prepared to handle a `ValidationException` of type `ISOLATED_SESSION_IS_NO_LONGER_AVAILABLE`.

TopLink throws an `ISOLATED_SESSION_IS_NO_LONGER_AVAILABLE` when a client triggers the indirection on an isolated object when the isolated session used to load that object is no longer available, that is, after you call the isolated session method `release`.

Fore more information, see:

- "Exception Handlers" on page 72-12
- "Configuring an Exception Handler" on page 74-12

78

Configuring Historical Sessions

This chapter describes the various components that you must configure in order to be able to use historical sessions.

For more information about historical sessions, see "Historical Sessions" on page 72-25.

Historical Session Configuration Overview

There are two ways to configure TopLink to access the historical versions of objects maintained by your data source:

- Configuring Historical Sessions Using an Oracle Platform
- Configuring Historical Sessions Using a TopLink HistoryPolicy

Configuring Historical Sessions Using an Oracle Platform

Oracle9*i* Database Server (or later) automatically maintains historical versions of objects and extends SQL with an `AS_OF` clause used to query this historical data. Oracle refers to these as flashback queries.

If you configure your `Session` with an `OraclePlatform` (see "Configuring a Relational Database Platform at the Session Level" on page 83-1) for Oracle9*i* Database Server (or later), you can query the historical versions of objects automatically maintained by Oracle Database.

No further session configuration is required.

For more information, see the following:

- "Acquiring a Historical Session" on page 75-9
- "Historical Queries" on page 93-21.

Configuring Historical Sessions Using a TopLink HistoryPolicy

If you use a schema that you designed to maintain historical versions of objects and if that schema can be described by TopLink `HistoryPolicy`, you can query the historical versions of objects maintained by your database in accordance with your schema.

For more information, see the following:

- "Configuring a History Policy" on page 25-76
- "Acquiring a Historical Session" on page 75-9
- "Historical Queries" on page 93-21.

79

Configuring Session Broker and Client Sessions

This chapter describes the various components that you must configure in order to use session broker sessions.

Session Broker and Client Session Configuration Overview

Table 79–1 lists the configurable options for session broker sessions.

Table 79–1 Configurable Options for Session Broker Session

Option	Type	TopLink Workbench	Java
"Removing, Renaming, or Adding Sessions" on page 79-1	Basic	✔	✔
"Configuring a Primary Mapping Project" on page 74-2	Basic	✔	✔
"Configuring a Session Login" on page 74-4	Basic	✔	✔
"Configuring Logging" on page 74-4	Basic	✔	✔
"Configuring Multiple Mapping Projects" on page 74-9	Advanced	✔	✔
"Configuring a Performance Profiler" on page 74-10	Advanced	✔	✔
"Configuring an Exception Handler" on page 74-12	Advanced	✔	✔
"Configuring Customizer Class" on page 74-13	Advanced	✔	✔
"Configuring the Server Platform" on page 74-14	Advanced	✔	✔
"Configuring Session Event Listeners" on page 74-17	Advanced	✔	✔
"Configuring a Coordinated Cache" on page 88-1	Advanced	✔	✔
"Configuring the Integrity Checker" on page 74-18	Advanced	✔	✔
"Configuring Named Queries at the Session Level" on page 74-21	Basic		✔

Removing, Renaming, or Adding Sessions

Using TopLink Workbench, you can manage the sessions contained by a session broker.

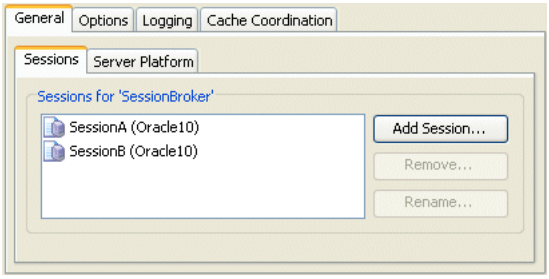
Note: Add only sessions of the same type to any given session broker. Do not mix sessions of different types within a session broker.

Using TopLink Workbench

To add sessions to, remove sessions from, or rename sessions in a session broker, use this procedure:

1. Select a session broker in the **Navigator**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.
3. Click the **Sessions** subtab. The Sessions subtab appears.

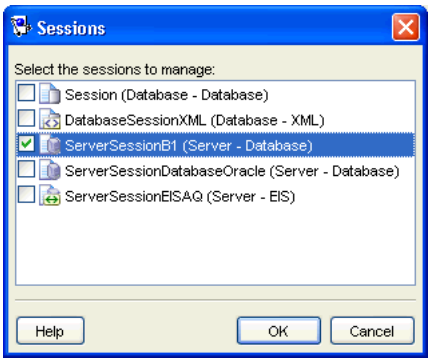
Figure 79–1 General Tab, Sessions Subtab



To manage the sessions in this session broker, choose one of the following:

- To remove a session, select the session in the Sessions tab's list and click **Remove**.
- To rename a session, select the session in the Sessions tab's list and click **Rename**. The Rename dialog box appears. Enter a new name and click **OK**.
- To add a session, click **Add Session**. The Sessions dialog box appears showing a list of all the sessions currently configured in the session configuration that owns this session broker.

Figure 79–2 Sessions Dialog Box



Check the sessions in the Session dialog that you want to add to the session broker and click **OK**.

80

Configuring Database Sessions

This chapter describes the various components that you must configure in order to use database sessions.

Database Session Configuration Overview

Table 80–1 lists the configurable options for database sessions.

Table 80–1 Configurable Options for Database Session

Option	Type	TopLink Workbench	Java
"Configuring External Connection Pools" on page 80-1	Basic	✔	✔
"Configuring a Primary Mapping Project" on page 74-2	Basic	✔	✔
"Configuring a Session Login" on page 74-4	Basic	✔	✔
"Configuring Logging" on page 74-4	Basic	✔	✔
"Configuring Multiple Mapping Projects" on page 74-9	Advanced	✔	✔
"Configuring a Performance Profiler" on page 74-10	Advanced	✔	✔
"Configuring an Exception Handler" on page 74-12	Advanced	✔	✔
"Configuring Customizer Class" on page 74-13	Advanced	✔	✔
"Configuring the Server Platform" on page 74-14	Advanced	✔	✔
"Configuring Session Event Listeners" on page 74-17	Advanced	✔	✔
"Configuring a Coordinated Cache" on page 88-1	Advanced	✔	✔
"Configuring the Integrity Checker" on page 74-18	Advanced	✔	✔
"Configuring Named Queries at the Session Level" on page 74-21	Basic		✔

Configuring External Connection Pools

Unlike a server session, a database session does not provide internal connection pools. A database session only has a single database connection that it uses for its life cycle.

Oracle recommends that you use a server and client session in a three-tier environment. Alternatively, you can use a database session with an external connection pool (see "Configuring External Connection Pooling" on page 82-2): in this case, you should allocate a new database session per user/thread or request.

WARNING: Do not allow the concurrent use of a database session by multiple users/threads.

The usage of an external connection pool reduces the number of the database session login and logout attempts to acquire the database connection.

Part XVI

Data Access

This part describes how TopLink defines connections to a data source. It contains the following chapters:

- Chapter 81, "Understanding Data Access"

  This chapter describes each of the different TopLink data source login types and important data access concepts.

- Chapter 82, "Configuring a Data Source Login"

  This chapter explains how to configure TopLink data source login options common to two or more data source login types.

- Chapter 83, "Configuring a Database Login"

  This chapter explains how to configure a TopLink database login for a session used in a relational project.

- Chapter 84, "Configuring an EIS Login"

  This chapter explains how to configure a TopLink EIS login for a session used in an EIS project.

- Chapter 85, "Creating an Internal Connection Pool"

  This chapter explains how to create an internal connection pool.

- Chapter 86, "Configuring an Internal Connection Pool"

  This chapter explains how to configure an internal connection pool.

81

Understanding Data Access

One of the most important functions of a session is to provide access to a data source. This chapter explains session components specific to accessing a data source.

This chapter explains the following:

- Data Access Concepts
- Understanding Data Access API

Data Access Concepts

This section describes concepts unique to TopLink data access, including the following:

- Externally Managed Transactional Data Sources
- Data Source Login Types
- Data Source Platform Types
- Authentication
- Connections
- Connection Pools

Externally Managed Transactional Data Sources

A TopLink transactional data source is *externally managed* if the connection pool is managed by a transaction service (such as an application server controlled transaction or a JTA transaction). A JTA managed data source or connection pool is commonly used in J2EE applications and normally required in EJB applications. Use an externally-managed connection pool as follows:

- Configure the session to use an `ExternalTransactionController` to integrate TopLink's unit of work with the external transaction service (see "Integrating the Unit of Work With an External Transaction Service" on page 99-21).
- Use the external-transaction-control option to specify the connection's login and inform TopLink that the connection is maintained by the external controller (see "Configuring External Connection Pooling" on page 82-2).
- You may need to configure the TopLink read connection pool or sequence connection pool to use a non-JTA connection pool in order to avoid transactional overhead (see "Default (Write) and Read Connection Pools" on page 81-8).

For more information on transactional data sources, see the following:

- "JTA Controlled Transactions" on page 97-3
- "OTS Controlled Transactions" on page 97-3
- "CMP Controlled Transactions" on page 97-3

Refer to Chapter 97, "Understanding TopLink Transactions" for more information on TopLink transactions.

Data Source Login Types

The login (if any) associated with a session determines how the TopLink runtime connects to the project's data source.

A login includes details of data source access, such as authentication, use of connection pools, and use of external transaction controllers. A `Login` owns a data source platform.

A data source platform includes options specific to a particular data source including such as binding, use of native SQL, use of batch writing, and sequencing. For more information about platforms, see "Data Source Platform Types" on page 81-3.

For projects that do not persist to a data source, a login is not required. For projects that do persist to a data source, a login is always required.

In TopLink Workbench, the project type determines the type of login that the project uses, if applicable.

You can use a login in a variety of roles. A login's role determines where and how you create it. The login role you choose depends on the type of project you are creating and how you intend to use the login:

- "Non-CMP Session Role: Session Login" on page 17-3
- "CMP Deployment Role: Deployment Login" on page 17-3
- "Development Role: Development Login" on page 17-4

There is a session login type for each project type that persists to a data source:

- DatabaseLogin
- EISLogin

Note that there is no XML login. TopLink XML projects are used for nonpersistent, in-memory object to XML data transformation and consequently there is no data source to log in to. For more information about persistent and nonpersistent projects, see "Persistent and Nonpersistent Projects" on page 17-2.

For additional information, see the following:

- "Projects and Login" on page 17-3
- "Configuring Common Data Source Login Options" on page 82-1

DatabaseLogin

If you are creating a project that accesses a relational database, you must configure the project with a `DatabaseLogin`. Your choice of `DatabasePlatform` further customizes your project for a particular type of database (see "Database Platforms" on page 81-3).

For more information, see "Database Login Configuration Overview" on page 83-1.

EISLogin

If you are creating a project that accesses a nonrelational data source using a J2C adapter, you must configure the project with an `EISLogin`. Your choice of `EISPlatform` further customizes your project for a particular J2C adapter and specifies what record type TopLink uses to exchange data with the EIS (see "EIS Platforms" on page 81-4).

For more information, see "EIS Login Configuration Overview" on page 84-1.

Data Source Platform Types

TopLink abstracts the details of your underlying data source using data source platform classes. TopLink provides the following data source platforms:

- Database Platforms
- EIS Platforms

A data source platform is owned by your project's `Login`. For more information about logins, see "Data Source Login Types" on page 81-2.

In general, TopLink Workbench provides minimal access to a platform API. To configure most platform options, you must use an amendment method (see "Configuring Amendment Methods" on page 25-81), or a `preLogin` event listener (see "Managing Session Events With the Session Event Manager" on page 72-5).

Database Platforms

TopLink interacts with databases using structured query language (SQL). Because each database platform uses its own variation on the basic SQL language, TopLink must adjust the SQL it uses to communicate with the database to ensure that the application runs smoothly.

The type of database platform you choose determines the specific means by which the TopLink runtime accesses the database, including the type of Java Database Connectivity (JDBC) driver to use. JDBC is an application programming interface (API) that gives Java applications access to a database. TopLink relational projects rely on JDBC connections to read objects from, and write objects to, the database. TopLink applications use either individual JDBC connections or a JDBC connection pool (see "Connection Pools" on page 81-7), depending on the application architecture.

TopLink provides a variety of database-specific platforms that let you customize your project for your target database.

Oracle Database platforms are located in `oracle.toplink.platform.database.oracle` package and include the following:

- `OraclePlatform`
- `Oracle8Platform`
- `Oracle9Platform`

Non-Oracle Database platforms are located in `oracle.toplink.platform.database` package and include the following:

- `AccessPlatform` for Microsoft Access databases
- `AttunityPlatform` for Attunity Connect JDBC drivers
- `CloudscapePlatform`

- `DB2MainframePlatform`
- `DB2Platform`
- `DBasePlatform`
- `HSQLPlatform`
- `InformixPlatform`
- `MySQL4Platform`
- `PointBasePlatform`
- `SQLAnyWherePlatform`
- `SQLServerPlatform`
- `SybasePlatform`

Specify your database platform at the project level (see "Configuring Relational Database Platform at the Project Level" on page 20-2) for all sessions, or override this project-level configuration at the session level (see "Configuring a Relational Database Platform at the Session Level" on page 83-1).

If you set your database platform in TopLink Workbench, then TopLink Workbench manages the database platform configuration for you automatically.

EIS Platforms

TopLink interacts with an EIS data source indirectly by way of a J2C adapter. TopLink abstracts the details of an EIS data source using the `oracle.toplink.eis.EISPlatform` class.

The type of EIS platform you choose determines the specific means by which the TopLink runtime accesses the EIS, including the type of J2C adapter to use. TopLink EIS projects rely on EIS connections to read objects from, and write objects to, the EIS. TopLink applications use individual EIS connections returned by the EIS connection factory specified by the EIS platform.

TopLink provides a variety of `EISPlaform` classes that let you customize your project for your target EIS.

EIS platforms for production are located in `oracle.toplink.eis.adapters` package and include the following:

- `oracle.toplink.eis.adapters.aq.AQPlatform` to access an EIS using Oracle Advanced Queuing messages.
- `oracle.toplink.eis.adapters.attunity.AttunityPlatform` to access an EIS using an Attunity J2C adapter.
- `oracle.toplink.eis.adapters.jms.JMSPlatform` to access an EIS using JMS messages.
- `oracle.toplink.eis.adapters.mqseries.MQPlatform` to access an EIS using IBM MQSeries messages.

EIS platforms for testing are also located in `oracle.toplink.eis.adapters` and include the following:

- `oracle.toplink.eis.adapters.blackbox.BlackBoxPlatform` for testing your EIS project with the Sun BlackBox reference adapter using indexed records only.

- `oracle.toplink.eis.adapters.xmlfile.XMLFilePlatform` for testing your EIS project with an EIS emulated as one or more XML files in the local file system using XML records.

Specify your EIS platform at the session level (see "Configuring an EIS Data Source Platform at the Session Level" on page 84-1).

If you set your platform in TopLink Workbench, then TopLink Workbench manages the EIS platform configuration for you automatically.

Authentication

Authentication is the means by which a data source validates a user's identity and determines whether or not the user has sufficient privileges to perform a given action.

For two-tier applications, simple JDBC authentication is usually sufficient (see "Simple JDBC Authentication" on page 81-5).

For three-tier applications, you can use simple JDBC authentication or, proxy authentication (see "Oracle Database Proxy Authentication" on page 81-5) when using the Oracle Call Interface (OCI) JDBC driver.

Authentication plays a central role in data security and user accountability and auditing (see "Auditing" on page 81-6).

Simple JDBC Authentication

When you configure a TopLink database login with a user name and password ("Configuring User Name and Password" on page 82-1), TopLink provides these credentials to the JDBC driver that you configure your application to use (see "Configuring Database Login Connection Options" on page 83-2).

By default, TopLink writes passwords to and reads them from the `sessions.xml` file in encrypted form using JCE encryption. Optionally, you can configure a different encryption class (see "Configuring Password Encryption" on page 82-2).

Oracle Database Proxy Authentication

TopLink supports proxy authentication with the Oracle Database in JSE applications and JEE applications using OC4J native or managed data sources with Oracle JDBC driver release 10.1.0.2.0 or later and external connection pools (see "External Connection Pools" on page 81-8) only.

> **Note:** TopLink does not support Oracle Database proxy authentication with JTA.

Oracle Database proxy authentication delivers the following security benefits:

- A limited trust model, by controlling the users on whose behalf middle tiers can connect, and the roles the middle tiers can assume for the user.
- Scalability, by supporting user sessions through Oracle Call Interface (OCI) and thick JDBC, and eliminating the overhead of reauthenticating clients.
- Accountability, by preserving the identity of the real user through to the database, and enabling auditing of actions taken on behalf of the real user.
- Flexibility, by supporting environments in which users are known to the database, and in which users are merely "application users" of which the database has no awareness.

Note: Oracle Database supports proxy authentication in three-tiers only; it does not support it across multiple middle tiers.

For more information about authentication in an Oracle Database, see "Preserving User Identity in Multitiered Environments" in the *Oracle Database Security Guide*.

Configure your TopLink database login to use proxy authentication (see "Configuring Oracle Database Proxy Authentication" on page 83-12) to do the following:

- address the complexities of authentication in a three-tier architecture (such as client-to-middle-tier and middle-tier-to-database authentication, and client reauthentication through the middle -tier to the database)
- enhance database audit information (for even triggers and stored procedures) by using a specific user for database operations, rather than the generic pool user
- simplify VPD/OLS configuration (see "Isolated Client Sessions and Oracle Virtual Private Database (VPD)" on page 72-21) by using a proxy user, rather than setting user information directly in the session context with stored procedures

Auditing

Regardless of what type of authentication you choose, TopLink logs the name of the user associated with all database operations. Example 81–1 shows the `CONFIG` level TopLink logs when a `ServerSession` connects through the main connection for the sample user "scott", and a `ClientSession` uses proxy connection "jeff".

Example 81–1 TopLink Logs with Oracle Database Proxy Authentication

```
[TopLink
Config]--ServerSession(13)--Connection(14)--Thread(Thread[main,5,main])--connecting(DatabaseL
ogin( platform=>Oracle9Platform   user name=> "scott" connector=>OracleJDBC10_1_0_
2ProxyConnector datasource name=>DS))
[TopLink Config]--ServerSession(13)--Connection(34)--Thread(Thread[main,5,main])--Connected:
jdbc:oracle:thin:@localhost:1521:orcl
User: SCOTT
[TopLink
Config]--ClientSession(53)--Connection(54)--Thread(Thread[main,5,main])--connecting(DatabaseL
ogin(platform=>Oracle9Platform user name=> "scott" connector=>OracleJDBC10_1_0_
2ProxyConnector datasource name=>DS))
[TopLink Config]--ClientSession(53)--Connection(56)--Thread(Thread[main,5,main])--Connected:
jdbc:oracle:thin:@localhost:1521:orcl
User: jeff
```

For more information on configuring TopLink log level and log options, see "Configuring Logging" on page 74-4.

Your database server likely provides additional user auditing options. Consult your database server documentation for details.

Alternatively, you may consider using the TopLink unit of work in conjunction with your database schema for auditing purposes (see "Implementing User and Date Auditing With the Unit of Work" on page 99-21).

Connections

A connection is an object that provides access to a data source by way of the driver you configure your application to use (see "Configuring Database Login Connection Options" on page 83-2). Relational projects use JDBC to connect to the data source; EIS and XML projects use JCA. TopLink uses the interface

`oracle.toplink.internal.databaseaccess.Accessor` to wrap data source connections. This interface is accessible from certain events (see "Descriptor Event Manager" on page 23-8).

Typically, when using a server session, TopLink uses a a different connection for both reading and writing. This lets you use nontransactional connections for reading and avoid maintaining connections when not required. See "Reading Through the Write Connection" on page 99-28 and "Exclusive Write Connections" on page 74-19 for more information.

By default, a TopLink server session acquires connections lazily: that is, only during the commit operation of a unit of work. Alternatively, you can configure TopLink to acquire a write connections at the time you acquire a client sessions (see "Lazy Connection Acquisition" on page 74-20).

Connections can be allocated from internal or external connection pools (see "Connection Pools" on page 81-7).

Connection Pools

A **connection pool** is a service that creates and maintains a shared collection (pool) of data source connections on behalf of one or more clients. The connection pool provides a connection to a process on request, and returns the connection to the pool when the process is finished using it. When it is returned to the pool, the connection is available for other processes. Because establishing a connection to a data source can be time-consuming, reusing such connections in a connection pool can improve performance.

TopLink uses connection pools to manage and share the connections used by server and client sessions. This feature reduces the number of connections required and allows your application to support many clients.

You can configure your session to use internal connection pools provided by TopLink or external connection pools provided by a JDBC driver or J2EE container.

You can use connection pools in your TopLink application for a variety of purposes, such as reading, writing, sequencing, and other application-specific functions.

This section describes the following:

- Internal Connection Pools
- External Connection Pools
- Default (Write) and Read Connection Pools
- Sequence Connection Pools
- Application-Specific Connection Pools

Internal Connection Pools

For non-J2EE applications, you typically use *internal* connection pools. By default, TopLink sessions use internal connection pools.

Using internal connection pools, you can use TopLink Workbench to configure the default (write) and read connection pools (see "Default (Write) and Read Connection Pools" on page 81-8) and you can create additional connection pools for object identity (see "Sequence Connection Pools" on page 81-8), or any other purpose (see "Application-Specific Connection Pools" on page 81-9).

Using internal connection pools, you can optimize the creation of read connections for applications that read data only to display it and only infrequently modify data (see "Configuring a Nontransactional Read Login" on page 86-3).

For information on selecting the type of connection pool to use, see "Configuring External Connection Pooling" on page 82-2.

For more information on creating and configuring internal connection pools, see:

- "Internal Connection Pool Creation Overview" on page 85-1
- "Internal Connection Pool Configuration Overview" on page 86-1

External Connection Pools

For J2EE applications, you typically use *external* connection pools.

If you are using an external transaction controller (JTA), you must use external connection pools to integrate with the JTA (see "Integrating the Unit of Work With an External Transaction Service" on page 99-21).

Using external connection pools, you can use either TopLink Workbench or Java to configure the default (write) and read connection pools (see "Default (Write) and Read Connection Pools" on page 81-8) and create additional connection pools for object identity (see "Sequence Connection Pools" on page 81-8), or any other purpose (see "Application-Specific Connection Pools" on page 81-9).

For more information on selecting the type of connection pool to use, see "Configuring External Connection Pooling" on page 82-2.

Default (Write) and Read Connection Pools

A server session provides a read connection pool and a write connection pool. These could be different pools, or if you use external connection pooling, the same connection pool.

All read queries use connections from the read connection pool and all queries that write changes to the data source use connections from the write connection pool. You can configure attributes of the default read and write connection pools.

Whenever a new connection is established, TopLink uses the connection configuration you specify in your session's `DatasourceLogin`. Alternatively, when you use an external transaction controller, you can define a separate connection configuration for a read connection pool to avoid the additional overhead, if appropriate (see "Configuring a Nontransactional Read Login" on page 86-3).

For more information on configuring read and write connection pools, see "Internal Connection Pool Configuration Overview" on page 86-1.

Sequence Connection Pools

An essential part of maintaining object identity (see "Cache Type and Object Identity" on page 87-3) is sequencing–managing the assignment of unique values to distinguish one instance from another. For more information, see "Projects and Sequencing" on page 17-4.

Sequencing involves reading and writing a special sequence resource maintained by your data source.

By default, TopLink includes sequence operations in a separate transaction. This avoids complications during the write transaction, which may lead to deadlocks over the sequence resource. However, when using an external transaction controller (such as a JTA data source or connection pool), TopLink cannot use a different transaction

for sequencing. Use a sequence connection pool to configure a non-JTA transaction pool for sequencing. This is required only for table sequencing–not native sequencing.

In each server session, you can create one connection pool, called a sequence connection pool, that TopLink uses exclusively for sequencing. With a sequence connection pool, TopLink satisfies a request for a new object identifier outside of the transaction from which the request originates. This allows TopLink to immediately commit an update to the sequence resource, which avoids deadlocks.

> **Note:** If you use a sequence connection pool and the original transaction fails, the sequence operation does not roll back.

You should use a sequence connection pool, if the following applies:

- You use table sequencing (that is, non-native sequencing). See "Table Sequencing" on page 17-16 and "Unary Table Sequencing" on page 17-17 for more information.
- You use external transaction controller (JTA).

You should not use a sequence connection pool, if the following applies:

- You do not use sequencing, or use the data source's native sequencing (see "Native Sequencing With an Oracle Database Platform" on page 17-18 and "Native Sequencing With a Non-Oracle Database Platform" on page 17-19).
- You have configured the sequence table to avoid deadlocks.
- You use non-JTA data sources.

For more information, see

- "Internal Connection Pool Creation Overview" on page 85-1
- "Internal Connection Pool Configuration Overview" on page 86-1

Application-Specific Connection Pools

When you use internal TopLink connection pools in a session, you can create one or more connection pools that you can use for any application purpose. These are called named connection pools, as you can give them any name you want and use them for any purpose.

Typically, use these named connection pools to provide pools of different security levels. For example, the "default" connection pool may only allow access to specific tables but the "admin" connection pool may allow access to all tables.

For more information, see the following:

- "Internal Connection Pool Creation Overview" on page 85-1
- "Internal Connection Pool Configuration Overview" on page 86-1
- "Acquiring a Client Session That Uses a Named Connection Pool" on page 75-8

Understanding Data Access API

This section describes the following:

- Login Inheritance Hierarchy
- Platform Inheritance Hierarchy

Login Inheritance Hierarchy

Example 81–2 illustrates the login types that are derived from abstract class `oracle.toplink.sessions.DatasourceLogin`.

Example 81–2 Login Inheritance Hierarchy

```
class oracle.toplink.sessions.DatasourceLogin
    class oracle.toplink.sessions.DatabaseLogin
    class oracle.toplink.eis.EISLogin
```

Platform Inheritance Hierarchy

Example 81–3 illustrates the platform type class hierarchy.

Example 81–3 Platform Inheritance Hierarchy

```
oracle.toplink.platform.database
    AccessPlatform
    AttunityPlatform
    CloudscapePlatform
    DatabasePlatform
    DB2MainframePlatform
    DB2Platform
    DBasePlatform
    HSQLPlatform
    InformixPlatform
    PointBasePlatform
    SQLAnyWherePlatform
    SQLServerPlatform
    SybasePlatform
oracle.toplink.platform.database.oracle
    Oracle8Platform
    Oracle9Platform
    OraclePlatform
```

82

Configuring a Data Source Login

This chapter describes how to configure TopLink data source logins.

Table 82–1 lists the types of TopLink data source logins that you can configure and provides a cross-reference to the type-specific chapter that lists the configurable options supported by that type.

Table 82–1 Configuring TopLink Data Source Logins

If you are configuring a...	See...
DatabaseLogin	Chapter 83, "Configuring a Database Login"
EISLogin	Chapter 84, "Configuring an EIS Login"

Table 82–2 lists the configurable options shared by two or more TopLink data source login types.

When using the `sessions.xml` file to configure login information, TopLink will override any login information in the `project.xml` and instead use the information from the `sessions.xml` configuration. For more information, see "Understanding Data Access" on page 81-1.

Configuring Common Data Source Login Options

Table 82–2 lists the configurable options shared by two or more TopLink data source login types. In addition to the configurable options described here, you must also configure the options described for the specific Data Source Login Types, as shown in Table 82–1

Table 82–2 Common Data Source Login Options

Option	Type	TopLink Workbench	Java
"Configuring User Name and Password" on page 82-1	Basic	✔	✔
"Configuring Password Encryption" on page 82-2	Advanced		✔
"Configuring External Connection Pooling" on page 82-2	Advanced	✔	✔
"Configuring Properties" on page 82-4	Advanced	✔	✔
"Configuring a Default Null Value at the Login Level" on page 82-5	Advanced		✔

Configuring User Name and Password

Optionally, you can specify the user name and password of a login.

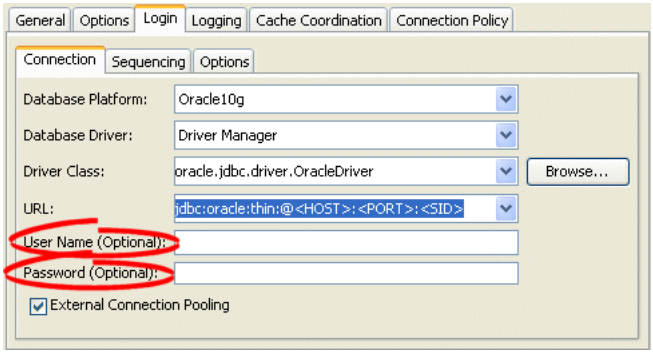
If you specify a password using TopLink Workbench, enter the plain text (not encrypted) value. By default, TopLink Workbench writes passwords to and reads passwords from the `sessions.xml` file in encrypted form using JCE encryption. For information on configuring password encryption, see "Configuring Password Encryption" on page 82-2.

Using TopLink Workbench

To specify a user name and password, use this procedure:

1. Select a server or database session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Login** tab. The Login tab appears.
3. Click the **Connection** subtab. The Connection subtab appears.

Figure 82–1 Login Tab, Connection Subtab, User Name and Password Fields



Enter a user name and password in plain text (not encrypted).

Configuring Password Encryption

By default, passwords are written to and read from the `sessions.xml` file in encrypted form using JCE encryption.

Currently, TopLink Workbench does not support specifying the encryption class used. To change the encryption class used, you must modify the login in Java using a `preLogin` event listener.

Using Java

To specify the encryption class that TopLink should use to write a password to or read a password from the sessions.xml file, use `DatasourceLogin` method `setEncryptionClassName`, passing in the name of the encryption class as a String.

To configure a `DatasourceLogin` with an encrypted password, use `DatasourceLogin` method `setEncryptedPassword`, passing in the encrypted password as a `String`.

Configuring External Connection Pooling

For non-J2EE applications, you typically use internal connection pools provided by TopLink (see "Internal Connection Pools" on page 81-7). In this case, you can use

TopLink Workbench to configure connection pool options and to create a sequence connection pool and application-specific (named) connection pools.

For J2EE applications, you typically use external connection pools provided by a JDBC driver or J2EE container (see "External Connection Pools" on page 81-8). Optionally, you can configure a read connection pool to use a nontransactional login, and you can configure a sequence connection pool to use a separate (preferably nontransactional) login of its own.

Because JTA external transaction controllers are dependent upon the external transaction service that the application server provides, you must configure TopLink to use external connection pools if you are using an external transaction controller (see "Integrating the Unit of Work With an External Transaction Service" on page 99-21).

External connection pools enable your TopLink application to do the following:

- Integrate into a J2EE-enabled system.
- Integrate with JTA transactions (JTA transactions require a JTA-enabled data source).
- Leverage a shared connection pool in which multiple applications use the same data source.
- Use a data source configured and managed directly on the server.

For more information about connection pools, see "Connection Pools" on page 81-7.

Using TopLink Workbench

To specify if the session login uses external connection pooling, use this procedure:

1. Configure a data source on the application server.

   If you are using the external connection pool with an external transaction controller (see "Configuring the Server Platform" on page 74-14), be sure to configure a JTA-enabled data source.

   For more information, see your J2EE container documentation.

2. Select a server or database session in the **Navigator**. Its properties appear in the Editor.
3. Click the **Login** tab. The Login tab appears.
4. Click the **Connection** subtab. The Connection subtab appears.

Figure 82–2 Login Tab, Connection Subtab, External Connection Pooling Field, Database Driver

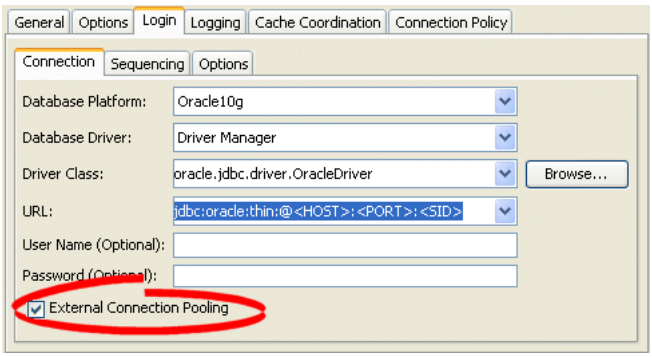
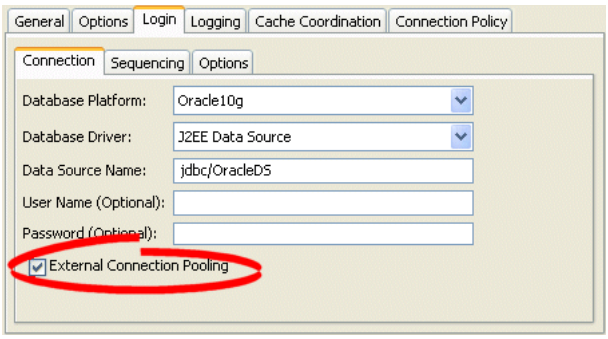


Figure 82–3 Connection Tab, External Connection Pooling Field, J2EE Data Source



Specify if this login uses External Connection Pooling. For a database driver, external connection pooling is optional. For a J2EE data source, external connection pooling is mandatory.

Configuring Properties

For all `DatasourceLogin` types, you can specify custom named values, called properties. Some data sources require additional, driver-specific properties not supported in the `DatasourceLogin` API (for example, see "JDBC Driver Properties Optimization" on page 11-14). Add these properties to the `DatasourceLogin` so that TopLink can pass them to the driver.

For relational sessions, you must first enable advanced option **Use Properties** (see "Configuring Advanced Options" on page 83-11).

For EIS sessions, properties are always enabled.

Note: Do not set a password as a property. Always use TopLink Workbench or `DatabaseLogin` method `setPassword`. For more information on configuring a password, see "Configuring User Name and Password" on page 82-1.

When using TopLink Workbench, you can only set character values, which TopLink returns as `String` objects (see "Using TopLink Workbench" on page 82-4).

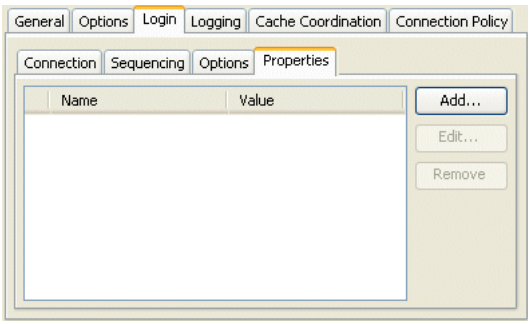
When using Java, you can set any `Object` value (see "Using Java" on page 82-5).

Using TopLink Workbench

To specify arbitrary named value pairs that TopLink associates with a `DatasourceLogin`, use this procedure:

1. Select a server or database session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Login** tab. The Login tab appears.
3. If necessary, enable support for properties:
   - For relational sessions, you must first enable advanced option **Use Properties** (see "Configuring Advanced Options" on page 83-11)
   - For EIS sessions, properties are always enabled.
4. Click the **Properties** subtab. The Properties subtab appears.

Figure 82–4 Login Tab, Properties Subtab



To add (or change) a new **Name**/**Value** property, click **Add** (or **Edit**). Add Property dialog box appears.

Use the following information to add or edit a login property on the Add Property dialog box:

Option	Description
Name	The name by which TopLink retrieves the property value using the `DatasourceLogin` method `getProperty`.
Value	The value TopLink retrieves using the `DatasourceLogin` method `getProperty` passing in the corresponding property name. Using TopLink Workbench, you can set only character values that TopLink returns as `String` objects.

To delete an existing property, select the **Name**/**Value** row and click **Remove**.

Using Java

Using Java, you can set any `Object` value using `DatasourceLogin` method `setProperty`. To remove a property, use `DatasourceLogin` method `removeProperty`.

Configuring a Default Null Value at the Login Level

A default null value is the Java `Object` type and value that TopLink uses instead of `null` when TopLink reads a `null` value from a data source.

When you configure a default null value at the login level, it applies to all mappings used in a session. In this case, TopLink uses it to translate in one direction only: when TopLink reads `null` from the data source, it converts this `null` to the specified type and value.

You can also use TopLink to set a default null value on a per-mapping basis (see "Configuring a Default Null Value at the Mapping Level" on page 32-12).

Note: A default null value must be an `Object`. To specify a primitive value (such as `int`), you must use the corresponding `Object` wrapper (such as `Integer`).

Using Java

Using Java API, you can configure a default null value for all mappings used in a session with the `DatabaseLogin` method `setDefaultNullValue(Class,Object)`.

For example:

```
// Defaults all null String values read from the database to empty String
session.getLogin().setDefaultNullValue(String.class, "");
```

83

Configuring a Database Login

In a relational database project, TopLink retrieves the table information from the database, for each descriptor. Each TopLink Workbench project contains an associated database. You can create multiple logins for each database.

Database Login Configuration Overview

Table 83–1 lists the configurable options for a database login.

Table 83–1 Configurable Options for Database Login

Option	Type	TopLink Workbench	Java
"Configuring a Relational Database Platform at the Session Level" on page 83-1	Basic	✔	✔
"Configuring Database Login Connection Options" on page 83-2	Basic	✔	✔
"Configuring Sequencing at the Session Level" on page 83-4	Basic	✔	✔
"Configuring JDBC Options" on page 83-9	Basic	✔	✔
"Configuring User Name and Password" on page 82-1	Basic	✔	✔
"Configuring a Table Qualifier" on page 83-8	Advanced	✔	✔
"Configuring Advanced Options" on page 83-11	Advanced	✔	✔
"Configuring Password Encryption" on page 82-2	Advanced	✔	✔
"Configuring External Connection Pooling" on page 82-2	Advanced	✔	✔
"Configuring Properties" on page 82-4	Advanced	✔	✔
"Configuring Oracle Database Proxy Authentication" on page 83-12	Advanced		✔

Configuring a Relational Database Platform at the Session Level

For each database session, you must specify the database platform (such as Oracle9*i* Database Server). This platform configuration overrides the platform at the project level, if configured.

For more information, see the following:

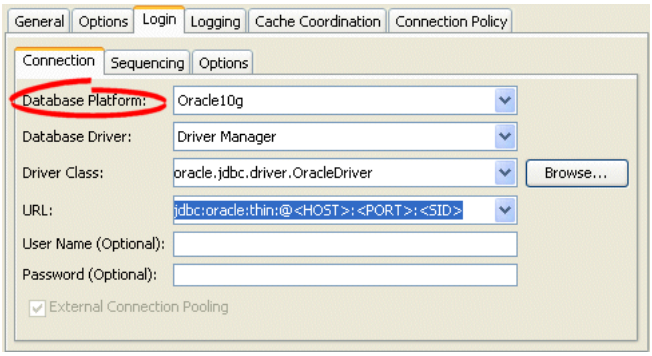
- "Configuring Relational Database Platform at the Project Level" on page 20-2
- "Data Source Platform Types" on page 81-2

Using TopLink Workbench

To specify the database platform options for a relational server (or database) session login, use this procedure:

1. Select a relational server (or database) session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Login** tab. The Login tab appears.
3. Click the **Connection** subtab. The Connection subtab appears.

Figure 83–1 Login Tab, Connection Subtab, Database Platform Option



Select the database platform from the menu of options. This menu includes all instances of `DatabasePlatform` in the TopLink classpath.

Configuring Database Login Connection Options

You configure connection information at the session level for a non-CMP TopLink application. This information is stored in the `sessions.xml` file. The TopLink runtime uses this information whenever you perform a persistence operation using the session in your non-CMP TopLink application.

This connection configuration overrides the connection information at the project level, if configured. For more information about project-level configuration, see "Configuring Development and Deployment Logins" on page 20-6.

This connection configuration is overridden by the connection information at the connection pool level. For more information, see "Configuring Connection Pool Connection Options" on page 86-4.

Using TopLink Workbench

To specify the connection options for a relational server (or database) session login, use this procedure:

1. Select a relational server (or database) session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Login** tab. The Login tab appears.
3. Click the **Connection** subtab. The Connection subtab appears.

Figure 83–2 Login Tab, Connection Subtab, Database Driver

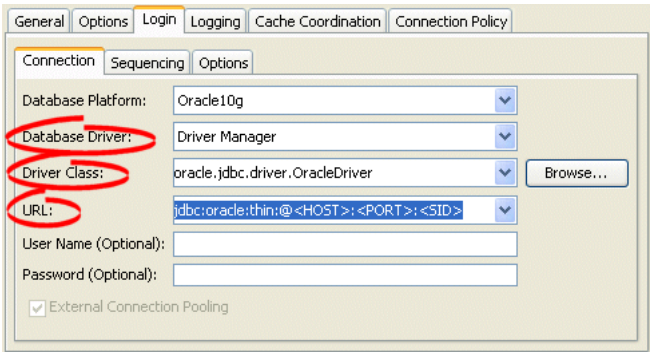
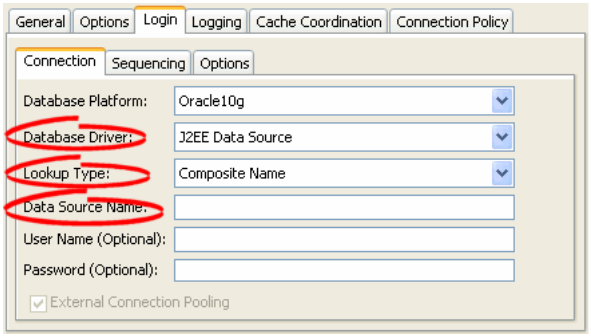


Figure 83–3 Login Tab, Connection Subtab



Use the following information to enter data in the driver fields on the tab:

Field	Description
Database Driver	Specify the appropriate database driver: ▪ **Driver Manager**: specify this option to configure the driver class and URL used to connect to the database. In this case, you must configure the **Driver Class** and **Driver URL** fields. ▪ **J2EE Data Source**: specify this option to use a J2EE data source already configured on your target application server. In this case, you must configure the **Datasource Name** field. Note: If you select **J2EE Datasource**, you must use external connection pooling. You cannot use internal connection pools with this **Database Driver** option (for more information, see "Configuring External Connection Pooling" on page 82-2).
Driver Class[1]	Configure this field when **Database Driver** is set to **Driver Manager**. Select from the menu of options. This menu includes all JDBC drivers in the TopLink classpath.
Driver URL[1]	Configure this field when **Database Driver** is set to **Driver Manager**. Select from the menu of options relevant to the selected **Driver Class**, and edit the URL to suit your data source.

Field	Description
Data Source Name[2]	Configure this field when **Database Driver** is set to **J2EE Datasource**. Specify any valid JNDI name that identifies the J2EE data source preconfigured on your target application server (example: `jdbc/EmployeeDB`). By convention, all such names should resolve to the JDBC subcontext (relative to the standard `java:comp/env` naming context that is the root of all provided resource factories).
Lookup Type[2]	Configure this field when **Database Driver** is set to **J2EE Datasource**. Specify the lookup method for determining the JNDI name: ▪ Composite Name ▪ Compound Name ▪ String

[1] Applicable only when **Database Driver** is set to **Driver Manager**.
[2] Applicable only when **Database Driver** is set to **J2EE Datasource**.

Configuring Sequencing at the Session Level

You configure TopLink sequencing at the session or project level to tell TopLink how to obtain sequence values: that is, what type of sequences to use.

In a CMP project, you do not configure a session directly: in this case, you must configure sequences at the project level (see "Configuring Sequencing at the Project Level" on page 20-3). In a non-CMP project, you can configure a session directly: in this case, you can use session-level sequence configuration to override project-level sequence configuration, on a session-by-session basis, if required.

Using TopLink Workbench (see "Using TopLink Workbench" on page 83-5), you can configure table sequencing (see "Table Sequencing" on page 17-16) and native sequencing ("Native Sequencing With an Oracle Database Platform" on page 17-18 and "Native Sequencing With a Non-Oracle Database Platform" on page 17-19), and you can configure a preallocation size that applies to all sequences (see "Sequencing and Preallocation Size" on page 17-20).

Using Java (see "Using Java" on page 83-5), you can configure any sequence type that TopLink supports ("Sequencing Types" on page 17-16). You can create any number and combination of sequences. You can create a sequence object explicitly or use the default sequence that the platform creates. You can associate the same sequence with more than one descriptor and you can configure a separate preallocation size for each descriptor's sequence.

If you are migrating a BEA WebLogic CMP application to OC4J and TopLink persistence (see "Migrating BEA WebLogic Persistence to OC4J TopLink Persistence" on page 7-16), the TopLink migration tool does not migrate BEA WebLogic single column sequence tables to TopLink unary sequence tables (see "Unary Table Sequencing" on page 17-17). After migration, you must manually configure your project to use TopLink unary sequence tables if your application originally used single-column sequence tables in BEA WebLogic.

After configuring the sequence type at the session (or project) level, to enable sequencing, you must configure a descriptor with a sequence field and a sequence name (see "Configuring Sequencing at the Descriptor Level" on page 26-3).

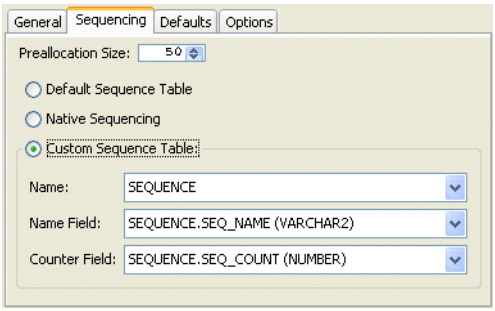
For more information about sequencing, see "Understanding Sequencing in Relational Projects" on page 17-14.

Using TopLink Workbench

To specify the sequencing information for a relational server (or database) session, use this procedure:

1. Select the session object in the **Navigator**.
2. Click the **Login** tab in the **Editor**.
3. Click the **Sequencing** subtab. The Sequencing subtab appears.

Figure 83–4 Login Tab, Sequencing Subtab



Use the following information to enter data in each field of the Sequencing subtab to configure the persistence type:

Field	Description
Preallocation Size	Select the default preallocation size (see "Sequencing and Preallocation Size" on page 17-20). Default is **50**. The preallocation size you configure applies to all sequences.
Default Sequence Table	Select this option to use table sequencing (see "Table Sequencing" on page 17-16) with default sequence table name `SEQUENCE`, default sequence name field `SEQ_NAME`, and default sequence count field `SEQ_COUNT`.
Native Sequencing	Select this option to use a sequencing object (see "Native Sequencing With an Oracle Database Platform" on page 17-18 or "Native Sequencing With a Non-Oracle Database Platform" on page 17-19) created by the database platform. This option applies only to Oracle, Sybase, Microsoft SQL, IBM Informix and IBM DB2 database platforms.
Custom Sequence Table	Select this option to use table sequencing (see "Table Sequencing" on page 17-16) with a sequence table name, sequence name field, and sequence count field name that you specify.
Name	Select the name of the sequence table.
Name Field	Select the name of the column used to store the sequence name.
Counter Field	Select the name of the column used to store the sequence count.

Using Java

Using Java, you can perform the following sequence configurations:

- Using the Platform Default Sequence
- Configuring Multiple Sequences
- Configuring Query Sequencing

Using the Platform Default Sequence

After you configure your login with a platform (see "Configuring a Relational Database Platform at the Session Level" on page 83-1), you can use the default sequence that the platform provides.

If you associate a descriptor with an unspecified sequence, the TopLink runtime will create an instance of `DefaultSequence` to provide sequencing for that descriptor. For more information, see "Configuring the Platform Default Sequence" on page 26-6.

You can access the default platform sequence directly as Example 83–1 shows. For example, by default, a `DatabasePlatform` creates a table sequence using the default table and column names (see "Table Sequencing" on page 17-16).

Example 83–1 Accessing the Platform Default Sequence

```
// assume that dbLogin owns a DatabasePlatform
TableSequence tableSeq2 = ((TableSequence)dbLogin.getDefaultSequence()).clone();
tableSeq2.setName("EMP_SEQ");
tableSeq2.setPreallocationSize(75);
dbLogin.addSequence(tableSeq2);
```

To avoid having to clone the platform default sequence, you can use the `DefaultSequence` class–a wrapper for the platform default sequence–as Example 83–2 shows. The new sequence named `EMP_SEQ` will be of the same type as the platform default sequence.

Example 83–2 Using the DefaultSequence Class

```
login.addSequence(
    new DefaultSequence("EMP_SEQ", 75)
);
```

You can override the default platform sequence as Example 83–3 shows. In this example, `dbLogin` owns a `DatabasePlatform` that provides a default sequence of type `TableSequence`. After setting the default sequence to type `UnaryTableSequence`, when you use the `DefaultSequence` class, it will access the new default sequence type. In this example, the sequence named `EMP_SEQ` will be of type `UnaryTableSequence` and have a preallocation size of 75.

Example 83–3 Overriding the Platform Default Sequence

```
// assume that dbLogin owns a DatabasePlatform
Sequence unaryTableSequence = new UnaryTableSequence();
unaryTableSequence.setPreallocationSize(40);
dbLogin.setDefaultSequence(unaryTableSequence);
dbLogin.addSequence(
    new DefaultSequence("EMP_SEQ", 75) // UnaryTableSequence
);
```

Configuring Multiple Sequences

In addition to using the platform default sequence (see "Using the Platform Default Sequence" on page 83-6), you can explicitly create sequence instances and configure a `Login` with any combination of sequence types, each with their own preallocation size as Example 83–4 shows. In this example, the sequence named `EMP_SEQ` will provide sequence values exclusively for instances of the `Employee` class and `ADD_SEQ` will provide sequence values exclusively for instances of the `Address` class. The sequence named `PHONE_SEQ` will use the platform default sequence with a preallocation size of 30 to provide sequence values for the `Phone` class.

Example 83–4 Configuring Multiple Sequences Explicitly

```
login.addSequence(new TableSequence("EMP_SEQ", 25));
login.addSequence(new DefaultSequence("PHONE_SEQ", 30));
login.addSequence(new UnaryTableSequence("ADD_SEQ", 55));
login.addSequence(new NativeSequence("NAT_SEQ", 10));
```

If login owned a `DatabasePlatform` (whose default sequence type is `TableSequence`), you could configure your sequences using the platform default sequence type as Example 83–5 shows. In this example, sequences `EMP_SEQ` and `PHONE_SEQ` share the same `TableSequence` table: `EMP_SEQ` and `PHONE_SEQ` represent rows in this table.

Example 83–5 Configuring Multiple Sequences Using the Default Sequence Type

```
login.addSequence(new DefaultSequence("EMP_SEQ", 25));
login.addSequence(new DefaultSequence("PHONE_SEQ", 30));
login.addSequence(new UnaryTableSequence("ADD_SEQ", 55));
login.addSequence(new NativeSequence("NAT_SEQ", 10));
```

Configuring Query Sequencing

You can configure the query that TopLink uses to update or read a sequence value for any sequence type that extends `QuerySequence`.

In most applications, the queries that TopLink automatically uses are sufficient. However, if your application has special sequencing needs–for example, if you want to use stored procedures for sequencing–then you can configure the update and read queries that the TopLink sequence uses.

Example 83–7 illustrates how to specify a stored procedure that updates a sequence and returns the new sequence value with a single SQL select query. In this example, the stored procedure is named `UPDATE_SEQ`. It contains one input argument–the name of the sequence to update (`SEQ_NAME`), and one output argument–the value of the sequence after the updated (`SEQ_COUNT`). The stored procedure increments the sequence value associated with the sequence named `SEQ_NAME` and returns the new sequence value in the output argument named `SEQ_COUNT`.

Example 83–6 Using a Stored Procedure for both Sequence Update and Select

```
ValueReadQuery seqReadQuery = new ValueReadQuery();
StoredProcedureCall spCall = new StoredProcedureCall();
spCall.setProcedureName("UPDATE_SEQ");
seqReadQuery.addNamedArgument("SEQ_NAME");
seqReadQuery.addNamedOutputArgument("SEQ_COUNT");
seqReadQuery.setCall(spCall);
((QuerySequence)login.getDefaultSequence()).setSelectQuery(seqReadQuery);
```

Example 83–7 and Example 83–8 illustrate how to specify separate stored procedures for sequence update and select actions.

In Example 83–7, the stored procedure is named UPDATE_SEQ and it contains one input argument: the name of the sequence to update (`SEQ_NAME`). The stored procedure increments the sequence value associated with the sequence named `SEQ_NAME`.

Example 83–7 Using a Stored Procedure for Sequence Updates Only

```
DataModifyQuery seqUpdateQuery = new DataModifyQuery();
StoredProcedureCall spCall = new StoredProcedureCall();
```

```
spCall.setProcedureName("UPDATE_SEQ");
seqUpdateQuery.addNamedArgument("SEQ_NAME");
seqUpdateQuery.setCall(spCall);
((QuerySequence)login.getDefaultSequence()).setUpdateQuery(seqUpdateQuery);
```

In Example 83–8, the stored procedure is named `SELECT_SEQ` and it takes one argument: the name of the sequence to select from (`SEQ_NAME`). The stored procedure reads one data value: the current sequence value associated with the sequence name `SEQ_NAME`.

Example 83–8 Using a Stored Procedure for Sequence Selects Only

```
ValueReadQuery seqReadQuery = new ValueReadQuery();
StoredProcedureCall spCall = new StoredProcedureCall();
spCall.setProcedureName("SELECT_SEQ");
seqReadQuery.addArgument("SEQ_NAME");
seqReadQuery.setCall(spCall);
login.((QuerySequence)getDefaultSequence()).setSelectQuery(seqReadQuery)
```

You can also create a `QuerySequence` directly and add it to your login, as Example 83–9 shows.

Example 83–9 Using the QuerySequence Class

```
// Use the two-argument constructor: pass in sequence name and preallocation size.
// Alternatively, you can use zero- or one-argument (sequence name) constructor
login.addSequence(new QuerySequence("SEQ1", 75));
```

Configuring a Table Qualifier

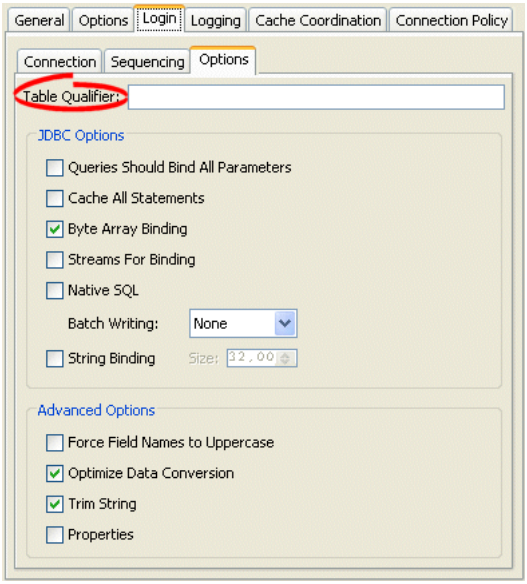
Some databases (such as Oracle Database and DB2) require that all tables be qualified by an identifier. This can be the creator of the table or database name on which the table exists. When you specify a table qualifier, TopLink uses this qualifier for all of the tables it references. Specify a table qualifier only if required and only if all of the tables have the same qualifier.

Using TopLink Workbench

To specify a table qualifier, use this procedure:

1. Select a relational server (or database) session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Login** tab. The Login tab appears.
3. Click the **Options** subtab. The Options subtab appears.

Figure 83–5 Login Tab, Options Subtab, Table Qualifier Field



In the **Table Qualifier** field enter the identifier used to qualify references to all tables in this database.

Configuring JDBC Options

Most JDBC drivers support the run-time configuration of various options to customize driver operation to meet user needs. TopLink provides direct support (in API and TopLink Workbench) for many of the most important options, as this section describes, as well as more advanced options (see "Configuring Advanced Options" on page 83-11)

You can also configure additional options by specifying properties (see "Configuring Properties" on page 82-4).

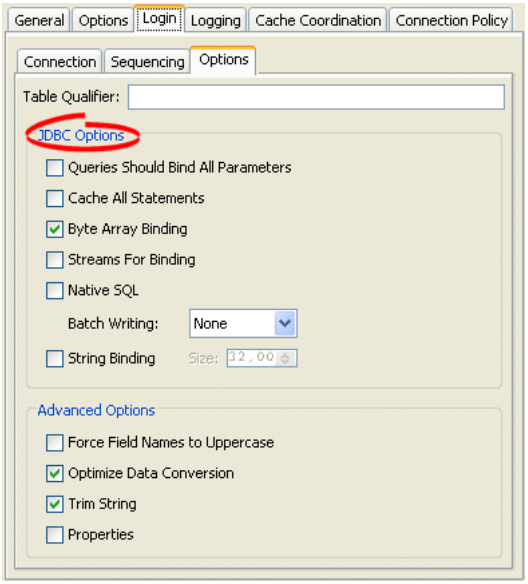
> **Note:** Not all drivers support all JDBC options. Selecting a combination of options may result in different behavior from one driver to another. Before selecting JDBC options, consult your JDBC driver documentation.

Using TopLink Workbench

To specify the JDBC options for a relational server (or database) session login, use this procedure:

1. Select a relational server (or database) session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Login** tab. The Login tab appears.
3. Click the **Options** subtab. The Options subtab appears.

Figure 83–6 Login Tab, Options Subtab, JDBC Options



Option	Description
Queries Should Bind All Parameters[1]	Select this option to bind all of the query's parameters
Cache All Statements[1]	When selected, TopLink caches each prepared statement so that when reexecuted, you avoid the SQL preparation time which improves performance.
Byte Array Binding[1]	Select this option if you query binary large object (BLOB) data.
Streams for Binding[1]	Select this option if you use a JDBC driver that is more efficient at handling BLOB data using `java.io.InputStream` and `java.io.OutputStream`.
Native SQL	By default, TopLink generates SQL using JDBC SQL grammar. Select this option if you want TopLink to use database specific SQL grammar, for example, if your database driver does not support the full JDBC SQL grammar.
Batch Writing[2]	Select this option if you use a JDBC driver that supports sending groups of `INSERT`, `UPDATE`, and `DELETE` statements to the database in a single transaction, rather than individually. Select **JDBC** to use the batch writing capabilities of your JDBC driver. Select **TopLink** to use the native batch writing capabilities that TopLink provides. Select this option if your JDBC driver does not support batch writing.
String Binding[1]	Select this option if you query large `java.lang.String` objects. You can configure the maximum `String` length (default: 32000 characters).

[1] For more information, see "Parameterized SQL (Binding) and Prepared Statement Caching" on page 11-15.

[2] If you are using the `MySQL4` database platform (see "Data Source Platform Types" on page 81-3), use **JDBC** batch writing (do not use **TopLink** batch writing). For more information, see "Batch Writing" on page 11-15.

Using Java

To enable parameterized SQL and prepared statement caching for all queries, configure at the `Login` level, as Example 83–10 shows. For more information, see "Parameterized SQL (Binding) and Prepared Statement Caching" on page 11-15.

Example 83–10 Parameterized SQL and Binding at the Login Level

```
databaseLogin.bindAllParameters();
databaseLogin.cacheAllStatements();
databaseLogin.setStatementCacheSize(100);
```

To enable JDBC batch writing, use `Login` method `useBatchWriting` as Example 83–11 shows:

Example 83–11 Using JDBC Batch Writing

```
project.getLogin().useBatchWriting();
project.getLogin().bindAllParameters();
project.getLogin().setMaxBatchWritingSize(100);
```

Configuring Advanced Options

Most JDBC drivers support the run-time configuration of various options to customize driver operation to meet user needs. TopLink provides direct support (in API and TopLink Workbench) for many of the most important options (see "Configuring JDBC Options" on page 83-9), as well as more advanced options, as this section describes.

You can also configure additional options by specifying properties (see "Configuring Properties" on page 82-4).

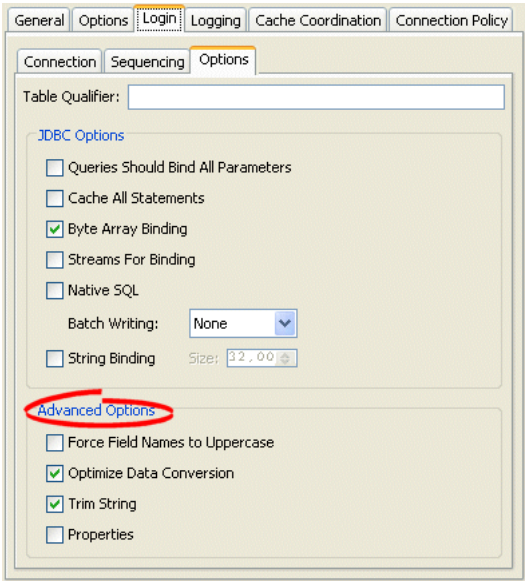
> **Note:** Not all drivers support all JDBC options. Selecting a combination of options may result in different behavior from one driver to another. Before selecting JDBC options, consult your JDBC driver documentation.

Using TopLink Workbench

To specify the advanced options for a relational server (or database) session login, use this procedure:

1. Select a relational server (or database) session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Login** tab. The Login tab appears.
3. Click the **Options** subtab. The Options subtab appears.

Figure 83–7 Login Tab, Options Subtab, Advanced Options



Option	Description
Force Field Names to Uppercase	By default, TopLink uses the case of field names as returned by the database. If your application expects field names to be uppercase but the database does not return consistent case (for example, if you accessing different databases), enable this option.
Optimize Data Conversion	By default, TopLink optimizes data access by accessing the data from JDBC in the format the application requires. If you are using an older JDBC driver that does not perform data conversion correctly and conflicts with this optimization, disable this optimization.
Trim String	By default, TopLink discards the trailing blanks from `CHAR` field values. To read and write `CHAR` field values literally (including any trailing blanks), disable this option.
Properties	Check this option to enable the use of properties for this `DatabaseLogin` (see "Configuring Properties" on page 82-4).

Configuring Oracle Database Proxy Authentication

You can configure a database login to use Oracle Database proxy authentication with an Oracle Database platform in JSE applications and JEE applications using OC4J native or managed data sources with Oracle JDBC driver release 10.1.0.2.0 or later and external connection pools only.

There is no TopLink Workbench support for this feature. To configure TopLink to use Oracle Database proxy authentication, you must use Java (see "Using Java" on page 83-13).

For more information, see "Oracle Database Proxy Authentication" on page 81-5.

You can use TopLink support for Oracle Database proxy authentication by doing the following:

- Providing Authenticated Reads and Writes of Secured Data Through the Use of an Exclusive Isolated Client Session

- Providing Authenticated Writes for Database Auditing Purposes With a Client Session
- Providing Authenticated Writes for Database Auditing Purposes With a Client Session

Providing Authenticated Reads and Writes of Secured Data Through the Use of an Exclusive Isolated Client Session

In this configuration, the client Session is an isolated client session (see "Isolated Client Sessions" on page 72-19) that uses an exclusive proxy connection. You must acquire the client session using a `ConnectionPolicy` that specifies the proxy authentication user credentials.

Reads and writes of secured data are performed through the proxy-authenticated connection. Reads of nonsecured data occur through nonproxy-authenticated connections.

If you are using Oracle Private Virtual Database (VPD) (see "Isolated Client Sessions and Oracle Virtual Private Database (VPD)" on page 72-21), use this configuration to set up VPD support entirely in the database. That is, rather than making the isolated client session execute SQL (see "PostAcquireExclusiveConnection Event Handler" on page 77-1 and "PreReleaseExclusiveConnection Event Handler" on page 77-2), the database performs the required set up in an after login trigger using the proxy `session_user`.

Providing Authenticated Writes for Database Auditing Purposes With a Client Session

In this configuration, isolated data or exclusive connections are not required. You must acquire client session using a `ConnectionPolicy` that specifies the proxy authentication user credentials.

Writes are performed through the proxy-authenticated connection. Reads occur through nonproxy-authenticated connections. This enables the database auditing process to access the user that performed the write operations.

Providing Authenticated Reads and Writes With a Database Session

In this configuration, you use a `DatabaseSession` object with a proxy-authenticated login. All reads and writes occur through the proxy-authenticated connection.

Note: Oracle recommends that you exclusively use server and client sessions in a three-tier environment.

Do not use database sessions in a three-tier environment. Ensure that a database session is used by a single user and not accessed concurrently.

Using Java

You configure Oracle Database proxy authentication by customizing your session in your Java code, such as through a `SessionCustomizer` when using the `sessions.xml` file. You can wrap a configured TopLink `DatasourceLogin` `JNDIConnector` with a TopLink proxy connector instance (from `oracle.toplink.platform.database.oracle`) appropriate for your JDBC driver and to configure proxy authentication properties.

If you are using the Oracle JDBC OCI driver, use the `OracleOCIProxyConnector` and property constants defined in `oracle.jdbc.pool.OracleOCIConnectionPool`.

If you are using the Oracle JDBC Thin driver, use the `OracleJDBC10_1_0_2ProxyConnector` and the property constants defined in `oracle.jdbc.OracleConnection`.

The properties to set are shown in Tables 83–2 through 83–5.

Note: Property constant names and values are consistent between the two classes except for `PROXYTYPE_` constants (such as `PROXYTYPE_USER_NAME`). In `OracleOCIConnectionPool` these are of type `String` and in `OracleConnection` they are of type `int`. If you are using the Oracle JDBC Thin driver and `OracleJDBC10_1_0_2ProxyConnector`, you must always set these properties as a `String`. For example:

```
login.setProperty(
    "proxytype", Integer.toString(OracleConnection.PROXYTYPE_USER_NAME));
```

To configure TopLink to use Oracle Database proxy authentication, do the following:

1. Decide on the proxy type you want to use and create appropriate users and roles.
   a. User Name Authentication:

      To authenticate a proxy user `sarah` by user name only, create the user account on the Oracle Database using the following:

      ```
      alter user sarah grant connect through dbadminuser
          with roles clerk, reports;
      ```

      In this case, you will need to set the proxy properties shown in Table 83–2.

Table 83–2 Proxy Properties for User Name Authentication

Property Name	Property Value
"proxytype"	PROXYTYPE_USER_NAME
PROXY_USER_NAME	"sarah"
PROXY_ROLES	String[] {"role1", "role2", ...}

   b. User Name and Password Authentication:

      To authenticate a proxy user `sarah` by user name and password, create the user account on the Oracle Database using the following:

      ```
      alter user sarah grant connect through dbadminuser
          authenticated using password
          with roles clerk, reports;
      ```

      In this case, you will need to set the proxy properties shown in Table 83–3.

Table 83–3 Proxy Properties for User Name and Password Authentication

Property Name	Property Value
"proxytype"	PROXYTYPE_USER_NAME

Table 83–3 (Cont.) Proxy Properties for User Name and Password Authentication

Property Name	Property Value
PROXY_USER_NAME	"sarah"
PROXY_PASSWORD	"passwordforsarah"
PROXY_ROLES	String[] {"role1", "role2", ...}

c. Distinguished Name Authentication:

To authenticate a proxy user `sarah` by globally unique distinguished name, create the user account on the Oracle Database using the following:

```
create user sarah identified globally as
    'CN=sarah,OU=americas,O=oracle,L=city,ST=ca,C=us';
alter user sarah grant connect through dbadminuser
    authenticated using distinguished name
    with roles clerk, reports;
```

In this case, you will need to set the proxy properties shown in Table 83–4.

Table 83–4 Proxy Properties for Distinguished Name Authentication

Property Name	Property Value
"proxytype"	PROXYTYPE_DISTINGUISHED_NAME
PROXY_DISTINGUISHED_NAME	"CN=sarah,OU=americas,O=oracle,L=city,ST=ca,C=us"
PROXY_ROLES	String[] {"role1", "role2", ...}

d. Certificate Authentication:

To authenticate a proxy user `sarah` by encrypted distinguished name, create the user account on the Oracle Database using the following:

```
alter user sarah grant connect through dbadminuser
    authenticated using certificate
    with roles clerk, reports;
```

In this case, you will need to set the proxy properties shown in Table 83–2.

Table 83–5 Proxy Properties for User Name Authentication

Property Name	Property Value
"proxytype"	PROXYTYPE_CERTIFICATE
PROXY_CERTIFICATE	byte[] {<EncryptedCertificate>}
PROXY_ROLES	String[] {"role1", "role2", ...}

2. Configure your session login using Java code. Do this through a `SessionCustomizer` when using the `sessions.xml` file.

The following example demonstrates how you can wrap the already specified `JNDIConnector` with the appropriate TopLink proxy authentication connector. You can set the server session's default connection policy to the same proxy authenticated login.

If you use Oracle VPD (ref VPD), you should set the connection policy to use exclusive connections, and the descriptor for secured data to isolated (ref isolated).

```
Login login = server.getDatasourceLogin();
// Make sure that external connection pooling is used
login.setUsesExternalConnectionPooling(true);
// Wrap JNDIConnector with either
// OracleOCIProxyConnector or OracleJDBC10_1_0_2ProxyConnector
login.setConnector(
    new OracleOCIProxyConnector(
        ((JNDIConnector)login.getConnector()).getName()));
ConnectionPolicy policy = server.getDefaultConnectionPolicy();
policy.setPoolName(null);
policy.setLogin(login);
// If using Oracle VPD support,set the connection policy to exclusive
policy.setShouldUseExclusiveConnection(true);
```

3. Acquire a proxy-authenticated client session through specifying a `ConnectionPolicy` with this user's credentials.

```
ConnectionPolicy policy =
    (ConnectionPolicy)server.getDefaultConnectionPolicy().clone();
Login login = (Login)policy.getLogin().clone;
// Set proxy properties into connection policy's login
login.setProperty("proxytype" , OracleOCIConnectionPool.PROXYTYPE_USER_NAME);
login.setProperty(OracleOCIConnectionPool.PROXY_USER_NAME ,"sarah");
policy.setLogin(login);
Session session = server.acquireClientSession(policy);
```

84

Configuring an EIS Login

This chapter describes the various components that you must configure to use an EIS login.

EIS Login Configuration Overview

Table 84–1 lists the configurable options for an EIS login.

Table 84–1 Configurable Options for EIS Login

Option	Type	TopLink Workbench	Java
"Configuring an EIS Data Source Platform at the Session Level" on page 84-1	Basic	✔	✔
"Configuring EIS Connection Specification Options at the Session Level" on page 84-2	Basic	✔	✔
"Configuring User Name and Password" on page 82-1	Basic	✔	✔
"Configuring Password Encryption" on page 82-2	Advanced	✔	✔
"Configuring External Connection Pooling" on page 82-2	Advanced	✔	✔
"Configuring Properties" on page 82-4	Advanced	✔	✔

Configuring an EIS Data Source Platform at the Session Level

For each EIS session, you must specify the platform (such as AQ). This platform configuration overrides the platform at the project level, if configured.

For more information, see the following:

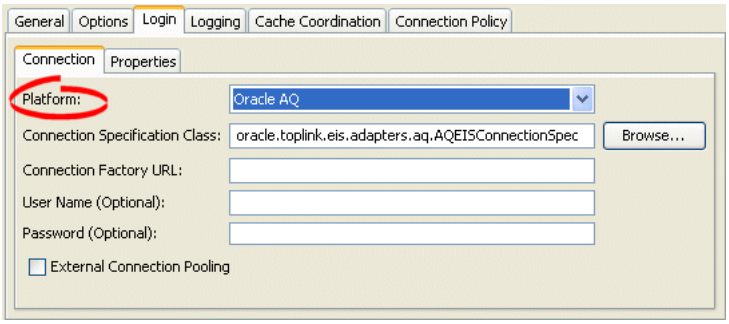
- "Configuring Relational Database Platform at the Project Level" on page 20-2
- "Data Source Login Types" on page 81-2

Using TopLink Workbench

To specify the database platform options for an EIS session login, use this procedure:

1. Select an EIS session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Login** tab. The Login tab appears.
3. Click the **Connection** subtab. The Connection subtab appears.

Figure 84–1 Login Tab, Connection Subtab, Platform Options



Use the following information to enter data in the Platform field on the Connection tab to configure the platform:

Field	Description
Platform	The EIS platform for the session. Select from the menu of options. This menu includes all instances of `EISPlatform` in the TopLink classpath.

Configuring EIS Connection Specification Options at the Session Level

You can configure connection information at the session level for an EIS application. This information is stored in the `sessions.xml` file. The Oracle TopLink runtime uses this information whenever you perform a persistence operation using the session in your EIS application.

This connection configuration overrides the connection information at the project level, if configured. For more information about project-level configuration, see "Configuring Development and Deployment Logins" on page 20-6 and "Configuring EIS Connection Specification Options at the Project Level" on page 21-2.

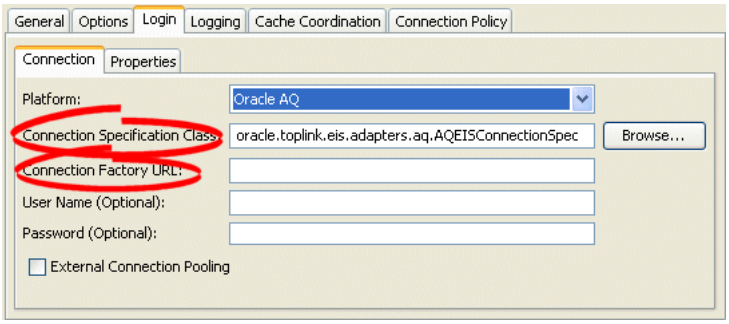
This connection configuration is overridden by the connection information at the connection pool level. For more information about connection pool-level configuration, see "Configuring Connection Pool Connection Options" on page 86-4.

Using TopLink Workbench

Use this procedure to specify the connection options for an EIS session login.

1. Select an EIS session in the Navigator window. Its properties appear in the Editor window.
2. Click the **Login** tab. The Login tab appears.
3. Click the **Connection** subtab. The Connection tab appears.

Figure 84–2 Login Tab, Connection Subtab



Use the following information to enter data in the connection fields on the tab:

Field	Description
Connection Specification Class	Specify the appropriate connection specification class for the selected **Platform**. Click **Browse** to choose from all the classes in the TopLink classpath. (For example: if **Platform** is `oracle.toplink.eis.aq.AQPlatform`, use `oracle.toplink.eis.aq.AQEISConnectionSpec`). For more information on platform configuration, see "Configuring an EIS Data Source Platform at the Session Level" on page 84-1.
Connection Factory URL	Specify the appropriate connection factory URL for the selected **Connection Specification Class** (For example: `jdbc:oracle:thin@:localhost:1521:orcl`).

85

Creating an Internal Connection Pool

This chapter explains how to create TopLink internal connection pools, including the following:

- Internal Connection Pool Creation Overview

For information, see "Internal Connection Pools" on page 81-7.

Internal Connection Pool Creation Overview

You can create internal connection pools only for server sessions (not for any other session type, including database sessions).

You can create an internal connection pool using TopLink Workbench or Java code. Oracle recommends that you use TopLink Workbench to create and manage your internal connection pools. For more information, see "Using TopLink Workbench" on page 85-1.

Alternatively, you can create internal connection pools in Java. For more information on creating sessions in Java, see the Oracle TopLink API Reference.

After you create an internal connection pool, you must configure its various options (see "Internal Connection Pool Configuration Overview" on page 86-1).

After you create and configure a sequence connection pool, TopLink uses it whenever it needs to assign an identifier to a new object.

After you create and configure a named connection pool, you use it in your application by passing in a `ConnectionPolicy` when you acquire a client session (see "Acquiring a Client Session That Uses a Named Connection Pool" on page 75-8).

Using TopLink Workbench

Before you create a connection pool, you must first create a server session (see "Creating a Server Session" on page 73-4).

To create a new TopLink internal connection pool, use this procedure:

1. Select the server session in the **Navigator** in which you want to create a connection pool.



2. Click the appropriate button on the toolbar to create the type of connection pool you want:
   - To create a named connection pool, select **Create a New Named Connection Pool**, enter a name, and click **OK**.

- To create a sequence connection pool, select **Add the Sequence Connection Pool**.
- To create a write connection pool, select **Add the Write Connection Pool**.

You can also create a new internal connection pool by right-clicking the server session configuration in the **Navigator** and selecting **New > Named Connection Pool**, **Sequence Connection Pool**, or **Write Connection Pool** from the context menu.

86

Configuring an Internal Connection Pool

This chapter describes the various components that you must configure to use an internal connection pool.

Internal Connection Pool Configuration Overview

When you are using server sessions, you can configure the default read connection pool and write connection pool. You can also configure the optional named connection pools and sequence connection pool you may have created (see "Internal Connection Pool Creation Overview" on page 85-1).

Table 86–1 lists the configurable options for an internal connection pool.

Table 86–1 Configurable Options for Connection Pool

Option	Type	TopLink Workbench	Java
"Configuring Connection Pool Sizes" on page 86-1	Basic	✔	✔
"Configuring Exclusive Read Connections" on page 86-6[1]	Advanced	✔	✔
"Configuring a Nontransactional Read Login" on page 86-3[1]	Advanced	✔	✔
"Configuring Properties" on page 86-2	Advanced	✔	✔
"Configuring Connection Pool Connection Options" on page 86-4[2]	Advanced	✔	✔

[1] Read connection pools only.

[2] Not applicable to write connection pools.

Configuring Connection Pool Sizes

By default, if using TopLink internal connection pooling, the TopLink write connection pool maintains a minimum of five connections and a maximum of ten. The read connection pool maintains a minimum and maximum of two connections.

Connection pool size can significantly influence the concurrency of your application and should be set to be large enough to handle your expected application load.

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 smallest value you can enter is 0. Setting the maximum number of connections to 0 will make it impossible for TopLink to allocate any connections.

The minimum number of connections should always be less than or equal to the maximum number of connections.

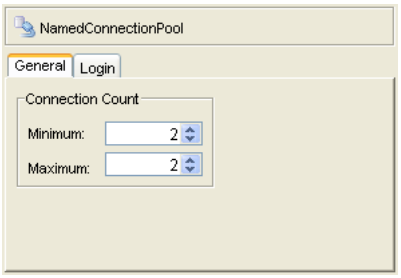
If the maximum number of connections is in use, the next connection request will be blocked until a connection is available.

Using TopLink Workbench

To specify the minimum and maximum number of connections in a TopLink internal connection pool, use this procedure:

1. Expand a server session to reveal its connection pools in the **Navigator**.
2. Select a connection pool in the **Navigator**. Its properties appear in the Editor.
3. Click the **General** tab. The General tab appears.

Figure 86–1 General Tab, Connection Count Options



Enter the desired minimum and maximum number of connections and press **Enter** or use the increment and decrement arrows.

Configuring Properties

For all connection pools, except write connection pools, you can specify arbitrary named values, called properties.

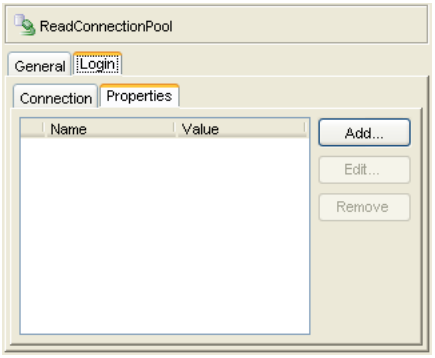
Some data sources require additional, driver-specific properties not supported in the `ConnectionPool` API. Add these properties to the `ConnectionPool` so that TopLink can pass them to the driver.

Using TopLink Workbench

To specify arbitrary named value pairs that TopLink associates with a `ConnectionPool`, use this procedure:

1. Expand a server session to reveal its connection pools in the **Navigator**.
2. Select a read, named, or sequence connection pool in the **Navigator**. Its properties appear in the Editor.
3. Click the **Login** tab. The Login tab appears.
4. Click the **Properties** subtab. The Properties subtab appears.

Figure 86–2 Login Tab, Properties Subtab



Use the following information to add or edit a login property on the Add Property dialog box to add or edit a login property:

Option	Description
Name	The name by which TopLink retrieves the property value using the `DatasourceLogin` method `getProperty`.
Value	The value TopLink retrieves using the `DatasourceLogin` method `getProperty` passing in the corresponding property name. Using TopLink Workbench, you can set only character values which TopLink returns as `String` objects.

To add (or change) a new **Name**/**Value** property, click **Add** (or **Edit**).

To delete an existing property, select the **Name**/**Value** row and click **Remove**.

Using Java

Using Java, you can set any `Object` value using the `DatasourceLogin` method `setProperty`. To remove a property, use the `DatasourceLogin` method `removeProperty`.

Configuring a Nontransactional Read Login

When you use an external transaction controller (see "Configuring the Server Platform" on page 74-14), establishing a connection requires not only the usual connection setup overhead, but also transactional overhead. If your application reads data only to display it and only infrequently modifies data, you can configure an internal read connection pool to use its own connection specification that does not use the external transaction controller. This may improve performance by reducing the time it takes to establish a new read connection.

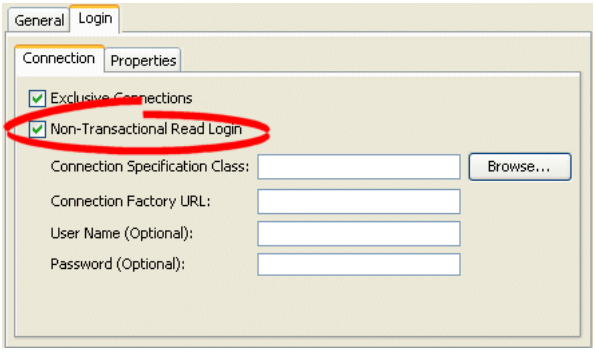
Using TopLink Workbench

To enable the configuration of nontransactional connection information for a TopLink read connection pool, use this procedure:

1. Expand a server session to reveal its connection pools in the **Navigator**.
2. Select a read connection pool in the **Navigator**. Its properties appear in the Editor.
3. Click the **Login** tab. The Login tab appears.

4. Click the **Connection** subtab. The Connection subtab appears.

Figure 86–3 Login Tab, Connection Subtab



To enable a nontransactional read login, select the **Use Non-Transactional Read Login** option (see "Externally Managed Transactional Data Sources" on page 81-1). Continue with "Configuring Connection Pool Connection Options" on page 86-4 to specify the connection information.

Configuring Connection Pool Connection Options

By default, connection pools use the login configuration specified for their session (see "Configuring Database Login Connection Options" on page 83-2 and "Configuring EIS Connection Specification Options at the Session Level" on page 84-2).

For read, named, and sequence connection pools, you can override the session login configuration on a per-connection pool basis.

To configure login configuration for a read connection pool, you must first enable it for a nontransactional read login (see "Configuring a Nontransactional Read Login" on page 86-3).

Using TopLink Workbench

To configure connection information for a TopLink read, named, or sequence connection pool, use this procedure:

1. Expand a server session to reveal its connection pools in the **Navigator**.
2. Select a read, named, or sequence connection pool in the **Navigator**. Its properties appear in the Editor.
3. Click the **Login** tab. The Login tab appears.
4. Click the **Connection** subtab. The Connection subtab appears.

Figure 86–4 Login Tab, Connection Subtab, Relational Session Connection Pool Options

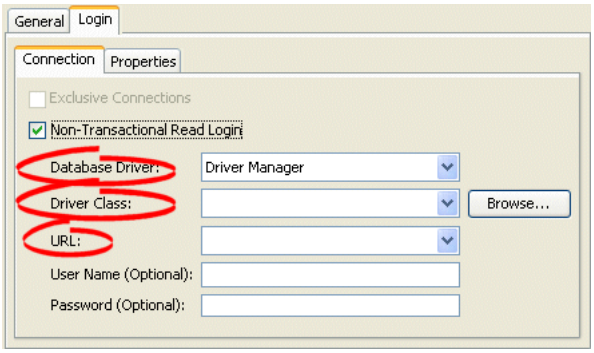
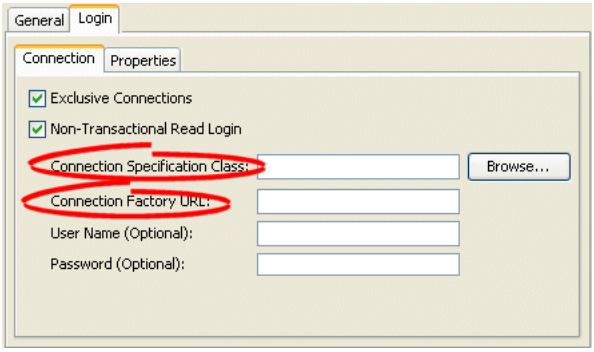


Figure 86–5 Login Tab, Connection Subtab, EIS Session Connection Pool Options



5. Ensure the **Use Non-Transaction Read Login** option is selected.

Use the following information to complete fields on the Connection subtab:

Field	Description
Database Driver[1]	Specify the appropriate database driver: ▪ **Driver Manager**: Specify this option to configure the driver class and URL used to connect to the database. In this case, you must configure the **Driver Class** and **Driver URL** fields. ▪ **J2EE Datasource**: Specify this option to use a J2EE data source already configured on your target application server. In this case, you must configure the **Datasource Name** field. Note: If you select **J2EE Datasource**, you must use external connection pooling. You cannot use internal connection pools with this **Database Driver** option (for more information, see "Configuring External Connection Pooling" on page 82-2).
Driver Class[1]	Configure this field when **Database Driver** is set to **Driver Manager**. Select from the menu of options. This menu includes all JDBC drivers in the TopLink application classpath.
URL[1]	Configure this field when **Database Driver** is set to **Driver Manager**. Select from the menu of options relevant to the selected **Driver Class** and edit the URL to suit your data source.

Field	Description
Datasource Name[1]	Configure this field when **Database Driver** is set to **J2EE Datasource**. Specify any valid JNDI name that identifies the J2EE data source preconfigured on your target application server (For example: `jdbc/EmployeeDB`). By convention, all such names should resolve to the JDBC subcontext (relative to the standard `java:comp/env` naming context that is the root of all provided resource factories).
Connection Specification Class[2]	Specify the appropriate connection specification class for the selected **Platform**. Click **Browse** to choose from all the classes in the TopLink classpath. (For example: if **Platform** is `oracle.toplink.eis.aq.AQPlatform`, use `oracle.toplink.eis.aq.AQEISConnectionSpec`). For more information on platform configuration, see "Configuring an EIS Data Source Platform at the Session Level" on page 84-1.
Connection Factory URL[2]	Specify the appropriate connection factory URL for the selected **Connection Specification Class** (For example: `jdbc:oracle:thin@:localhost:1521:orcl`).

[1] For sessions that contain a `DatabaseLogin`.

[2] For sessions that contain an `EISLogin`.

Configuring Exclusive Read Connections

An exclusive connection is one that TopLink allocates specifically to a given session and one that is never used by any other session.

Allowing concurrent reads on the same connection reduces the number of read connections required and reduces the risk of having to wait for an available connection. However, many JDBC drivers do not support concurrent reads.

If you are using internal connection pools (see "Internal Connection Pools" on page 81-7), you can configure TopLink to acquire an exclusive connection from the read connection pool.

By default, TopLink acquires exclusive read connections.

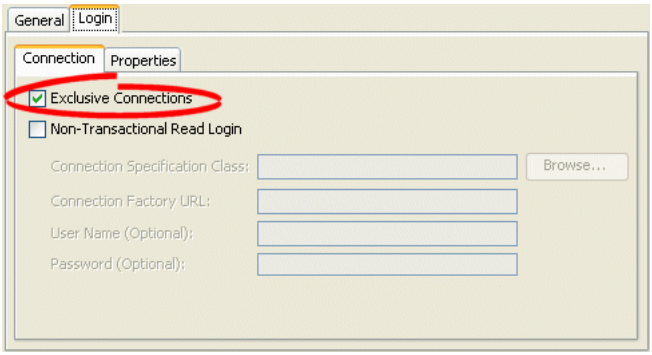
If you are using external connection pools, read connections are always exclusive.

Using TopLink Workbench

To configure a TopLink read connection pool to allocate exclusive connections, use this procedure:

1. Expand a server session to reveal its connection pools in the **Navigator**.
2. Select a read connection pool in the **Navigator**. Its properties appear in the Editor.
3. Click the **Login** tab. The Login tab appears.
4. Click the **Connection** subtab. The Connection subtab appears.

Figure 86–6 Login Tab, Connection Subtab, Exclusive Connections Option



Select the **Exclusive Connections** option to configure TopLink to acquire an exclusive connection from the read connection pool.

Deselect the **Exclusive Connections** option to configure TopLink to share read connections and allow concurrent reads. Before selecting this option, ensure that your JDBC driver supports concurrent reads.

Part XVII

Cache

This part describes using the TopLink object cache in both distributed and nondistributed applications. It contains the following chapters:

- Chapter 87, "Understanding the Cache"

  This chapter describes each of the different TopLink cache types and important cache concepts.

- Chapter 88, "Configuring a Coordinated Cache"

  This chapter explains how to configure TopLink coordinated cache options common to two or more coordinated cache types.

- Chapter 89, "Configuring a JMS Coordinated Cache"

  This chapter explains how to configure a TopLink JMS coordinated cache.

- Chapter 90, "Configuring an RMI Coordinated Cache"

  This chapter explains how to configure a TopLink RMI coordinated cache.

- Chapter 91, "Configuring a CORBA Coordinated Cache"

  This chapter explains how to configure a TopLink CORBA coordinated cache.

87

Understanding the Cache

The TopLink cache is an in-memory repository that stores recently read or written objects based on class and primary key values. TopLink uses the cache to do the following:

- Improve performance by holding recently read or written objects and accessing them in-memory to minimize database access
- Manage locking and isolation level
- Manage object identity

This section describes the following:

- Cache Architecture
- Cache Concepts
- Understanding the Cache API

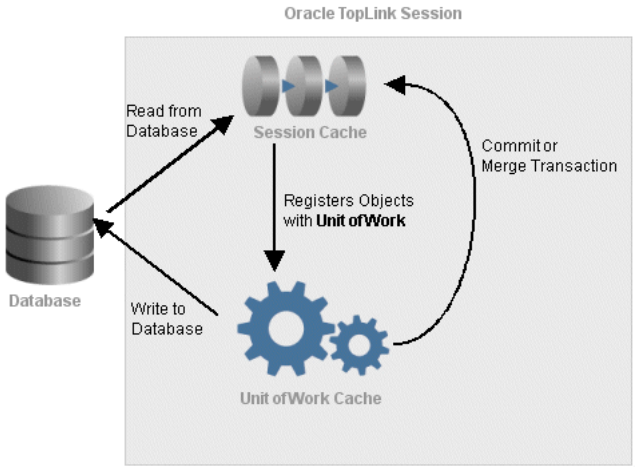
Cache Architecture

TopLink uses two types of cache: the **session cache** maintains objects retrieved from and written to the data source; and the **unit of work cache** holds objects while they participate in transactions. When a unit of work successfully commits to the data source, TopLink updates the session cache accordingly.

> **Note:** You can also configure a query to cache its results (see "Caching Results in a ReadQuery" on page 96-20)

As Figure 87–1 shows, the session cache and the unit of work cache work together with the data source connection to manage objects in a TopLink application. The object life cycle relies on these three mechanisms.

Figure 87–1 Object Life Cycle and the TopLink Caches



Session Cache

The session cache is a shared cache that services clients attached to a given session. When you read objects from or write objects to the data source using a client session, TopLink saves a copy of the objects in the parent server session's cache and makes them accessible to all other processes in the session.

TopLink adds objects to the session cache from the following:

- The data store, when TopLink executes a read operation
- The unit of work cache, when a unit of work successfully commits a transaction

An isolated client session is a special type of client session that provides its own session cache isolated from the shared object cache of its parent server session. The isolated client session cache can be used to improve user-based security or to avoid caching highly volatile data. For more information, see "Isolated Client Sessions" on page 72-19.

Unit of Work Cache

The unit of work cache services operations within the unit of work. It maintains and isolates objects from the session cache, and writes changed or new objects to the session cache after the unit of work commits changes to the data source.

Cache Concepts

This section describes concepts unique to the TopLink cache, including the following:

- Cache Type and Object Identity
- Querying and the Cache
- Handling Stale Data
- Explicit Query Refreshes
- Cache Invalidation
- Cache Coordination
- Cache Isolation

- Cache Locking and Transaction Isolation
- Cache Optimization

Cache Type and Object Identity

TopLink preserves object identity through its cache using the primary key attributes of a persistent entity. These attributes may or may not be assigned through sequencing (see "Projects and Sequencing" on page 17-4). In a Java application, object identity is preserved if each object in memory is represented by one, and only one, object instance. Multiple retrievals of the same object return references to the same object instance–not multiple copies of the same object.

Maintaining object identity is extremely important when the application's object model contains circular references between objects. You must ensure that the two objects are referencing each other directly, rather than copies of each other. Object identity is important when multiple parts of the application may be modifying the same object simultaneously.

Oracle recommends that you always maintain object identity. Disable object identity only if absolutely necessary, for example, for read-only objects (see "Configuring Read-Only Descriptors" on page 25-5).

You can configure how object identity is managed on a class-by-class basis. The `Descriptor` object provides the cache and identity map options described in Table 87–1.

Table 87–1 Cache and Identity Map Options

Option (Identity Map)	Caching	Guaranteed Identity	Memory Use	Client/Server Transaction Save
Full Identity Map	Yes	Yes	High	Yes
Weak Identity Map	Yes	Yes	Low	No
Soft and Hard Cache Weak Identity Maps	Yes	Yes	Lower	Yes
No Identity Map	No	No	None	No

For more information, see "Guidelines for Configuring the Cache and Identity Maps" on page 87-4.

Full Identity Map

This option provides full caching and guaranteed identity: objects are never flushed from memory unless they are deleted.

It caches all objects and does not remove them. Cache size doubles whenever the maximum size is reached. This method may be memory-intensive when many objects are read. Do not use this option on batch operations.

Oracle recommends using this identity map when the data set size is small and memory is in large supply.

Weak Identity Map

This option is similar to the full identity map, except that the map holds the objects by using weak references. This method allows full garbage collection and provides full caching and guaranteed identity.

The weak identity map uses less memory than full identity map but also does not provide a durable caching strategy across client/server transactions. Objects are available for garbage collection when the application no longer references them on the server side (that is, from within the server JVM).

Oracle recommends using this identity map for transactions that, once started, stay on the server side. Do not use this option for applications that expect objects to remain cached across client/server invocations.

Soft and Hard Cache Weak Identity Maps

This option is similar to the weak identity map except that it maintains a most frequently used subcache. The subcache uses soft or hard references to ensure that these objects are garbage-collected only if the system is low on memory.

The soft cache weak identity map and hard cache weak identity map provide more efficient memory use. They release objects as they are garbage-collected, except for a fixed number of most recently used objects. Note that weakly cached objects might be flushed if the transaction spans multiple client/server invocations. The size of the subcache is proportional to the size of the identity map as specified by the `Descriptor` method `setIdentityMapSize`. You should set this cache size to be as large as the maximum number of objects (of the same type) referenced within a transaction (see "Configuring Cache Type and Size at the Descriptor Level" on page 25-35).

Oracle recommends using this identity map in most circumstances as a means to control memory used by the cache.

For more information, see "Understanding the Internals of Soft and Hard Cache Weak Identity Map" on page 87-5.

No Identity Map

This option does not preserve object identity and does not cache objects.

Oracle does not recommend using the no identity map option.

Guidelines for Configuring the Cache and Identity Maps

You can configure the cache at the project ("Configuring Cache Type and Size at the Project Level" on page 19-13) or descriptor ("Configuring Cache Type and Size at the Descriptor Level" on page 25-35) level.

Use the following guidelines when configuring your cache and identity map:

- If objects with a long life span and object identity are important, use a `SoftCacheWeakIdentityMap` or `HardCacheWeakIdentityMap` policy. For more information on when to choose one or the other, see "Understanding the Internals of Soft and Hard Cache Weak Identity Map" on page 87-5.
- If object identity is important, but caching is not, use a `WeakIdentityMap` policy.
- If an object has a long life span or requires frequent access, or object identity is important, use a `FullIdentityMap` policy.

> **WARNING: Use the `FullIdentityMap` only if the class has a small number of finite instances. Otherwise, a memory leak will occur.**

- If an object has a short life span or requires frequent access, and identity is not important, use a `CacheIdentityMap` policy.
- If objects are discarded immediately after being read from the database, such as in a batch operation, use a `NoIdentityMap` policy. The `NoIdentityMap` does not preserve object identity.

> **Note:** Oracle does not recommend the use of `CacheIdentityMap` and `NoIdentityMap` policies.

Understanding the Internals of Soft and Hard Cache Weak Identity Map

The `SoftCacheWeakIdentityMap` and `HardCacheWeakIdentityMap` types of identity map contain two caches:

- Reference cache: implemented as a `LinkedList` that contains soft or hard references, respectively
- Weak cache: implemented as a `HashMap` that contains weak references

When you create a `SoftCacheWeakIdentityMap` or `HardCacheWeakIdentityMap` with a specified size, the reference cache `LinkedList` is exactly this size. The weak cache `HashMap` is initialized to 100 percent of the specified size: the weak cache will grow when more objects than the specified size are read in. Because TopLink does not control garbage collection, the JVM can reap the weakly held objects whenever it sees fit.

Because the reference cache is implemented as a `LinkedList`, new objects are added to the end of the list. Because of this, it is by nature a least recently used (LRU) cache: fixed size, object at the top of the list is deleted, provided the maximum size has been reached.

The `SoftCacheWeakIdentityMap` and `HardCacheWeakIdentityMap` are essentially the same type of identity map. The `HardCacheWeakIdentityMap` was constructed to work around an issue with some JVMs.

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 reference cache may be garbage-collected so often that you will not benefit from the performance improvement provided by it. 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 reference cache. This guarantees that your application will benefit from the performance improvement provided by it.

When an object in a `HardCacheWeakIdentityMap` or `SoftCacheWeakIdentityMap` is pushed out of the reference cache, it gets put in the weak cache. Although it is still cached, TopLink cannot guarantee that it will be there for any length of time because the JVM can decide to garbage-collect weak references at anytime.

TopLink cleans up dead cache keys after every n^{th} access to a `HardCacheWeakIdentityMap` or `SoftCacheWeakIdentityMap`. For example, if you set the size to 100, then TopLink locks the cache and cleans up these cache keys on the 100^{th} access, 200^{th} access, 300^{th} access, and so on. Although this may appear to be a memory leak, it is not: it is a compromise that provides improved performance at the expense of holding on to memory until the n^{th} access. While reducing n frees memory more frequently, it does so at the unacceptable performance cost of performing an enumeration of the cache too frequently.

If you are querying in memory and you absolutely need the guarantee that the objects are in memory, then you should have those classes set to `FullIdentityMap` (see "Full Identity Map" on page 87-3). Querying in memory with `SoftCacheWeakIdentityMap` or `HardCacheWeakIdentityMap` does not 100 percent guarantee that the query will retrieve the objects that you expect.

When you query in memory, TopLink does not move the objects around to maintain a strict LRU. This behavior occurs only when you merge or query the database. Not supporting in-memory LRU functionality is a compromise that enhances performance. To support in-memory LRU functionality, the identity map would have to be locked for the duration of the move, resulting in performance degradation.

Querying and the Cache

A query that is run against the shared session cache is known as an **in-memory query**. Careful configuration of in-memory querying can improve performance (see "Using In-Memory Queries" on page 93-30).

By default, a query that looks for a single object based on primary key attempts to retrieve the required object from the cache first, searches the data source only 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 "Queries and the Cache" on page 93-29.

Handling Stale Data

Stale data is an artifact of caching, in which an object in the cache is not the most recent version committed to the data source. To avoid stale data, implement an appropriate cache locking strategy.

By default, TopLink optimizes concurrency to minimize cache locking during read or write operations. Use the default TopLink isolation level, unless you have a very specific reason to change it. For more information on isolation levels in TopLink, see "Database Transaction Isolation Levels" on page 99-26.

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 within another process.

A well-managed cache makes your application more efficient. There are very few cases in which you turn the cache off entirely, because 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, Oracle recommends the following:

- Configure a Locking Policy
- Configure the Cache on a Per-Class Basis
- Force a Cache Refresh When Required on a Per-Query Basis
- Configure Cache Invalidation
- Configure Cache Coordination

Configure a Locking Policy

Make sure you configure a locking policy so that you can prevent or at least identify when values have already changed on an object you are modifying. Typically, this is

done using optimistic locking. TopLink offers several locking policies such as numeric version field, time-stamp version field, and some or all fields.

For more information, see "Configuring Locking Policy" on page 25-64.

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 an object whose reference has been removed.

For more information, see "Configuring Cache Type and Size at the Descriptor Level" on page 25-35.

Force a Cache Refresh When Required on a Per-Query Basis

Any query can include a flag that forces TopLink to go to the data source for the most up-to-date version of selected objects and update the cache with this information.

For more information, see the following:

- "Cache Refresh API" on page 87-14
- "Using DatabaseQuery Queries" on page 94-4
- "Using Named Queries" on page 94-18

Configure Cache Invalidation

Using descriptor API, you can designate an object as invalid: when any query attempts to read an invalid object, TopLink will go to the data source for the most up to date version of that object and update the cache with this information. You can manually designate an object as invalid or use a `CacheInvalidationPolicy` to control the conditions under which an object is designated invalid.

For more information, see "Cache Invalidation" on page 87-8.

Configure Cache Coordination

If your application is primarily read-based and the changes are all being performed by the same Java application operating with multiple, distributed sessions, you may consider using the TopLink cache coordination feature. Although this will not prevent stale data, it should greatly minimize it.

For more information, see "Cache Coordination" on page 87-9.

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 coordination.

To implement this type of strategy, do the following:

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, TopLink returns the cached objects without checking the database for a more recent version. This reduces the number of objects that TopLink must build from database results, and is optimal for noncoordinated cache environments. However, this may not always be the best strategy for a coordinated cache 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 TopLink descriptor, or just on certain queries, depending upon the nature of the application.

For more information, see the following:

- "Configuring Cache Refreshing" on page 25-27
- "Refreshing the Cache" on page 93-35

> **Note:** Refreshing does not prevent phantom reads from occurring. See "Refreshing Finder Results" on page 93-38.

EJB Finders and Refresh Policy

When you invoke a `findByPrimaryKey` finder, if the object exists in the cache, 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 calling `refreshIdentityMapResult` method on it.

For more information, see the following:

- "Queries and the Cache" on page 93-29
- "Configuring Named Query Options" on page 25-22

Cache Invalidation

By default, objects remain in the session cache until they are explicitly deleted (see "Deleting Objects" on page 98-7) or garbage collected when using a weak identity map (see "Configuring Cache Type and Size at the Project Level" on page 19-13).

Alternatively, you can configure any object with a `CacheInvalidationPolicy` that lets you specify, either automatically or manually, under what circumstances a cached object is invalid: when any query attempts to read an invalid object, TopLink will go to the data source for the most up-to-date version of that object, and update the cache with this information.

You can use any of the following `CacheInvalidationPolicy` instances:

- `DailyCacheInvalidationPolicy`: the object is automatically flagged as invalid at a specified time of day.
- `NoExpiryCacheInvalidationPolicy`: the object can only be flagged as invalid by explicitly calling `oracle.toplink.sessions.IdentityMapAccessor` method `invalidateObject`.
- `TimeToLiveCacheInvalidationPolicy`: the object is automatically flagged as invalid after a specified time period has elapsed since the object was read.

You can configure a cache invalidation policy in the following ways:

- At the project level that applies to all objects ("Configuring Cache Expiration at the Project Level" on page 19-19)
- At the descriptor level to override the project level configuration on a per-object basis ("Configuring Cache Expiration at the Descriptor Level" on page 25-42)
- At the query level that applies to the results returned by the query ("Configuring Cache Expiration at the Query Level" on page 96-21)

If you configure a query to cache results in its own internal cache (see "Caching Query Results in the Query Cache" on page 93-36), the cache invalidation policy you configure at the query level applies to the query's internal cache in the same way it would apply to the session cache.

If you are using a coordinated cache (see "Cache Coordination" on page 87-9), you can customize how TopLink communicates the fact that an object has been declared invalid. For more information, see "Configuring Cache Coordination Change Propagation at the Descriptor Level" on page 25-40.

Cache Coordination

The need to maintain up-to-date data for all applications is a key design challenge for building a distributed application. The difficulty of this increases as the number of servers within an environment increases. TopLink provides a distributed cache coordination feature that ensures data in distributed applications remains current.

Cache coordination reduces the number of optimistic lock exceptions encountered in a distributed architecture, and decreases the number of failed or repeated transactions in an application. However, cache coordination in no way eliminates the need for an effective locking policy. To effectively ensure working with up-to-date data, cache coordination must be used with optimistic or pessimistic locking. Oracle recommends that you use cache coordination with an optimistic locking policy (see "Configuring Locking Policy" on page 25-64).

You can use cache invalidation to improve cache coordination efficiency. For more information, see "Cache Invalidation" on page 87-8.

For more information, see "Understanding Cache Coordination" on page 87-10.

Cache Isolation

Isolated client sessions provide a mechanism for disabling the shared server session cache. Any classes marked as isolated only cache objects relative to the life cycle of their client session. These classes never utilize the shared server session cache. This is the best mechanism to prevent caching as it is configured on a per-class basis allowing caching for some classes, and denying for others.

For more information, see "Isolated Client Sessions" on page 72-19.

Cache Locking and Transaction Isolation

By default, TopLink optimizes concurrency to minimize cache locking during read or write operations. Use the default TopLink transaction isolation configuration unless you have a very specific reason to change it.

For more information, see "Database Transaction Isolation Levels" on page 99-26.

Cache Optimization

Tune the TopLink cache for each class to help eliminate the need for distributed cache coordination. Always tune these settings before implementing cache coordination.

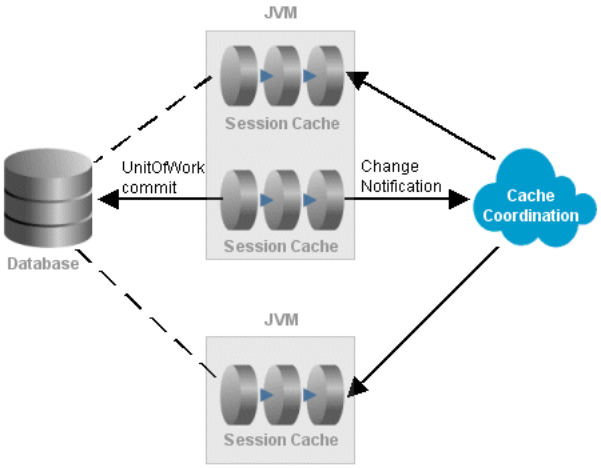
For more information, see "Cache Optimization" on page 11-13.

Understanding Cache Coordination

As Figure 87–2 shows, cache coordination is a session feature that allows multiple, possibly distributed, instances of a session to broadcast object changes among each other so that each session's cache is either kept up-to-date or notified that the cache must update an object from the data source the next time it is read.

Note: You cannot use isolated client sessions (see "Isolated Client Sessions" on page 72-19) with cache coordination.

Figure 87–2 Cache Coordination



When sessions are distributed, that is, when an application contains multiple sessions (in the same JVM, in multiple JVMs, possibly on different servers), as long as the servers hosting the sessions are interconnected on the network, sessions can participate in cache coordination. Coordinated cache types that require discovery services also require the servers to support User Datagram Protocol (UDP) communication and multicast configuration (for more information, see "Coordinated Cache Architecture" on page 87-11).

This section describes the following:

- When to use Cache Coordination
- Coordinated Cache Architecture
- Coordinated Cache Types
- Custom Coordinated Cache

For more information, see "Configuring a Coordinated Cache" on page 88-1.

When to use Cache Coordination

Cache coordination can enhance performance and reduce the likelihood of stale data for applications that have the following characteristics:

- Changes are all being performed by the same Java application operating with multiple, distributed sessions
- Primarily read-based
- Regularly requests and updates the same objects

To maximize performance, avoid cache coordination for applications that do not have these characteristics. For more information about alternatives to cache coordination, see "Cache Optimization" on page 11-13.

Cache coordination enhances performance mainly by avoiding data source access.

Cache coordination reduces the occurrence of stale data by increasing the likelihood that distributed caches are kept up-to-date with changes and are notified when one of the distributed caches must update an object from the data source the next time it is read.

Cache coordination reduces the number of optimistic lock exceptions encountered in a distributed architecture, and decreases the number of failed or repeated transactions in an application. However, cache coordination in no way eliminates the need for an effective locking policy. To effectively ensure working with up-to-date data, cache coordination must be used with optimistic or pessimistic locking. Oracle recommends that you use cache coordination with an optimistic locking policy (see "Configuring Locking Policy" on page 25-64).

For other options to reduce the likelihood of stale data, see "Handling Stale Data" on page 87-6.

Coordinated Cache Architecture

TopLink provides coordinated cache implementations that perform discovery and message transport services using various technologies including the following:

- Java Message Service (JMS)–See "JMS Coordinated Cache" on page 87-12
- Remote Method Invocation (RMI)–See "RMI Coordinated Cache" on page 87-12
- Common Object Request Broker Architecture (CORBA)–See "CORBA Coordinated Cache" on page 87-13

Regardless of the type of discovery and message transport you choose to use, the following are the principal objects that provide coordinated cache functionality:

- Session
- Descriptor
- Unit of Work

Session

When you enable a session for change propagation, the session provides discovery and message transport services using either JMS, RMI, or CORBA.

Discovery services ensure that sessions announce themselves to other sessions participating in cache coordination. Discovery services use UDP communication and multicast configuration to monitor sessions as they join and leave the coordinated cache. All coordinated cache types (except JMS) require discovery services.

Message transport services allow the session to broadcast object change notifications to other sessions participating in cache coordination when a unit of work from this session commits a change.

Descriptor

You can configure how object changes are broadcast on a descriptor-by-descriptor basis. This lets you fine-tune the type of notification to make.

For example, for an object with few attributes, you can configure its descriptor to send object changes. For an object with many attributes, it may be more efficient to configure its descriptor so that the object is flagged as invalid (so that other sessions will know to update the object from the data source the next time it is read).

Unit of Work

Only changes committed by a unit of work are subject to propagation when cache coordination is enabled. The unit of work computes the appropriate change set based on the descriptor configuration of affected objects.

Coordinated Cache Types

You can create the following types of coordinated cache:

- JMS Coordinated Cache
- RMI Coordinated Cache
- CORBA Coordinated Cache

JMS Coordinated Cache

For a JMS coordinated cache, when a particular session's coordinated cache starts up, it uses its JNDI naming service information to locate and create a connection to the JMS server. The coordinated cache is ready when all participating sessions are connected to the same topic on the same JMS server. At this point, sessions can start sending and receiving object change messages. You can then configure all sessions that are participating in the same coordinated cache with the same JMS and JNDI naming service information.

Because you must supply the necessary information to connect to the JMS Topic, a JMS coordinated cache does not use a discovery service.

If you do use cache coordination, Oracle recommends that you use JMS cache coordination: JMS is robust, easy to configure, and provides efficient support for asynchronous change propagation.

For more information, see Chapter 89, "Configuring a JMS Coordinated Cache".

RMI Coordinated Cache

For an RMI coordinated cache, when a particular session's coordinated cache starts up, the session binds its connection in its naming service (either an RMI registry or JNDI), creates an announcement message (that includes its own naming service information), and broadcasts the announcement to its multicast group (see "Configuring a Multicast Group Address" on page 88-4 and "Configuring a Multicast Port" on page 88-5). When a session that belongs to the same multicast group receives this announcement, it uses the naming service information in the announcement message to establish bidirectional connections with the newly announced session's coordinated cache. The coordinated cache is ready when all participating sessions are interconnected in this way, at which point sessions can start sending and receiving object change messages.

You can then configure each session with naming information that identifies the host on which the session is deployed.

If you do use cache coordination, Oracle recommends that you use RMI cache coordination only if you require synchronous change propagation (see "Configuring the Synchronous Change Propagation Mode" on page 88-2).

TopLink also supports cache coordination using RMI over the Internet Inter-ORB Protocol (IIOP). An RMI/IIOP coordinated cache uses RMI (and a JNDI naming service) for discovery and message transport services.

> **Note:** If you use an RMI coordinated cache, Oracle recommends that you use RMI/IIOP only if absolutely necessary.

For more information, see Chapter 90, "Configuring an RMI Coordinated Cache".

CORBA Coordinated Cache

For a CORBA coordinated cache, when a particular session's coordinated cache starts up, the session binds its connection in JNDI, creates an announcement message (that includes its own JNDI naming service information), and broadcasts the announcement to its multicast group (see "Configuring a Multicast Group Address" on page 88-4 and "Configuring a Multicast Port" on page 88-5). When a session that belongs to the same multicast group receives this announcement, it uses the naming service information in the announcement message to establish bidirectional connections with the newly announced session's coordinated cache. The coordinated cache is ready when all participating sessions are interconnected in this way, at which point, sessions can start sending and receiving object change messages. You can then configure each session with naming information that identifies the host on which the session is deployed.

Currently, TopLink provides support for the Sun Object Request Broker.

For more information on configuring a CORBA coordinated cache, see Chapter 91, "Configuring a CORBA Coordinated Cache".

Custom Coordinated Cache

Using the classes in `oracle.toplink.remotecommand` package, you can define your own coordinated cache for custom solutions. For more information, contact your TopLink support representative.

Once you have created the required cache coordination classes, for more information on configuring a user-defined coordinated cache, see Chapter 92, "Configuring a Custom Coordinated Cache".

Understanding the Cache API

To configure the TopLink cache, you use the appropriate API in the following objects:

- Object Identity API
- Cache Refresh API
- Cache Invalidation API
- Cache Coordination API

Object Identity API

You configure object identity using the `Descriptor` API summarized in Example 87–1.

For more information, see "Configuring Cache Type and Size at the Descriptor Level" on page 25-35.

Example 87–1 Object Identity Descriptor API

```
useCacheIdentityMap()
useFullIdentityMap()
useHardCacheWeakIdentityMap()
useNoIdentityMap()
useSoftCacheWeakIdentityMap()
useWeakIdentityMap()
```

Cache Refresh API

You configure cache refresh using the `Descriptor` API summarized in Example 87–2.

Example 87–2 Cache Refresh Descriptor API

```
alwaysRefreshCache()
alwaysRefreshCacheOnRemote()
disableCacheHits()
disableCacheHitsOnRemote()
onlyRefreshCacheIfNewerVersion()
```

You can also configure cache refresh using the following API calls:

- `Session`: `refreshObject` method
- `DatabaseSession` and `UnitOfWork: refreshAndLockObject` methods
- `ObjectLevelReadQuery: refreshIdentityMapResult` and `refreshRemoteIdentityMapResult` methods

For more information, see "Configuring Cache Refreshing" on page 25-27.

Cache Invalidation API

You configure cache invalidation using `Descriptor` methods `getCacheInvalidationPolicy` and `setCacheInvalidationPolicy` to configure an `oracle.toplink.descriptors.invalidation.CacheInvalidationPolicy`.

You can use any of the following `CacheInvalidationPolicy` instances:

- `DailyCacheInvalidationPolicy`: The object is automatically flagged as invalid at a specified time of day.
- `NoExpiryCacheInvalidationPolicy`: The object can only be flagged as invalid by explicitly calling `oracle.toplink.sessions.IdentityMapAccessor` method `invalidateObject`.
- `TimeToLiveCacheInvalidationPolicy`: The object is automatically flagged as invalid after a specified time period has elapsed since the object was read.

You can also configure cache invalidation using a variety of API calls accessible through the `Session`. The `oracle.toplink.sessions.IdentityMapAccessor` provides the following methods:

- `getRemainingValidTime`: Returns the remaining life of the specified object. This time represents the difference between the next expiry time of the object and its read time.
- `invalidateAll`: Sets all objects for all classes to be invalid in TopLink identity maps.
- `invalidateClass(Class klass)` and `invalidateClass(Class klass, boolean recurse)`: Set all objects of a specified class to be invalid in TopLink identity maps.
- `invalidateObject(Object object)`, `invalidateObject(Record rowWithPrimaryKey, Class klass)` and `invalidateObject(Vector primaryKey, Class klass)`: Set an object to be invalid in TopLink identity maps.
- `invalidateObjects(Expression selectionCriteria)` and `invalidateObjects(Vector collection)`: Set all objects from the specified `Expression/collection` to be invalid in TopLink identity maps.
- `isValid(DatabaseRow rowContainingPrimaryKey, Class theClass)`, `isValid(Object object)` and `isValid(java.util.Vector primaryKey, Class theClass)`: Return `true` if the object is valid in TopLink identity maps.

For more information, see the following:

- "Configuring Cache Expiration at the Project Level" on page 19-19
- "Configuring Cache Expiration at the Descriptor Level" on page 25-42
- "Configuring Cache Expiration at the Query Level" on page 96-21

Cache Coordination API

You configure cache coordination using the `Session` methods summarized in Example 87–3.

You configure how object changes are propagated using the `Descriptor` methods summarized in Example 87–4.

For more information, see "Configuring Common Coordinated Cache Options" on page 88-1.

Example 87–3 Cache Coordination Session API

```
Session.getCommandManager()
    setShouldPropagateAsynchronously(boolean)
Session.getCommandManager().getDiscoveryManager()
    setAnnouncementDelay()
    setMulticastGroupAddress()
    setMulticastPort()
    setPacketTimeToLive()
Session.getCommandManager().getTransportManager()
    setEncryptedPassword()
    setInitialContextFactoryName()
    setLocalContextProperties(Hashtable)
    setNamingServiceType() passing in one of:
        TransportManager.JNDI_NAMING_SERVICE
```

```
        TransportManager.REGISTRY_NAMING_SERVICE
    setPassword()
    setRemoteContextProperties(Hashtable)
    setShouldRemoveConnectionOnError()
    setUserName()
```

Example 87–4 Cache Coordination Descriptor API

```
setCacheSynchronizationType() passing in one of:
    ClassDescriptor.DO_NOT_SEND_CHANGES
    ClassDescriptor.INVALIDATE_CHANGED_OBJECTS
    ClassDescriptor.SEND_NEW_OBJECTS_WITH_CHANGES
    ClassDescriptor.SEND_OBJECT_CHANGES
```

88

Configuring a Coordinated Cache

This chapter describes how to configure a TopLink coordinated cache.

Table 88–1 Configuring TopLink Coordinated Caches

If you are configuring a...	See...
JMS Coordinated Cache	Chapter 89, "Configuring a JMS Coordinated Cache"
RMI Coordinated Cache	Chapter 90, "Configuring an RMI Coordinated Cache"
CORBA Coordinated Cache	Chapter 91, "Configuring a CORBA Coordinated Cache"
Custom Coordinated Cache	Chapter 92, "Configuring a Custom Coordinated Cache"

Table 88–1 lists the configurable options shared by two or more TopLink coordinated cache types.

For more information, see "Understanding Cache Coordination" on page 87-10.

Configuring Common Coordinated Cache Options

Table 88–1 lists the configurable options shared by two or more TopLink coordinated cache types. In addition to the configurable options described here, you must also configure the options described for the specific Coordinated Cache Types, as shown in Table 88–2.

Table 88–2 Common Coordinated Cache Options

Option	Type	TopLink Workbench	Java
"Configuring Cache Coordination Change Propagation at the Descriptor Level" on page 25-40	Basic	✔	✔
"Configuring the Synchronous Change Propagation Mode" on page 88-2	Basic	✔	✔
"Configuring a Service Channel" on page 88-3	Basic	✔	✔
"Configuring a Multicast Group Address" on page 88-4	Basic	✔	✔
"Configuring a Multicast Port" on page 88-5	Basic	✔	✔
"Configuring a Naming Service Type" on page 88-7	Basic	✔	✔
"Configuring an Announcement Delay" on page 88-11	Basic	✔	✔
"Configuring Connection Handling" on page 88-13	Advanced	✔	✔
"Configuring Context Properties" on page 88-14	Advanced	✔	✔
"Configuring a Packet Time-to-Live" on page 88-15	Advanced	✔	✔

Configuring the Synchronous Change Propagation Mode

You can configure whether the coordinated cache should propagate object changes asynchronously or synchronously.

Table 88–3 summarizes which coordinated caches support propagation mode configuration.

Table 88–3 Coordinated Cache Support for Propagation Mode Configuration

Coordinated Cache	Using TopLink Workbench	Using Java
JMS Coordinated Cache (asynchronous only)		
RMI Coordinated Cache	✔	✔
CORBA Coordinated Cache	✔	✔
Custom Coordinated Cache		

Synchronous propagation mode forces the session to wait for an acknowledgement before sending the next object change notification: this reduces the likelihood of stale data at the expense of performance.

Asynchronous propagation mode allows the session to create separate threads to propagate changes to remote servers. TopLink returns control to the client immediately after the local commit operation, whether or not the changes merge successfully on the remote servers. This offers superior performance for applications that are somewhat tolerant of stale data.

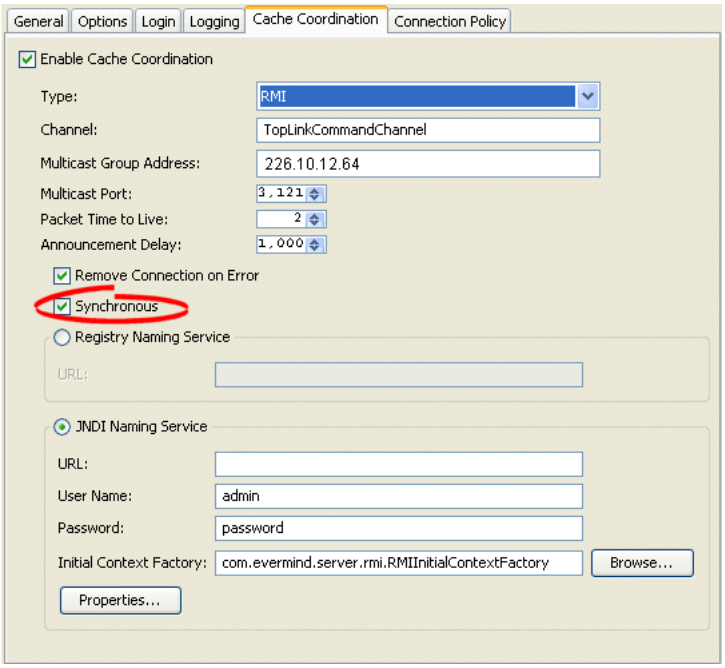
For more information, "Handling Stale Data" on page 87-6.

Using TopLink Workbench

To specify the change propagation mode, use this procedure:

1. Select a session or session broker in the **Navigator**. Its properties appear in the Editor.
2. Click the **Cache Coordination** tab. The Cache Coordination tab appears.
3. Ensure the **Enable Cache Coordination** option is selected, then select the appropriate coordinated cache **Type** (see Table 88–3). The cache coordination options appear on the tab.

Figure 88–1 Cache Coordination Tab, Synchronous Field



4. Select the **Synchronous** option to use synchronous change propagation. Do not select this option to use asynchronous change propagation.

Configuring a Service Channel

The **service channel** is the name of the TopLink coordinated cache channel to which sessions subscribe in order to participate in the same coordinated cache. Such sessions use the service channel to exchange messages with each other. Messages sent on other service channels will not be exchanged with this coordinated cache.

Table 88–4 summarizes which coordinated caches support service channel configuration.

Table 88–4 Coordinated Cache Support for Service Channel Configuration

Coordinated Cache	Using TopLink Workbench	Using Java
JMS Coordinated Cache		
RMI Coordinated Cache	✔	✔
CORBA Coordinated Cache	✔	✔
Custom Coordinated Cache	✔	✔

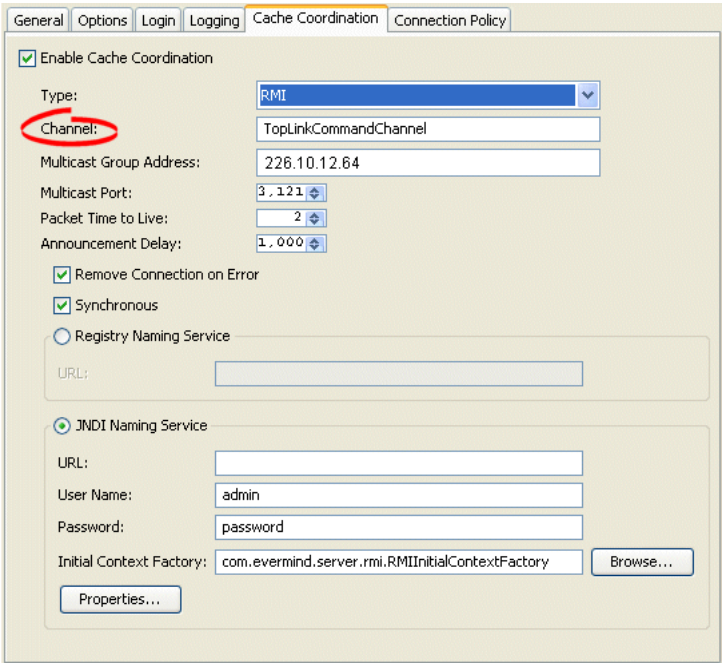
Using TopLink Workbench

To specify the service channel, use this procedure:

1. Select a session or session broker in the **Navigator**. Its properties appear in the Editor.
2. Click the **Cache Coordination** tab. The Cache Coordination tab appears.

3. Ensure the **Enable Cache Coordination** option is selected, then select the appropriate coordinated cache **Type** (see Table 88–4). The cache coordination options appear on the tab.

Figure 88–2 Cache Coordination Tab, Channel Field



4. In the **Channel** field, enter the name of the service channel for this coordinated cache.

Configuring a Multicast Group Address

A multicast group address is an Internet Protocol (IP) address in the range 224.0.0.0 to 239.255.255.255 that identifies the members of an IP multicast group. To efficiently broadcast the same message to all members of an IP multicast group, you configure each recipient with the same multicast group address and send the message to that address.

Table 88–5 summarizes which coordinated caches support multicast group address configuration.

Table 88–5 Coordinated Cache Support for Multicast Group Address Configuration

Coordinated Cache	Using TopLink Workbench	Using Java
JMS Coordinated Cache		
RMI Coordinated Cache	✔	✔
CORBA Coordinated Cache	✔	✔
Custom Coordinated Cache		

Note: Ensure your host and network are configured to support multicast operation before configuring this option.

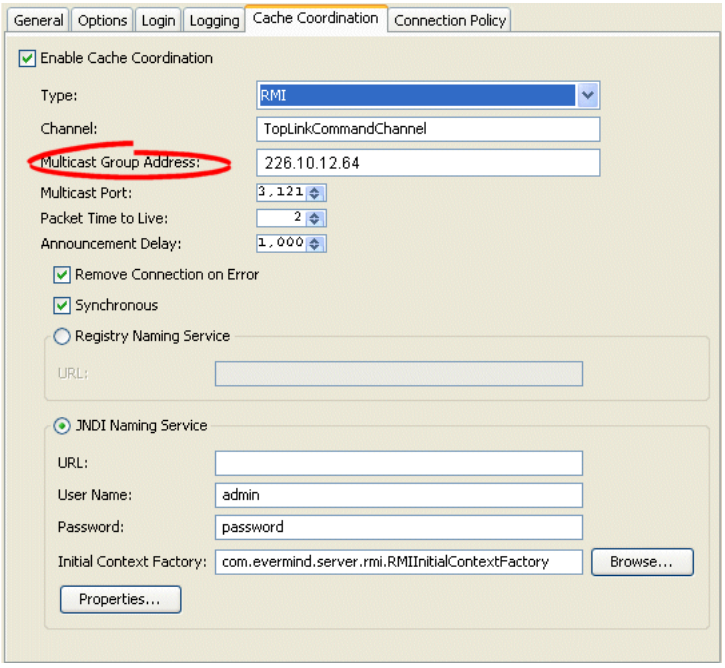
In addition to configuring the multicast group address, you must also configure the multicast port (see "Configuring a Multicast Port" on page 88-5) for the coordinated cache types shown in Table 88–5.

Using TopLink Workbench

To specify the multicast group address, use this procedure:

1. Select a session or session broker in the **Navigator**. Its properties appear in the Editor.
2. Click the **Cache Coordination** tab. The Cache Coordination tab appears.
3. Ensure the **Enable Cache Coordination** option is selected, then select the appropriate coordinated cache **Type** (see Table 88–5). The cache coordination options appear on the tab.

Figure 88–3 Cache Coordination Tab, Multicast Group Address Field



4. Enter the multicast group address in the range 224.0.0.0 to 239.255.255.255 to subscribe this session to a given coordinated cache.

Configuring a Multicast Port

The multicast port is the port on which multicast messages are received. Members of a multicast group (see "Configuring a Multicast Group Address" on page 88-4) rely on messages broadcast to their multicast group address to communicate with one another.

Table 88–6 summarizes which coordinated caches support multicast port configuration.

Table 88–6 Coordinated Cache Support for Multicast Port Configuration

Coordinated Cache	Using TopLink Workbench	Using Java
JMS Coordinated Cache		
RMI Coordinated Cache	✔	✔
CORBA Coordinated Cache	✔	✔
Custom Coordinated Cache		

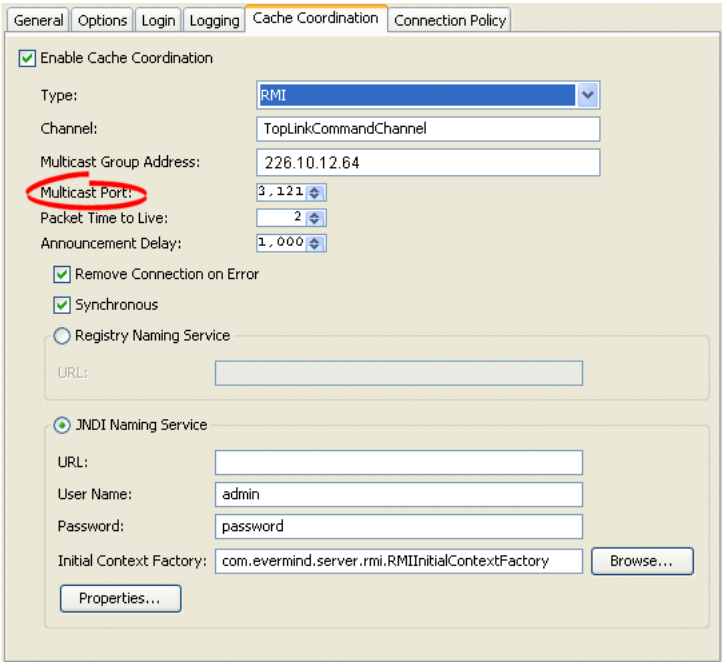
Note: Ensure your host and network are configured to support multicast operation before configuring this option

Using TopLink Workbench

To specify the multicast port, use this procedure:

1. Select a session or session broker in the **Navigator**. Its properties appear in the Editor.
2. Click the **Cache Coordination** tab. The Cache Coordination tab appears.
3. Ensure the **Enable Cache Coordination** option is selected, then select the appropriate coordinated cache **Type** (see Table 88–6). The cache coordination options appear on the tab.

Figure 88–4 Cache Coordination Tab, Multicast Port Field



4. Enter the multicast port on which messages broadcast to the multicast group address are received.

Configuring a Naming Service Type

The session's message transport service uses a naming service when it looks up connections to other sessions in the coordinated cache. You can configure the message transport service to look up remote objects using an RMI registry or Java Naming and Directory Interface (JNDI). By default, JNDI is used.

Table 88–7 summarizes which coordinated caches support naming service configuration.

Table 88–7 Coordinated Cache Support for Naming Service Configuration

Coordinated Cache	JNDI Naming Service	RMI Registry Naming Service
JMS Coordinated Cache	✔	
RMI Coordinated Cache	✔	✔
CORBA Coordinated Cache	✔	
Custom Coordinated Cache		

For information, see:

- "Configuring RMI Registry Naming Service Information" on page 88-9
- "Configuring JNDI Naming Service Information" on page 88-7

Configuring JNDI Naming Service Information

The session's message transport service uses a naming service when it looks up connections to other sessions in the coordinated cache. If you choose to use a JNDI naming service, you must configure JNDI naming service information.

Table 88–8 summarizes which coordinated caches support JNDI naming service configuration.

Table 88–8 Coordinated Cache Support for JNDI Naming Service Configuration

Coordinated Cache	Using TopLink Workbench	Using Java
JMS Coordinated Cache	✔	✔
RMI Coordinated Cache	✔	✔
CORBA Coordinated Cache	✔	✔
Custom Coordinated Cache		

TopLink uses JNDI naming service information differently, depending on the type of coordinated cache.

For a JMS coordinated cache, when a particular session's coordinated cache starts up, it uses its JNDI naming service information to locate and create a connection to the JMS server. The coordinated cache is ready when all participating sessions are connected to the JMS server. At this point, sessions can start sending and receiving object change messages. You can then configure all sessions that are participating in the same coordinated cache with the same JNDI naming service information.

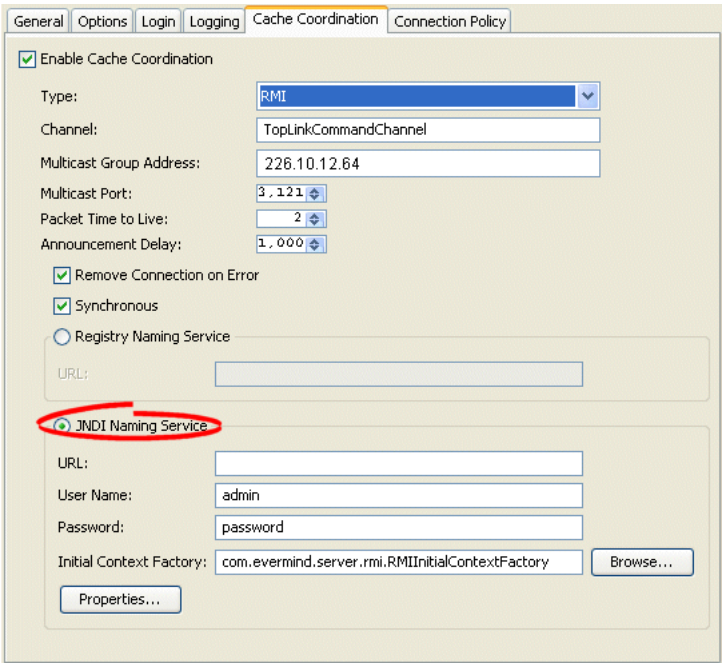
For an RMI or CORBA coordinated cache, when a particular session's coordinated cache starts up, the session binds its connection in JNDI, creates an announcement message (that includes its own JNDI naming service information), and broadcasts the announcement to its multicast group (see "Configuring a Multicast Group Address" on page 88-4 and "Configuring a Multicast Port" on page 88-5). When a session that belongs to the same multicast group receives this announcement, it uses the JNDI naming service information in the announcement message to establish bidirectional connections with the newly announced session's coordinated cache. The coordinated cache is ready when all participating sessions are interconnected in this way, at which point, sessions can start sending and receiving object change messages. You can then configure each session with JNDI naming information that identifies the host on which the session is deployed.

Using TopLink Workbench

To specify the sessions's JNDI naming service, use this procedure:

1. Select a session or session broker in the **Navigator**. Its properties appear in the Editor.
2. Click the **Cache Coordination** tab. The Cache Coordination tab appears.
3. Ensure the **Enable Cache Coordination** option is selected, then select the appropriate coordinated cache **Type** (see Table 88–8). The cache coordination options appear on the tab.

Figure 88–5 Cache Coordination Tab, JNDI Naming Service Options



Use the following information to enter data in the fields of the Cache Coordination tab to configure the naming service options:

Field	Description

URL	The location of the JNDI naming service.
	
	For a JMS coordinated cache: If you are using the Oracle Containers for J2EE (OC4J) JNDI naming service and all the hosts in your coordinated cache can communicate using the OC4J proprietary RMI protocol ORMI, use a URL similar to the following:
	
	`ormi://<JMS-host-IP>:<JMS-host-port>`
	
	where `JMS-host-IP` is the IP address of the host on which the JMS service provider is running and `JMS-host-port` is the port on which the JMS service provider is listening for JMS requests.
	
	For an RMI or CORBA coordinated cache: If you are using the OC4J JNDI naming service and all the hosts in your coordinated cache can communicate using the OC4J proprietary RMI protocol ORMI on OC4J default port 23791, use a URL similar to the following:
	
	`ormi://<session-host-IP>:23791`
	
	where `session-host-IP` is the IP address of the host on which this session is deployed.

Username	The user name required to log in to the JNDI naming service. The value you enter defines the `Context.SECURITY_PRINCIPAL` environment property.
Password	The plain text (unencrypted) password required to log in to the JNDI naming service. The password appears in plain text in TopLink Workbench, but it is encrypted when written to the `sessions.xml` file The value you enter defines the `Context.SECURITY_CREDENTIALS` environment property.
Initial Context Factory	The name of the factory class, provided by your JNDI naming service provider, that implements the `javax.naming.spi.InitialContextFactory` interface. This factory class is used to create a `javax.naming.Context` instance that can access the JNDI naming service provider's context implementation. The value you enter defines the `Context.INITIAL_CONTEXT_FACTORY` environment property.
Properties	The JNDI context properties. Click **Properties** to configure custom JNDI context properties (see "Configuring Context Properties" on page 88-14).

Configuring RMI Registry Naming Service Information

The session's message transport service uses a naming service when it looks up connections to other sessions in the coordinated cache. If you choose to use an RMI registry naming service, you can configure RMI registry naming service options.

Table 88–8 summarizes which coordinated caches support RMI registry naming service configuration.

Table 88–9 Coordinated Cache Support for RMI Registry Naming Service Configuration

Coordinated Cache	Using TopLink Workbench	Using Java
JMS Coordinated Cache		
RMI Coordinated Cache	✔	✔
CORBA Coordinated Cache		
Custom Coordinated Cache		

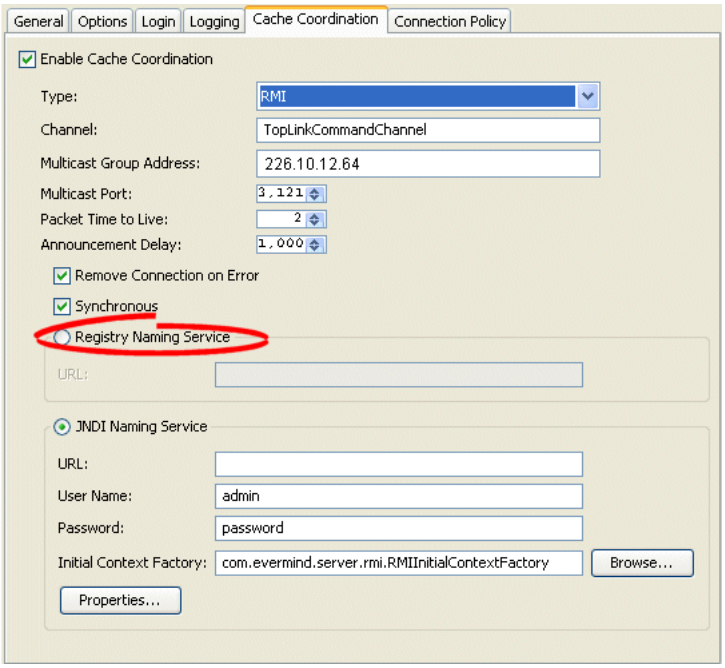
For an RMI coordinated cache, when a particular session's coordinated cache starts up, the session binds its connection in its RMI registry, creates an announcement message (that includes its own naming service information), and broadcasts the announcement to its multicast group (see "Configuring a Multicast Group Address" on page 88-4 and "Configuring a Multicast Port" on page 88-5). When a session that belongs to the same multicast group receives this announcement, it uses the JNDI naming service information in the announcement message to establish bidirectional connections with the newly announced session's coordinated cache. The coordinated cache is ready when all participating sessions are interconnected in this way, at which point, sessions can start sending and receiving object change messages. You can then configure each session with RMI registry naming information that identifies the host on which the session is deployed.

Using TopLink Workbench

To specify the sessions's registry naming service, use this procedure:

1. Select a session or session broker in the **Navigator**. Its properties appear in the Editor window.
2. Click the **Cache Coordination** tab. The Cache Coordination tab appears.
3. Ensure the **Enable Cache Coordination** option is selected, then select the appropriate coordinated cache **Type** (see Table 88–9). The cache coordination options appear on the tab.

Figure 88–6 Cache Coordination Tab, Naming Service Options



Use the following information to configure the naming service options:

Field	Description
URL	Assuming that you are using the OC4J JNDI naming service and that all the hosts in your coordinated cache can communicate using the OC4J proprietary RMI protocol ORMI on OC4J default port 23791, use a URL similar to the following: `ormi://<session-host-IP>:23791` where `session-host-IP` is the IP address of the host on which this session is deployed.

Configuring an Announcement Delay

Use the announcement delay option to set the amount of time (in milliseconds) that a session should wait between the time that it is available and the time that it broadcasts its announcement message to the members of the coordinated cache. This additional delay may be necessary to give some systems more time to post their connections into the naming service (see "Configuring a Naming Service Type" on page 88-7).

Table 88–10 summarizes which coordinated caches support announcement delay configuration.

Table 88–10 Coordinated Cache Support for Announcement Delay Configuration

Coordinated Cache	Using TopLink Workbench	Using Java
JMS Coordinated Cache		
RMI Coordinated Cache	✔	✔

Table 88–10 (Cont.) Coordinated Cache Support for Announcement Delay Configuration

Coordinated Cache	Using TopLink Workbench	Using Java
CORBA Coordinated Cache	✔	✔
Custom Coordinated Cache		

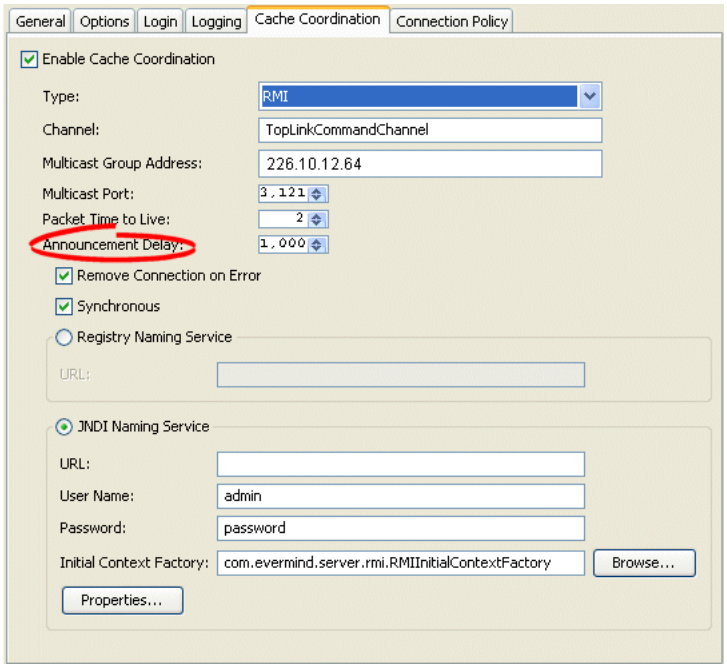
In addition to announcement delay, you may also need to consider packet time-to-live configuration (see "Configuring a Packet Time-to-Live" on page 88-15).

Using TopLink Workbench

To specify the announcement delay (in milliseconds) for an RMI coordinated cache, use this procedure:

1. Select a session or session broker in the **Navigator**. Its properties appear in the Editor.
2. Click the **Cache Coordination** tab. The Cache Coordination tab appears.
3. Ensure the **Enable Cache Coordination** option is selected, then select the appropriate coordinated cache **Type** (see Table 88–10). The cache coordination options appear on the tab.

Figure 88–7 Cache Coordination Tab, Announcement Delay Field



4. Select the amount of time (in milliseconds) that this session should wait between the time that it is available and the time that it broadcasts its announcement message to the members of the coordinated cache.

Configuring Connection Handling

The session's transport manager creates connections to the various members of the coordinated cache. If a communication error occurs on one of these connections, you can configure the session to either ignore the error or remove the connection.

Table 88–11 summarizes which coordinated caches support connection handling configuration.

Table 88–11 Coordinated Cache Support for Connection Handling Configuration

Coordinated Cache	Using TopLink Workbench	Using Java
JMS Coordinated Cache	✔	✔
RMI Coordinated Cache	✔	✔
CORBA Coordinated Cache	✔	✔
Custom Coordinated Cache	✔	✔

If you configure the session to remove the connection on error, the next time the session tries to communicate with that coordinated cache member, it will construct a new connection.

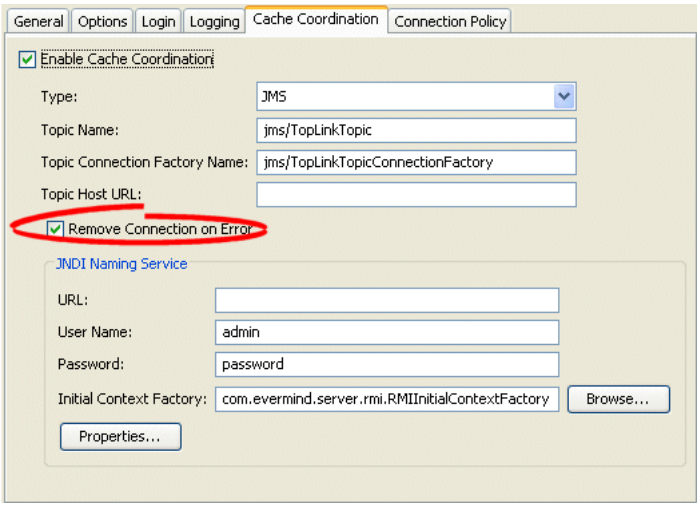
If you configure the session to ignore the error, the next time the session tries to communicate with that coordinated cache member, it will continue to use the same connection.

Using TopLink Workbench

To specify how TopLink handles session connections in the event of an error, use this procedure:

1. Select a session or session broker in the **Navigator**. Its properties appear in the Editor.
2. Click the **Cache Coordination** tab. The Cache Coordination tab appears.
3. Ensure the **Enable Cache Coordination** option is selected, then select the appropriate coordinated cache **Type** (see Table 88–11). The cache coordination options appear on the tab.

Figure 88–8 Cache Coordination Tab, Remove Connection on Error Option



4. Select the **Remove Connection on Error** option to configure the session to remove the data source connection in the event of an error.

Configuring Context Properties

When you configure a coordinated cache to use a JNDI naming service (see "Configuring a Naming Service Type" on page 88-7), you can add new environment properties to the environment of the initial JNDI context.

Table 88–12 summarizes which coordinated caches support context properties.

Table 88–12 Coordinated Cache Support for Context Properties

Coordinated Cache	Using TopLink Workbench	Using Java
JMS Coordinated Cache	✔	✔
RMI Coordinated Cache[1]	✔	✔
CORBA Coordinated Cache[1]	✔	✔
Custom Coordinated Cache		

[1] When JNDI naming service is selected (see "Configuring a Naming Service Type" on page 88-7).

Using TopLink Workbench, TopLink uses the new environment properties you add to create the initial context for both local and remote JNDI access.

Using Java, you can configure different properties for local and remote JNDI access using a session customizer class to call `TransportManager` methods `setLocalContextProperties` and `setRemoteContectProperties` (for more information, see "Configuring Customizer Class" on page 74-13).

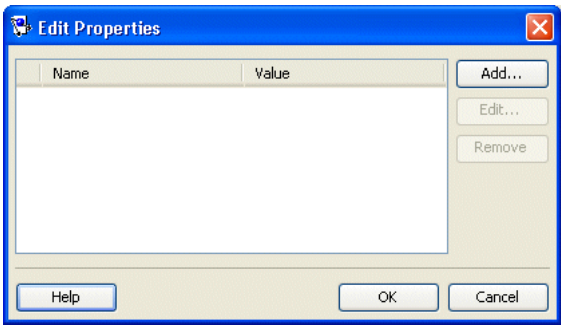
Using TopLink Workbench

To define JNDI context properties, use this procedure:

1. Select a session or session broker in the **Navigator**. Its properties appear in the Editor.

2. Ensure the **Enable Cache Coordination** option is selected, then select the appropriate coordinated cache **Type** (see Table 88–12). The cache coordination options appear on the tab.
3. Ensure the **JNDI Naming Service** option is selected. See "Configuring a Naming Service Type" on page 88-7.
4. In the JNDI Naming Service area, click **Properties**. The Edit Properties dialog box appears.

Figure 88–9 Edit Properties Dialog Box



5. Click **Add** to create a new property. The Add New Property dialog box appears.

   To change (or delete) an existing property, select the property and click **Edit** (or **Remove**).

Use this table to enter data in the following fields on the dialog box.

Field	Description
Name	The name of the property.
Value	The value of the property.

Configuring a Packet Time-to-Live

The **packet time-to-live** is the number of hops that session data **packets** can take before expiring. The default is 2. This allows for a **hub** and an interface card, and prevents the data packets from leaving the local network. If sessions are hosted on different local area networks (LANs) that are part of wide area network (WAN), or if a firewall configuration prevents it, the announcement sent by one session may not reach the other sessions in the coordinated cache. In this case, consult your network administrator for the correct time-to-live value.

Table 88–13 summarizes which coordinated caches support packet time-to-live configuration.

Table 88–13 Coordinated Cache Support for Packet Time-to-Live Configuration

Coordinated Cache	Using TopLink Workbench	Using Java
JMS Coordinated Cache		
RMI Coordinated Cache	✔	✔

Table 88–13 (Cont.) Coordinated Cache Support for Packet Time-to-Live Configuration

Coordinated Cache	Using TopLink Workbench	Using Java
CORBA Coordinated Cache	✔	✔
Custom Coordinated Cache		

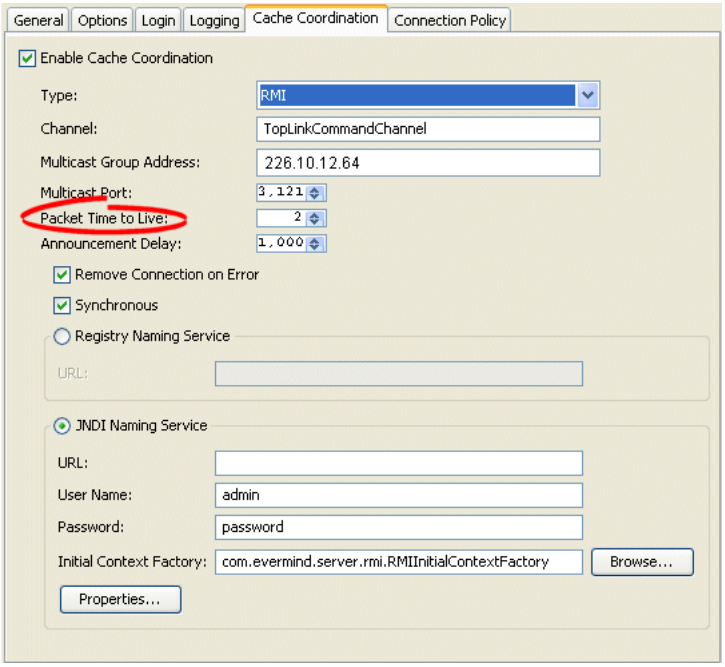
In addition to configuring packet time-to-live, you may also need to configure announcement delay (see "Configuring an Announcement Delay" on page 88-11).

Using TopLink Workbench

To specify the number of hops that session data packets can take before expiring, use this procedure:

1. Select a session or session broker in the **Navigator**. Its properties appear in the Editor.
2. Ensure the **Enable Cache Coordination** option is selected, then select the appropriate coordinated cache **Type** (see Table 88–12). The cache coordination options appear on the tab.

Figure 88–10 Cache Coordination Tab, Packet Time to Live Field



In the **Packet Time to Live** field, specify the number of hops (default = 2) that session data packets can take before expiring.

89

Configuring a JMS Coordinated Cache

This chapter describes the various components that you must configure in order to use a JMS coordinated cache.

JMS Coordinated Cache Configuration Overview

Table 89–1 lists the configurable options for a JMS coordinated cache.

Table 89–1 Configurable Options for a JMS Coordinated Cache

Option	Type	TopLink Workbench	Java
"Configuring Cache Coordination Change Propagation at the Descriptor Level" on page 25-40	Basic	✔	✔
"Configuring the Synchronous Change Propagation Mode" on page 88-2	Basic	✔	✔
"Configuring JNDI Naming Service Information" on page 88-7	Basic	✔	✔
"Configuring a Topic Name" on page 89-1	Basic	✔	✔
"Configuring a Topic Connection Factory Name" on page 89-2	Basic	✔	✔
"Configuring a Topic Host URL" on page 89-2	Advanced	✔	✔
"Configuring Connection Handling" on page 88-13	Advanced	✔	✔
"Configuring Context Properties" on page 88-14	Advanced	✔	✔
"Configuring a Packet Time-to-Live" on page 88-15	Advanced		✔

Configuring a Topic Name

A JMS topic identifies a publish/subscribe destination for a JMS server. JMS users who wish to share messages subscribe to the same JMS topic.

The topic name you configure is the name that TopLink uses to look up the `javax.jms.Topic` instance from the JNDI service. You must provide a fully qualified JNDI name, such as `jms/<topic_name>`.

All the members of the same JMS coordinated cache must use the same JMS topic.

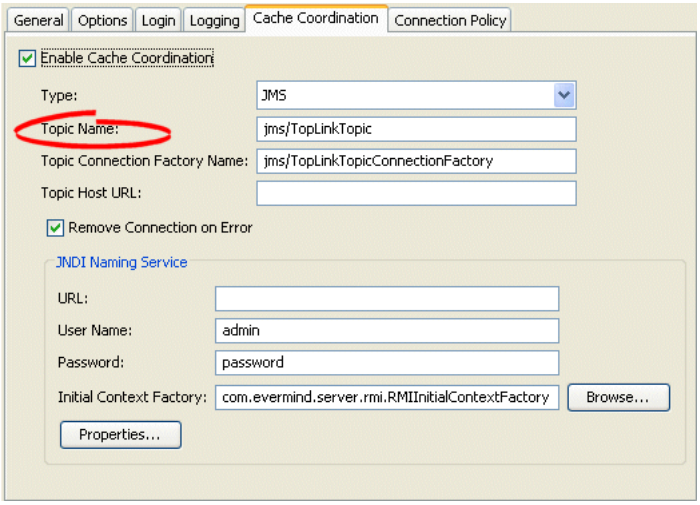
Using TopLink Workbench

To specify the topic name for JMS cache coordination, use this procedure:

1. Select a server session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Cache Coordination** tab. The Cache Coordination tab appears.

3. Ensure **Enable Cache Coordination** is selected and the **Type** is **JMS** (see "Understanding Cache Coordination" on page 87-10 for more information).

Figure 89–1 Cache Coordination Tab, Topic Name Field, JMS



4. Enter the topic name to use with the JMS coordinated cache for this session. This must be a fully qualified JNDI name, such as `jms/<topic_name>`.

Configuring a Topic Connection Factory Name

A JMS topic connection factory creates connections with the JMS provider for a specific JMS destination. Each connection factory contains the specific configuration information to create a connection to a JMS destination.

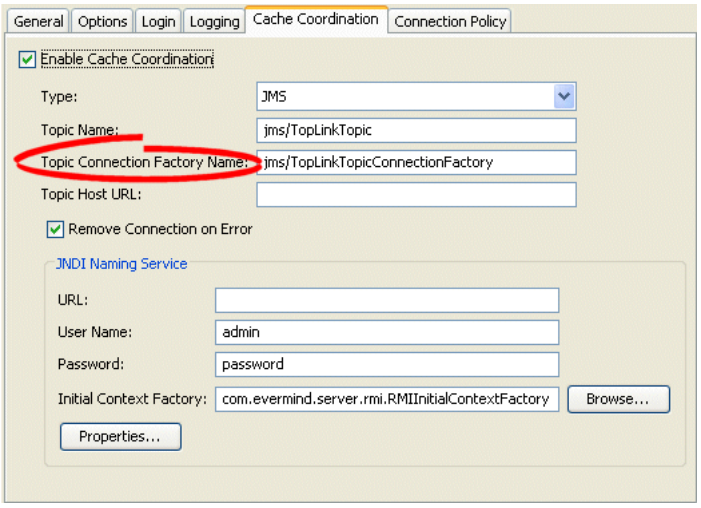
The topic connection factory name you configure is the name that TopLink uses to look up the `javax.jms.TopicConnectionFactory` instance from the JNDI service. This must be a fully qualified JNDI name, such as `jms/<resource_name>`.

Using TopLink Workbench

To specify the topic connection factory for a JMS coordinated cache, use this procedure:

1. Select a server session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Cache Coordination** tab. The Cache Coordination tab appears.
3. Ensure **Enable Cache Coordination** is selected and the **Type** is **JMS** (see "Understanding Cache Coordination" on page 87-10 for more information).

Figure 89–2 Cache Coordination Tab, Topic Connection Factory Name Field



4. Enter the topic connection factory name to use with the JMS coordinated cache for this session. This must be a fully qualified JNDI name, such as `jms/<resource_name>`.

Configuring a Topic Host URL

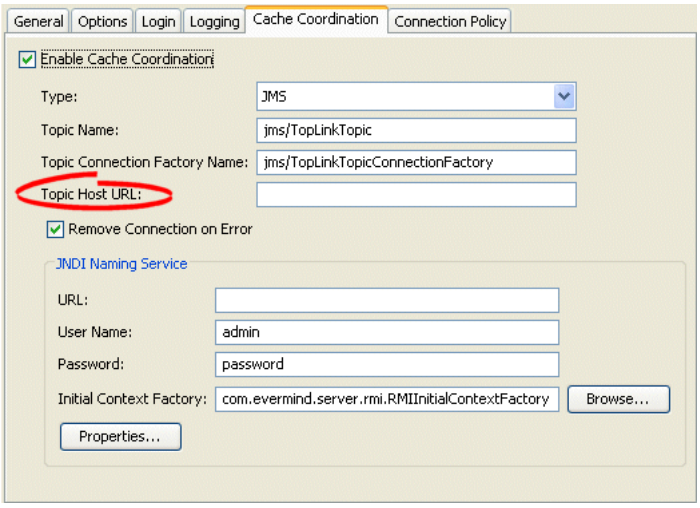
The JMS topic host URL is the URL of the machine on the network that hosts the JMS topic (see "Configuring a Topic Name" on page 89-1).

Using TopLink Workbench

To specify the topic host URL for a JMS coordinated cache, use this procedure:

1. Select a server session in the **Navigator**. Its properties appear in the Editor.
2. Click the **Cache Coordination** tab. The Cache Coordination tab appears.
3. Ensure **Enable Cache Coordination** is selected and the **Type** is **JMS** (see "Understanding Cache Coordination" on page 87-10 for more information).

Figure 89–3 Cache Coordination Tab, Topic Host URL Field



Enter the URL of the machine on the network that hosts the JMS topic (see "Configuring a Topic Name" on page 89-1) to use with the JMS coordinated cache for this session.

90

Configuring an RMI Coordinated Cache

This chapter describes the various components that you must configure in order to use an RMI coordinated cache.

RMI Coordinated Cache Configuration Overview

Table 90–1 lists the configurable options for an RMI coordinated cache.

Table 90–1 Configurable Options for an RMI Coordinated Cache

Option	Type	TopLink Workbench	Java
"Configuring Cache Coordination Change Propagation at the Descriptor Level" on page 25-40	Basic	✔	✔
"Configuring the Synchronous Change Propagation Mode" on page 88-2	Basic	✔	✔
"Configuring a Service Channel" on page 88-3	Basic	✔	✔
"Configuring a Multicast Group Address" on page 88-4	Basic	✔	✔
"Configuring a Multicast Port" on page 88-5	Basic	✔	✔
"Configuring a Naming Service Type" on page 88-7	Basic	✔	✔
"Configuring an Announcement Delay" on page 88-11	Basic	✔	✔
"Configuring Connection Handling" on page 88-13	Advanced	✔	✔
"Configuring Context Properties" on page 88-14	Advanced	✔	✔
"Configuring a Packet Time-to-Live" on page 88-15	Advanced	✔	✔

91

Configuring a CORBA Coordinated Cache

This chapter describes the various components that you must configure to use a CORBA coordinated cache.

CORBA Coordinated Cache Configuration Overview

Table 91–1 lists the configurable options for a CORBA coordinated cache.

Table 91–1 Configurable Options for a CORBA Coordinated Cache

Option	Type	TopLink Workbench	Java
"Configuring Cache Coordination Change Propagation at the Descriptor Level" on page 25-40	Basic	✔	✔
"Configuring the Synchronous Change Propagation Mode" on page 88-2	Basic	✔	✔
"Configuring a Service Channel" on page 88-3	Basic	✔	✔
"Configuring a Multicast Group Address" on page 88-4	Basic	✔	✔
"Configuring a Multicast Port" on page 88-5	Basic	✔	✔
"Configuring a Naming Service Type" on page 88-7	Basic	✔	✔
"Configuring an Announcement Delay" on page 88-11	Basic	✔	✔
"Configuring Connection Handling" on page 88-13	Advanced	✔	✔
"Configuring Context Properties" on page 88-14	Advanced	✔	✔
"Configuring a Packet Time-to-Live" on page 88-15	Advanced		✔

92

Configuring a Custom Coordinated Cache

This chapter describes the various components that you must configure to use a custom, user-defined coordinated cache. For more information, see "Custom Coordinated Cache" on page 87-13.

Custom Coordinated Cache Configuration Overview

Table 92–1 lists the configurable options for a custom coordinated cache.

Table 92–1 Configurable Options for a Custom Coordinated Cache

Option	Type	Using TopLink Workbench	Java
"Configuring Cache Coordination Change Propagation at the Descriptor Level" on page 25-40	Basic	✔	✔
"Configuring a Service Channel" on page 88-3	Basic	✔	✔
"Configuring Transport Class" on page 92-1	Basic	✔	✔
"Configuring Connection Handling" on page 88-13	Advanced	✔	✔

Configuring Transport Class

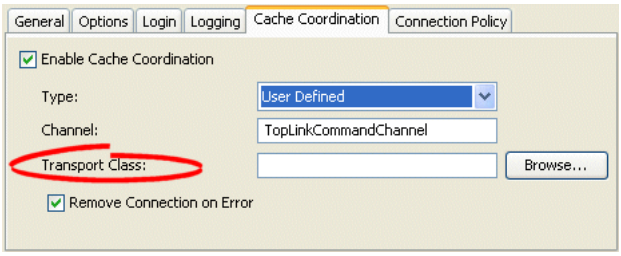
To configure a custom coordinated cache, you must specify your custom instance of `oracle.toplink.remotecommand.TransportManager`.

Using TopLink Workbench

To select the transport class for the user defined coordinated cache, use this procedure:

1. Select a session or session broker in the **Navigator**. Its properties appear in the Editor.
2. Click the **Cache Coordination** tab. The Cache Coordination tab appears.
3. Ensure the **Enable Cache Coordination** option is selected and the **Type** is **User Defined** (see "Understanding Cache Coordination" on page 87-10).

Figure 92–1 Cache Coordination, Transport Class Option



4. Click **Browse** and select the transport class for the user-defined coordinated cache.

Part XVIII

Queries

This part describes building TopLink queries and using them to create, read, update, and delete objects. It contains the following chapters.:

- Chapter 93, "Understanding TopLink Queries"

  This chapter describes each of the different TopLink query types and important query concepts.

- Chapter 94, "Using Basic Query API"

  This chapter explains how to use basic TopLink query options.

- Chapter 95, "Understanding TopLink Expressions"

  This chapter describes the TopLink expressions framework and how to use it with TopLink queries.

- Chapter 96, "Using Advanced Query API"

  This chapter explains how to use advanced TopLink query options.

93

Understanding TopLink Queries

TopLink enables you to create, read, update, and delete persistent objects or data using queries in both J2EE and non-J2EE applications for both relational and nonrelational data sources.

This chapter explains the following:

- Query Types
- Query Concepts
- Building Queries
- Executing Queries
- Handling Query Results
- Queries and the Cache
- Understanding the Query API

Query Types

Table 93–1 lists the query types that you can build in TopLink, and classifies them as basic or advanced.

Table 93–1 TopLink Query Types

Query Type	Description	Type	TopLink Workbench	Java
Session Queries	A query implicitly constructed and executed by a `Session` based on input parameters used to perform the most common data source actions on objects.	Basic		✔
Database Queries	A query also known as a *query object query*. An instance of `DatabaseQuery` that you create and then execute to perform any data source action on either objects or data. You can further refine a `DatabaseQuery` by also creating and configuring its `Call` (see "Call Queries" on page 93-17).	Basic		✔
Named Queries	An instance of `DatabaseQuery` stored by name in a `Session` or a descriptor's `DescriptorQueryManager` where it is constructed and prepared once. Such a query can then be repeatedly executed by name.	Basic	✔	✔
Call Queries	An instance of `Call` that you create and then either execute directly, using a special `Session` API to perform limited data source actions on data only, or execute indirectly in the context of a `DatabaseQuery`. TopLink supports `Call` instances for custom SQL, stored procedures, and EIS interactions.	Basic	✔	✔

Table 93–1 (Cont.) TopLink Query Types

Query Type	Description	Type	TopLink Workbench	Java
Redirect Queries	An instance of `MethodBasedQueryRedirector` (taking the name of a static method and the `Class` in which it is defined as parameters) set on a named query. When the query is executed, the static method is invoked.	Advanced		✔
Historical Queries	Any query executed in the context of a historical session using the time-aware features of the TopLink `Expression` framework.	Advanced		✔
Interface and Inheritance Queries	Any query that references an interface type or super and subclasses of an inheritance hierarchy.	Advanced		✔
Descriptor Query Manager Queries	The `DescriptorQueryManager` defines a default `DatabaseQuery` for each basic data source operation (create, read, update, and delete), and provides an API with which you can customize either the `DatabaseQuery` or its `Call`.	Advanced	✔	✔
EJB Finders	A query defined on the home interface of an enterprise bean that returns enterprise beans. You can implement finders using any TopLink query type, including `EJBQLCall`, a `Call` that takes EJB QL.	Advanced	✔	✔

For more information, see the following:

- "Using Basic Query API" on page 94-1
- "Using Advanced Query API" on page 96-1
- "Query Concepts" on page 93-2

Query Concepts

In general, querying a data source means performing an action on or interacting with the contents of the data source. To do this, you must be able to perform the following:

- Define an action in a syntax native to the data source being queried
- Apply the action in a controlled fashion
- Manage the results returned by the action (if any)

Specific to TopLink, you must also consider how the query affects the TopLink cache. For more information, see "Queries and the Cache" on page 93-29.

This section introduces query concepts unique to TopLink, including the following:

- Call
- DatabaseQuery
- Data-Level and Object-Level Queries
- Summary Queries
- Descriptor Query Manager
- TopLink Expressions
- Query Keys
- Query Languages

Call

In TopLink, the `Call` object encapsulates an operation or action on a data source. TopLink provides a variety of `Call` types such as structured query language (SQL), Enterprise Java Beans Query Language (EJB QL), Extensible Markup Language (XML), and enterprise information system (EIS).

You can execute a `Call` directly or in the context of a `DatabaseQuery`.

DatabaseQuery

A `DatabaseQuery` object is an abstraction that associates additional customization and optimization options with the action encapsulated by a `Call`. By separating these options from the `Call`, TopLink can provide sophisticated query capabilities across all `Call` types.

For more information, see "Database Queries" on page 93-10.

Data-Level and Object-Level Queries

In TopLink, queries can be defined for objects or data.

- **Object-level** queries (see "Object-Level Read Query" on page 93-11 and "Object-Level Modify Query" on page 93-14) are object-specific and return data as objects in your domain model. They are the preferred type of query for mapped data. By far, object-level `DatabaseQuery` queries are the most common query used in TopLink.
- **Data-level** queries (see "Data-Level Read Query" on page 93-13 and "Data-Level Modify Query" on page 93-15) are used to query database tables directly, and are an appropriate way to work with unmapped data, such as foreign keys and object version fields.

Summary Queries

While data-level queries return raw data and object-level queries return objects in your domain model, summary queries return data about objects. TopLink provides partial object queries (see "Partial Object Queries" on page 93-11) to return a set of objects with only specific attributes populated, and report queries (see "Report Query" on page 93-15) to return summarized (or rolled-up) data for specific attributes of a set of objects.

Descriptor Query Manager

In addition to storing named queries applicable to a particular class (see "Named Queries" on page 93-16), you can also use the `DescriptorQueryManager` to override the default action that TopLink defines for common data source operations. For more information, see "Descriptor Query Manager Queries" on page 93-23.

TopLink Expressions

TopLink expressions let you specify query search criteria based on your domain object model. When you execute the query, TopLink translates these search criteria into the appropriate query language for your platform.

TopLink provides two public classes to support expressions:

- The `Expression` class represents an expression that can be anything from a simple constant to a complex clause with boolean logic. You can manipulate, group, and integrate expressions.
- The `ExpressionBuilder` class is the factory for constructing new expressions.

You can specify a selection criterion as an `Expression` with `DatabaseQuery` method `setSelectionCriteria` (see "Database Queries" on page 93-10), and in a finder that takes an `Expression` (see "Expression Finders" on page 93-27).

For more information about using TopLink expressions, see "Understanding TopLink Expressions" on page 95-1.

Query Keys

A query key is a schema-independent alias for a database field name. Using a query key, you can refer to a field using a schema-independent alias. In relational projects only, TopLink automatically creates query keys for all mapped attributes. The name of the query key is the name of the class attribute specified in your object model.

You can configure query keys in a class descriptor (see "Configuring Query Keys" on page 25-30) or interface descriptor (see "Configuring Interface Query Keys" on page 25-33).

You can use query keys in expressions (see "Query Keys and Expressions" on page 95-10) and to query variable one-to-one mappings (see "Using Queries on Variable One-to-One Mappings" on page 96-5).

Query Languages

Using TopLink, you can express a query using any of the following query languages:

- SQL Queries
- EJB QL Queries
- XML Queries
- EIS Interactions
- Query-by-Example

In most cases, you can compose a query directly in a given query language or, preferably, you can construct a `DatabaseQuery` with an appropriate `Call` and specify selection criteria using a TopLink `Expression`. Although composing a query directly appears to be the simplest approach (and for simple operations or operations on unmapped data, it is), using the `DatabaseQuery` approach offers the compelling advantage of confining your query to your domain object model and avoiding dependence on data source schema implementation details. Oracle recommends that you compose your queries using `DatabaseQuery`, `Call`, and `Expression`.

SQL Queries

SQL is the most common query language for applications that use a relational database data source.

You can execute custom SQL directly using `Session` methods `executeSelectingCall` and `executeNonSelectingCall`, or you can construct a `DatabaseQuery` with an appropriate `Call`.

TopLink provides a variety of SQL `Call` objects for use with stored procedures and, with Oracle databases, stored functions. For more information, see "SQL Calls" on page 93-17.

EJB QL Queries

Like SQL, EJB QL is a query language; but unlike SQL, it presents queries from an object model perspective, allowing you to declare queries using the attributes of each abstract entity bean in your object model. It also includes path expressions that enable navigation over the relationships defined between entity beans and dependent objects.

Using EJB QL offers the following advantages:

- You do not need to know the database structure (such as tables and fields).
- You can construct queries using the attributes of the entity beans instead of using database tables and fields.
- You can use relationships in a query to provide navigation from attribute to attribute.
- EJB QL queries are portable because they are database-independent.
- You can specify the reference class in the `SELECT` clause.

The disadvantage of EJB QL queries is that it is difficult to use when you construct complex queries.

TopLink provides the full support for the EJB QL specification.

> **Note:** TopLink supports the `LOCATE` string function and will generate the correct SQL with it. However, not all data sources support `LOCATE`. Before using the `LOCATE` string function, consult your data source documentation.

EJB QL is the standard query language first defined in the EJB 2.0 specification. Consequently, TopLink lets you specify selection criteria using EJB QL in an EJB finder (see "EJB QL Finders" on page 93-27).

Although EJB QL is usually associated with EJB, TopLink also lets you specify selection criteria using EJB QL in queries for regular Java objects as well. TopLink provides an EJB QL `Call` that you can execute directly or in the context of a `DatabaseQuery`. For more information, see "EJB QL Calls" on page 93-19 and "DatabaseQuery" on page 93-3.

XML Queries

You can use TopLink XML to query XML data stored in an Oracle Database XMLType field. For more information, see "Direct-to-XMLType Mapping" on page 33-4 and "XMLType Functions" on page 95-5.

EIS Interactions

When you execute a TopLink query using an EIS `Call` (see "Enterprise Information System (EIS) Interactions" on page 93-19), TopLink converts your selection criteria into an XML format appropriate for your J2C adapter.

If supported by your J2C adapter, you can use the XQuery language by executing an XQuery interaction (see "XQueryInteraction" on page 93-20) either directly or in the context of a `DatabaseQuery`.

Query-by-Example

Query-by-example is a simple and intuitive way to express a query. To specify a query-by-example, provide a sample instance of the persistent object to query, and set appropriate values on only the data members on which you wish to query.

You can use a constructor with a reference class argument to create a sample instance or example object. Alternatively, you can use a combination of any other type of constructor and a `setReferenceClass` method of your query object. If you fail to specify the reference class, a `QueryException` will be thrown.

Query-by-example lets you query for an object based on any attribute that uses a direct mapping or a one-to-one relationship (including those with nesting).

Note: Query-by-example does not support any other relationship mapping types, nor does it support EJB beans.

Set only the attributes on which you base the query; set all other attributes to `null`. By default, TopLink ignores attributes in the sample instance that contain `null`, zero (0), empty strings, and `FALSE`. You can modify this list of values (and define other query by example options) by specifying a `QueryByExamplePolicy` (see "Defining a QueryByExamplePolicy" on page 94-7).

Query-by-example uses the `AND` operator to tie the attribute comparisons together. By default, attribute values in the sample instance are compared against corresponding values of candidate objects using `EQUALS` operator. You can modify this behaviour using the `QueryByExamplePolicy`.

Both `ReadAllQuery` and `ReadObjectQuery` provide a `setExampleObject` method and `setQueryByExamplePolicy` method that you can use to specify selection criteria based on an example object instance.

For more information and examples, see "Reading Objects Using Query-By-Example" on page 94-6.

Building Queries

You can build queries using TopLink Workbench or Java, using the TopLink API.

Some queries are implicitly constructed for you based on passed in arguments and executed in one step (for example, session queries, as described in "Session Queries" on page 93-9) and others you explicitly create, configure, and then execute (for example, "Database Queries" on page 93-10).

For more information, see the following:

- "Using Basic Query API" on page 94-1
- "Using Advanced Query API" on page 96-1

Executing Queries

In TopLink, you execute most queries using the `Session` API summarized in Table 93–2.

Table 93–2 Session Methods for Executing a Query

Query Type	Session Method	Advantages and Disadvantages

Session Queries	`readObject`
	`readAllObjects`
	`writeObject`
	`writeAllObjects`
	`deleteObject`
	`deleteAllObjects`
	`insertObject`
	`updateObject`
		Advantages: the most convenient way to perform common data source operations on objects.
		Disadvantages: less control over query execution and results; less efficient for frequently executed queries.

Database Queries Named Queries Redirect Queries	`executeQuery`	Advantages: greatest configuration and execution flexibility; can take advantage of named queries for efficiency. Disadvantages: you must explicitly create and configure `DatabaseQuery` and possibly `Call` objects.
Call Queries	`executeSelectingCall` `executeNonSelectingCall`	Advantages: convenient way to directly apply an action to unmapped data. Disadvantages: least control over query execution and results; your application must do more work to handle raw data results.

Note: Oracle recommends that you perform all data source operations using a unit of work: doing so is the most efficient way to manage transactions, concurrency, and referential constraints. For more information, see "Understanding TopLink Transactions" on page 97-1.

Alternatively, you can execute queries outside of a unit of work using a session API directly, but doing so places greater responsibility on your application to manage transactions, concurrency, and referential constraints.

TopLink executes `DescriptorQueryManager` queries when you execute a session query. For more information, see "Descriptor Query Manager Queries" on page 93-23.

You execute EJB finders when you call the appropriate finder method on an EJB. For more information, see "EJB Finders" on page 93-24.

WARNING: Allowing an unverified SQL string to be passed into methods (for example: `setSQLString(String sql)`, `readAllObjects(Class class, String sql)` methods) makes your application vulnerable to SQL injection attacks.

For more information, see the following:

- "Using Basic Query API" on page 94-1
- "Using Advanced Query API" on page 96-1

Handling Query Results

TopLink queries generally return Java objects as their result set. TopLink queries can return any of the following:

- Entire objects, with all attributes populated and the object reflected in the cache.
- Collections of objects (see "Collection Query Results" on page 93-8).
- Partial objects, with only the attributes you specify populated and without the object reflected in the cache (see "Report Query Results").
- Streams of objects (see "Stream and Cursor Query Results" on page 93-8).
- EJB (see "EJB Finders" on page 93-24).

Collection Query Results

A collection is a group of Java objects contained by an instance of `Collection` or `Map`

By default, queries that return more than one object return their results in a `Vector`.

You can configure TopLink to return query results in any concrete instance of `Collection` or `Map`.

Collection results are supported by all TopLink query types.

For information and examples on how to configure and handle collection query results, see "Handling Collection Query Results" on page 94-25.

Report Query Results

A `ReportQuery` (a type of partial object query) returns summary data for selected objects using the database reporting functions and features supported by your platform. Although the report query returns data (not objects), it does enable you to query the returned data and specify it at the object level.

By default, a `ReportQuery` returns a collection (see "Collection Query Results" on page 93-8) of `ReportQueryResult` objects, one collection per database row returned. You can use the `ReportQuery` API to configure how a `ReportQuery` returns its results. For more information see "Handling Report Query Results" on page 94-26.

For more information, see the following:

- "Report Query" on page 93-15
- "Handling Report Query Results" on page 94-26
- "Partial Object Queries" on page 93-11.

Stream and Cursor Query Results

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.

Large result sets can be resource-intensive to collect and process. To improve performance and give the client more control over the returned results, configure TopLink queries to use a cursor or stream.

Cursors & streams are supported by all subclasses of `DataReadQuery` and `ReadAllQuery`.

For more information, see "Handling Cursor and Stream Query Results" on page 96-15.

Session Queries

Sessions provide query methods that lets you perform the object operations listed in Table 93–3.

Table 93–3 Session Object Query Summary

Session Type	Create	Read	Update	Delete
`UnitOfWork`	NA	`readObject` `readAllObjects`	NA	`deleteObject` `deleteAllObjects`
`Server`	NA	NA	NA	NA
`ClientSession`	NA	`readObject` `readAllObjects`	NA	NA

`DatabaseSession`	`insertObject`	`readObject`
		`readAllObjects`
			`updateObject`
			`writeObject`
			`writeAllObjects`
				`deleteObject`
				`deleteAllObjects`

Note: Oracle recommends that you perform all data source operations using a unit of work: doing so is the most efficient way to manage transactions, concurrency, and referential constraints. For more information, see "Understanding TopLink Transactions" on page 97-1.

These methods implicitly construct and execute a `DatabaseQuery` based on any of the following input parameters and return `Object` or `Object` collection:

- Reference `Class` (the `Class` of objects that the query accesses)
- Reference `Class` and `Call`
- Reference `Class` and `Expression`
- Example object with primary key set

These methods are a convenient way to perform the most common data source operations on objects.

WARNING: Allowing an unverified SQL string to be passed into these methods makes your application vulnerable to SQL injection attacks.

To access all configurable options to further refine and optimize a query, consider using a corresponding `DatabaseQuery` directly. For more information, see "Database Queries" on page 93-10.

For more information, see "Using Session Queries" on page 94-1.

Read-Object Session Queries

Read-object queries return the first instance of an `Object` that matches the specified selection criteria, and read-all object queries return all such instances.

You can also pass in a domain `Object` with its primary key set and TopLink will construct and execute a read-object query to select that object. This is one form of query by example. For more information on query by example, see "Query-by-Example" on page 93-6.

For more information, see "Reading Objects With a Session Query" on page 94-1.

Create, Update, and Delete Object Session Queries

Oracle recommends that you create and update objects using a unit of work: doing so is the most efficient way to manage transactions, concurrency, and referential constraints. For more information, see "Understanding TopLink Transactions" on page 97-1.

However, you can also create and update objects using a session query. These session queries are a convenient way to modify objects directly on the database when you manage simple, nonbusiness object data that has no relationships (for example, user preferences).

If you know an object is new, you can use an `insertObject` method to avoid having TopLink perform an existence check. If you do not know if an object is new, use the `updateObject`, `writeObject`, or `writeAllObject` methods: TopLink performs an existence check if necessary.

When you execute a write session query, it writes both the object and its privately owned parts to the database. To manage this behavior, use a corresponding `DatabaseQuery` (see "Object-Level Modify Queries and Privately Owned Parts" on page 93-15).

Using the `Session` method `deleteObject`, you can delete a specific object. Using the Session method `deleteAllObjects`, you can delete a collection of objects. Each specified object and all its privately owned parts are deleted. In the case of `deleteAllObjects`, all deletions are performed within a single transaction.

For more information, see "Creating, Updating, and Deleting Objects With a Session Query" on page 94-3.

Database Queries

All session types provide an `executeQuery` method that takes any of the following types of `DatabaseQuery`:

- Object-Level Read Query
- Data-Level Read Query
- Object-Level Modify Query
- Data-Level Modify Query
- Report Query

Using `DatabaseQuery` method `setCall`, you can define your own `Call` to accommodate a variety of data source options such as SQL (including stored procedures and stored functions), EJB QL queries, and EIS interactions. For more information, see "Call Queries" on page 93-17.

Using `DatabaseQuery` method `setSelectionCriteria`, you can specify your selection criteria using a TopLink `Expression`. For more information, see "TopLink Expressions" on page 93-3.

For more information, see "Using DatabaseQuery Queries" on page 94-4.

Object-Level Read Query

Using an `ObjectLevelReadQuery`, you can query your data source and return `Object` instances that match the specified selection criteria. This section describes the following:

- ReadObjectQuery
- ReadAllQuery
- Partial Object Queries
- Join Reading and Object-Level Read Queries
- Fetch Groups and Object-Level Read Queries

For more information, see "Reading Objects Using a DatabaseQuery" on page 94-4.

ReadObjectQuery

Using a `ReadObjectQuery`, you can query your data source and return the first object that matches the specified selection criteria.

ReadAllQuery

Using a `ReadAllQuery`, you can query your data source and return a `Collection` of all the objects that match the specified selection criteria.

Partial Object Queries

By default, an `ObjectLevelReadQuery` returns all attributes of the objects read.

If you require only certain attributes from selected objects, you can create a partial object query by using `ObjectLevelReadQuery` method `addPartialAttributes`. Using this method, you can improve query performance by making TopLink return objects with only specified attributes populated.

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. Later retrieval of a complete object is simplified because TopLink always includes the primary key attribute (even if you do not add it as a partial attribute.

Consider the following when you use partial object queries:

- You cannot edit or cache partial objects.
- Unspecified attributes will be left `null`.
- You cannot have two partial attributes of the same type.
- You cannot add a partial attribute which is of the same type as the class being queried.

If you require only summary information for certain attributes from selected objects, it is more efficient to use a `ReportQuery` (see "Report Query" on page 93-15).

For more information, see "Reading Objects Using Partial Object Queries" on page 94-6.

Join Reading and Object-Level Read Queries

Join reading is a query optimization feature that allows a single query for a class to return the data to build the instances of that class and its related objects. Use this feature to improve query performance by reducing database access. By default, relationships are not join-read: each relationship is fetched separately when accessed if you are using indirection (see "Indirection" on page 30-5), or as a separate database query if you are not using indirection.

You can use join reading with `ReadObjectQuery` and `ReadAllObjectQuery` to join only one-to-one or one-to-many mapped relationships. Join reading is not currently supported for any other relationship mappings.

Join reading can specify multiple and nested relationships to be joined. Nested joins are expressed through using expressions (see "Expressions for Joining and Complex Relationships" on page 95-6).

Outer joins can also be used with join reading through using the expression outer join API. If an outer join is not used, objects with missing one-to-one relationships or empty one-to-many relationships will be filtered from the result set.

You can use join reading between relationships to the same class, or to concrete or leaf inherited classes, but not to root or branch inherited classes that have multiple tables. For more information about inheritance, see "Understanding Descriptors and Inheritance" on page 23-12.

> **Note:** TopLink does not support joining to a class that takes multiple SQL calls to read. For example, joining to a superclass (potentially).

You can use join reading with custom SQL or stored procedures, but the query must ensure that all of the required data to build all of the join-read objects is returned. If the result set includes the same tables or fields, they must be returned in the same table order as TopLink would have generated.

For more information, see "Using Join Reading" on page 94-11.

Avoiding Join-Reading Duplicate Data Join reading can result in returning duplicate data if a one-to-many or a shared one-to-one relationship is joined. Although TopLink correctly filters the duplicate results from the object result, the duplicate data still must be fetched from the database and can degrade performance, especially if multiple one-to-many relationships are joined. In general, batch reading can be used as a better alternative to join reading, as it does not require fetching duplicate data.

Oracle recommends that you use one-to-many joining with caution, because it does not scale well in many situations.

Because the main cost of a `ReadObjectQuery` is SQL execution, the performance of a one-to-many join in this case is usually better than a query without joining.

However, because the main cost of a `ReadAllObjectQuery` is row-fetching, which the duplicate data of a join increases, the performance of a one-to-many join in this case is less efficient than batch reading in many scenarios (even though one-to-many joining is more efficient than reading the objects one-by-one).

This is mainly due to the fact that a one-to-many join reads in duplicate data: the data for each source object will be duplicated for each target object. Depending on the size of the one-to-many relationship and the size of the source object's row, this can become very inefficient, especially if the source object has a Large Object (LOB).

If you use multiple or nested one-to-many joins in the same query, the problem is compounded: the source object's row is duplicated *n***m* times, and each target object *n* and *m* times respectively. This can become a major performance issue.

To handle empty collections, you must use outer joins, so the queries can easily become very database intensive. Batch reading has the advantage of only returning the required data, and does not require outer joins.

Oracle recommends that you use batch reading to optimize querying relationships in read-all applications.

For more information, see the following:

- "Batch and Join Reading" on page 11-17
- "Reading Case 2: Batch Reading Objects" on page 11-23

Fetch Groups and Object-Level Read Queries

You can use a fetch group with a `ReadObjectQuery` or `ReadAllQuery`. When you execute the query, TopLink retrieves only the attributes in the fetch group. TopLink automatically executes a query to fetch all the attributes excluded from this subset when and if you call a getter method on any one of the excluded attributes.

Note: Currently, fetch groups only provide support for EJB.

For more information, see the following:

- "Fetch Groups" on page 23-5
- "Using Queries With Fetch Groups" on page 96-2

Data-Level Read Query

Using a `DataLevelReadQuery`, you can query your data source and return `Object` instances that match the specified selection criteria. This section describes the following:

- DataReadQuery
- DirectReadQuery
- ValueReadQuery

For more information, see "Reading Data With a DatabaseQuery" on page 94-15.

WARNING: Allowing an unverified SQL string to be passed into constructors of such objects as `DataReadQuery`, `DirectReadQuery` and `ValueReadQuery` makes your application vulnerable to SQL injection attacks.

DataReadQuery

Use a `DataReadQuery` to execute a selecting SQL string that returns a `Collection` of the `DatabaseRows` representing the result set.

DirectReadQuery

Use a `DirectReadQuery` to read a single column of data (that is, one field) that returns a `Collection` of values representing the result set.

ValueReadQuery

Use a `ValueReadQuery` to read a single data value (that is, one field). A single data value is returned, or null if no rows are returned.

Object-Level Modify Query

Using an `ObjectLevelModifyQuery`, you can query your data source to create, update, and delete objects. This section describes the following:

- WriteObjectQuery
- UpdateObjectQuery
- InsertObjectQuery
- DeleteObjectQuery
- UpdateAllQuery
- Object-Level Modify Queries and Privately Owned Parts

For more information, see "Creating, Updating, and Deleting Objects With a DatabaseQuery" on page 94-13.

Note: Oracle recommends that you create and update objects using a TopLink `UnitOfWork`: doing so is the most efficient way to manage transactions, concurrency, and referential constraints. For more information, see "Understanding TopLink Transactions" on page 97-1.

WriteObjectQuery

If you do not know whether or not an object is new, use a `WriteObjectQuery`: TopLink performs an existence check if necessary to determine whether to perform an insert or an update.

If you do know whether or not an object exists, you can avoid the existence check by using an `UpdateObjectQuery` (see "UpdateObjectQuery" on page 93-14) or `InsertObjectQuery` (see "InsertObjectQuery" on page 93-14).

UpdateObjectQuery

If you know that the object you want to modify exists, use an `UpdateObjectQuery` to avoid having TopLink perform an existence check.

InsertObjectQuery

If you know an object is new, you can use an `InsertObjectQuery` to avoid having TopLink perform an existence check.

DeleteObjectQuery

To delete a specific object, construct a `DeleteObjectQuery` with a single specific object as an argument.

UpdateAllQuery

The `UpdateAllQuery` allows you to take an expression and update a set of objects (at the object level) without loading the objects into memory. You can updated to either a specific or relative value. For example, you can set the value to 5 or to increase by 5 percent.

For more information, see "Creating, Updating, and Deleting Objects With a DatabaseQuery" on page 94-13.

Object-Level Modify Queries and Privately Owned Parts

When you execute a create or update object `DatabaseQuery`, it writes both the object and its privately owned parts to the database by default. To create a query that does not update privately owned parts, use the `DatabaseQuery` method `dontCascadeParts`. Use this method to do the following:

- Increase performance when you know that only the object's direct attributes have changed.
- Manually 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 data source. For more information, see "Understanding TopLink Transactions" on page 97-1.

Data-Level Modify Query

Using a `DataModifyQuery`, you can query your data source to execute a nonselecting SQL statement. It is equivalent to `Session` method `executeNonSelectingCall`.

For more information, see "Updating Data With a DatabaseQuery" on page 94-16.

Report Query

If you want to summarize (or roll up) certain attributes of a set of objects, you can use a `ReportQuery`.

A `ReportQuery` returns summary data from a set of objects and their related objects. That is, it returns data about objects, rather than the objects themselves. However, it still lets you query and specify the data at the object level. To build a report query, you specify the search criteria, the data you require about the objects, and how that data should be summarized.

For example, you can create a report query to compute 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).

A `ReportQuery` lets you do the following:

- 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 data source aggregation functions (supported by your platform), 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.

A `ReportQuery` is the most efficient form of partial object query (see "Partial Object Queries" on page 93-11) because it takes advantage of the reporting capabilities of your

data source (if available). Oracle recommends that you use `ReportQuery` to do partial object queries.

The `ReportQuery` API returns a collection of `ReportQueryResult` objects, similar in structure and behavior to a `DatabaseRow` or a `Map`. For more information, see "Report Query Results" on page 93-8.

For more information, see the following:

- "Reading Case 1: Displaying Names in a List" on page 11-20
- "Reading Objects Using Report Queries" on page 94-6
- "Configuring Named Queries at the Descriptor Level" on page 25-10

Named Queries

When you use a session query method like `readAllObjects` (see "Session Queries" on page 93-9), TopLink creates a corresponding `ReadAllQuery`, which builds other objects it needs to perform its task. When TopLink finishes execution of the `readAllObjects` method, these objects are discarded. Each time you call this session method, TopLink creates these related objects again, uses them once, and then discards them.

Alternatively, you can create a `DatabaseQuery` (see "Database Queries" on page 93-10) and store it by name at the descriptor- (see "Configuring Named Queries at the Descriptor Level" on page 25-10) or session-level (see "Configuring Named Queries at the Session Level" on page 74-21).

TopLink prepares a named query once, and it (and all its associated supporting objects) can be efficiently reused thereafter making a named query well suited for frequently executed operations.

Using the `Session` API (see "Using Named Queries" on page 94-18), you can execute these queries by name, passing in any required arguments.

WARNING: Allowing an unverified SQL string to be passed into methods (for example: `readAllObjects(Class class, String sql)` method) makes your application vulnerable to SQL injection attacks.

When to Use Named Queries

For a reasonably complex query that you execute frequently, you should consider making the query a named query.

If a query is global to a project, configure the named query at the session level ("Configuring Named Queries at the Session Level" on page 74-21).

If a query is global to a `Class` or you want to configure CMP finders, configure the named query at the descriptor level (see "Configuring Named Queries at the Descriptor Level" on page 25-10). For more information about descriptor level query configuration, see "Descriptor Query Manager Queries" on page 93-23.

For a very complex query, you can delegate query execution to your own static method using a special form of a named query called a redirect query. For more information about redirect queries, see "Redirect Queries" on page 93-20).

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.

Call Queries

All session types provide `executeSelectingCall` and `executeNonSelectingCall` methods that take any of the following `Call` types:

- SQL Calls
- EJB QL Calls
- Enterprise Information System (EIS) Interactions

You can also execute a `Call` in the context of a `DatabaseQuery`. For more information on `DatabaseQuery`, see "Database Queries" on page 93-10.

> **WARNING: Allowing an unverified SQL string to be passed into methods (for example: `executeSelectingCall(String sql)` method) makes your application vulnerable to SQL injection attacks.**

SQL Calls

SQL calls access fields in a relational database. TopLink supports the following SQL calls:

- SQLCall
- StoredProcedureCall
- StoredFunctionCall

Using the `Call` API (or SQL string conventions), you can specify input, output, and input-output parameters and assign values for input and input/output parameters.

Using a descriptor `ReturningPolicy`, you can control whether or not TopLink writes a parameter out, retrieves a value generated by the database, or both. For more information, see "Configuring Returning Policy" on page 25-67.

SQLCall

Using a `SQLCall`, you can specify any arbitrary SQL statement and execute it on a data source.

> **WARNING: Allowing an unverified SQL string to be passed into methods makes your application vulnerable to SQL injection attacks.**

For more information, see "Using an SQLCall" on page 94-19.

StoredProcedureCall

A stored procedure is composed of one or more procedural language statements, such as Procedural Language/Structured Query Language (PLSQL), stored by name in the database. Most relational databases support stored procedures.

You invoke a stored procedure to execute logic and access data from the data source.

Using a `StoredProcedureCall`, you can detect execution errors, specify input parameters, output parameters, and input/output parameters. However, stored procedures do not provide a return value.

For more information, see "Using a StoredProcedureCall" on page 94-21.

StoredFunctionCall

A stored function is an Oracle database feature that provides all the functionality of a stored procedure as well as the ability to return a value.

Using a `StoredFunctionCall`, you can specify all the features of a `StoredProcedureCall` as well as the field name of the return value.

For more information, see "Using a StoredFunctionCall" on page 94-23.

Oracle Extensions

When you use TopLink with an Oracle database, you can make use of the following Oracle specific query features from within your TopLink applications:

- Hints
- Hierarchical Queries
- Flashback Queries
- Stored Functions

Hints Oracle lets you specify SQL query additions called hints that can influence how the database server SQL optimizer works. This lets you influence decisions usually reserved for the optimizer. You use hints to specify things such as join order for a join statement, or the optimization approach for a SQL call.

You specify hints using the `DatabaseQuery` method `setHintString`.

For more information, see the following:

- "Database Queries" on page 93-10
- "Oracle Hints" on page 96-6
- Your database *Performance Tuning Guide and Reference*.

Hierarchical Queries Oracle database Hierarchical Queries mechanism lets you 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.

You specify a hierarchical query clause using `DatabaseQuery` subclass `ReadAllQuery` method `setHierarchicalQueryClause`. For more information on `DatabaseQuery` queries, see "Database Queries" on page 93-10.

For more information on configuring a `ReadAllQuery` with an Oracle hierarchical query clause, see "Hierarchical Queries" on page 96-6.

Flashback Queries When using TopLink with Oracle9i (or later), you can acquire a special historical session where all objects are read as of a past time, and then you can express read queries depending on how your objects are changing over time.

For more information, see "Historical Queries" on page 93-21.

Stored Functions A stored function is an Oracle database mechanism that provides all the capabilities of a stored procedure in addition to returning a value.

For more information, see "StoredFunctionCall" on page 93-18.

EJB QL Calls

In TopLink, EJB QL calls represent EJB QL strings. An `EJBQLCall` object is an abstraction of a database invocation. You can execute an EJB QL call directly from a session or in the context of a `DatabaseQuery`.

For more information, see the following:

- "Using EJB QL Calls" on page 94-24
- "Specifying a Custom EJB QL String in a DatabaseQuery" on page 94-17
- "EJB QL Queries" on page 93-5

Enterprise Information System (EIS) Interactions

To invoke a query through a J2EE Connector Architecture (J2C) adapter to a remote EIS, you use an `EISInteraction`, an instance of `Call`. TopLink supports the following `EISInteraction` types:

- IndexedInteraction
- MappedInteraction
- XMLInteraction
- XQueryInteraction
- QueryStringInteraction

In each of these interactions, you specify a functional interface (similar to a stored procedure) that identifies the function to invoke on the EIS. This functional interface contains the following:

- The function name
- The record name (if different than the function name)
- A list of input arguments
- A list of output arguments

For more information, see the following:

- "EIS Projects" on page 17-7
- "Using EIS Interactions" on page 94-24

IndexedInteraction

In an `IndexedInteraction`, you exchange data with the EIS using indexed records. The order of the specification of the arguments must match the order of the values defined in the indexed record.

MappedInteraction

In a `MappedInteraction`, you exchange data with the EIS using mapped records. The arguments you specify map by name to fields in the mapped record.

XMLInteraction

An `XMLInteraction` is a `MappedInteraction` that maps data to an XML record. For an `XMLInteraction`, you may also provide an optional root element name.

XQueryInteraction

If your J2C adapter supports the XQuery dynamic query language, you can use an `XQueryInteraction`, which is an `XMLInteraction` that lets you specify your XQuery string.

QueryStringInteraction

If your J2C adapter supports a query string based dynamic query language, you can use a `QueryStringInteraction`, which is a `MappedInteraction` that lets you specify the dynamic query string.

Redirect Queries

To accommodate complex query logic, you can implement a **redirect query**: a named query that delegates query execution control to your application. For more information, see "Named Queries" on page 93-16.

Redirect queries lets you define the query implementation in code as a static method. When you invoke the query, the call redirects to the specified static method. Redirect queries accept any arbitrary parameters passed into them packaged in a `Vector`.

Although most TopLink queries search for objects directly, a redirect query generally invokes a method that exists on another class and waits for the results. Redirect queries let you build and use complex operations, including operations that might not otherwise be possible within the query framework.

By delegating query invocation to a method you provide, redirect queries let you dynamically make decisions about how a query should be executed based on argument values.

Using a redirect query, you can do the following:

- Dynamically configure the query options based on the arguments (for example, ordering and query optimization)
- Dynamically define the selection criteria based on the arguments
- Pass query-by-example objects or expressions as the arguments
- Post-process the query results
- Perform multiple queries or special operations

If you execute the query on a `UnitOfWork`, the results register with that instance of `UnitOfWork`, so any objects you attempt to retrieve with the invoke method must come from the `Session` cache.

To create a redirect query, you implement the `QueryRedirector` interface and set your implementation on a named query.

Oracle recommends that you take advantage of the `MethodBasedQueryRedirector`, an instance of `QueryRedirector` that TopLink provides. It takes the name of a static method and the `Class` in which it is defined as parameters. When you set a `MethodBasedQueryRedirector` on a named query, whenever `invokeQuery` method is called on this instance, TopLink uses reflection to invoke your static method instead.

The advantages of using a `MethodBasedQueryRedirector` are as follows:

- You can specify the static method and its `Class` dynamically.
- The class that provides the static method does not need to implement `QueryRedirector`.
- Your static method can have any name.
- You can restrict the parameters to your static method to only a `Session` and a `Vector` of arguments.

For more information, see "Using Redirect Queries" on page 96-1.

Historical Queries

By default, a session represents a view of the most current version of objects and when you execute a query in that session, it returns the most current version of selected objects.

If your data source maintains past or historical versions of objects, you can configure TopLink to access this historical data (see "Historical Sessions" on page 72-25).

Once you configure TopLink to take advantage of this historical data, you can access historical versions using the historical queries that Table 93–4 summarizes.

> **Note:** Flashback queries do not support view selects. This means you cannot use a flashback query on objects with an inheritance policy for read-all-subclasses views. For more information, see "Understanding Descriptors and Inheritance" on page 23-12.

Table 93–4 Historical Queries

Historical Query Type	Session	Cache	Must set `maintainCache` to false?	Query both current and historical versions?
Using an ObjectLevelReadQuery With an AsOfClause	Regular[1]	▪ Global ▪ Read and write ▪ Contains current versions	Yes	No
Using an ObjectLevelReadQuery With Expression Operator asOf	Regular[1]	▪ Global ▪ Read and write ▪ Contains current versions	Yes	Yes
Using an ObjectLevelReadQuery in a Historical Session	Historical[2]	▪ Isolated ▪ Read-only ▪ Contains static snapshot as of specified time	No	No

[1] A server or database session based on an `OraclePlatform` for an Oracle9i (or later) or based on a TopLink `HistoryPolicy`.

[2] A session returned by a server or database session based on an `OraclePlatform` or TopLink `HistoryPolicy` using the `acquireHistoricalSession` method passing in an `AsOfClause`.

Using an ObjectLevelReadQuery With an AsOfClause

You can query historical versions of objects using an `ObjectLevelReadQuery` configured with an `AsOfClause` (set by `ObjectLevelReadQuery` method `setAsOfClause`) that specifies a point in time that applies to every `Expression` used in the query.

This type of historical query lets you query a static snapshot of object versions as of the specified time.

> **Note:** To prevent corrupting the global shared cache with old versions of objects, you must set `ObjectLevelReadQuery` method `maintainCache` to `false` in this historical query. If you do not, TopLink will throw an exception when you execute the query.

For more information and examples of using an `ObjectLevelReadQuery` with an `AsOfClause`, see "Using Historical Queries" on page 96-2.

Using an ObjectLevelReadQuery With Expression Operator asOf

You can query historical versions of objects using an `ObjectLevelReadQuery` (such as `ReadObjectQuery` or `ReadAllQuery`) containing one or more expressions that use `Expression` operator `asOf` to specify a point in time on an `Expression`-by-`Expression` basis.

This type of historical query lets you combine both current and historical versions of objects in the same query.

If you configure the `ObjectLevelReadQuery` with an `AsOfClause`, that point in time overrides the point in time specified in any `Expression` in the query (see "Using an ObjectLevelReadQuery With an AsOfClause" on page 93-21).

> **Note:** To prevent corrupting the global shared cache with old versions of objects, you must set `ObjectLevelReadQuery` method `maintainCache` to `false` in this historical query. If you do not, TopLink will throw an exception when you execute the query.

For more information and examples of using an `ObjectLevelReadQuery` with `Expression` operator `asOf`, see "Using Historical Queries" on page 96-2.

Using an ObjectLevelReadQuery in a Historical Session

Given a session that maintains historical versions of objects (based on an appropriate `OraclePlatform` or TopLink `HistoryPolicy`), you can use `Session` method `acquireHistoricalSession` passing in an `AsOfClause` that specifies a point in time that applies to all queries and expressions.

This method returns a lightweight, read-only snapshot of object versions as of the specified time. The cache used in this type of session is isolated from the global shared cache. You do not need to set `ObjectLevelReadQuery` method `maintainCache` to `false` in this case.

For more information and examples of using an `ObjectLevelReadQuery` with a historical session, see "Using Historical Queries" on page 96-2.

Interface and Inheritance Queries

When you define an interface descriptor (see "Relational Interface Descriptors" on page 24-2), you can perform queries on interfaces and inheritance hierarchies.

For more information, see the following:

- "Querying on Interfaces" on page 96-4
- "Querying on an Inheritance Hierarchy" on page 96-4

Descriptor Query Manager Queries

Each `Descriptor` owns an instance of `DescriptorQueryManager` that you can use for the following:

- Configuring Named Queries
- Configuring Default Query Implementations
- Configuring Additional Join Expressions

Configuring Named Queries

The `DescriptorQueryManager` provides API for storing and retrieving frequently used queries by name.

For more information, see "Named Queries" on page 93-16.

Configuring Default Query Implementations

The `DescriptorQueryManager` of each `Descriptor` lets you customize the query implementation that TopLink uses for the following data source operations:

- insert object
- update object
- read object
- read all objects
- delete object

For example, if you need to insert an object using a stored procedure, you can override the default `SQLCall` used by the `DescriptorQueryManager` insert object query.

Whenever you execute a query on a given `Class`, TopLink consults the `DescriptorQueryManager` to determine how to perform the given data source operation.

You can use this capability for a variety of purposes such as to extend TopLink behavior, access nonrelational data, or use stored procedures or customized SQL calls.

WARNING: Allowing an unverified SQL string to be passed into methods makes your application vulnerable to SQL injection attacks.

For information and examples on customizing these default query implementations, see:

- "Configuring Custom SQL Queries for Basic Persistence Operations" on page 26-6
- "Configuring Custom EIS Interactions for Basic Persistence Operations" on page 28-6

Configuring Additional Join Expressions

You can configure the `DescriptorQueryManager` to automatically append an expression to every query it performs on a class. For example, you can add an expression that filters the data source for the valid instances of a given class.

For more information, see "Appending Additional Join Expressions" on page 96-4.

EJB Finders

An EJB finder is a query as defined by the EJB specification. It returns EJB, collections, and enumerations. The difference between a finder and a query is that queries return Java objects, but finders return EJB. The TopLink query framework lets you 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.

You can implement finders using any TopLink query feature and you can take advantage of predefined finder implementations that TopLink provides for both CMP and BMP entity beans.

This section describes the following:

- Predefined Finders
- Default Finders
- Call Finders
- DatabaseQuery Finders
- Named Query Finders
- Primary Key Finders
- Expression Finders
- EJB QL Finders
- SQL Finders
- Redirect Finders
- The ejbSelect Method

For more information, see "Using EJB Finders" on page 96-7.

Predefined Finders

TopLink provides predefined finder implementations that provide a rich API that lets you dynamically specify query properties at run time and take full advantage of TopLink query features.

TopLink provides the following predefined finders:

- Predefined CMP Finders
- Predefined BMP Finders

For more information, see the following:

- "Understanding the EJB Entity Beans With CMP Architecture" on page 2-27
- "Understanding the EJB Entity Beans With BMP Architecture" on page 2-31

Predefined CMP Finders

Table 93–5 lists the predefined finders you can use with TopLink CMP (using OC4J or BEA WebLogic Server).

The TopLink runtime reserves the method names listed in Table 93–5.

Table 93–5 Predefined CMP Finders

Method	Arguments	Return
`findAll`	`()`	`Collection`
`findManyByEJBQL`	`(String ejbql)` `(String ejbql, Vector arguments)`	`Collection`
`findManyByQuery`	`(DatabaseQuery query)` `(DatabaseQuery query, Vector arguments)`	`Collection`
`findManyBySQL`	`(String sql)` `(String sql, Vector arguments)`	`Collection`
`findByPrimaryKey`	`(Object primaryKeyObject)`	`EJBObject`
`findOneByEJBQL`	`(String ejbql)` `(String ejbql, Vector arguments)`	`EJBObject`
`findOneByQuery`	`(DatabaseQuery query)` `(DatabaseQuery query, Vector arguments)`	`EJBObject`
`findOneBySQL`	`(String sql)` `(String sql, Vector arguments)`	`EJBObject`

Note: 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.

For more information, see "Using EJB Finders" on page 96-7.

Predefined BMP Finders

Table 93–6 lists the predefined finders you can use if you extend your BMP EJB from `oracle.toplink.ejb.bmp.BMPEntityBase` (see "Understanding the EJB Entity Beans With BMP Architecture" on page 2-31).

The TopLink runtime reserves the method names listed in Table 93–6.

Table 93–6 Predefined BMP Finders

Method	Arguments	Return

`findAll`	`()`
	`(Call)`
	`(Expression)`
	`(ReadAllQuery)`
		`Enumeration`

`findAllByNamedQuery`	`(String queryName, Vector arguments)`	`Enumeration`
`findByPrimaryKey`	`(Object primaryKeyObject)`	`Object`

`findOne`	`(Call)`
	`(Expression)`
	`(ReadObjectQuery)`
		`Object`

`findOneByNamedQuery`	`(String queryName, Vector arguments)`	`Object`

For more information about using EJB finders, see "Using EJB Finders" on page 96-7.

Default Finders

For each finder method defined on the home interface of an entity bean, whose name matches `findBy<CMP-FIELD-NAME>` where `<CMP-FIELD-NAME>` is the name of a persistent field on the bean, TopLink generates a finder implementation including a TopLink query that uses the TopLink expressions framework. If the return type is a single bean type, TopLink creates a `ReadObjectQuery`; if the return type is a `Collection`, TopLink creates a `ReadAllQuery`.

Although you must still define the finder in the entity home, you do not need to declare the finder in the `ejb-jar.xml` file.

For more information, see "Creating a Finder" on page 96-8.

Call Finders

Finders that use a `Call` lets you create dynamic queries that you generate at run time rather than at deployment time.

For more information, see the following:

- "Call Queries" on page 93-17.
- "Predefined Finders" on page 93-24

DatabaseQuery Finders

Finders that use a `DatabaseQuery` lets you create dynamic queries that you generate at run time rather than at deployment time.

In addition to finders that take a `DatabaseQuery`, TopLink also provides a default `findAll` finder that returns all the EJB of a given type. As with other dynamic finders, the TopLink runtime reserves the name `findAll`.

For more information, see "Database Queries" on page 93-10.

For more information on TopLink predefined finders that take a `DatabaseQuery`, see "Predefined Finders" on page 93-24.

Named Query Finders

Finders that use a named `DatabaseQuery` stored in a `DescriptorQueryManager` or `Session` lets you efficiently reuse frequently executed queries.

For more information, see

- "Named Queries" on page 93-16
- "Predefined Finders" on page 93-24

Primary Key Finders

TopLink provides predefined finder implementations that take a primary key class as a Java `Object`.

Because the EJB 2.0 and 2.1 specifications requires the container to implement the `findByPrimaryKey` call on each bean `Home` interface, do not delete this finder from a bean.

For more information, see "Predefined Finders" on page 93-24.

Expression Finders

Using a finder based on a TopLink `Expression` offers the following advantages:

- Version-controlled standardized queries in Java code
- Ability to simplify most complex operations
- A more complete set of querying features than is available through EJB QL

Because expressions lets you specify finder search criteria based on the object model, they are frequently the best choice for constructing your finders.

For more information, see "TopLink Expressions" on page 93-3.

For more information on TopLink predefined finders that take an `Expression`, see "Predefined Finders" on page 93-24.

You can also use an `Expression` in a finder that takes a `DatabaseQuery` by using `DatabaseQuery` method `setSelectionCriteria`. For more information on TopLink predefined finders that take a DatabaseQuery, see "DatabaseQuery Finders" on page 93-26.

EJB QL Finders

TopLink supports EJB QL. EJB QL finders let you specify an EJB QL string as the implementation of the query.

EJB QL offers the following advantages:

- It is the EJB 2.0 and 2.1 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.

For more information about EJB QL support in TopLink, see "Query Languages" on page 93-4.

For more information on TopLink predefined finders that take EJB QL, see "Predefined Finders" on page 93-24.

SQL Finders

Using SQL to define a finder offers the following advantages:

- You can implement logic that cannot be expressed when you use EJB QL or a TopLink `Expression`.
- It allows for the use of a stored procedure instead of 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.
- Hard-coded SQL limits portability to other databases.
- No validation is performed on the SQL string. Errors in SQL statements will not be detected until run time.
- The use of SQL for a function other than `SELECT` may result in unpredictable errors.

For more information on TopLink predefined finders that take SQL, see "Predefined Finders" on page 93-24.

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, 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.

For more information, see the following:

- "Redirect Queries" on page 93-20
- "Using EJB Finders" on page 96-7

The ejbSelect Method

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.

ejbSelect 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.

- The `result-type-mapping` tag in the `ejb-jar.xml` file determines the return type for `ejbSelect` methods. Set the flag to `Remote` to return `EJBObjects`; set it to `Local` to return `EJBLocalObjects`.

The format for an `ejbSelect` method definition looks as follows:

```
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.

For more information and examples on using TopLink queries in the `ejbSelect` method, see "Using EJB Finders" on page 96-7.

Queries and the Cache

When you execute a query, TopLink retrieves the information from either the database or the TopLink session cache. You can configure the way queries use the TopLink cache to optimize performance.

TopLink maintains a client-side cache to reduce the number of read operations required from the database. 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 TopLink descriptor. For more information, see "Explicit Query Refreshes" on page 87-7.

This section illustrates ways to manipulate the relationship between query and cache, including the following:

- Configuring the Cache
- Using In-Memory Queries
- Primary Key Queries and the Cache
- Disabling the Identity Map Cache Update During a Read Query
- Refreshing the Cache
- Caching Query Results in the Session Cache
- Caching Query Results in the Query Cache
- Caching and EJB Finders

Configuring the Cache

The cache in a TopLink application holds objects that have already been read from or written to the database. Use of the cache in a TopLink application reduces the number of accesses to the database. Because accessing the database consumes time and

resources, an effective caching strategy is important to the efficiency of your application.

For more information about configuring and using the cache, see Chapter 87, "Understanding the Cache".

Using In-Memory Queries

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 not based on 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 to 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.

In-memory querying lets you 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 value holders to enable in-memory querying to work across relationships.

This section describes the following:

- Configuring Cache Usage for In-Memory Queries
- Expression Options for In-Memory Queries
- Handling Exceptions Resulting From In-Memory Queries

Configuring Cache Usage for In-Memory Queries

You can configure in-memory query cache usage at the query level using `ReadObjectQuery` and `ReadAllQuery` methods:

- `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 parent session cache (not the unit of work cache) and return the result from the parent session 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. For more information and limitations on conforming, see "Using Conforming Queries and Descriptors" on page 99-8.

Alternatively, you can configure cache usage using the `ObjectLevelReadQuery` method `setCacheUsage`, passing in the appropriate `ObjectLevelReadQuery` field: `CheckCacheByPrimaryKey`, `CheckCacheByExactPrimaryKey`, `CheckCacheThenDatabase`, `CheckCacheOnly`, `ConformResultsInUnitOfWork`, or `DoNotCheckCache`.

Expression Options for In-Memory Queries

You can use a subset of `Expression` (see Table 93–7) and `ExpressionMath` (see Table 93–8) methods with in-memory queries. For more information about these options, see "Understanding TopLink Expressions" on page 95-1.

Table 93–7 Expressions Operator Support for In-Memory Queries

Expressions Operator	In-Memory Query Support
`addMonths`	
`and`	✔
`anyof`[1]	✔
`anyofAllowingNone`[1]	✔
`asciiValue`	
`between`	✔
`concat`	✔
`currentDate`	
`dateToString`	
`decode`	
`equal`	✔
`get`[1]	✔
`getAllowingNull`[1]	✔
`getFunction`	
`greaterThan`	✔
`greaterThanEqual`	✔
`hexToRaw`	
`ifNull`	
`in`	✔
`isNull`	✔
`lastDay`	

Table 93–7 (Cont.) Expressions Operator Support for In-Memory Queries

Expressions Operator	In-Memory Query Support
leftPad	
leftTrim	
length	✔
lessThan	✔
lessThanEqual	✔
like	✔
monthsBetween	
newTime	
nextDay	
notBetween	✔
notIn	
notIn	✔
notNull	✔
or	✔
ref	
replace	
rightPad	
rightTrim	
subQuery	
substring	✔
toCharacter	
toDate	
toLowerCase	✔
toNumber	✔
toUpperCase	✔
toUpperCasedWords	
translate	
trim	✔
truncateDate	

[1] For more information, see "Join Reading and Object-Level Read Queries" on page 93-12.

Table 93–8 ExpressionMath Operator Support for In-Memory Queries

ExpressionMath Operator	In-Memory Query Support
abs	✔
acos	✔

Table 93–8 (Cont.) ExpressionMath Operator Support for In-Memory Queries

ExpressionMath Operator	In-Memory Query Support
`add`	✔
`asin`	✔
`atan`	✔
`atan2`	
`ceil`	✔
`chr`	
`cos`	✔
`cosh`	
`exp`	✔
`floor`	✔
`ln`	
`log`	✔
`max`	✔
`min`	✔
`mod`	
`none`	
`power`	✔
`round`	✔
`sign`	
`sin`	✔
`sinh`	
`sqrt`	✔
`subtract`	✔
`tan`	✔
`tanh`	
`trunc`	

Handling Exceptions Resulting From In-Memory Queries

In-memory queries may fail for several reasons, the most common of which are the following:

- The query expression is too complex to execute in memory.
- There are untriggered value holders in which indirection is used. All object models that use indirection must first trigger value holders before they conform on the relevant objects.

TopLink provides a mechanism to handle indirection exceptions. To specify how the application must handle these exceptions, use the following `InMemoryQueryIndirectionPolicy` methods:

- `throwIndirectionException`: The default setting; it is the only setting that throws indirection exceptions.
- `triggerIndirection`: Triggers all valueholders to eliminate the problem.
- `ignoreIndirectionExceptionReturnConformed`: Returns conforming if an untriggered value holder is encountered. That is, results from the database are expected to conform, and an untriggered value holder is taken to mean that the underlying attribute has not changed.
- `ignoreIndirectionExceptionReturnNotConformed`: Returns not conforming if an untriggered value holder is encountered.

Note: When you build new applications, consider throwing all conform exceptions. This provides more detailed feedback for unsuccessful in-memory queries. For more information, see "Exceptions During Conforming" on page 99-34.

Primary Key Queries and the Cache

When a query searches for a single object by a primary key, 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 nonprimary key selection criteria or is a read-all query, the query executes against the database (unless you are using `ReadObjectQuery` or `ReadAllQuery` method `checkCacheOnly`). 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, TopLink builds the object. If the query is a refreshing query, TopLink updates the contents of any objects with the results from the query. Use "equals" on the object identity to 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.

Traversing Relationships With Compound Primary Keys

When getting objects by using compound primary keys to traverse relationships, you must create use query keys (see "Query Keys and Expressions" on page 95-10). By adding a query key for each mapped attribute in a class with a complex primary key, TopLink can use the primary key on the cache.

Consider the class `MyClass` with two attributes: `A` and `B`. Both `A` and `B` are mapped as 1:1 mappings to the database and designated primary keys.

You should create a query key for each attribute (such as `MyQueryKeyA` and `MyQueryKeyB`) that will map the attributes of the primary key of `MyClass` *without* going through the other classes. You can then use the query key to find the object in the cache and query the object's primary key:

```
builder.get("MyQueryKeyA").equal(new Long("123456"));
```

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 and can avoid exceptions during partial object queries (see "Reading Objects Using Partial Object Queries" on page 94-6).

Example 93–1 Disabling the Identity Map Cache Update

Example 93–1 demonstrates how code reads `Employee` objects from the database and writes the information to a file.

```
// Reads objects from the employee table and writes them to an employee file
void writeEmployeeTableToFile(String filename, Session session) {
    Vector employeeObjects;
    // Create ReadAllQuery and set Employee as its reference class
    ReadAllQuery query = new ReadAllQuery(Employee.class);
    ExpressionBuilder builder = query.getExpressionBuilder();
    query.setSelectionCriteria(builder.get("id").greaterThan(100));
    query.dontMaintainCache();
    Vector employees = (Vector) session.executeQuery(query);
    // Write all the employee data to a file
    Employee.writeToFile(filename, employees);
}
```

Refreshing the Cache

You can refresh objects in the cache to ensure that they are current with the database, while preserving object identity. This section describes how to use query API to perform the following:

- Configure query refreshing at the descriptor level (see "Configuring Cache Refreshing" on page 25-27) to apply cache refreshing to all queries of a particular object type. Before configuring cache refresh options, consider their effect on performance (see "Cache Optimization" on page 11-13).

Object Refresh

To refresh objects in the cache with the data in the database, call the `Session` method `refreshObject` or the `ReadObjectQuery` method `setShouldRefreshIdentityMapResult(true)`.

Cascading Object Refresh

You can control the depth at which a refreshing updates objects and their related objects. There are the following three options:

1. *CascadePrivateParts*: Default refresh behavior. Refreshes the local level object and objects that are referenced in privately owned relationships.
2. *CascadeNone*: Refreshes only the first level of the object, but does not refresh related objects.
3. *CascadeAll*: Refreshes the entire object tree, stopping when it reaches leaf objects.

Refreshing the Identity Map Cache During a Read Query

Include the `refreshIdentityMapResult` method in a query to force refreshing of an identity map with the results of the query.

Example 93–2 Refreshing the Result of a Query in the Identity Map Cache During a Read Query

```
// Create ReadObjectQuery and set Employee as its reference class
ReadObjectQuery query = new ReadObjectQuery(Employee.class);
ExpressionBuilder builder = query.getExpressionBuilder();
query.setSelectionCriteria(builder.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, you should 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 93–3 Using the cascadePrivateParts Method

```
ReadAllQuery query = new ReadAllQuery(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 in the Session Cache

By default, TopLink stores query results in the session cache enabling TopLink to execute the query repeatedly, without accessing the database. This is useful when you execute queries that run against static data.

By default, a read-all query always goes to the database, as it does not know how many objects it is seeking. However if the object already exists in the cache, time can be saved by not having to build a new object from the row.

For more information, see "Understanding the Cache" on page 87-1.

Caching Query Results in the Query Cache

In addition to TopLink's object cache, TopLink also supports a query cache:

- The **object cache** indexes objects by their primary key, allowing primary key queries to obtain cache hits. By using the object cache, queries that access the data source can avoid the cost of building the objects and their relationships if the object is already present.
- The **query cache** is distinct from the object cache. The query cache is indexed by the query and the query parameters – not the object's primary key. This allows for any query executed with the same parameters to obtain a query cache hit and return the same result set.

By default, a `ReadQuery` does not cache its query result set. You can, however, configure the query to cache its result set. This is useful for frequently executed

queries whose result set infrequently changes. The query cache always maintains hard references to the result set; the number of results sets for distinct parameters stored in the query cache is configurable. The query cache maintains its size number of the last executed queries with distinct parameters.

For more information, see "Caching Results in a ReadQuery" on page 96-20.

You can apply a cache invalidation policy to the query's internal cache (see "Configuring Cache Expiration at the Query Level" on page 96-21). For more information, see "Cache Invalidation" on page 87-8.

Internal Query Cache Restrictions

TopLink does not support the use of the query cache with cursors: if you use query caching with cursors, TopLink will throw an exception. For information on cursor query results, see "Stream and Cursor Query Results" on page 93-8 and "Handling Cursor and Stream Query Results" on page 96-15.

Caching and EJB Finders

TopLink caches enterprise beans that EJB finders retrieve. For your application, you can configure the caching of the EJB finders' results in a variety of ways, force the cache to be refreshed, or disable the caching.

This section describes the following:

- Caching Options
- Disabling Cache for Returned Finder Results
- Refreshing Finder Results

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 using TopLink Workbench.

You can set the caching options on a per-finder basis. Table 93–9 lists the valid values.

Table 93–9 Finder Caching Options

This Setting . . .	Causes Finders to . . .	When the Search Involves a Finder That . . .
`ConformResultsInUnitOfWork`[1]	Check the unit of work cache before querying the session cache or the database. The finder's results always conform to uncommitted new, deleted, and changed objects.	Returns either a single bean or a collection.
`DoNotCheckCache`	Query the database, bypassing the 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 databas.e	Returns a single bean.

Table 93–9 (Cont.) Finder Caching Options

This Setting . . .	Causes Finders to . . .	When the Search Involves a Finder That . . .
`CheckCacheOnly`	Search the parent session cache only (not the unit of work cache), but not the database.	Returns either a single bean or a collection.

[1] Default.

For more information about the TopLink queries, as well as the TopLink unit of work and how it integrates with JTS, see Chapter 97, "Understanding TopLink Transactions".

> **Note:** To apply caching options to finders with manually created queries (`findOneByQuery`, `findManyByQuery`), use the TopLink API.

Disabling Cache for Returned Finder Results

By default, 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` method on the query, or disable returned object caching for the query in TopLink Workbench.

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 TopLink 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.

> **Note:** When you invoke this option from within a transaction, the refresh action 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.

Understanding the Query API

Table 93–10 summarizes the query support provided by each type of session. For each session type, it shows the type of query operation (create, read, update, delete) that you can perform and whether or not you can execute a `DatabaseQuery` or `Call`. For

example, using a unit of work, you can use session queries to read and delete; using a server session, you can use session queries to create, read, update, and delete.

Table 93–10 Session Query API Summary

Session	Create	Read	Update	Delete	Execute Database Query	Execute Call
Unit of work		✔		✔	✔	✔
Database	✔	✔	✔	✔	✔	✔
Server						
Client		✔			✔	✔

Example 93–4 summarizes the important TopLink packages that provide query and expression support:

Example 93–4 Query and Expression Packages

```
oracle.toplink.queryframework
oracle.toplink.expressions
oracle.toplink.querykeys
oracle.toplink.descriptors.DescriptorQueryManager
```

94

Using Basic Query API

This chapter explains the following essential TopLink query API calls most commonly used throughout the development cycle:

- Using Session Queries
- Using DatabaseQuery Queries
- Using Named Queries
- Using SQL Calls
- Using EJB QL Calls
- Using EIS Interactions
- Handling Exceptions
- Handling Collection Query Results
- Handling Report Query Results

For more information, see "Using Advanced Query API" on page 96-1.

Using Session Queries

This section provides examples of using the session query methods for the following:

- Reading Objects With a Session Query
- Creating, Updating, and Deleting Objects With a Session Query

> **Note:** Oracle recommends that you perform all data source operations using a unit of work: doing so is the most efficient way to manage transactions, concurrency, and referential constraints. For more information, see "Understanding TopLink Transactions" on page 97-1.

For more information, see "Session Queries" on page 93-9.

Reading Objects With a Session Query

Using the session query API, you can perform the following read operations:

- Reading an Object With a Session Query
- Reading All Objects With a Session Query
- Refreshing an Object With a Session Query

Reading an Object With a Session Query

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, a null value 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`. 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` method 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 94–1 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"));
```

Reading All Objects With a Session Query

The `readAllObjects` method retrieves a `Vector` of objects from the database and does not put the returned objects in order. 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 94–2.

Example 94–2 readAllObjects Using an Expression

```
// Returns a Vector of employees whose employee salary is greater than 10000
Vector employees = session.readAllObjects(Employee.class,new
ExpressionBuilder.get("salary").greaterThan(10000));
```

Refreshing an Object With a Session Query

The `refreshObject` method causes TopLink to update the object in memory using 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.

Creating, Updating, and Deleting Objects With a Session Query

Using the session query API, you can perform the following create, update, and delete operations:

- Writing a Single Object to the Database With a Session Query
- Writing All Objects to the Database With a Session Query
- Adding New Objects to the Database With a Session Query
- Modifying Existing Objects in the Database With a Session Query
- Deleting Objects in the Database With a Session Query

Writing a Single Object to the Database With a Session Query

When you invoke the `writeObject` method, the method performs a *does-exist* check to determine whether or not 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 in the database.

Example 94–3 Writing a Single Object Using writeObject

```
// Create an instance of the 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 With a Session Query

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 94–4 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 not currently in
the database will be inserted. All the other employees currently stored in the
database will be updated */
session.writeAllObjects(employees);
```

Adding New Objects to the Database With a Session Query

The `insertObject` method creates a new object in 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 in the database. If the object does exist, the database throws an exception when you execute the `insertObject` method.

Modifying Existing Objects in the Database With a Session Query

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` method.

Deleting Objects in the Database With a Session Query

To delete a 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.

Using DatabaseQuery Queries

This section describes creating and executing `DatabaseQuery` queries to perform a variety of basic persistence operations, including the following:

- Reading Objects Using a DatabaseQuery
- Creating, Updating, and Deleting Objects With a DatabaseQuery
- Reading Data With a DatabaseQuery
- Updating Data With a DatabaseQuery
- Specifying a Custom SQL String in a DatabaseQuery
- Specifying a Custom EJB QL String in a DatabaseQuery
- Using Parameterized SQL and Statement Caching in a DatabaseQuery

Reading Objects Using a DatabaseQuery

This section provides examples that illustrate how to read objects using a `DatabaseQuery`, including the following:

- Basic DatabaseQuery Read Operations
- Reading Objects Using Partial Object Queries
- Reading Objects Using Report Queries
- Reading Objects Using Query-By-Example
- Specifying Read Ordering
- Specifying a Collection Class
- Specifying the Maximum Rows Returned
- Configuring Query Timeout at the Query Level
- Using Batch Reading
- Using Join Reading

Basic DatabaseQuery Read Operations

Example 94–5 illustrates a simple read query. It uses a 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 94–5 A Simple ReadAllQuery

```
// This example returns a Vector of employees whose employee ID is > 100

// Initialize the DatabaseQuery by specifying the query type
// and set the reference class for the query
ReadAllQuery query = new ReadAllQuery(Employee.class);

// Retrieve ExpressionBuilder from the query
ExpressionBuilder builder = query.getExpressionBuilder();

/* Configure the query execution. Because this example uses an expression, it uses
the setSelectionCriteria method */
query.setSelectionCriteria(builder.get("id").greaterThan(100));

// Execute the query
Vector employees = (Vector) session.executeQuery(query);
```

Example 94–6 illustrates a complex `readObject` query that uses all available configuration options.

Example 94–6 A Named Read Query with Two Arguments

```
// Initialize the DatabaseQuery by specifying the query type
// and set the reference class for the query
ReadObjectQuery query = new ReadObjectQuery(Employee.class);
// Retrieve ExpressionBuilder from the query
ExpressionBuilder builder = query.getExpressionBuilder();
// Define two expressions that map to the first and last names of the employee
Expression firstNameExpression =
emp.get("firstName").equal(emp.getParameter("firstName"));
Expression lastNameExpression =
emp.get("lastName").equal(emp.getParameter("lastName"));

/* Configure the query execution. Because this example uses an expression, it uses
the setSelectionCriteria method */
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");
```

Reading Objects Using Partial Object Queries

Example 94–7 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 94–7 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.
Since TopLink automatically includes the primary key of the object, the full
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");
// 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;
```

Reading Objects Using Report Queries

Example 94–8 reports the total and average salaries for Canadian employees grouped by their city.

Example 94–8 Querying Reporting Information on Employees

```
ExpressionBuilder emp = new ExpressionBuilder();
ReportQuery query = new ReportQuery(Employee.class, emp);
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);
```

The `ReportQuery` class provides an extensive reporting API, including methods for computing average, maximum, minimum, sum, standard deviation, variance, and count of attributes. For more information about the available methods for the `ReportQuery`, see the *Oracle TopLink API Reference*.

> **Note:** Because `ReportQuery` inherits from `ReadAllQuery`, it also supports most `ReadAllQuery` properties.

Reading Objects Using Query-By-Example

Query-by-example enables you to specify query selection criteria in the form of a sample object instance that you populate with only the attributes you want to use for the query.

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, optionally, a `QueryByExamplePolicy` (see "Defining a QueryByExamplePolicy" on page 94-7) that specifies configuration settings, such as the operators to use and the attribute values to ignore. You can also combine a query-by-example with an expression (see "Combining Query-by-Example and Expressions" on page 94-8).

For more information, see "Query-by-Example" on page 93-6.

Example 94–9 Using Query-by-Example to Query an Employee

Example 94–9 queries the employee Bob Smith.

```
Employee employee = new Employee();
employee.setFirstName("Bob");
employee.setLastName("Smith");

// Create a query and set Employee as its reference class
ReadObjectQuery query = new ReadObjectQuery(Employee.class);
query.setExampleObject(employee);

Employee result = (Employee) session.executeQuery(query);
```

Example 94–10 Using Query-by-Example to Query an Employee's Address

Example 94–10 queries across the employee's address.

```
Employee employee = new Employee();
Address address = new Address();
address.setCity("Ottawa");
employee.setAddress(address);

// Create a query and set Employee as its reference class
ReadObjectQuery query = new ReadObjectQuery(Employee.class);
query.setExampleObject(employee);

Vector results = (Vector) session.executeQuery(query);
```

Defining a QueryByExamplePolicy

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 do the following:

- 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 94–11 Query-by-Example Policy Using like Operator

Example 94–11 uses `like` operator for strings and includes only objects whose salary is greater than zero.

```
Employee employee = new Employee();
employee.setFirstName("B%");
employee.setLastName("S%");
employee.setSalary(0);

// Create a query and set Employee as its reference class
ReadAllQuery query = new ReadAllQuery(Employee.class);
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 94–12 Query-by-Example Policy Using Keywords

Example 94–12 uses keywords for strings and ignores the value -1.

```
Employee employee = new Employee();
employee.setFirstName("bob joe fred");
employee.setLastName("smith mc mac");
employee.setSalary(-1);

// Create a query and set Employee as its reference class
ReadAllQuery query = new ReadAllQuery(Employee.class);
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 and Expressions

To create more complex query-by-example queries, combine query-by-example with TopLink expressions, as shown in Example 94–13.

Example 94–13 Combining Query-by-Example with Expressions

```
Employee employee = new Employee();
employee.setFirstName("Bob");
employee.setLastName("Smith");

// Create a query and set Employee as its reference class
ReadAllQuery query = new ReadAllQuery(Employee.class);

query.setExampleObject(employee);

// Specify expression
ExpressionBuilder builder = query.getExpressionBuilder();
query.setSelectionCriteria(builder.get("salary").between(100000,200000);
Vector results = (Vector) session.executeQuery(query);
```

Specifying Read Ordering

Ordering is a common `DatabaseQuery` option. Use the Order tab in TopLink Workbench to order the collection of objects returned from a `ReadAllQuery`, or the `addOrdering`, `addAscendingOrdering`, or `addDescendingOrdering` methods in Java code. You can apply order based on attribute names or query keys and expressions.

Example 94–14 A Query with Simple Ordering

```
// Retrieves objects ordered by last name then first name in ascending order
ReadAllQuery query = new ReadAllQuery(Employee.class);
query.addAscendingOrdering ("lastName");
query.addAscendingOrdering ("firstName");
Vector employees = (Vector) session.executeQuery(query);
```

Example 94–15 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(Employee.class);
ExpressionBuilder emp = query.getExpressionBuilder();
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 about configuring read ordering, see "Configuring Read All Query Order" on page 25-15.

Specifying a Collection Class

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, as shown in Example 94–16.

Example 94–16 Specifying the Collection Class for a Collection

```
ReadAllQuery query = new ReadAllQuery(Employee.class);
query.useCollectionClass(LinkedList.class);
LinkedList employees = (LinkedList) getSession().executeQuery(query);
```

Example 94–17 Specifying the Collection Class for a Map

```
ReadAllQuery query = new ReadAllQuery(Employee.class);
query.useMapClass(HashMap.class, "getFirstName");
HashMap employees = (HashMap) getSession().executeQuery(query);
```

Specifying the 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, as shown in Example 94–18.

Example 94–18 Setting the Maximum Returned Object Size

```
ReadAllQuery query = new ReadAllQuery(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 let you 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 "Handling Cursor and Stream Query Results" on page 96-15.

Configuring Query Timeout at the Query Level

You can set the maximum amount of time that TopLink waits for results from a query. This forces a hung or lengthy query to abort after the specified time has elapsed. TopLink throws a `DatabaseException` after the timeout interval.

To specify a timeout interval on a per-query basis, use `DatabaseQuery` method `setQueryTimeout` and pass the timeout interval as an integer representing the number of seconds before the timeout interval should occur, as Example 94–19 shows.

Example 94–19 DatabaseQuery Timeout

```
// Create the appropriate query and set timeout limits
ReadAllQuery query = new ReadAllQuery(Employee.class);
query.setQueryTimeout(2);
try{
    Vector employees = (Vector)session.executeQuery(query);
}
catch (DatabaseException ex) {
    // timeout occurs
}
```

To specify a timeout interval for all queries on a particular object type, configure a query timeout interval at the descriptor level (see "Configuring Query Timeout at the Descriptor Level" on page 25-26).

Using Batch Reading

Batch reading propagates query selection criteria through an object's relationship attribute mappings. You can also nest batch read operations down through complex object graphs. This significantly reduces the number of required SQL select statements and improves database access efficiency.

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 read operations, because they result in multiple joins on the database, slowing query execution.

For more information, see "Reading Case 2: Batch Reading Objects" on page 11-23.

For example, in reading *n* employees and their related projects, TopLink may require *n + 1* select operations. 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 operation by using the original selection criteria, for a total of only two select operations.

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 TopLink 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 back pointer relationship attributes, you can defer back pointer instantiation until the end of the query, when all parent and owning objects are instantiated. This prevents unnecessary database access and optimizes TopLink cache use.

Using Join Reading

Use join reading to configure a query for a class to return the data to build the instances of that class and its related objects. For more information, see "Join Reading and Object-Level Read Queries" on page 93-12.

To add one or more joined attributes to a query, you can use either TopLink Workbench or Java.

Using TopLink Workbench To add one or more joined attributes to a query using TopLink Workbench, configure joined attributes when you define named queries (see "Configuring Named Query Optimization" on page 25-16) or Java. You cannot use TopLink Workbench to create an `ObjectLevelReadQuery` with a join expression on a one-to-many mapped attribute: you must use Java.

Using Java You can use `ObjectLevelReadQuery` API to add joined attributes for one-to-one and one-to-many relationships.

Use the `ObjectLevelReadQuery` method `addJoinedAttribute(Expression attributeExpression)` to add join expressions to the query. Using this method, you can add multiple joined attributes for one-to-one and one-to-many relationships, including nested joins. The source and target can be the same class type. You cannot use the `ObjectLevelReadQuery` method `addJoinedAttribute` with a join expression on a many-to-many mapped attribute.

- Use the `ObjectLevelReadQuery` method `addJoinedAttribute` with a join expression on a one-to-one mapped attribute to get the class of the `ObjectLevelReadQuery` and the target of the one-to-one mapped attribute of that class with a single database hit.
- Use the `ObjectLevelReadQuery` method `addJoinedAttribute` with a join expression on a one-to-many mapped attribute to get the class of the `ObjectLevelReadQuery` and the target collection of the one-to-many mapped attribute of that class with a single database hit.

Example 94–20 is based on the TopLink `ThreeTierEmployee` example project. It shows a `ReadAllQuery` configured to join-read multiple attributes. This query produces the SQL that Example 94–21 shows.

Example 94–20 Join Reading Multiple Attributes

```
ReadAllQuery query = new ReadAllQuery(Employee.class);

Expression managedEmployees = query.getExpressionBuilder().anyOfAllowingNone(
    "managedEmployees"
);
query.addJoinedAttribute(managedEmployees);
query.addJoinedAttribute(managedEmployees.get("address"));
query.addJoinedAttribute(managedEmployees.anyOf("phoneNumbers"));

Vector employees = (Vector)getSession().executeQuery(query);
```

Example 94–21 SQL for Multiple Attribute Join Reading

```
SELECT DISTINCT
    t2.VERSION, t3.EMP_ID, t2.GENDER, t3.SALARY, t2.EMP_ID, t2.F_NAME, t2.L_NAME,
    t2.MANAGER_ID, t2.ADDR_ID, t2.END_DATE, t2.START_DATE, t2.END_TIME,
    t2.START_TIME, t0.VERSION, t1.EMP_ID, t0.GENDER, t1.SALARY, t0.EMP_ID,
    t0.F_NAME, t0.L_NAME, t0.MANAGER_ID, t0.ADDR_ID, t0.END_DATE, t0.START_DATE,
    t0.END_TIME, t0.START_TIME
FROM
    SALARY t3, EMPLOYEE t2, SALARY t1, EMPLOYEE t0
WHERE
    ((t3.EMP_ID = t2.EMP_ID) AND
     ((t0.MANAGER_ID (+) = t2.EMP_ID) AND
     (t1.EMP_ID (+) = t0.EMP_ID)))
```

Use the `ObjectLevelReadQuery` method `addJoinedAttribute(java.lang.String attributeName)` to configure the query to join-read a single attribute, as Example 94–22 shows.

Example 94–22 Join Reading a Single Attribute

```
ReadAllQuery query = new ReadAllQuery(Employee.class);
query.addJoinedAttribute("address");
Vector employees = (Vector)getSession().executeQuery(query);
```

Creating, Updating, and Deleting Objects With a DatabaseQuery

This section describes the following:

- Write Query Overview
- UpdateAll Queries
- Noncascading Write Queries
- Disabling the Identity Map Cache During a Write Query

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 94–23 Using a WriteObjectQuery

```
WriteObjectQuery writeQuery = new WriteObjectQuery();
writeQuery.setObject(domainObject);
session.executeQuery(writeQuery);
```

Example 94–24 Using InsertObjectQuery, UpdateObjectQuery, and DeleteObjectQuery

```
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);
```

UpdateAll Queries

Use an `UpdateAllQuery` to update a large number of objects at once. With this query, you can update a large number of objects with a single SQL statement, instead of reading the objects into memory and updating them individually. Example 94–25 shows an `UpdateAllQuery` to give all full-time employees a raise.

Example 94–25 Using UpdateAllQuery

```
// Give all full time employees a 10% raise
UpdateAllQuery updateQuery = new UpdateAllQuery(Employee.class);
ExpressionBuilder employee = updateQuery.getExpressionBuilder();
updateQuery.setSelectionCriteria(eb.get("status").equal("FULL_TIME"));
updateQuery.addUpdateExpression(employee.get("salary"),
    ExpressionMath.multiply(employee.get("salary"), new Float(1.10)));
```

Since multiple tables cannot be updated from the same SQL statement, the `UpdateAllQuery` does not support objects that span multiple tables, or inheritance. Additionally, the `UpdateAllQuery` must be executed from its own transaction–the

unit of work must contain only the query. Use the `UnitOfWork` method `executeQuery`.

In a non-JTA transaction, TopLink provides support for the unit of work and session execute; in a JTA transaction, only the unit of work is supported–there is no support for the session execute. For more information on transactions, see Chapter 97, "Understanding TopLink Transactions".

`UpdateAllQuery` takes the cache into consideration and ensures that the cache is kept up to date. You can configure the `UpdateAllQuery` to invalidate cache (see "Cache Invalidation" on page 87-8) by setting the cache usage to `INVALIDATE_CACHE` (default), or to not use the cache by specifying `NO_CACHE` option. You can manipulate these settings through the `setCacheUsage` method. You can only update the cache for expressions that can conform. For more information on cache, see Chapter 87, "Understanding the Cache".

UpdateAll queries only support unidirectional one-to-one relationships; there is a full support for direct, and a partial support for aggregate mappings (see Part IX, "Mappings").

Note: You can set an attribute within an aggregate only, but not an entire aggregate.

These queries do not support foreign key updates, therefore you cannot use these queries to set foreign key fields to `null`.

You can use an UpdateAll query with optimistic locking (see "Understanding Descriptors and Locking" on page 23-18) at the level of updating a row in a database–there should be no updates in the cache. we will update the locking field on the database. There is also support for version and timestamp locking, as well as indirect support for field locking.

Noncascading 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 do the following:

- 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 94–26 Performing a Noncascading Write Query

```
// the Employee 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, TopLink copies them to the session cache by default. To disable this within a query, call the `dontMaintainCache` method within the query. This improves query performance when you insert objects into the database, but must be used only on objects that will not be required later by the application.

Example 94–27 Disabling the Identity Map Cache During a Write Query

Example 94–27 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);
   }
}
```

> **Note:** Disable the identity map only when object identity is unimportant in subsequent operations.

Reading Data With a DatabaseQuery

This section describes the following:

- Using a DataReadQuery
- Using a DirectReadQuery
- Using a ValueReadQuery

Using a DataReadQuery

You can use a `DataReadQuery` to execute a selecting SQL string that returns a `Collection` of the `DatabaseRows` representing the result set, as Example 94–28 shows.

Example 94–28 Using a DataReadQuery

```
DataReadQuery dataReadQuery = new DataReadQuery();
dataReadQuery.setSQLStatement(sqlStatement);

// queryResults is a Vector of DatabaseRow objects
Vector queryResults = (Vector)session.executeQuery(dataReadQuery);
```

Using a DirectReadQuery

You can use a `DirectReadQuery` to read a single column of data (that is, one field) that returns a `Collection` of the `DatabaseRows` representing the result set, as Example 94–29 shows.

Example 94–29 Using a DirectReadQuery

```
DirectReadQuery directReadQuery = new DirectReadQuery();
directReadQuery.setSQLStatement(sqlStatement);

// queryResults is a Vector of DatabaseRow objects
Vector queryResults = (Vector)session.executeQuery(directReadQuery);
```

Using a ValueReadQuery

You can use a `ValueReadQuery` to read a single data value (that is, one field). A single data value is returned, or null if no rows are returned, as Example 94–30 shows.

Example 94–30 Using a ValueReadQuery

```
ValueReadQuery valueReadQuery = new ValueReadQuery();
valueReadQuery.setSQLString("SELECT DISTINCT CURRENT TIMESTAMP FROM SYSTABLES");

// result is a single Object value
Object result = session.executeQuery(valueReadQuery);
```

WARNING: Allowing an unverified SQL string to be passed into methods (for example: `setSQLString` method) makes your application vulnerable to SQL injection attacks.

Updating Data With a DatabaseQuery

You can use a `DataModifyQuery` to execute a nonselecting SQL statement (directly or as an `SQLCall`), as Example 94–31 shows. This is equivalent to `Session` method `executeNonSelectingCall` (see "Using an SQLCall" on page 94-19).

Example 94–31 Using a DataModifyQuery

```
DataModifyQuery query = new DataModifyQuery(new SQLCall("Delete from Employee"));
session.executeQuery(query);
```

Specifying a Custom SQL String in a DatabaseQuery

All `DatabaseQuery` objects provide a `setSQLString` method that you can use to define a custom SQL string.

For more information about using custom SQL in queries, see "Using SQL Calls" on page 94-19.

Example 94–32 uses SQL to read all employee IDs.

Example 94–32 A Direct Read Query with SQL

```
DirectReadQuery query = new DirectReadQuery();
query.setSQLString("SELECT EMP_ID FROM EMPLOYEE");
Vector ids = (Vector) session.executeQuery(query);
```

Example 94–33 uses SQL to switch to a different database.

Example 94–33 A Data Modify Query with SQL

```
DataModifyQuery query = new DataModifyQuery();
query.setSQLString("USE SALESDATABASE");
session.executeQuery(query);
```

WARNING: Allowing an unverified SQL string to be passed into methods (for example: `setSQLString` method) makes your application vulnerable to SQL injection attacks.

Specifying a Custom EJB QL String in a DatabaseQuery

All `DatabaseQuery` objects provide a `setEJBQLString` method that you can use to specify a custom EJB QL string.

For more information about using custom EJB QL in queries, see "Using EJB QL Calls" on page 94-24.

Provide both a reference class and a `SELECT` clause, and execute the query in the usual manner.

Example 94–34 EJB QL

```
ReadAllQuery query = new ReadAllQuery(EmployeeBean.class);
query.setEJBQLString("SELECT OBJECT(emp) FROM EmployeeBean emp");
…
Vector returnedObjects = (Vector)session.executeQuery(query);
```

Example 94–35 defines the query similarly to Example 94–34, but creates, fills, and passes a vector of arguments to the `executeQuery` method.

Example 94–35 A Simple ReadAllQuery Using EJB QL and Passing Arguments

```
// First define the query
ReadAllQuery query = new ReadAllQuery(EmployeeBean.class);
query.setEJBQLString("SELECT OBJECT(emp) FROM EmployeeBean emp WHERE emp.firstName = ?1");
query.addArgument("1", String.class);
...
// Next define the arguments
Vector arguments = new Vector();
arguments.add("Bob");
...
// Finally, execute the query passing in the arguments
Vector returnedObjects = (Vector)session.executeQuery(query, arguments);
```

Using Parameterized SQL and Statement Caching in a DatabaseQuery

To enable the parameterized SQL on individual queries, use `DatabaseQuery` methods `bindAllParameters` and `cacheStatement`. This causes TopLink to use a prepared statement, binding all SQL parameters and caching the prepared statement. When you reexecute this query, you avoid the SQL preparation, which improves performance.

Example 94–36 A Simple ReadObjectQuery with Parameterized SQL

```
ReadObjectQuery query = new ReadObjectQuery(Employee.class);
query.setShouldBindAllParameters(true);
query.setShouldCacheStatement(true);
```

Alternatively, you can configure parameterized SQL and binding at the `Login` level for all queries (see "Configuring JDBC Options" on page 83-9).

For more information about using parameterized SQL and binding for data access optimization, see "Parameterized SQL (Binding) and Prepared Statement Caching" on page 11-15.

> **Note:** For applications using a J2EE data source or external connection pool, you must configure statement caching in the J2EE server's data source–not in TopLink.

Using Named Queries

Named queries improve application performance because they are prepared once and they (and all their associated supporting objects) can be efficiently reused thereafter making them well suited for frequently executed operations.

You can configure named queries at the session (see "Configuring Named Queries at the Session Level" on page 74-21) or descriptor (see "Configuring Named Queries at the Descriptor Level" on page 25-10) level.

For a session-level named query, you can execute the query using any of the following `Session` API calls:

- `executeQuery(String queryName)`
- `executeQuery(String queryName, arg1)`
- `executeQuery(String queryName, arg1, arg2)`
- `executeQuery(String queryName, arg1, arg2, arg3)`
- `executeQuery(String queryName, Vector args)`

Example 94–37 Executing a Session-Level Named Query

```
Vector args = new Vector();
args.add("Sarah");
Employee sarah = (Employee)session.executeQuery(
    "employeeReadByFirstName",
    args
);
```

For a descriptor-level named query, you can execute the query using any of the following `Session` API calls, as Example 94–38 shows:

- `executeQuery(String queryName, Class domainClass)`
- `executeQuery(String queryName, Class domainClass, arg1)`
- `executeQuery(String queryName, Class domainClass, arg1, arg2)`
- `executeQuery(String queryName, Class domainClass, arg1, arg2, arg3)`
- `executeQuery(String queryName, Class domainClass, Vector args)`

Example 94–38 Executing a Descriptor Level Named Query

```
Vector args = new Vector();
args.add("Sarah");
Employee sarah = (Employee)session.executeQuery(
    "ReadByFirstName",
    Employee.class,
    args
);
```

For more information, see "Named Queries" on page 93-16

Using SQL Calls

The TopLink expression framework enables you to define complex queries at the object level. If your application requires a more complex query or one that accesses data or stored procedures directly, you can specify a custom SQL string in an SQL `Call` object and provide that `Call` object to any query.

You can also specify an SQL string directly on `DatabaseQuery`. For more information, see "Specifying a Custom SQL String in a DatabaseQuery" on page 94-16.

When using SQL calls, you can use a `ReturningPolicy` to control whether or not TopLink writes a parameter out or retrieves a value generated by the database. For more information, see "Configuring Returning Policy" on page 25-67.

This section describes the following:

- Using an SQLCall
- Using a StoredProcedureCall
- Using a StoredFunctionCall

Using an SQLCall

You can provide an `SQLCall` object to any query instead of an expression, but the SQL string contained in the `SQLCall` must return all data required to build an instance of the queried class.

The SQL string can be a complex SQL query, a stored procedure call, or a stored function call. You can specify input, output, and input/output parameters.

You can invoke an `SQLCall` through a session query method (as Example 94–39 illustrates) or through a `DatabaseQuery`.

Example 94–39 Session Read Query With Custom SQL

```
List result = session.executeSelectingCall(
    new SQLCall("SELECT * FROM EMPLOYEE WHERE EMP_ID = 44"));
```

WARNING: Allowing an unverified SQL string to be passed into methods makes your application vulnerable to SQL injection attacks.

TopLink assumes that a token in the custom SQL string of an `SQLCall` is a parameter if it is prefixed with one or more number signs (#). You can bind values to these parameters using query API, as the following sections describe:

- Specifying a SQLCall Input Parameter
- Specifying a SQLCall Output Parameter
- Specifying a SQLCall Input / Output Parameter

Specifying a SQLCall Input Parameter

In Example 94–40, you specify `last_name` as an input parameter by prefixing its name with one number sign (#). Example 94–41 shows how to bind a value to this input parameter when you execute the query.

Example 94–40 Specifying an SQLCall with an Input Parameter Using the # Prefix

```
SQLCall sqlCall = new SQLCall(
    "INSERT INTO EMPLOYEE (L_NAME) VALUES (#last_name)");
```

Example 94–41 Executing an SQLCall with an Input Parameter

```
DataModifyQuery query = new DataModifyQuery();
query.setCall(sqlCall);
query.addArgument("last_name");   // input

Vector arguments = new Vector();
arguments.add("MacDonald");
session.executeQuery(query, arguments);
```

Specifying a SQLCall Output Parameter

In Example 94–42, you specify `employee_id` as an output parameter by prefixing its name with three number signs (###). You specify the type of the output parameter with `SQLCall` method `setCustomSQLArgumentType`. You continue to specify `last_name` as an input parameter by prefixing its name with a number sign (#).

Example 94–42 Specifying a SQLCall with an Output Parameter Using the ### Prefix

```
SQLCall sqlCall = new SQLCall("begin;
    INSERT INTO EMPLOYEE (L_NAME) VALUES (#L_NAME) RETURNING EMP_ID INTO ###employee_id;
    end");
sqlCall.setCustomSQLArgumentType("employee_id", Integer.class);
```

Example 94–43 Executing a SQLCall with an Output Parameter

```
ValueReadQuery query = new ValueReadQuery();
query.setCall(sqlCall);
query.addArgument("last_name");   // input

Vector args = new Vector();
args.add("MacDonald");

Number employeeID = (Number)getSession().executeQuery(query, args);
```

You can also obtain results for an output parameter declared to be of type `CURSOR`.

Specifying a SQLCall Input / Output Parameter

In Example 94–44, you specify `in_out` as an input and output parameter by prefixing its name with four number signs (####). The type of the input value determines the type of the output value. In this example, a `String` ("MacDonald") is passed in and the output value (for `EMP_ID`) is returned as a `String`.

Example 94–44 Specifying an Input and Output Parameter Using the #### Prefix

```
SQLCall sqlCall = new SQLCall(
    "INSERT INTO EMPLOYEE (L_NAME) VALUES (####in_out) RETURNING EMP_ID INTO ####in_out");
```

Example 94–45 Executing a SQLCall with an Input and Output Parameter

```
ValueReadQuery query = new ValueReadQuery();
query.setCall(sqlCall);
query.addArgument("in_out");   // input and outpu

Vector args = new Vector();
args.add("MacDonald");

Number employeeID = (Numbere)getSession().executeQuery(query, args);
```

Using a StoredProcedureCall

You can provide a `StoredProcedureCall` object to any query instead of an expression or a SQL string, but the procedure must return all data required to build an instance of the class you query.

Example 94–46 A Read-All Query with a Stored Procedure

```
ReadAllQuery readAllQuery = new ReadAllQuery();
call = new StoredProcedureCall();
call.setProcedureName("Read_All_Employees");
call.useNamedCursorOutputAsResultSet("RESULT_SET");
readAllQuery.setCall(call);
Vector employees = (Vector) session.executeQuery(readAllQuery);
```

Using a `StoredProcedureCall`, you can access the following:

- Specifying an Input Parameter
- Specifying an Output Parameter
- Specifying an Input / Output Parameter
- Using an Output Parameter Event

> **Note:** You no longer need to use `DatabaseQuery` method `bindAllParameters` when using a `StoredProcedureCall` with `OUT` or `INOUT` parameters. However, you should always specify the Java type for all `OUT` and `INOUT` parameters. If you do not, be aware of the fact that they default to type `String`.

Specifying an Input Parameter

In Example 94–47, you specify the parameter `POSTAL_CODE` as an input parameter using the `StoredProcedureCall` method `addNamedArgument`, and you can specify the value of the argument using method `addNamedArgumentValue`.

Example 94–47 Stored Procedure Call with an Input Parameter

```
StoredProcedureCall call = new StoredProcedureCall();
call.setProcedureName("CHECK_VALID_POSTAL_CODE");
call.addNamedArgument("POSTAL_CODE");
call.addNamedArgumentValue("POSTAL_CODE", "L5J1H5");
call.addNamedOutputArgument(
    "IS_VALID",    // procedure parameter name
```

```
    "IS_VALID",    // out argument field name
    Integer.class  // Java type corresponding to type returned by procedure
);
ValueReadQuery query = new ValueReadQuery();
query.setCall(call);
Number isValid = (Number)session.executeQuery(query);
```

Specifying an Output Parameter

Output parameters enable the stored procedure to return additional information. You can use output parameters to define a `ReadObjectQuery` if they return all the fields required to build the object.

In Example 94–48, you specify the parameter IS_VALID as an output parameter using the `StoredProcedureCall` method `addNamedOutputArgument`.

Example 94–48 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",    // procedure parameter name
    "IS_VALID",    // out argument field name
    Integer.class  // Java type corresponding to type returned by procedure
);
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);
```

> **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.

If you are using an Oracle database, you can make use of TopLink cursor and stream query results.

Specifying an Input / Output Parameter

In Example 94–49, you specify the parameter `LENGTH` as an input/output parameter and specify the value of the argument when it is passed to the stored procedure using the `StoredProcedureCall` method `addNamedInOutputArgumentValue`. If you do not want to specify a value for the argument, use method `addNamedInOutputArgument`.

Example 94–49 Stored Procedure Call with an Input/Output Parameter

```
StoredProcedureCall call = new StoredProcedureCall();
call.setProcedureName("CONVERT_FEET_TO_METERs");
call.addNamedInOutputArgumentValue(
    "LENGTH",          // procedure parameter name
    new Integer(100),  // in argument value
    "LENGTH",          // out argument field name
    Integer.class      // Java type corresponding to type returned by procedure
)
```

```
ValueReadQuery query = new ValueReadQuery();
query.setCall(call);
Integer metricLength = (Integer)session.executeQuery(query);
```

Using an Output Parameter Event

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, TopLink raises the session event `OutputParametersDetected` to allow the application to process the output parameters.

Example 94–50 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",  // procedure parameter name
    "ERROR_CODE",  // out argument field name
    Integer.class  // Java type corresponding to type returned by procedure
);
call.useNamedCursorOutputAsResultSet("RESULT_SET");
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);
```

Using a StoredFunctionCall

You use a `StoredProcedureCall` to invoke stored procedures defined on databases that support them. You can also use a `StoredFunctionCall` to invoke stored functions defined on databases that support them, that is, on databases for which the `DatabasePlatform` method `supportsStoredFunctions` returns `true`.

In general, both stored procedures and stored functions let you specify input parameters, output parameters, and input and output parameters. For more information, see "Using a StoredProcedureCall" on page 94-21. However, stored procedures need not return values, while stored functions always return a single value.

The `StoredFunctionCall` class extends `StoredProcedureCall` to add one new method: `setResult`. Use this method to specify the name (and alternatively both the name and type) under which TopLink stores the return value of the stored function.

When TopLink prepares a `StoredFunctionCall`, it validates its SQL and throws a `ValidationException` under the following circumstances:

- If your current platform does not support stored functions. Stored functions are supported only for Oracle.
- If you fail to specify the return type

In Example 94–51, note that the name of the stored function is set using `StoredFunctionCall` method `setProcedureName`.

Example 94–51 Creating a StoredFunctionCall

```
StoredFunctionCall functionCall = new StoredFunctionCall();
functionCall.setProcedureName("CHECK_VALID_EMPLOYEE");
functionCall.addNamedArgument("EMP_ID");
functionCall.setResult("FUNCTION_RESULT", String.class);
ValueReadQuery query = new ValueReadQuery();
query.setCall(functionCall);
query.addArgument("EMP_ID");
Vector args = new Vector();
args.addElement(new Integer(44));
String valid = (String) session.executeQuery(query, args);
```

Using EJB QL Calls

The TopLink expression framework lets you define complex queries at the object level. Alternatively, you can specify a custom EJB QL string in an EJB QL `Call` object and provide that `Call` object to any query.

You can also specify an EJB QL string directly in a `DatabaseQuery`. For more information, see "Specifying a Custom EJB QL String in a DatabaseQuery" on page 94-17.

You can provide an `EJBQLCall` object to any query instead of an expression or EJB QL string, but the procedure must return all data required to build an instance of the class you query.

You can invoke EJB QL queries through the session query methods or through a `DatabaseQuery`.

Example 94–52 Session Read Query With Custom EJB QL

```
Vector theObjects = (Vector)aSession.readAllObjects(
    EmployeeBean.class,
    new EJBQLCall("SELECT OBJECT (emp) from EmployeeBean emp"));
```

Using EIS Interactions

For an EIS root descriptor, you can define EIS interactions to invoke methods on an EIS.

TopLink represents EIS interactions using instances of `oracle.toplink.eis.interactions.EISInteraction`. These classes implement the `Call` interface and can be used wherever a `Call` can be used.

Table 94–1 lists the type of EIS interactions that TopLink supports.

Table 94–1 EIS Interactions

EIS Interaction Type	Description
`IndexedInteraction`	Defines the specification for a call to a J2C interaction that uses indexed records. Builds the input and output records from the arguments by position.
`MappedInteraction`	Defines the specification for a call to a J2C interaction that uses mapped records. Builds the input and output records from the arguments by name.

Table 94–1 (Cont.) EIS Interactions

EIS Interaction Type	Description
`XMLInteraction`	Specifies an instance of `MappedInteraction` that defines the specification for a call to a J2C interaction that uses XML records defined by the XML schema document (XSD) associated with the EIS project (for more information, see "Importing an XML Schema" on page 4-34).
`QueryStringInteraction`	Specifies an instance of `MappedInteraction` that defines the specification for a call to a J2C interaction that uses a query string. Prefix arguments in the query string with a number sign (#) character.
`XQueryInteraction`	Specifies an instance of `XMLInteraction` that defines the specification for a call to a J2C interaction that uses XQuery. Translates the XQuery from the query arguments.

You can use TopLink to define an interaction for each basic persistence operation (`insert`, `update`, `delete`, `read object`, `read all`, or `does exist`) so that when you query and modify your EIS-mapped objects, the TopLink runtime will use the appropriate EIS interaction. For more information, see "Configuring Custom EIS Interactions for Basic Persistence Operations" on page 28-6.

You can also use TopLink to define an interaction as a named query for read object and read-all object queries. These queries are not called for basic persistence operations; you can call these additional queries by name in your application for special purposes. For more information, see "Creating an EIS Interaction for a Named Query" on page 25-20.

Handling Exceptions

Most exceptions in queries are database exceptions, resulting from a failure in the database operation (see "Database Exceptions (4002 – 4018)" on page 13-27). 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
}
```

See Chapter 14, "TopLink Workbench Error Reference" for more information about exceptions in a TopLink application.

Handling Collection Query Results

TopLink provides a `useCollectionClass` method to all subclasses of `DataReadQuery` and `ReadAllQuery`, that you can use to configure a query to return results as any concrete instance of `Collection` or `Map`.

Do not confuse collection query result configuration with a mapping container policy (see "Configuring Container Policy" on page 32-26): there is no relationship between the two. Collection query result configuration determines how TopLink returns multiobject results from a particular query. A mapping container policy tells TopLink how your domain object implements a data member that contains a collection.

For example, consider a class `Employee` with a data member `phoneNumbers`. In your implementation of `Employee`, the `getPhoneNumbers` method returns a `Vector`. Using TopLink Workbench, you map the `phoneNumbers` data member as a one-to-many mapping. You configure the mapping container policy so that the mapping contains its value (many `PhoneNumber` objects) in a `Vector`. This corresponds to your implementation of `Employee`.

You define a `ReadAllQuery` named `localPhoneNumbers` on the `DescriptorQueryManager` of the `PhoneNumber`. The `localPhoneNumbers` query takes one argument, the ID of an `Employee` object, and returns all the phone numbers from its `phoneNumbers` data member whose area code is 613.

You get this query by name from the `DescriptorQueryManager` for `PhoneNumber`. You call the `useCollectionClass` method on this `ReadAllQuery`, passing in the `ArrayList` class. You execute the query, passing in the ID of an `Employee`. The query returns all the `PhoneNumber` objects from the `Employee` object's `phoneNumbers` data member whose area code is 613. The query returns these results as an `ArrayList`.

Handling Report Query Results

Table 94–2 lists the `ReportQuery` methods you can use to configure how a `ReportQuery` returns its results.

Table 94–2 Report Query Result Options

Method	Query Returns	Description
`setShouldReturnSingleAttribute`	`DatabaseRow`	Returns a single attribute (not wrapped in a `ReportQueryResult`). Use this option if you know that the `ReportQuery` returns only one attribute.
`setShouldReturnSingleResult`	`ReportQueryResult`	Returns only the first `ReportQueryResult` object (not wrapped in a `Collection` or `Map`). Use this option if you know that the `ReportQuery` returns only one row.
`setShouldReturnSingleValue`	Object	Returns only a single value. Use this option if you know that the `ReportQuery` returns only one row that contains only one attribute.

For more information, see the following:

- "Report Query" on page 93-15
- "Report Query Results" on page 93-8
- "Reading Objects Using Report Queries" on page 94-6
- "Configuring Named Query Attributes" on page 25-17

95

Understanding TopLink Expressions

Using the TopLink expressions framework, you can specify query search criteria based on your domain object model. This section describes the following:

- Understanding the Expression Framework
- Expression Components
- Parameterized Expressions
- Query Keys and Expressions
- Using Multiple Expressions
- Data Queries and Expressions
- Creating an Expression
- Creating and Using a User-Defined Function

Understanding the Expression Framework

The 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.

Example 95–1 illustrates how to use the `Expression` class.

Example 95–1 Using the Expression Class

```
expressionBuilder.get("lastName").equal("Smith");
```

Example 95–2 illustrates how to use the `ExpressionMath` class.

Example 95–2 Using the ExpressionMath Class

```
ExpressionMath.abs(ExpressionMath.subtract(emp.get("salary"),
emp.get("spouse").get("salary")).greaterThan(10000)
```

This division of functionality enables TopLink expressions to provide similar mathematical functionality to the `java.lang.Math` class, but keeps both the `Expression` and `ExpressionMath` classes from becoming unnecessarily complex.

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, 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)));
  ```

  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))
  ```

Expression Components

A simple expression usually consists of the following three parts:

1. The *attribute*, which represents a mapped attribute or query key of the persistent class
2. The *operator*, which is an expression method that implements boolean logic, such as `GreaterThan`, `Equal`, or `Like`
3. 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.

You can use the following components when constructing an `Expression`:

- Boolean Logic
- Database Functions and Operators

- Mathematical Functions
- XMLType Functions
- Platform and User-Defined Functions
- Expressions for One-to-One and Aggregate Object Relationships
- Expressions for Joining and Complex Relationships

Boolean Logic

Expressions use standard boolean operators, such as `AND`, `OR`, and `NOT`, and you can combine multiple expressions to form more complex expressions. Example 95–3 illustrates a code fragment that queries for projects managed by a selected person, and that have a budget greater than or equal to 1,000,000.

Example 95–3 Using Boolean Logic in an Expression

```
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 and Operators

Functions

TopLink expressions support a variety of database functions, including, but not limited to, the following:

- `toUpperCase`
- `toLowerCase`
- `toDate`
- `decode`
- `locate`
- `monthsBetween`
- `nextDay`
- `replace`
- `reverse`
- `substring`
- `translate`

> **Note:** Some functions may be database platform specific.

Database functions let you define more flexible queries. You can use these functions in either a report query items using a `SELECT` clause, or with comparisons in a query's selection criteria using a `WHERE` clause. Example 95–4 illustrates a code fragment that matches several last names, including "SMART", "Smith", and "Smothers":

Example 95–4 Using a Database Function Supported by the Expression API

```
emp.get("lastName").toUpperCase().like("SM%")
```

You access most functions using `Expression` methods such as `toUpperCase`.

Some functions have very specific purpose: you can use `ascending` and `descending` functions only within an ordering expression to place the result in ascending or descending order, as Example 95–5 shows:

Example 95–5 Using an Ordering Database Function

```
readAllQuery.addOrderBy(expBuilder.get("address").get("city").ascending())
```

You can use aggregate functions, such as `average`, `minimum`, `maximum`, `sum` and so forth, with `ReportQuery` (see "Report Query" on page 93-15).

Operators

Operators are relation operations that compare two values. TopLink expressions support the following operators:

- `like`
- `notLike`
- `equal`
- `notEqual`
- `lessThan`
- `lessThanEqual`
- `equalsIgnoreCase`
- `greaterThan`
- `greaterThanEqual`
- `in`
- `notIn`
- `between`
- `notBetween`

Example 95–4 demonstrates the use of the `like` operator.

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`. Example 95–6 illustrates using the `abs` and `subtract` methods.

Example 95–6 Using Mathematical Functions in an Expression

```
ExpressionMath.abs(ExpressionMath.subtract(emp.get("salary"),emp.get("spouse")
  .get("salary")).greaterThan(10000)
```

XMLType Functions

You can use the following operators when constructing queries against data mapped to an Oracle Database XMLType column:

- `extract`: Takes an XPath string and returns an XMLType which corresponds to the part of the original document that matches the XPath.
- `extractValue`: Takes an Xpath string and returns either a numerical or string value based on the contents of the node pointed to by the XPath.
- `existsNode`: Takes an Xpath expression and returns the number of nodes that match the Xpath.
- `getStringVal`: Gets the string representation of an XMLType object.
- `getNumberVal`: Gets the numerical representation of an XMLType object.
- `isFragment`: Evaluates to 0 if the XML is a well formed document. Evaluates to 1 if the document is a fragment.

Example 95–7 illustrates how to use the extract operator in a query:

Example 95–7 Using the XMLType Extract Operator

```
Expression criteria =
builder.get("resume").extract("//education/degree/text()").getStringVal().equal("BCS");
Vector employees = session.readAllObject(Employee.class, criteria);
```

Platform and User-Defined Functions

You can use the `Expression` method `getFunction` to access database functions that TopLink does not support directly. Example 95–8 illustrates how to access a database function named `VacationCredit` from within an expression, even though there is no support for such a function in the `Expression` API.

Example 95–8 Using a Database Function Not Supported by the Expression API

```
emp.get("lastName").getFunction("VacationCredit").greaterThan(42)
```

This expression produces the following SQL:

```
SELECT . . . WHERE VacationCredit(EMP.LASTNAME) > 42
```

The `Expression` API includes additional forms of the `getFunction` method that allow you to specify arguments. For more information, see *Oracle TopLink API Reference*.

You can also access a custom function that you create. For more information on creating a custom function in TopLink, see "Creating and Using a User-Defined Function" on page 95-16.

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 relationship translates naturally into a SQL join that returns a single row.

Example 95–9 illustrates a code fragment that accesses fields from an employee's address.

Example 95–9 Using an Expression with a One-to-One Relationship

```
emp.get("address").get("country").like("S%")
```

Example 95–9 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 Joining and 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.

This section describes the following:

- Understanding Joins
- Using TopLink Expression API For Joins

Understanding Joins

A **join** is a relational database query that combines rows from two or more tables. Relational databases perform a join whenever multiple tables appear in the query's `FROM` clause. The query's select list can select any columns from any of these tables.

An inner join (sometimes called a "simple join") is a join of two or more tables that returns only those rows that satisfy the join condition.

An outer join extends the result of an inner join. An outer join returns all rows that satisfy the join condition and also returns some or all of those rows from one table for which no rows from the other satisfy the join condition. Outer joins can be categorized as left or right:

- A query that performs a left outer join of tables A and B returns all rows from A. For all rows in A that have no matching rows in B, the database returns null for any select list expressions containing columns of B.
- A query that performs a right outer join of tables A and B returns all rows from B. For all rows in B that have no matching rows in A, the database returns null for any select list expressions containing columns of A.

When you query with a join expression, TopLink 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 following 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 `Expression` method `getAllowingNull`, rather than `get`, and `Expression` method `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. 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.

Using TopLink Expression API For Joins

You can use joins anywhere expressions are used, including: selection-criteria, ordering (see "Specifying Read Ordering" on page 94-9), report queries (see "Report Query" on page 93-15), partial objects (see "Partial Object Queries" on page 93-11), one-to-one relational mappings (see "Configuring Joining at the Mapping Level" on page 37-1), and join reading (see "Join Reading and Object-Level Read Queries" on page 93-12).

Use the expression API shown in Table 95–1 to configure inner and outer join expressions.

Table 95–1 Expression API for Joins

Expression API	Type of Join	Type of Mapping
`get`	inner	one-to-one
`getAllowingNull`	outer	one-to-one
`anyOf`	inner	one-to-many, many-to-many
`anyOfAllowingNone`	outer	one-to-many, many-to-many

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.

Example 95–10 illustrates an expression that returns employees who manage at least one employee (through a one-to-many relationship) with a salary less than $10,000.

Example 95–10 Using an Expression with a One-to-Many Relationship

```
emp.anyOf("managedEmployees").get("salary").lessThan(10000);
```

Example 95–11 illustrates how to query across a many-to-many relationship using a similar strategy:

Example 95–11 Using an Expression with a Many-to-Many Relationship

```
emp.anyOf("projects").equal(someProject)
```

TopLink translates these queries into SQL that joins the relevant tables using a `DISTINCT` clause to remove duplicates. TopLink translates Example 95–10 into the following SQL:

```
SELECT DISTINCT . . . FROM EMP t1, EMP t2 WHERE
t2.MANAGER_ID = t1.EMP_ID AND t2.SALARY < 10000
```

You can use one-to-one and one-to-many join expressions in an `ObjectLevelReadQuery` to configure joins on a per-query basis (see "Join Reading and Object-Level Read Queries" on page 93-12).

You can also configure joins at the mapping level (see "Configuring Joining at the Mapping Level" on page 37-1).

Note: Calling `anyOf` once would result in a different outcome than if you call it twice. For example, if you query for an employee with a telephone area code of 613 and a number of 123-4599, you would use a single `anyOf` and a temporary variable. If you query for an employee, whose telephone has an area code of 613, and whose telephone has a number of 123-4599, you would call anyOf twice.

Parameterized Expressions

A relationship mapping differs from a regular query because it retrieves data for many different objects. To be able 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 lets you build expressions that retrieve context-sensitive information. This technique is useful when you define EJB finders (see "EJB Finders" on page 93-24).

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 TopLink constructs automatically based on the information provided in the mapping. To specify expressions yourself, use the mapping customization mechanisms.

You can use parameterized expressions to create reusable queries (see "Named Queries" on page 93-16).

Expression Method getParameter

The `Expression` method `getParameter` returns an expression that becomes a parameter in the query. This lets you create a query that includes user input in 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 95–12 illustrates how to use a custom query to find an employee by first name.

Example 95–12 Using a Parameterized Expression in a Custom Query

```
Expression firstNameExpression;
```

```
ReadObjectQuery query = new ReadObjectQuery(Employee.class);
ExpressionBuilder emp = query.getExpressionBuilder();
firstNameExpression = emp.get("firstName").equal(emp.getParameter("firstName"));
query.setSelectionCriteria(firstNameExpression);
query.addArgument("firstName");
Vector v = new Vector();
v.addElement("Sarah");
Employee e = (Employee) session.executeQuery(query, v);
```

Example 95–13 illustrates how to use a custom query to find all employees that live in the same city as a given employee.

Example 95–13 Using Nested Parameterized Expressions

```
Expression addressExpression;
ReadObjectQuery query = new ReadObjectQuery(Employee.class);
ExpressionBuilder emp = query.getExpressionBuilder();
addressExpression =
    emp.get("address").get("city").equal(
    emp.getParameter("employee").get("address").get("city"));
query.setName("findByCity");
query.setSelectionCriteria(addressExpression);
query.addArgument("employee");
Vector v = new Vector();
v.addElement(employee);
Employee e = (Employee) session.executeQuery(query, v);
```

Example 95–14 illustrates how to obtain 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.

Example 95–14 Using a Parameterized Expression in a Mapping

```
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")));
```

Expression Method getField

The `Expression` method `getField` returns an expression that represents a database field with the given name. Use this 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 and Expressions" on page 95-12.

Example 95–15 illustrates how to use the `Expression` method `getField`.

Example 95–15 Using Expression Method getParameter

```
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));
```

Query Keys and Expressions

A query key is a schema-independent alias for a database field name.

Query keys are supported in relational database projects only.

Query keys are generated automatically for all direct and relationship mappings. The name of the query key is the class attribute name.

For more information on how query keys are created and modified, see "Configuring Query Keys" on page 25-30.

Example 95–16 illustrates how to use the query key `firstName` for the corresponding directly mapped `Employee` attribute.

Example 95–16 Using an Automatically Generated Query Key in an Expression

```
Vector employees = session.readAllObjects(Employee.class,
  new ExpressionBuilder().get("firstName").equal("Bob"));
```

Example 95–17 illustrates how to use a one-to-one query key within the TopLink expression framework.

Example 95–17 Using a One-to-One Query Key in an Expression

```
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 `Expression` method `anyOf`.

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 lets you 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. TopLink expressions provide subqueries to support subselects.

Subqueries lets you define complex 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 an 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 95–18 illustrates how to create an expression that matches all employees with more than five managed employees.

Example 95–18 A Subquery Expression Using a Comparison and Count Operation

```
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 95–19 illustrates how to create an expression that matches the employee with the highest salary in the city of Ottawa.

Example 95–19 A Subquery Expression Using a Comparison and Max Operation

```
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 95–20 illustrates how to create an expression that matches all employees that have no projects.

Example 95–20 A Subquery Expression Using a Not Exists Operation

```
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 TopLink obtains its class from the query.

Example 95–21 illustrates how to create an expression that matches all employees with the same last name as another employee of different gender, and accounts for the possibility that returned results could be a spouse.

Example 95–21 A Parallel Expression on Two Independent Employees

```
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"));
```

Data Queries and Expressions

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. TopLink provides two main methods for expressions that query for data: `getField` and `getTable`.

getField

The `getField` method lets you 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, TopLink raises an exception when you execute the query.

You can also use the `getField` method to retrieve the foreign key information for an object.

Example 95–22 illustrates how to use the data expression method (operator) `getField` with an object.

Example 95–22 Using getField with an Object

```
builder.getField("[FIELD_NAME]").greaterThan("[ARGUMENT]");
```

getTable

The `getTable` method 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` method.

Example 95–23 illustrates how to combine both `getField` and `getTable` in the same expression.

Example 95–23 Using getTable and getField Together

```
builder.getTable("[TABLE_NAME]").getField("[FIELD_NAME]").equal("[ARGUMENT]");
```

A common use for the `getTable` and `getField` methods is to retrieve information from a link table (or reference table) that supports a many-to-many relationship.

Example 95–24 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 95–24 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));
```

Creating an Expression

You can create an expression using TopLink Workbench or Java code.

Use TopLink Workbench for creating basic expressions for use in named queries (see "Using TopLink Workbench" on page 95-13).

Use Java code to create more complex expressions and to take full advantage of the features in the expressions API (see "Using Java" on page 95-15).

Using TopLink Workbench

To create TopLink expressions for named queries, use this procedure:

1. From the **Named Queries Format** tab, click **Edit** (or double-click a query string). The Expression Builder dialog box appears.

   See "Named Queries" on page 93-16 for more information.

Figure 95–1 Expression Builder Dialog Box

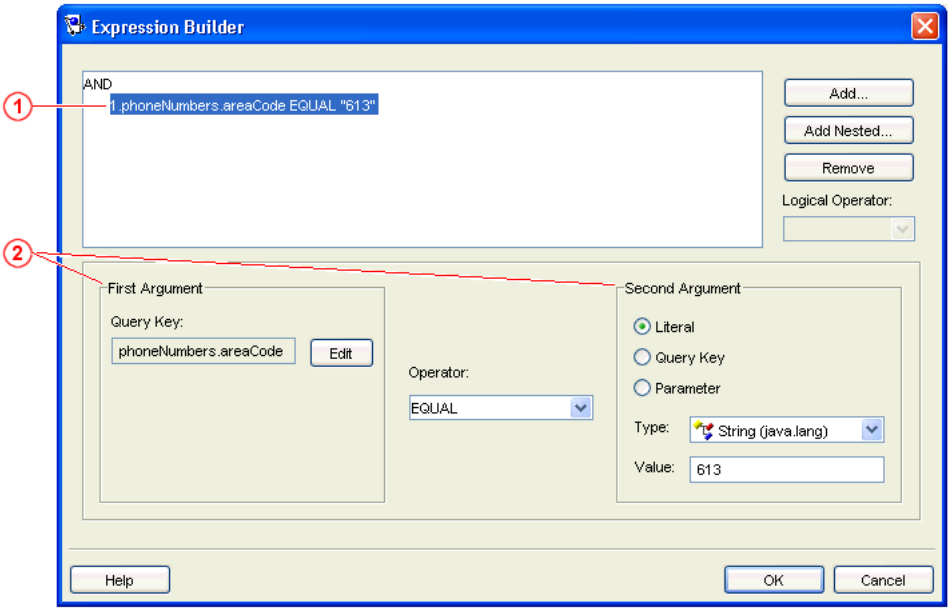


Figure 95–1 numbered callouts identify the following user-interface components:

1. Expression tree
2. Arguments

2. Click **Add** or **Add Nested** to create a new expression. TopLink assigns a sequence number to each node and nested node.

   Click **Remove** to remove an existing expression.

3. Select the node and use the **Logical Operator** list to specify the operator for the node (**AND**, **OR**, **Not AND**, or **Not OR**).

Use this table to complete the argument fields for each expression:

Field	Description
First Argument	Click **Edit** and select the query key for the first argument. The Choose Query Key dialog box appears. Continue with "Adding Arguments" on page 95-14.
Operator	Specify how TopLink should evaluate the expression. Valid operators include: Equal, Not Equal, Equal Ignore Case, Greater Than, Greater Than Equal, Less Than, Less Than Equal, Like, Not Like, Like Ignore Case, Is Null, and Not Null.
Second Argument	Specify the second argument: ■ **Literal**–Select the **Type** and enter a literal value for **Value**. ■ **Query Key**–Click **Edit** and select the query key. ■ **Parameter**–Click **Add** to add a new parameter and then select from the list. Continue with "Adding Arguments" on page 95-14

Click **OK**. TopLink Workbench adds the expression to the **Named Queries** tab.

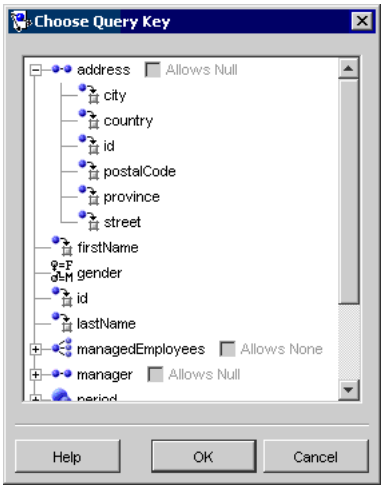
Adding Arguments

Each expression contains elements (*arguments*) to evaluate. Expressions using the Is Null or Not Null operators require only a single argument.

To add new arguments, use this procedure:

1. Select an existing expression or click **Add** (or **Add Nested**) to add a new expression to the named query.
2. For the **First Argument**, click **Edit**. The Choose Query Key dialog box appears.

Figure 95–2 Choose Query Key



3. Select the attribute, specify if the query allows a null value, and click **OK**.

   Use the **Allows Null** and **Allows None** options to define an expression with an outer join.

   Check the **Allows Null** option to use the `ExpressionBuilder` method `getAllowingNull`.

   Check the **Allows None** option to use the `ExpressionBuilder` method `anyOfAllowingNone`.

   For more information, see "Using TopLink Expression API For Joins" on page 95-7.

4. Use the **Operator** list to specify how TopLink should evaluate the expression.

5. For the **Second Argument**, select **Literal**, **Query Key**, or **Parameter**.

   - For **Literal** arguments, choose the literal type (such as **String** or **Integer**) and enter the literal value.
   - For **Query Key** arguments, click **Edit**. The Choose Query Key dialog box appears (see step 3 and Figure 95–2 on page 95-14).
   - For **Parameter** arguments, click **Add** to add a parameter and then use the list to select it.

Repeat this procedure for each expression or subexpression.

Example 95–25 Sample Expression

The following expression will find employees who:

- Have a manager with the last name Jones or have no manager, and
- work on projects with the name Beta or project ID 4, and
- live in Canada and have a salary of more than 25,000, or

  live in the United States and have a salary of more than 37,500

```
AND
  1.manager(Allows Null).lastName EQUAL "Jones"
  2.OR
    2.1.projects.name LIKE "BETA"
    2.2.projects.id EQUAL "4"
  3.OR
    3.1.AND
      3.1.1.address.country EQUAL "Canada"
      3.1.2.salary GREATER THAN "25000"
    3.2.AND
      3.1.1.address.country EQUAL "United States"
      3.1.2.salary GREATER THAN "37500"
```

Using Java

To create an expression in Java code, use the `Expression` class or `ExpressionBuilder` method `get`.

The `ExpressionBuilder` acts as a substitute for the objects that you query. To construct a query, call methods on 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 95–26 illustrates how to use the query key `lastName` to reference the field name L_NAME.

Example 95–26 Using ExpressionBuilder to Build a Simple Expression

```
Expression expression = new ExpressionBuilder().get("lastName").equal("Young");
```

Example 95–27 illustrates how to create a complex expression by combining two smaller expressions with a logical `and` operator.

Example 95–27 Combining Two Expressions with a Logical AND Operator

```
ExpressionBuilder emp = new ExpressionBuilder();
    Expression exp1, exp2;
    exp1 = emp.get("firstName").equal("Ken");
    exp2 = emp.get("lastName").equal("Young");
    return exp1.and(exp2);
```

Example 95–28 illustrates how to create an expression using the `notLike` operator.

Example 95–28 Using Database Function notLike in an Expression

```
Expression expression = new ExpressionBuilder().get("lastName").notLike("%ung");
```

Creating and Using a User-Defined Function

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, TopLink recognizes functions and other operators that vary according to the relational database.

Although most platform-specific operators exist in TopLink, if necessary, you can create your own operators.

To create a user-defined function, use the `ExpressionOperator` class.

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.

Where you create a user-defined function and how you add it to the TopLink expression framework depends on whether you want the new function available to all database platforms or to only a specific database platform.

This section describes the following:

- Making a User-Defined Function Available to a Specific Platform
- Making a User-Defined Function Available to All Platforms

Making a User-Defined Function Available to a Specific Platform

To make the function that overrides a specific operation on your own platform, use the following procedure:

1. Create a subclass of the desired `DatabasePlatform` (from `oracle.toplink.platform.database` or `oracle.toplink.platform.database.oracle` package) that provides a public method that calls the protected superclass method `addOperator`:

```
...
public class MyDatabasePlatform extends DatabasePlatform {
    protected void initializePlatformOperators() {
        super.initializePlatformOperators();
        // Create user-defined function
        ExpressionOperator toUpper = new ExpressionOperator();
        toUpper.setSelector(ExpressionOperator.ToUpperCase);
        Vector v = new Vector();
        v.addElement("UPPERCASE(");
        v.addElement(")");
        toUpper.printAs(v);
        toUpper.bePrefix();
        toUpper.setNodeClass(FunctionExpression.class);

        // Make it available to this platform only
        addOperator(toUpper);
    }
}
```

2. Configure your session to use your platform subclass (see "Configuring Relational Database Platform at the Project Level" on page 20-2 or "Configuring a Relational Database Platform at the Session Level" on page 83-1).

Making a User-Defined Function Available to All Platforms

To make the function available to all platforms, use `ExpressionOperator` method `addOperator` as Example 95–29 shows.

Example 95–29 Adding a toUpper Function for All Platforms

```
ExpressionOperator toUpper = new ExpressionOperator();
toUpper.setSelector(600);
Vector v = new Vector();
v.addElement("NUPPER(");
v.addElement(")");
toUpper.printAs(v);
toUpper.bePrefix();
toUpper.setNodeClass(FunctionExpression.class);

ExpressionOperator.addOperator(toUpper);
```

> **Note:** Represent the number in the `setSelector` method by a constant value. Ensure that this number is greater than 500 (numbers below 500 are reserved in TopLink).

Using a User-Defined Function

Regardless of whether you added the function for all platforms or for a specific platform, Example 95–30 illustrates how to use the `Expression` method

`getFunction` to access the user-defined expression operator represented by a constant with the value 600.

Example 95–30 Accessing a User-Defined Function

```
ReadObjectQuery query = new ReadObjectQuery(Employee.class);
ExpressionBuilder builder = query.getExpressionBuilder();
Expression functionExpression = builder.get("firstName").
    getFunction(600).equal("BOB");
query.setSelectionCriteria(functionExpression);
session.executeQuery(query);
```

96

Using Advanced Query API

This section explains more advanced TopLink query API calls and techniques most commonly used later in the development cycle, including the following:

- Using Redirect Queries
- Using Historical Queries
- Using Queries With Fetch Groups
- Querying on Interfaces
- Querying on an Inheritance Hierarchy
- Appending Additional Join Expressions
- Using Queries on Variable One-to-One Mappings
- Using Oracle Database Features
- Using EJB Finders
- Handling Cursor and Stream Query Results
- Using Queries and the Cache

For more information about the available query API, see *Oracle TopLink API Reference*.

Using Redirect Queries

A redirect query is a named query that delegates query execution control to your application. redirect queried allow you to define the query implementation in code as a static method.

To perform complex operations, you can combine query redirectors with the TopLink query framework.

Creating a Redirect Query

To perform complex operations, you can combine query redirectors with the TopLink query framework. To create a redirector, implement the `oracle.toplink.queryframework.QueryRedirector` interface. The query mechanism executes the `Object invokeQuery(DatabaseQuery query, Record arguments, Session session)` method and waits for the results.

TopLink provides one preimplemented 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 96–1 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("findEmployeeByAnEmployee", Employee.class, 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);
    }
}
```

Using Historical Queries

To make a query time-aware, you specify an `AsOfClause` that TopLink appends to the query. Use the `AsOfClause` class if your historical schema is based on time stamps or the `AsOfSCNClause` class if your historical schema is based on database system change numbers. You can specify an `AsOfClause` at the time you acquire a historical session so that TopLink appends the same clause to all queries, or you can specify an `AsOfClause` on a query-by-query basis.

Example 96–2 shows how to create a query that uses a particular `AsOfClause`. This query will read all `Employee` objects as of the time specified by `timestamp` using the appropriate history tables described by the `HistoryPolicy` set on the `Employee` descriptor.

Example 96–2 Using a Historical Session

```
ReadAllQuery historicalQuery = new ReadAllQuery(Employee.class);
AsOfClause asOfClause = new AsOfClause(timestamp);
historicalQuery.setAsOfClause(asOfClause);
historicalQuery.dontMaintainCache();
Vector pastEmployees = (Vector)historicalSession.executeQuery(historicalQuery);
```

Using Queries With Fetch Groups

You can use a fetch group with a `ReadObjectQuery` or `ReadAllQuery`. When you execute the query, TopLink retrieves only the attributes in the fetch group. TopLink automatically executes a query to fetch all the attributes excluded from this subset when and if you call a getter method on any one of the excluded attributes.

Note: Fetch groups are only supported with CMP.

This section describes the following:

- Configuring Default Fetch Group Behavior
- Querying With a Static Fetch Group
- Querying With a Dynamic Fetch Group

For more information about fetch groups, see "Fetch Groups and Object-Level Read Queries" on page 93-13.

Configuring Default Fetch Group Behavior

You can optionally designate at most one fetch group as the default fetch group for a descriptor's reference class.

If you execute a `ReadObjectQuery` or `ReadAllQuery` without specifying a fetch group, TopLink will use the default fetch group unless you configure the query otherwise, as Example 96–3 shows.

Example 96–3 Configuring Default Fetch Group Behavior

```
// at the descriptor level
FetchGroup group = new FetchGroup("nameOnly");
group.addAttribute("firstName");
group.addAttribute("lastName");
employeeDescriptor.getFetchGroupManager().addFetchGroup(group);
// set the default fetch group
employeeDescriptor.getFetchGroupManager().setDefaultFetchGroup(group);

// when query1 is executed, the default fetch group applies
ReadAllQuery query1 = new ReadAllQuery(Employee.class);

// when query2 is executed, the default fetch group does not apply
ReadAllQuery query2 = new ReadAllQuery(Employee.class);
query2.setShouldUsedefaultFetchGroup(false);
```

Querying With a Static Fetch Group

Example 96–4 shows how to configure a `ReadObjectQuery` for the `Employee` class with a `FetchGroup` named `nameOnly` previously stored in the `FetchGroupManager` owned by the `Employee` class's descriptor.

Example 96–4 Configuring a Query with a FetchGroup Using the FetchGroupManager

In this example, only the `Employee` attributes `firstName` and `lastName` are fetched. If you call the `Employee` method `get` for any other attribute, TopLink executes another query to retrieve all unfetched attribute values. Thereafter, calling that `get` method will return the value directly from the object.

```
// create static fetch group at the descriptor level
FetchGroup group = new FetchGroup("nameOnly");
group.addAttribute("firstName");
group.addAttribute("lastName");
descriptor.getFetchGroupManager().addFetchGroup(group);

// use static fetch group at query level
ReadAllQuery query = new ReadAllQuery(Employee.class);
query.setFetchGroupName("nameOnly");
```

Querying With a Dynamic Fetch Group

Example 96–5 shows how to create a `FetchGroup` instance dynamically, at the time you create and execute a query, and configure the query with that `FetchGroup` directly.

Example 96–5 Configuring a Query with a FetchGroup Dynamically

In this example, only the `firstName`, `lastName`, and `salary` attributes are fetched. If you call the `Employee` method `get` for any other attribute, TopLink executes another query to retrieve all unfetched attribute values. Thereafter, calling that `get` method will return the value directly from the object.

```
// dynamic fetch group query
ReadAllQuery query = new ReadAllQuery(Employee.class);
FetchGroup group = new FetchGroup("nameAndSalary");
group.addAttribute("firstName");
group.addAttribute("lastName");
group.addAttribute("salary");
query. setFetchGroup(group);
```

Querying on Interfaces

When you define descriptors for an interface to enable querying, 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, as follows:

- 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 not to read subclasses, the query returns only instances of the queried class, but no instances of the subclasses.
- If neither of these conditions applies, the class is a leaf class and does not have any subclasses. The query returns instances of the queried class.

Appending 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 do the following:

- Filter logically deleted objects
- Enable two independent classes to share a single table without inheritance
- Filter historical versions of objects

Using Java

Using Java, configure a descriptor with additional join expressions by creating an amendment method (see "Configuring Amendment Methods" on page 25-81), and then using the `DescriptorQueryManager` methods `setAdditionalJoinExpression` or `setMultipleTableJoinExpression`, as Example 96–6 shows.

Example 96–6 Registering a Query That Includes a Join Expression

In Example 96–6, the `join` expression 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"))
    );
}
```

Using Queries on Variable One-to-One Mappings

TopLink does not provide a method to directly query against variable one-to-one mappings. To query against this type of mapping, combine TopLink `DirectQueryKeys` and 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 96–7 Creating DirectQueryKeys

```
/* The DirectQueryKeys as generated in the TopLink project Java source code from
TopLink Workbench */
…
descriptor.addDirectQueryKey("locationTypeCode","DEALLOCATION.DEALLOCATIONOBJECTTY
PE");
descriptor.addDirectQueryKey("locationTypeId","DEALLOCATION.DEALLOCATIONOBJECTID")
;
```

Using Oracle Database Features

If you are using Oracle Database, you can take advantage of TopLink support for the following Oracle Database features:

- Oracle Hints

- Hierarchical Queries

Oracle Hints

Oracle Hints is an Oracle Database feature through which you can make decisions usually reserved for the optimizer. You use hints to specify things such as join order for a join statement, or the optimization approach of an SQL call.

The TopLink query framework supports Oracle Hints with the following API:

```
setHintString("/*[hints or comments]*/");
```

TopLink adds the hint to the SQL string as a comment immediately following a `SELECT`, `UPDATE`, `INSERT`, or `DELETE` statement.

Add hints to a read query as follows:

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):

```
// Create the query and set Employee as its reference class
ReadObjectQuery query = new ReadObjectQuery(Employee.class);
// Retrieve ExpressionBuilder from the query
ExpressionBuilder builder = query.getExpressionBuilder();
query.setSelectionCritera(builder.get("id").equal(new Integer(1));
// Add the hint
query.setHintString("/*+ 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 TopLink query framework, then specify hints as required. For more information, see the following:

- "Configuring Custom SQL Queries for Basic Persistence Operations" on page 26-6
- "Configuring Custom EIS Interactions for Basic Persistence Operations" on page 28-6

For more information about the available hints, see the Oracle Database documentation.

Hierarchical Queries

Hierarchical Queries is an Oracle Database mechanism that lets you 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 this 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 described in the subsequent text.

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` method. Pass the vector as the third parameter in the `setHierarchicalQueryClause` method. If you do not specify an order, set this value to `null`.

Example 96–8 Hierarchical Query

```
ReadAllQuery raq = new ReadAllQuery(Employee.class);
// Specifies 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
```

Using EJB Finders

This section describes how to use EJB finders in TopLink, including the following:

- Creating a Finder
- Using DatabaseQuery Finders
- Using Named Query Finders
- Using Primary Key Finders
- Using EJB QL Finders

- Using SQL Finders
- Using Redirect Finders
- Using the ejbSelect Method

Creating a Finder

In general, to create a finder for an entity bean that uses the TopLink query framework, you must define, declare, and configure it.

For predefined finders (see "Predefined Finders" on page 93-24), you do not need to explicitly create a finder.

For default finders (see "Default Finders" on page 93-26), you only need to define the finder method.

To create a finder for an entity bean that uses the TopLink query framework, follow these steps:

1. Define the finder method on the entity bean's `remoteHome` or `localHome` interface.

   For entity beans with container-managed persistence, define the method on the entity bean's `Home` interface.

   For default finders (see "Default Finders" on page 93-26), you must define the finder as follows:

   - *<RETURN-TYPE>* `findBy`*<CMP-FIELD-NAME>*`(`*<CMP-FIELD-TYPE>*`)`
   - the first letter of *<CMP-FIELD-NAME>* must be capitalized
   - *<RETURN-TYPE>* may be a single bean type or `Collection`.

   For example:

   ```
   EmployeeBean (Integer id, String name)
   EmployeeHome ..{
      Employee findById(Integer id) throws...;
      Collection findByName(String name) throws...;
   }
   ```

   Note: If you are using default finders (see "Default Finders" on page 93-26), you are finished. TopLink will implement the finder for you at run time.

2. Declare the finder in the `ejb-jar.xml` file (see "ejb-jar.xml Finder Options" on page 96-9).
3. Start TopLink Workbench.
4. Click the project icon in the **Navigator** and select: **Selected > Update Project from ejb-jar.xml** to read in the finders.
5. Go to the **Queries** > **Named Queries** tab for the bean (see "Using Named Queries" on page 94-18).
6. Select and configure the finder.

Notes: For predefined finders `findOneByQuery` and `findManyByQuery`, the client creates a query at run time and passes it as a parameter to the finder. Because of this, do not configure query options on these finders. Instead, configure options on the query passed into the finder. For more information about predefined finders, see "Predefined Finders" on page 93-24.

7. 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 TopLink CMP, define finder methods on the bean's `Home` interface, not in the entity bean itself. TopLink CMP provides this functionality and offers several strategies to create and customize finders. The EJB container and 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 TopLink Workbench.

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 96–9 illustrates the structure of a typical finder defined within the `ejb-jar.xml` file.

Note: Use a combination of an escape character and a double-quote (\") when defining your query using EJB QL. For more information on correct query syntax, see a note at the end of Configuring Named Query Selection Criteria on page 25-14.

Example 96–9 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>
```

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.

Using DatabaseQuery Finders

TopLink provides a predefined finder that takes a `DatabaseQuery` such as a `ReadAllQuery`. To use this feature in a bean, add the following finder definition to the `Home` interface of your bean:

```
public Collection findManyByQuery(ReadAllQuery query) throws RemoteException,
FinderException;

public <EJBLocal/Remote> findOneByQuery(ReadObjectQuery query) throws
RemoteException, FinderException;
```

To execute a `ReadAllQuery` finder, create the query on the client, as Example 96–10 shows.

Example 96–10 A ReadAllQuery Finder

```
{
    ReadAllQuery query = new ReadAllQuery(Employee.class);
    query.addJoinedAttribute("address");
    Enumeration employees = getEmployeeHome().findManyByQuery(query);
}
```

Using Named Query Finders

You can implement an EJB finder method (including TopLink predefined finders) as a named query. For more information, see "Using Named Queries" on page 94-18. You execute such a finder as you would any other.

Using Primary Key Finders

TopLink provides a predefined finder (`findByPrimaryKey`) that takes a primary key as an `Object`.

Example 96–11 Executing a Primary Key Finder

```
{
    Employee employee = getEmployeeHome().findByPrimaryKey(primraryKey);
}
```

Using EJB QL Finders

EJB QL is the standard query language first defined in the EJB 2.0 specification. TopLink supports EJB QL. EJB QL finders let you specify an EJB QL string as the implementation of the query.

> **Note:** Use a combination of an escape character and a double-quote (\") when defining your query using EJB QL. For more information on correct query syntax, see a note at the end of "Configuring Named Query Selection Criteria" on page 25-14.

EJB QL offers several advantages:

- It is the EJB 2.0 and 2.1 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 use this procedure:

1. Declare the finder on either the `LocalHome` or the `RemoteHome` interface.
2. Start TopLink Workbench.
3. Reimport the `ejb-jar.xml` file to synchronize the project to the file.

   TopLink 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 an entity bean with container-managed persistence, use this procedure:

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 TopLink 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)
```

Using SQL Finders

You can use custom SQL code to specify finder logic. SQL lets you implement logic that might not be possible to express with TopLink expressions or EJB QL.

To create a SQL finder, use this procedure:

1. Declare the finder in the `ejb-jar.xml` file, and leave the `ejb-ql` tag empty.
2. Start TopLink 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, select 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 number sign character (#) is used to bind the argument `projectName` within the SQL string:

```
SELECT * FROM EJB_PROJECT WHERE (PROJ_NAME = #projectName)
```

Using Redirect Finders

Redirect finders let you 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.

For more information about redirect queries, see "Redirect Queries" on page 93-20.

The finder can have any arbitrary parameters. If the finder includes parameters, TopLink packages them into a `Vector` and passes them to the redirect method.

Redirect finders offer several 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 lets you 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 does the following:

- Includes logic to perform several 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 a TopLink query.

The main disadvantage of redirect finders is that they are complex and can be difficult to configure. They also 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.

To create a redirect finder, use the following procedure:

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 "Configuring Amendment Methods" on page 25-81.
4. Start TopLink 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 redirectedQuery(oracle.toplink.sessions.Sessions, Vector args)
```

and

```
public static Vector redirectedQuery(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`. TopLink converts this result to `java.util.Enumeration` (or Collection) if required.

> **Note:** The redirect method also interprets a TopLink session as a parameter. For more information about a TopLink session, see Part XV, "TopLink Sessions".

At run time, 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 a TopLink expression.

Example 96–12 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 containing 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 previously created
   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 and set Employee as its ref. class
   ReadAllQuery raq = new ReadAllQuery(Employee.class);
   raq.addArgument("lastName");

   // Create the selection criteria
   ExpressionBuilder employee = query.getExpressionBuilder();
   Expression whereClause =
   employee.get("lastName").equal(arguments.firstElement());

   // Set the selection criteria
   raq.setSelectionCriteria(whereClause);

   return (Vector)session.executeQuery(raq, arguments);
}
```

```
[...]
}
```

Using the ejbSelect Method

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.
- The `result-type-mapping` tag in the `ejb-jar.xml` file determines the return type for `ejbSelect` methods. Set the flag to `Remote` to return `EJBObjects`; set it to `Local` to return `EJBLocalObjects`.

The format for an ejbSelect method definition should be similar to the following:

```
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 `ejbSelect` 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 method, use this procedure:

1. Update the `ejb-jar.xml` file as follows:
   - Declare the `ejbSelect` method.
   - Enter the EJB QL string in the `ejb-ql` tag.
   - Specify the return type in the `result-type-mapping` tag (if required).
2. Declare the `ejbSelect` on the abstract bean class.
3. Start TopLink Workbench.
4. Click the project icon in the **Navigator**, and select: **Selected > Update Project from ejb-jar.xml** to read in the finders.
5. Go the **Queries** > **Named Queries** tab for the bean.
6. Select and configure the `ejbSelect` method.

Handling Cursor and Stream Query Results

Cursors and streams are related mechanisms that let you work with large result sets efficiently. See "Stream and Cursor Query Results" on page 93-8 for more information.

Table 96–1 lists the methods that TopLink provides for all subclasses of `DataReadQuery` and `ReadAllQuery` that you can use to make your query return its results as a cursor or stream.

Table 96–1 Stream and Cursor Query Result Options

Method	Query Returns	Description
`useScrollableCursor`	`ScrollableCursor`	Allows you access a database result set cursor, allowing you to move forward and backward through the result set.
`useCursoredStream`	`CursoredStream`	Allows you to access results one at a time in sequence, as results become available to the underlying database result set cursor.

Using a `ScrollableCursor` or `CursoredStream` combines the features of a TopLink with the ability of the database to cursor data, and breaks up the result set into smaller, more manageable pieces.

The behavior of a query that uses a `ScrollableCursor` or `CursoredStream` differs from other queries in that the elements requested by the client are sent to the client.

This section describes the following:

- Cursors and Java Iterators
- Java Streams
- Optimizing Streams
- Using Cursors and Streams With EJB Finders

Cursors and Java Iterators

The TopLink scrollable cursor lets you scroll through a result set from the database without reading the whole result set in a single database read operation. The `ScrollableCursor` class implements the Java `ListIterator` interface to allow for direct and relative access within the stream. Scrollable cursors also let you scroll forward and backward through the stream.

Traversing Data With Scrollable Cursors

Several methods let you 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 prior to 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 *Oracle TopLink API Reference*.

To use the `ScrollableCursor` object, the JDBC driver must be compatible with the JDBC 2.0 specifications.

Example 96–13 Traversing with a Scrollable Cursor

```
ReadAllQuery query = new ReadAllQuery(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 let you retrieve query results as individual records or groups of records, which can result in a performance increase. You can use streams to build efficient 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 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 96–14 Cursored Streams

```
CursoredStream stream;
ReadAllQuery query = new ReadAllQuery(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 96–15 returns a `Vector` that contains all employee objects. If ACME has 10,000 employees, the `Vector` contains references to 10,000 `Employee` objects.

Example 96–15 Using a Vector

```
ReadAllQuery query = new ReadAllQuery(Employee.class);
Enumeration employeeEnumeration;

Vector employees = (Vector) session.executeQuery(query);
employeeEnumeration = employee.elements();

while (employeeEnumeration.hasMoreElements())
{
Employee employee = (Employee) employeeEnumeration.nextElement();
employee.doSomeWork();
}
```

The following example returns a `CursoredStream` instance rather than a `Vector`. The `CursoredStream` collection appears to contain all 10,000 objects, but initially contains a reference to only 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:

```
ReadAllQuery query = new ReadAllQuery(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 TopLink loads each page. The default page size is 5.

When you execute a batch-type operation, use the `dontMaintainCache` method 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` ensures that these transient objects are not cached.

Using Cursors and Streams With EJB Finders

Large result sets can be resource-intensive to collect and process. To give the client more control over the returned results, configure TopLink finders to use cursors. This combines TopLink's `CursoredStream` with the ability of the database to cursor data, and breaks up the result set into smaller, more manageable pieces.

Note: If you use the transactional attribute `REQUIRED` for an entity bean, wrap all read operations in `UserTransaction` methods `begin` and `commit` to ensure that read operations 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.Collection` to use a cursor. When you create the query for the finder, add the `useCursoredStream` option to enable cursoring.

Example 96–16 Cursored Stream in a Finder

```
ReadAllQuery raq = new ReadAllQuery(ProjectBean.class);
ExpressionBuilder bldr = raq.getExpressionBuilder();
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

TopLink offers the following additional elements for traversing finder results:

- `isEmpty` method: As with `java.util.Collection`, `isEmpty` method returns a `boolean` value indicating whether or not the `Collection` is empty.
- `size` method: As with `java.util.Collection`, `size` method returns an integer indicating the number of elements in the `Collection`.
- `iterator` method: As with `java.util.Collection`, `iterator` method returns a `java.util.Iterator` for enumerating the elements in the `Collection`.

TopLink also offers an extended protocol for `oracle.toplink.ejb.cmp.wls.CursoredIterator` (based on `java.util.Iterator`):

- `close` method: Closes the cursor on the server. The client must call this method to close the database connection.
- `hasNext` method: Returns a `boolean` value indicating whether or not any more elements are in the result set.
- `next` method: Returns the next available element.
- `next(int count)` method: Retrieves a `Vector` of at most count elements from the available results, depending on how many elements remain in the result set.

Example 96–17 illustrates client code executing a cursored finder.

Example 96–17 Cursored Finder

```
// 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();
```

Using Queries and the Cache

This section describes how to use caching options in TopLink queries, including the following:

- Caching Results in a ReadQuery
- Configuring Cache Expiration at the Query Level

Caching Results in a ReadQuery

By default, each time you execute a `ReadQuery`, TopLink applies the current query configuration to the read operation. In doing so, TopLink will access the session cache, the data source, or both.

Some queries are known to return the same result set (for example, the number of units sold last year by the current sales person). After the first query execution, there is no need to actually execute the query if it is invoked again.

For these types of queries, you can use any TopLink `ReadQuery` and configure it to store its query results in an internal query cache.

After its first execution for a set of query parameters, the query will return its cached result set each time it is invoked with the same query parameters. This improves query performance for frequently executed queries. By default a query will cache the results sets for the last 100 queries of specific parameters. You can configure this query cache as part of the `QueryResultsCachePolicy`.

Enable this feature using `ReadQuery` method `cacheQueryResults` or by calling the `ReadQuery` method `setQueryResultsCachePolicy` with an instance of `QueryResultsCachePolicy`, and disable it using `ReadQuery` method `doNotCacheQueryResults`.

Before using this feature, consider the restrictions in "Internal Query Cache Restrictions" on page 93-37. For more information, see "Caching Query Results in the Query Cache" on page 93-36.

You can apply a cache invalidation policy to the query's internal cache (see "Configuring Cache Expiration at the Query Level" on page 96-21). For more information, see "Cache Invalidation" on page 87-8.

Example 96–18 shows how to configure a `ReadQuery` to cache its results.

Example 96–18 Configuring a ReadQuery to Cache Its Query Results

```
ReadObjectQuery query = new ReadObjectQuery(Employee.class);

// Instruct the ReadQuery to cache its query results
query.cacheQueryResults();

// The first time you invoke it, the ReadQuery reads from the database, session
// cache, or both and stores the result set in its internal query cache
Employee employeeFirst = (Employee) session.executeQuery(query);
```

Example 96–19 shows how to configure the `ReadQuery` to stop caching its results. The next time the query is executed, TopLink does not use the query cache. Instead, the query accesses the data source.

Example 96–19 Configuring a ReadQuery to Stop Caching Its Query Results

```
// Disable query caching
query.doNotCacheQueryResults();

// The ReadQuery does not use the query cahce and instead accesses the database
Employee employee = (Employee) session.executeQuery(query);
```

Alternativley, you can clear the query's internal cache using `ReadQuery` method `clearQueryResults` passing in your session. This clears the currently cached results and ensures that the next query execution reads from the database.

Configuring Cache Expiration at the Query Level

You can configure a `ReadQuery` with a `CacheInvalidationPolicy`.

If you configure a query to cache results in its own internal cache (see "Caching Results in a ReadQuery" on page 96-20), the cache invalidation policy allows the cached query result set to expire, based on a time-to-live or daily-expiry. This invalidation time is calculated from the time of the query execution that cached the query result set for the specific set of query parameters.

Example 96–20 shows how to configure a `ReadQuery` so that a `TimeToLiveCacheInvalidationPolicy` is applied to all the objects returned by the query and cached in the query's internal cache.

Example 96–20 Configuring a CacheInvalidationPolicy on a ReadQuery for the Query's Internal Cache

```
// The TimeToLiveCacheInvalidationPolicy applies to all objects returned by the query and
// cached in the query's internal cache
readQuery.setQueryResultsCachePolicy(
    new QueryResultsCachePolicy(new TimeToLiveCacheInvalidationPolicy(1000))
);
```

For more information, see "Cache Invalidation" on page 87-8.

Part XIX

Transactions

This part describes how to use the TopLink unit of work to transactionally perform create, read, update, and delete operations with and without an external transaction processor. It contains the following chapters:

- Chapter 97, "Understanding TopLink Transactions"

  This chapter describes how to use the unit of work, the TopLink wrapper for a transaction, and how TopLink integrates with transaction management and other important query concepts.

- Chapter 98, "Using Basic Unit of Work API"

  This chapter explains how to use basic TopLink unit of work options.

- Chapter 99, "Using Advanced Unit of Work API"

  This chapter explains how to use advanced TopLink unit of work options.

97

Understanding TopLink Transactions

This chapter explains how transactions are implemented in TopLink, including the following:

- Unit of Work Architecture
- Unit of Work Concepts
- Understanding the Unit of Work API
- Example Model Object and Schema

Unit of Work Architecture

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. Transactions may be *internal* (that is, provided by TopLink) or *external* (that is, provided by a source external to the application, such as an application server).

In TopLink, transactions are contained in the unit of work object. You acquire a unit of work from a session (see "Acquiring a Unit of Work" on page 98-1) and using its API, you can control transactions directly or through a Java 2 Enterprise Edition (J2EE) application server transaction controller such as the Java Transaction API (JTA).

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 97–1. Each offers a trade-off between performance and resistance from the following unwanted actions:

- Dirty read: a transaction reads uncommitted data written by a concurrent transaction.
- Nonrepeatable read: a transaction rereads data and finds it has been modified by some other transaction that was committed after the initial read operation.

- Phantom read: a transaction reexecutes a query and the returned data has changed due to some other transaction that was committed after the initial read operation.

Table 97–1 Transaction Isolation Levels

Transaction Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read
Read Uncommitted	Yes	Yes	Yes
Read Committed	No	Yes	Yes
Repeatable Read	No	No	Yes
Serializable	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.

Like any transaction, a unit of work transaction provides the following:

- Unit of Work Transaction Context
- Unit of Work Transaction Demarcation
- Unit of Work Transaction Isolation

Unit of Work Transaction Context

Unit of work operations occur within a unit of work *context*, in which writes are 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.

Unit of Work Transaction Demarcation

In a standalone TopLink application, your application demarcates transactions using the unit of work.

If your application includes a J2EE container that provides container-managed transactions, your application server demarcates transactions using its own transaction service. You configure TopLink to integrate with the container's transaction service by specifying a TopLink external transaction controller.

A TopLink external transaction controller class integrates the unit of work with an application server's transaction service. Using an external transaction controller, your application can participate in transactions that span multiple data sources and that are managed by the application server. The external transaction controller coordinates messages and callbacks between the application server's transaction service and the unit of work.

When you configure your application to use an external transaction controller (see "Configuring the Server Platform" on page 74-14), the unit of work executes as part of an external transaction. The unit of work still manages its own internal operations, but it waits for the external transaction to tell it to write changes back to the database and to the session cache.

Note that because the transaction happens outside of the unit of work context and is controlled by the application server's transaction service, errors can be more difficult to diagnose and fix because exceptions may occur outside of your application code, for example, during application server initiated call-backs.

You can integrate the unit of work with the following:

- JTA Controlled Transactions
- OTS Controlled Transactions
- CMP Controlled Transactions

JTA Controlled Transactions

The Java Transaction API (JTA) is the application programming interface you use to interact with a transaction manger.

Using JTA, your application can participate in a distributed transaction. A transaction manager that implements JTA provides transaction management and connection pooling and enables your application to interact with multiple data sources transparently by using JTA.

For more information, see "Integrating the Unit of Work With an External Transaction Service" on page 99-21.

OTS Controlled Transactions

The CORBA Object Transaction Service (OTS) specification is part of the Common Object Request Brokers Architecture (CORBA) Object Services model and is the standard for Object Request Broker (ORB) implementations. Some servers implement the Java APIs for the OTS rather than for JTA (see "JTA Controlled Transactions" on page 97-3).

Use TopLink OTS support with the unit of work to directly access the Java OTS interfaces of servers that do not support JTA.

To integrate your application with an OTS transaction service, you must configure your application to use a custom server platform (see "Configuring the Server Platform" on page 74-14).

For more information, see "Integrating the Unit of Work With an External Transaction Service" on page 99-21.

CMP Controlled Transactions

Entity beans that use container-managed persistence may participate in transactions that are either client demarcated or container demarcated.

A client demarcated transaction occurs when a client of an entity bean directly sets up transaction boundaries using the `javax.transaction.UserTransaction` interface.

A container demarcated transaction occurs when the container automatically wraps an invocation on an EJB in a transaction based upon the transaction attributes supplied in the EJB deployment descriptor.

In transactions involving EJB, TopLink waits until the transaction begins its two-phase commit process before updating the database. This allows for the following:

- SQL optimizations that ensure only changed data is written to the data source
- Proper ordering of updates to allow for database constraints

For more information, see "Integrating the Unit of Work With CMP" on page 99-24.

Unit of Work Transaction Isolation

The unit of work does not directly participate in database transaction isolation. Because the unit of work may execute queries outside the database transaction (and, by interacting with the cache, outside the database itself), the database does not have control over this data and its visibility.

However, by default, TopLink provides a degree of transaction isolation regardless of what database transaction isolation has been configured on the underlying database.

Each unit of work instance operates on its own copy (clone) of registered objects (see "Clones and the Unit of Work" on page 97-9). In this case, because the unit of work provides an API that allows querying to be done on object changes within a unit of work (see "Using Conforming Queries and Descriptors" on page 99-8), the unit of work provides read committed operations.

Optimistic locking, optimistic read locking, or pessimistic locking can be used to further manage concurrency (see "Locking and the Unit of Work" on page 97-12).

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).

For detailed information on configuring and using TopLink to achieve a particular level of transaction isolation and transaction isolation level limitations, see "Database Transaction Isolation Levels" on page 99-26.

Unit of Work Concepts

This section introduces transaction concepts unique to TopLink, including the following:

- Unit of Work Benefits
- Unit of Work Life Cycle
- Unit of Work and Change Policy
- Clones and the Unit of Work
- Nested and Parallel Units of Work
- Commit and Rollback Transactions
- Primary Keys
- Unit of Work Optimization
- Example Model Object and Schema

Unit of Work Benefits

The TopLink unit of work simplifies transactions and improves transactional performance. It is the preferred method of writing to a database in TopLink because it performs the following:

- 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 coordination, 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 insert, update, and delete operations 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 "Associating a New Source to an Existing Target Object" on page 98-6)

Unit of Work Life Cycle

TopLink uses the unit of work as follows:

1. The client application acquires a unit of work from a session object.
2. The client application queries TopLink to obtain a cache object it wants to modify, and then registers the cache object with the unit of work.
3. The unit of work registers the object according to the object's change policy. For more information about how change policy affects registration, see "Unit of Work and Change Policy" on page 97-6.

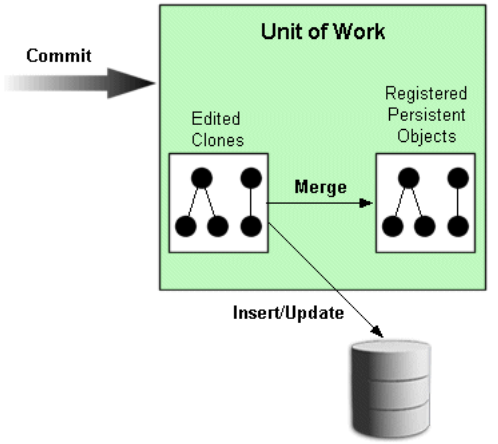
   By default, as each object is registered, the unit of work accesses the object from the session cache or database and creates a backup clone and working clone (see "Clones and the Unit of Work" on page 97-9). The unit of work returns the working clone to the client application.
4. The client application modifies the working object returned by the unit of work.
5. The client application (or external transaction controller) commits the transaction.
6. The unit of work calculates the change set for each registered object according to the object's change policy. For more information about how change policy affects change set calculation, see "Unit of Work and Change Policy" on page 97-6.

   By default, at commit time, the unit of work compares the working clones to the backup clones and calculates the change set (that is, determines the minimum changes required). The comparison is done with a backup clone so that concurrent changes to the same objects will not result in incorrect changes being identified. The unit of work then attempts to commit any new or changed objects to the database.

   If the commit transaction succeeds, the unit of work merges changes into the shared session cache. Otherwise, no changes are made to the objects in the shared cache. For more details, see "Commit and Rollback Transactions" on page 97-10.

   If there are no changes, the unit of work does not start a new transaction.

Figure 97–1 The Life Cycle of a Unit of Work



Example 97–1 shows the default life cycle in code.

Example 97–1 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);
    // 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);
    // 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
```

Unit of Work and Change Policy

The unit of work tracks changes for a registered object based on the change policy you configure for the object's descriptor. If there are no changes, the unit of work will not start a new transaction.

Table 97–2 lists the change policies that TopLink provides.

Table 97–2 TopLink Change Policies

Change Policy	Applicable to...
Deferred Change Detection Policy	Wide range of object change characteristics. The default change policy.
Object-Level Change Tracking Policy	Objects with few attributes or with many attributes and many changed attributes.
Attribute Change Tracking Policy	Objects with many attributes and few changed attributes. The most efficient change policy. The default change policy for EJB 3.0 persistent or 2.*n* CMP on OC4J.

For more information, see "Configuring Change Policy" on page 25-73.

Deferred Change Detection Policy

The `DeferredChangeDetectionPolicy` is the change policy that all persistent objects use by default.

This option provides good unit of work commit performance for a wide range of object change characteristics.

When you register in a unit of work an object whose descriptor is configured with a `DeferredChangeDetectionPolicy` (see "Configuring Deferred Change Detection Policy" on page 25-73), a backup clone is made of the object (see "Clones and the Unit of Work" on page 97-9) and at commit time, the unit of work computes changes by making an attribute-by-attribute comparison between the backup clone and the original object.

This change policy is applicable to all mapping types.

Object-Level Change Tracking Policy

The `ObjectChangeTrackingPolicy` optimizes the unit of work commit transaction by including objects in the change set calculation only if at least one attribute has changed.

This option provides improved unit of work commit performance for objects with few attributes, or with many attributes and many changed attributes.

When you register in a unit of work an object whose descriptor is configured with `ObjectChangeTracking` change policy, a backup clone is made of the object and at commit time, the unit of work computes changes by comparing the backup to the current object if and only if at least one attribute is changed (if the object's `hasChanges` method returns `true`). If a registered object has no changes, the unit of work does not compare it to the backup clone.

This change policy is applicable to a subset of mapping types (see "Change Policy Mapping Support" on page 97-9).

TopLink provides different levels of support for this change policy depending on the EBJ version and application server you are using:

EJB CMP For CMP applications deployed to an application server for which TopLink provides CMP integration (see "Application Server Support" on page 7-1), when you configure a descriptor for an entity bean with container-managed persistence with an `ObjectChangeTrackingPolicy`, TopLink code generates a concrete subclass to implement the TopLink `ChangeTracker` interface at deploy time (see "Configuring Object Change Tracking Policy" on page 25-73).

Note: The preceding paragraph is also true for applications that use EJB 3.0 persistence.

Attribute Change Tracking Policy

The `AttributeChangeTrackingPolicy` optimizes the unit of work commit transaction by tracking all object changes at the attribute level. This eliminates two unit of work operations: backup clone creation and change detection through comparison.

This option provides improved unit of work commit performance for objects with many attributes, and few changed attributes. Generally, this is the most efficient change policy.

This change policy is applicable to a subset of mapping types (see "Change Policy Mapping Support" on page 97-9).

Note: You cannot use the `AttributeChangeTrackingPolicy` if you are using any instance of `FieldsLockingPolicy` (see "Optimistic Field Locking Policies" on page 23-20).

TopLink provides different levels of support for this change policy depending on the EBJ version and application server you are using:

Plain Java Objects or Other Application Servers For plain Java objects or application servers other than OC4J, to use the `AttributeChangeTrackingPolicy` with a class, you must configure the class's descriptor with an `AttributeChangeTrackingPolicy` and you must implement the `ChangeTracker` interface in that class (see "Configuring Attribute Change Tracking Policy" on page 25-75).

EJB CMP on OC4J When using CMP on OC4J, if you want to benefit from this performance enhancement, configure your descriptors with the default `DeferredChangeDetectionPolicy` and allow TopLink to automatically apply an `AttributeChangeTrackingPolicy`. If you configure your project's descriptors with any other change policy, TopLink will honor that configuration and not apply an `AttributeChangeTrackingPolicy`.

When you deploy a TopLink-enabled CMP application to OC4J, for each mapped class configured with the default `DeferredChangeDetectionPolicy`, TopLink uses code generation to automatically override this configuration with an `AttributeChangeTrackingPolicy` and to make the class implement the required interfaces.

Note: The preceding information is applicable to applications built using EJB 3.0.

However, when you deploy a TopLink-enabled EJB 3.0 persistent application to OC4J, for each mapped class configured with the default `DeferredChangeDetectionPolicy`, TopLink uses bytecode weaving to automatically override this configuration with an `AttributeChangeTrackingPolicy` and to make the class implement the required interfaces.

Change Policy Mapping Support

TopLink supports alternative change tracking policies (policies other than `DeferredChangeDetectionPolicy`) for attributes that use any of the following mapping types:

- Direct-to-Field Mapping
- Transformation Mapping (immutable mappings only)
- One-to-One Mapping
- Variable One-to-One Mapping
- One-to-Many Mapping
- Many-to-Many Mapping
- Direct Collection Mapping
- Direct Map Mapping
- Aggregate Object Mapping
- EIS Transformation Mapping (immutable mappings only)

TopLink uses the `DeferredChangeDetectionPolicy` (see "Deferred Change Detection Policy" on page 97-7) for attributes that use any other type of mapping.

Clones and the Unit of Work

When using the `DefrerredChangeDetectionPolicy` or the `ObjectLevelChangeTrackingPolicy` (see "Deferred Change Detection Policy" on page 97-7), 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 clones to the backup 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 97-9) to exist, a requirement in multiuser three-tier applications.

The 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 "Configuring Indirection" on page 32-3.

You can customize the cloning process using the descriptor's copy policy. For more information, see "Configuring Copy Policy" on page 25-71.

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 transaction will raise an exception. The only time you can use a clone after a successful commit transaction is when you use the advanced API described in "Resuming a Unit of Work After Commit" on page 99-14.

Nested and Parallel Units of Work

You can use TopLink to create the following:

- Nested Unit of Work
- Parallel Unit of Work

For additional information and examples on using nested and parallel units of work, see "Using a Nested or Parallel Unit of Work" on page 99-15.

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 lets you 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 transaction 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. TopLink resolves any concurrency issues when the Units of Work commits the changes.

Commit and Rollback Transactions

When a unit of work transaction is committed, it either succeeds, or fails and rolls back. A commit transaction can be initiated by your application or by a J2EE container.

Commit Transactions

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 one 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 a commit transaction, 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 transaction acts 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 activity 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 "Integrating the Unit of Work With an External Transaction Service" on page 99-21.

Rollback Transactions

A unit of work commit transaction 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 "Integrating the Unit of Work With an External Transaction Service" on page 99-21.

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 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 99-17.

Unit of Work Optimization

By default, the unit of work performs change set calculation efficiently for a wide range of object change characteristics. However, there are various ways you can use the unit of work to enhance application performance.

One way to improve performance is to consider using an alternative change policy (see "Unit of Work and Change Policy" on page 97-6).

For more performance options, see "Unit of Work Optimization" on page 11-30.

Understanding the Unit of Work API

You do not instantiate an instance of `oracle.toplink.sessions.UnitOfWork`. Rather, you acquire a unit of work from an instance of `oracle.toplink.sessions.Session` or from another unit of work.

For more information on creating sessions, see "Creating Sessions" on page 73-1.

For more information on acquiring a unit of work, see "Acquiring a Unit of Work" on page 98-1.

For more information on using the basic API of the unit of work, see "Using Basic Unit of Work API" on page 98-1.

For more information on using the advanced API of the unit of work, see "Using Advanced Unit of Work API" on page 99-1.

Unit of Work as Session

The unit of work extends the interface `oracle.toplink.sessions.Session`, and implements all the usual session API. When using session API from a unit of work, you should consider the following:

- Reading and Querying Objects with the Unit of Work
- Locking and the Unit of Work

Reading and Querying Objects with the Unit of Work

A unit of work 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 98-2 and "Associating a New Source to an Existing Target Object" on page 98-6.

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.

Locking and the Unit of Work

For information on locking API generic to all sessions, see:

- "Locking" on page 2-11
- "Configuring Locking Policy" on page 25-64

For information on locking API specific to a unit of work, see "Using Optimistic Read Locking With forceUpdateToVersionField" on page 99-18.

Example Model Object and Schema

Throughout the chapters in this part, the following object model and schema are used in the examples provided. The example object model appears in Figure 97–2 and the example entity-relationship (data model) diagram appears in Figure 97–3.

Figure 97–2 Example Object Model

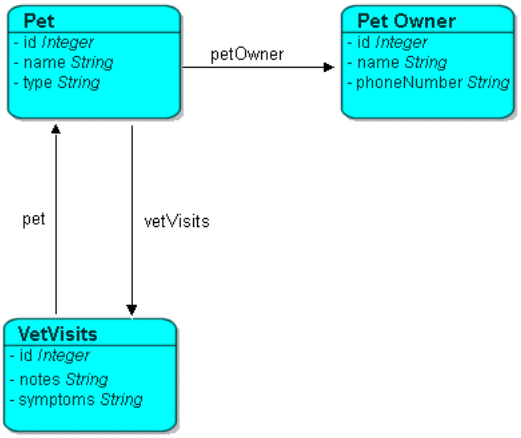
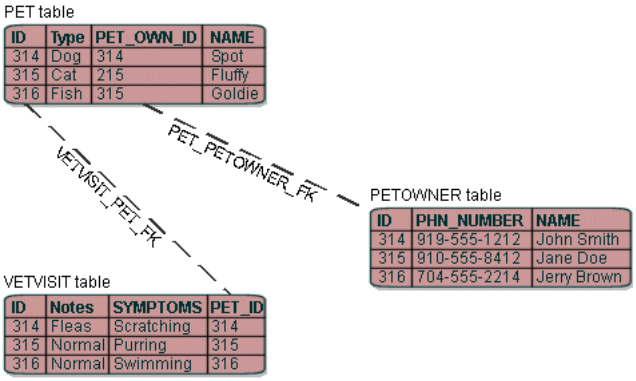


Figure 97–3 Example Data Model



98

Using Basic Unit of Work API

This chapter explains 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
- Associating a New Target to an Existing Source Object
- Associating a New Source to an Existing Target Object
- Associating an Existing Source to an Existing Target Object
- Deleting Objects

For more information, see Chapter 99, "Using Advanced Unit of Work API".

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. For more information about acquiring sessions at run time, see "Acquiring a Session at Run Time With the Session Manager" on page 72-5.

Note that you do not need to create a new session and log in before every transaction. The recommended pattern is to acquire a client session per client access (or thread), and then acquire the necessary unit of work from this client session.

The unit of work is valid until the `commit` or `release` method is called. After a commit or release transaction, 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 99-14.

When using a unit of work with JTA, you can also use the advanced API `getActiveUnitOfWork` method as described in "Integrating the Unit of Work With an External Transaction Service" on page 99-21.

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 a commit transaction, verify your mapping definitions. The order in which you register objects with the `registerObject` method does not affect the commit order.

Example 98–1 and Example 98–2 show how to create and persist a simple object (without relationships) using the clone returned by the unit of work `registerObject` method.

Example 98–1 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 98–2 shows a common alternative.

Example 98–2 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 98–1 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 98–3, a `Pet` is read prior to a unit of work: the variable `pet` is the cache copy clone for that `Pet`. Inside the unit of work, register the cache copy to get a working copy clone. We then modify the working copy clone and commit the unit of work.

Example 98–3 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 98–4, 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 clone.

If we wanted to do something with the updated `Pet` after the commit transaction, 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 98–4 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 register in a unit of work only the objects that need to be changed.

Associating a New Target to an Existing Source Object

There are two ways to associate a new target object with an existing source object with one-to-many and one-to-one 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 clone 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 98–5 shows the first way of associating a new target with an existing source.

Example 98–5 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 were a clone. That is, after the commit transaction:

```
petOwner != session.readObject(petOwner)
```

For a more detailed discussion of this fact, see "Using registerNewObject" on page 99-2).

Note: You cannot use `UnitOfWork` methods `registerObject`, `registerNewObject`, or `registerExistingObject` with an aggregate object (see "Relational Aggregate Descriptors" on page 24-2). Doing so will raise a `ValidationException` or other errors at commit time. For more information, see "Working With Aggregates" on page 99-6.

Therefore, after the unit of work commit transaction, 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 transaction, 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 98-4).

Associating With Reference to the Cache Object

Example 98–6 shows how to associate a new target with an existing source with reference to the cache object.

Example 98–6 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 transaction:

```
petOwner == session.readObject(petOwner)
```

This means that we have a handle to the cache copy after the commit transaction, rather than a clone.

Example 98–7 shows how to use unit of work method `registerNewObject` to add a new object when a bidirectional relationship exists. For more information, see "Using registerNewObject" on page 99-2.

> **Note:** You cannot use `UnitOfWork` methods `registerObject`, `registerNewObject`, or `registerExistingObject` with an aggregate object (see "Relational Aggregate Descriptors" on page 24-2). Doing so will raise a ValidationException or other errors at commit time. For more information, see "Working With Aggregates" on page 99-6.

Example 98–7 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 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 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();
```

Associating a New Source to an Existing Target Object

This section describes how to associate a new source object with an existing target object with one-to-many and one-to-one relationships.

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 "Associating a New Target to an Existing Source Object" on page 98-3, 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 TopLink will not write it to the database.

For example, the code shown in Example 98–8 shows how to create a new `Pet` and associate it with an existing `PetOwner`.

Example 98–8 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 99-2.

Note: You cannot use `UnitOfWork` methods `registerObject`, `registerNewObject`, or `registerExistingObject` with an aggregate object (see "Relational Aggregate Descriptors" on page 24-2). Doing so will raise a ValidationException or other errors at commit time. For more information, see "Working With Aggregates" on page 99-6.

If you fail to register the clone and accidentally associate the cache version of the existing object with the new object, then 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.

Associating an Existing Source to an Existing Target Object

This section explains how to associate an existing source object with an existing target object with one-to-many and one-to-one relationships.

As shown in Example 98–9, 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 98–9 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 98–10 shows another example of associating an existing source to an existing target object.

Example 98–10 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, TopLink deletes the object's privately owned child parts, because those parts cannot exist without the owning (parent) 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 98-8).

This section explains how to delete objects within a unit of work, including the following:

- 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 remove the reference to 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 have TopLink manage the garbage collection in the relational database for you.

As shown in Example 98–11, when you create a mapping using Java, use its `privateOwnedRelationship` method to tell TopLink that the referenced object is privately owned: that is, the referenced child object cannot exist without the parent object.

Example 98–11 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 TopLink Workbench, you can select the **Private Owned** check box under the **General** tab.

When you tell TopLink that a relationship is privately owned, you are specifying that:

- If the source of a privately owned relationship is deleted, then delete the target.
- If you remove the reference to a target from a source, then delete the target.

Do not configure privately 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 privately 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 the 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 98–12.

Example 98–12 Privately 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 privately 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 privately 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 privately owned relationships (especially root objects in your object model), then you can explicitly tell TopLink to delete the row representing the object using the `deleteObject` API, as shown in Example 98–13.

Example 98–13 Explicitly Deleting

```
UnitOfWork uow = session.acquireUnitOfWork();
    pet petClone = (Pet)uow.readObject(Pet.class);
    uow.deleteObject(petClone);
uow.commit();
```

The preceding 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 insulate you from having to modify your objects in the order the database requires.

By default, at commit time, the unit of work correctly puts in order all insert and update operations using the constraints defined by your schema. After all insert and update operations 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 TopLink as described in "Controlling the Order of Delete Operations" on page 99-16.

99

Using Advanced Unit of Work API

This chapter explains the advanced unit of work API calls and techniques commonly used later in the development cycle, including the following:

- Registering and Unregistering Objects
- Declaring Read-Only Classes
- Writing Changes Before Commit Time
- 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
- Controlling the Order of Delete Operations
- Using Optimistic Read Locking With forceUpdateToVersionField
- Implementing User and Date Auditing With the Unit of Work
- Integrating the Unit of Work With an External Transaction Service
- Integrating the Unit of Work With CMP
- Database Transaction Isolation Levels
- Troubleshooting a Unit of Work

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

Registering and Unregistering Objects

The unit of work provides the following object registration options:

- Creating and Registering an Object in One Step
- Using registerNewObject
- Using registerAllObjects
- Using Registration and Existence Checking
- Working With Aggregates
- Unregistering Working Clones

Creating and Registering an Object in One Step

Example 99–1 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 design pattern to create your objects (and have specified this in the query builder), the `newInstance` method will use the appropriate factory.

Example 99–1 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 section examines how to use the `registerNewObject` method, including the following the following:

- Registering a New Object With registerNewObject
- Associating New Objects With One Another

> **Note:** You cannot use `UnitOfWork` methods `registerObject`, `registerNewObject`, or `registerExistingObject` with an aggregate object (see "Relational Aggregate Descriptors" on page 24-2). Doing so will raise a `ValidationException` or other errors at commit time. For more information, see "Working With Aggregates" on page 99-6.

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 transaction and you do not want to work with clones of new objects.
- You must pass a clone into the constructor of a new object, then register the new object.

Example 99–2 shows how to register a new object with the `registerNewObject` method.

Example 99–2 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 been committed, then you should use the first approach described in "Associating a New Source to an Existing Target Object" on page 98-6. 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, TopLink can determine if an object is new or not. As described in "Associating a New Target to an Existing Source Object" on page 98-3, if a new object is reachable from a clone, you do not need to register it. TopLink effectively uses the `registerNewObject` method on 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 clone, and the argument, is considered the cache version.

Example 99–3 shows how to associate new objects with the `registerNewObject` method.

Example 99–3 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 commit, do not use the objects in the variables `newPet`, `newPetOwner`, and `newVetVisit` because they are technically copies from the unit of work.

If you need a handle to the cache version of these objects, query for them or use the unit of work as shown in Example 99–4.

Example 99–4 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 lets you register many objects at once as shown in Example 99–5.

> **Note:** You cannot use `UnitOfWork` methods `registerObject`, `registerNewObject`, or `registerExistingObject` with an aggregate object (see "Relational Aggregate Descriptors" on page 24-2). Doing so will raise a ValidationException or other errors at commit time. For more information, see "Working With Aggregates" on page 99-6.

Example 99–5 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 you register an object with the unit of work, TopLink runs an existence check to determine whether or not the object exists. TopLink uses this information at commit time to determine whether to perform an insert or an update operation. You can specify the default existence checking policy for a project as a whole (see "Configuring Existence Checking at the Project Level" on page 19-8) or on a per-descriptor basis ("Configuring Cache Existence Checking at the Descriptor Level" on page 25-43). By default, 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 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 Nonexistence

Check Database

If you configure a class's descriptor with an existence checking policy of check database, TopLink will check the database for existence for all instances of that class 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. TopLink will not check the database for existence on objects that you have registered with these methods. It will automatically perform an insert operation if `registerNewObject` is called, or an update operation if `registerExistingObject` is called.

Assume Existence

If you configure a class's descriptor with an existence checking policy of assume existence, TopLink will assume that all instances of that class registered with a unit of work exist and TopLink will always perform an update operation to the database on all such registered objects; even new objects that you registered with `registerObject` method. However, if you use the `registerNewObject` method on the new object, TopLink knows to perform an insert operation in the database even though the existence checking policy says assume existence.

Assume Nonexistence

If you configure a class's descriptor with an existence checking policy of assume nonexistence then TopLink assumes that all instances of that class registered with a unit of work do not exist and will always perform an insert operation on the database, even on objects read from the database. However, if you use the `registerExistingObject` method on existing objects, TopLink knows to perform an update operation on the database.

Working With Aggregates

Aggregate mapped objects should never be registered in a TopLink unit of work–doing so will generate an exception. 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.

When working with aggregates, you should always use an aggregate *within the context of its owner*:

- If you get an aggregate from a working clone owner, then the aggregate is a working clone.
- If you get an aggregate from a cache version owner, then the aggregate is the cache version.

For more information about aggregate objects, see "Relational Aggregate Descriptors" on page 24-2.

Unregistering Working Clones

The unit of work `unregisterObject` method lets you 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 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. The read-only classes are cached and must not be changed by the user.

Configuring 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 class A will change: you know that no class Bs will change. Before registering an instance of class B, use the following:

```
myUnitofWork.addReadOnlyClass(B.class);
```

You can then proceed with your transaction: registering class 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 clones with the working 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 class C are not be persisted because class C extends class B and class B was identified as read-only.

To identify multiple classes as read-only, add them to a `Vector` and use the following code:

```
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 99-15.

Configuring Default Read-Only Classes

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 declaration extends to its descriptors. You can declare a descriptor as read-only at development time, using either Java code or TopLink 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 declare a descriptor as read-only in TopLink Workbench, select the **Read Only** check box for the specific descriptor.

For more information, see "Configuring Read-Only Descriptors" on page 25-5.

Writing Changes Before Commit Time

By default, when you call the unit of work `commit` method, TopLink writes your changes to the data source and commits your changes.

Alternatively, you can perform a two-stage or partial commit transaction by using the unit of work `writeChanges` method prior to calling `commit` (either directly or by way of an external transaction service).

When you call the unit of work `writeChanges` method, the unit of work commit process begins, and all changes are written out to the data source. However, the data source transaction is not committed, nor will changes be merged into the shared session cache. To finalize your changes, the unit of work `commit` method must still be called (either directly or by way of an external transaction service).

After you call the unit of work `writeChanges` method, any attempt to register objects or execute object-level queries will throw an exception. You may execute report queries, noncaching queries, and data read and modify queries.

If any exception is thrown, the transaction will be rolled back (or marked rollback only) and you cannot recover the unit of work.

You can call this method only once. You cannot use this method to write out changes in an incremental fashion.

You can use the unit of work `writeChanges` method to address a variety of transaction issues, including the following:

- As an alternative to conforming (see "Using Unit of Work Method writeChanges Instead of Conforming" on page 99-12)
- To handle external transaction issues (see "Using the Unit of Work to Handle External Transaction Timeouts and Exceptions" on page 99-23)

Using Conforming Queries and Descriptors

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. 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.

Conforming is a query feature that lets you include new, changed, or deleted objects in queries within a unit of work prior to committing. This lets you to query against your relative logical or transaction view of the database.

Before you use conforming, be aware of its limitations (see "Guidelines for Using Conforming" on page 99-8) and make sure that conforming is actually necessary. For example, consider the alternative described in "Conforming Query Alternatives" on page 99-12.

This section explains the following:

- Guidelines for Using Conforming
- Using Conforming Queries
- Using Conforming Descriptors
- Conforming Query Alternatives

Note: By default, TopLink suppresses exceptions thrown during the memory search stage of conforming. For more information on handling exceptions during conforming, see "Exceptions During Conforming" on page 99-34.

Guidelines for Using Conforming

When using conforming, follow the guidelines that this section describes to ensure that conforming queries return the correct results:

- Ensure That the Query Supports Conforming
- Consider how Conforming Affects Database Results
- Register New Objects and Instantiate Relationships

Ensure That the Query Supports Conforming

Conforming is supported by the following queries:

- `ReadObjectQuery`
- `ReadAllQuery`
- Expressions
- EJB QL
- Query by example

- Query by selection object or primary key (only new or deleted objects apply)

Conforming is not supported by the following queries:

- `ReportQuery`
- `DataReadQuery`
- `DataReadQuery` (deleted objects can be conformed)
- `StoredProcedureCall` (deleted objects can be conformed)
- `EISCall` (deleted objects can be conformed)
- Expression or EJB QL queries that use database-specific functions, or subselects.

Consider how Conforming Affects Database Results

When conforming is used on a `ReadAllQuery`, the database result is first queried. If the unit of work has not yet committed any changes to the database, this result will not reflect the unit of work changes. The database results are then conformed in memory using the following criteria:

- Registered new objects that conform to the query are added.
- Modified existing objects that no longer conform are removed.
- Modified existing objects that conform are added.
- Deleted objects are removed.

> **Note:** If new objects are not explicitly registered, they are not conformed. Also, if removed object are not explicitly deleted, they are not conformed.

If the query uses ordering, ordering of conformed results is not maintained and conformed instances are added to the front of the result. To apply ordering, store the result in memory using `Collections` method `sort`, or a `TreeSet` result collection class. When using conforming on a `ReadObjectQuery`, first query the unit of work for a conforming object: if the conforming object is found, it is returned and the database access is avoided; if the conforming object is not found, the database is queried. If the unit of work has not yet committed any changes to the database, this result does not reflect the unit of work changes. The database results are then conformed in memory using the following criteria:

- If the database result no longer conforms, null is returned.
- If the database result has been deleted, null is returned.

> **Note:** If the database result returns multiple results, only the first result is considered, because it is an instance of the `ReadObjectQuery` and only a single result is expected. If the first result no longer conforms, null is returned, even if there were potential valid conforming results.
>
> If you expect the query to return multiple results, use a `ReadObjectQuery`.

Register New Objects and Instantiate Relationships

If a new object is only related to an existing object, and not explicitly registered, queries for this object are not able to conform it. If you remove, but do not explicitly delete a privately owned object, this object cannot be conformed.

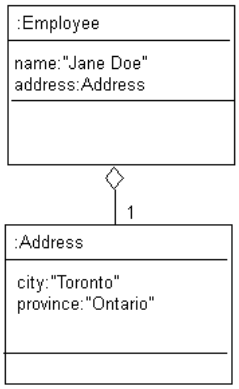
If a query traverses relationships (uses joins) and the related objects are changed, the query can only conform these objects if both of the following conditions are met:

- The source objects have been registered in the unit of work.
- The source objects' relationship has been instantiated.

TopLink provides a conforming option that forces an instantiation of indirection. However, you use this option with caution as it can cause an increased database access.

Consider the example shown in Example 99–1. In this example, you have `Employee` objects with an `address` attribute configured for indirection (see "Indirection" on page 30-5) mapped by a one-to-one mapping to an `Address` object.

Figure 99–1 Conforming Example



You want to read all employees who live in Ottawa, but first, you need to modify some of the `Address` objects to change city from Toronto to Ottawa. Jane Doe is one such employee.

First, using the `UnitOfWork`, you read all `Address` objects and change some `city` attributes (including Jane's) from Toronto to Ottawa. Then you run a conforming query to get all employees who live in Ottawa. However, for the following reasons Jane is not included in the results, even though she now lives in Ottawa:

- Jane is not returned from the database because the transaction has not yet been committed and in the database, her address still says Toronto.
- Jane cannot be added to the conformed result in memory because she is not registered in the `UnitOfWork` cache.

Conforming only recognizes *explicit* changes. In this example, Jane Doe's `Employee` object was only *implicitly* changed. In order to be considered explicitly changed, an `Employee` must meet the following criteria:

- be registered in the `UnitOfWork`
- have its `address` attribute changed (in this example, indirection must be triggered for the `address` attribute)

The correct way to handle this example would be as follows:

1. Using the `UnitOfWork`, read in all employees.

   All these `Employee` objects are now registered with the `UnitOfWork`

2. Using the same `UnitOfWork`, access the employees' addresses, instantiating the indirect relationships.
3. Modify the employees' addresses, changing some of the addresses to be in Ottawa.
4. Run the conforming query on employees with addresses inside Ottawa.

   All employees with addresses in Ottawa are returned, including both employees that were in Ottawa originally and employees whose addresses were changed in this transaction.

5. Commit the transaction.

If you do not register all employees whose address may be changed, and instantiate their address relationship, the conforming query will not include Jane.

An alternate approach is to use short transactions: the safest conforming query is one made immediately after a commit. For example:

1. Using the `UnitOfWork`, read in all addresses outside of Ottawa.
2. Modify the addresses, changing some of the addresses to be in Ottawa
3. Commit the transaction.
4. Using the `UnitOfWork`, read in all employees inside Ottawa.

Using Conforming Queries

Assume that a single `Pet` of type `Cat` already exists on the database. Examine the code shown in Example 99–6.

Example 99–6 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]
```

Using Conforming Descriptors

TopLink's support for conforming queries in the unit of work can be specified at the descriptor level.

You can define 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 TopLink Workbench.

You can configure a descriptor to always conform in the unit of work using TopLink Workbench or Java code.

To configure a descriptor to always conform in the unit of work in Java code, use `Descriptor` method `setShouldAlwaysConformResultsInUnitOfWork`, passing in an argument of `true`.

To configure a descriptor to always conform in the unit of work using TopLink, see "Configuring Unit of Work Conforming at the Descriptor Level" on page 25-6.

Conforming Query Alternatives

This section describes alternatives to conforming that may meet your needs without the performance penalty imposed by conforming. This section describes the following:

- Using Unit of Work Method writeChanges Instead of Conforming
- Using Unit of Work Properties Instead of Conforming

Using Unit of Work Method writeChanges Instead of Conforming

Using `UnitOfWork` method `writeChanges`, you can write uncommitted changes to the data source: you can execute report queries, noncaching queries, and data read and modify queries against these changes (see Example 99–7).

Example 99–7 Using Unit of Work Method writeChanges

```
UnitOfWork uow = session.acquireUnitOfWork();
    Pet pet = new Pet();
    Pet petClone = (Pet)uow.registerObject(pet);
    petClone.setId(100);
    petClone.setName("Fluffy");
    petClone.setType("Cat");

uow.writeChanges();

// Use uow to perform report, noncaching, and data read and modify queries
// against the changes made so far

uow.commit();
```

However, you can call `writeChanges` only once; any attempt to register objects or to execute object-level queries will throw an exception.

For more information, see "Writing Changes Before Commit Time" on page 99-7

Using Unit of Work Properties Instead of Conforming

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, note the following:

```
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, using unit of work properties is a simple and more efficient solution.

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 merge the serialized object with the original object in the session cache.

Using the unit of work methods listed in Table 99–1, you can merge a deserialized object into your session cache. Each method takes the serialized object as an argument and returns the original object.

Before doing so, you must ensure that the source object is in your session cache. Attempting to merge a deserialized object into a session cache that does not yet contain the object will result in a descriptor exception (see "200: ATTEMPT_TO_REGISTER_DEAD_INDIRECTION" on page 13-24). To avoid this, Oracle recommends that you first read the object instance that the deserialized object represents. If you are using a coordinated cache or your application is running in a cluster, the session you merge into may not yet contain your original object. By performing a read operation first, you guarantee that the object will be in the cache before you merge.

Table 99–1 Unit of Work Merge Methods

Method	Purpose	Used When
`mergeClone`	Merges the serialized object and all its privately owned parts (excluding non-private 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.
`mergeCloneWithReferences`	Merges the serialized object and all references into the working copy clone.	The client edits the object and the targets of its relationships and has not marked any attributes as transient.

Table 99–1 (Cont.) Unit of Work Merge Methods

Method	Purpose	Used When
`shallowMergeClone`	Merges only serialized object changes to attributes mapped with direct mappings into the working copy clone.	The client edits only 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.	The client traverses all relationships of the objects and makes changes. Note: Use `deepMergeClone` with caution. If two different copies of an object are in the same tree, TopLink 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 Sessions" on page 72-30). It automatically handles merging and lets you use a unit of work on the client.

You can merge clones with both existing and new objects. Because clones 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 99–8 shows one way to update the original object with the changes contained in the corresponding serialized object (`rmiClone`) received from a client.

Example 99–8 Merging a Serialized Object

```
update(Object original, Object rmiClone) {
  original = uow.registerObject(original);
  uow.mergeCloneWithRefereneces(rmiClone);
  uow.commit();
}
```

For more information, see "Indirection, Serialization, and Detachment" on page 30-9.

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, TopLink offers an API that lets you 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 transaction succeeds, the unit of work expires. However, if the commit transaction fails, this method does not invalidate the unit of work or its clones. This method lets you modify the registered objects in a failed unit of work and retry the commit transaction.

You should resume a unit of work only in an application that makes repeated changes to the same, small dataset. Reusing the same unit of work while accessing different datasets may result in poor performance.

Example 99–9 shows how to use the `commitAndResume` method.

Example 99–9 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 this SQL:

```
UPDATE PETOWNER SET NAME = 'Mrs. Newowner' WHERE (ID = 400)
```

Then, the `commit` call produces this 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 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.

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 copy clones 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 99–10 shows how to use nested units of work.

Example 99–10 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 execute native SQL or invoke a stored procedure within a unit of work by using unit of work method `executeNonSelectingCall` or by executing a `DataModifyQuery`. This makes the unit of work begin its database transaction early and execute the call to the data immediately.

If you release the unit of work, it will roll back the database changes. If you commit the unit of work and the commit succeeds, the unit of work will commit the changes to the database.

You can execute a `DataModifyQuery` only in a unit of work or a database session. You cannot execute a `DataModifyQuery` in a client or server session directly.

You can execute a `DataReadQuery` or use session method `executeSelectingCall` in any session type because these do not modify data.

Example 99–11 illustrates using `SQLCall` with the unit of work method `executeNonSelectingCall`.

Example 99–11 Using the executeNonSelectingCall Method

```
uow.executeNonSelectingCall(new SQLCall(mySqlString));
```

WARNING: Allowing an unverified SQL string to be passed into methods makes your application vulnerable to SQL injection attacks.

Controlling the Order of Delete Operations

"Deleting Objects" on page 98-7 explained that TopLink always properly arranges (orders) the SQL based on the mappings and foreign keys in your object model and schema. You can control the order of delete operations by the following:

- Using the Unit of Work setShouldPerformDeletesFirst Method
- Using the Descriptor addConstraintDependencies Method

Using the Unit of Work setShouldPerformDeletesFirst Method

By default, TopLink does insert and update operations first, before delete operations, to ensure that referential integrity is maintained. This is the preferred approach.

If you are forced to replace an object with unique constraints by deleting it and inserting a replacement, you may cause a constraint violation if the insert operation occurs before the delete operation. In this case, call `setShouldPerformDeletesFirst` to perform the delete operation before the insert operation.

Using the Descriptor addConstraintDependencies Method

The constraints used by 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 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, nonprivate relationship with C. You want to delete A (and in doing so the included Bs) but before deleting the Bs, for some of them (not all) you want to delete the associated object C.

There are two possible solutions:

1. Using deleteAllObjects Without addConstraintDependencies
2. Using deleteAllObjects With addConstraintDependencies

Using deleteAllObjects Without addConstraintDependencies

In the first option, you do not identify the one-to-many (A to B) relationship as privately owned. When deleting an A object, you must delete all of its B objects, as well as any C objects, as shown in the following example:

```
uow.deleteObject(existingA);
uow.deleteAllObjects(existingA.getBs());
// delete one of the Cs
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, keep the one-to-many (A to B) relationship privately owned and add a constraint dependency from A to C, as shown in the following example:

```
session.getDescriptor(A.class).addConstraintDependencies(C.class);
```

Now the delete code would be:

```
uow.deleteObject(existingA);
// delete one of the Cs
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 object is deleted before A and C. The main difference is that the second option generates fewer SQL statements, as it knows that it is deleting the entire set of Bs related from A.

Using Optimistic Read Locking With forceUpdateToVersionField

If your descriptors are configured to use an optimistic version locking policy (see "Optimistic Version Locking Policies" on page 23-18) or field locking policy (see "Optimistic Field Locking Policies" on page 23-20), use the unit of work method `forceUpdateToVersionField` to solve either or both of the following problems:

- You want an `OptimisticLockingException` thrown at commit time if an object you read in a transaction has changed since it was registered even though you have not changed the object in your transaction (see "Forcing a Check of the Optimistic Read Lock" on page 99-18).
- You modify an object in a transaction in such a way that its version field is not updated but you want to alert other threads of the change by way of the version field (see "Forcing a Version Field Update" on page 99-19).

  For example, you change a privately owned object that has its own database table so the parent object's version field is not, by default, updated. In this case, you can use `forceUpdateToVersionField` to update the parent's version field.

  As an alternative to this approach, consider "Optimistic Version Locking Policies and Cascading" on page 23-19.

To remove `forceUpdateToVersionField` configuration from an object before a commit operation, use the unit of work method `removeForceUpdateToVersionField` (see "Disabling forceUpdateToVersionField" on page 99-20).

Forcing a Check of the Optimistic Read Lock

When you read an object with the unit of work, optimistic lock checking is not applied to that object at commit time unless you change the object. However, there are times when you want your transaction to fail if the state of an object has changed since it was read, even though you have not modified the object.

Example 99–12 shows a transaction that updates a mortgage rate by multiplying the central bank prime rate by 1.25. The transaction forces 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 99–12 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();
} catch(OptimisticLockException exception) {
    /* Refresh the out-of-date object */
    session.refreshObject(exception.getObject());
    /* Retry… */
}
```

For another example that forces both optimistic locking and a version field update, see Example 99–13 in "Forcing a Version Field Update" on page 99-19.

Forcing a Version Field Update

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. In other words, when a relationship is changed in a one-to-many or one-to-one target foreign key mapping, by default, the version field (if any) of the affected object is not changed.

If you configure a descriptor to refresh the cache only if the database version is newer than the cache version (using descriptor method `onlyRefreshCacheIfNewerVersion`), and such a relationship changes, you will not be able to refresh the object at all. Because the version has not changed, the unit of work method `refreshObject` and even a query with `refreshIdentityMapResults` option set to `true` cannot refresh the object.

Using the unit of work method `forceUpdateToVersionField` passing in both the unit of work copy clone and `true` value will ensure that the object's version field is updated when such a change is made. It will also ensure that changes to the object before it is refreshed will result in optimistic locking exceptions, preventing the writing of stale data (see "Forcing a Check of the Optimistic Read Lock" on page 99-18).

Example 99–13 and Example 99–14 show transactions executing in separate threads that access the same customer object concurrently. The unit of work method forceUpdateToVersionField is used to ensure that changes to the customer object in one thread are detected by the other threads.

Example 99–13 shows a transaction in which an *invoice* thread calculates an invoice for a customer. Example 99–14 shows a transaction in which another thread, the *service* thread, adds a service to the same customer or modifies the current service. In either case, the *service* thread must inform the *invoice* thread, which adds the changes to the invoice.

Example 99–13 Optimistic Read Lock with Version Increment: Service Thread

```
/* 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… */
}
```

Example 99–14 Optimistic Read Lock with Version Increment: Invoice Thread

```
/* 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 service 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 privately 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… */
}
```

Disabling forceUpdateToVersionField

The `forceUpdateToVersionField` configuration you apply to an object stays in effect for the lifetime of your unit of work. After you commit your transaction, `forceUpdateToVersionField` configuration no longer applies.

To remove `forceUpdateToVersionField` configuration from an object before commit time, use the unit of work method `removeForceUpdateToVersionField`. TopLink will not apply optimistic read locking to the object unless you change it in this transaction (that is, unless you modify its direct-to-field or aggregate object mapping attribute).

Implementing User and Date Auditing With the Unit of Work

Auditing datasource changes is a common requirement: when a user commits a change to the datasource, the application updates a field in the datasource to record the user who made the change and the date.

For example, suppose each row in your database schema includes fields `lastUpdateBy` (to record the user name of the user who commits a change) and `lastUpdateOn` (to record the date of the change).

You can use `UnitOfWork` method `setProperty` to record the name of the user who acquires the `UnitOfWork` and implement a descriptor event listener for `AboutToUpdateEvent` descriptor events that extracts the property and updates the `lastUpdateBy` and `lastUpdateOn` fields.

For more information, see the following:

- "Acquiring a Unit of Work" on page 98-1
- "Configuring a Domain Object Method as an Event Handler" on page 25-59
- "Configuring a Descriptor Event Listener as an Event Handler" on page 25-62

Integrating the Unit of Work With an External Transaction Service

To support transactions managed by an application server's external transaction service, TopLink supports external connection pools and external transaction controller classes for supported servers. This lets you incorporate external transaction service support into your application, and use the unit of work with transactions managed externally by the server. For more information, see "Unit of Work Transaction Demarcation" on page 97-2.

To integrate a unit of work with an external transaction service, you must enable the use of the following:

- an external transaction controller (see "Configuring the Server Platform" on page 74-14)
- an external connection pool (see "Configuring External Connection Pooling" on page 82-2)

After you configure external connection pool and external transaction controller support, you use a unit of work in your TopLink application as you would typically, with few exceptions. This section describes these exceptions as follows:

- Acquiring a Unit of Work With an External Transaction Service
- Using a Unit of Work When an External Transaction Exists
- Using a Unit of Work When No External Transaction Exists
- Using the Unit of Work to Handle External Transaction Timeouts and Exceptions

Acquiring a Unit of Work With an External Transaction Service

You use a unit of work to commit changes to a data source even when using an external transaction service. 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 99–15.

> **Note:** Although there are other ways to commit changes to a data source using an external transaction service, Oracle recommends using the `getActiveUnitOfWork` method.

The `getActiveUnitOfWork` method searches for an existing external transaction in the following way:

- 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 TopLink returns a unit of work that is not `null`, use it exactly as you would typically: 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 99-22).

If TopLink returns a `null` unit of work, start an external transaction explicitly through the `UserTransaction` interface.

Example 99–15 Using a Unit of Work With an External Transaction Service

```
// Read in any pet
Pet pet = (Pet)clientSession.readObject(Pet.class);
UnitOfWork uow = clientSession.getActiveUnitOfWork();
if (uow == null) {
    throw new RuntimeException("No transaction started");;
}
Pet petClone = (Pet)uow.registerObject(pet);
petClone.setName("Furry");
```

Using a Unit of Work When an External Transaction Exists

When `getActiveUnitOfWork` returns a unit of work that is not `null`, 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 external 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, TopLink receives synchronization callbacks at key points during the commit transaction.

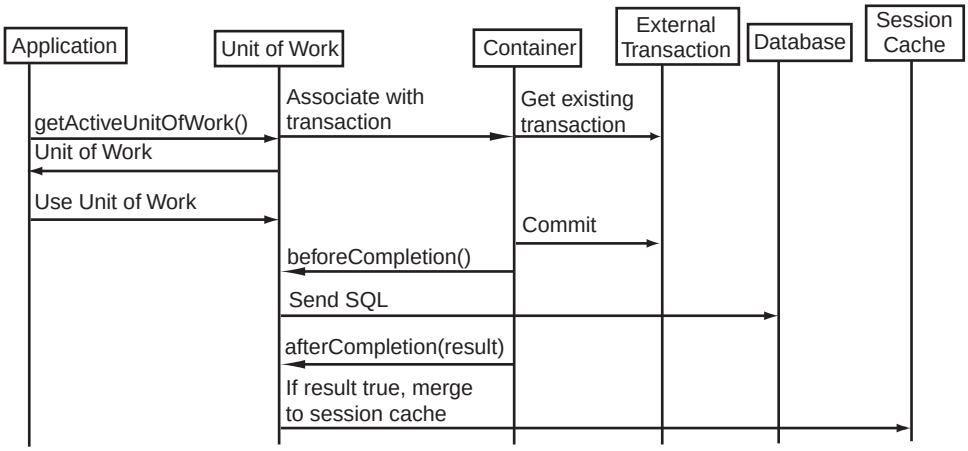
The unit of work sends the required SQL to the database when it receives the `beforeCompletion` callback.

The unit of work uses the `Boolean` argument received from the `afterCompletion` callback to determine if the commit was successful (`true`) or not (`false`).

If the commit transaction was successful, the unit of work merges changes to the session cache. If the commit transaction was unsuccessful, the unit of work discards the changes.

Figure 99–2 shows the life cycle of a unit of work when an external transaction exists.

Figure 99–2 Unit of Work When an External Transaction Exists



Using a Unit of Work When No External Transaction Exists

When the `getActiveUnitOfWork` method returns a null unit of work, there is no existing external transaction. You must start a new external transaction.

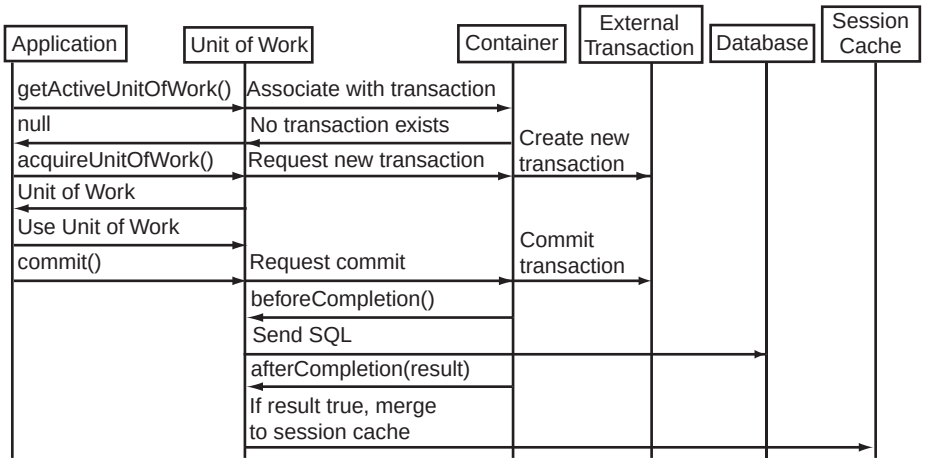
Do this either by starting an external transaction explicitly using the `UserTransaction` interface, or by acquiring a new unit of work using the `acquireUnitOfWork` method on the server session.

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, and the database updates and cache merge occur based upon those callbacks.

Figure 99–3 Unit of Work When No External Transaction Exists



Using the Unit of Work to Handle External Transaction Timeouts and Exceptions

This section describes two common problems with external transactions:

- External Transaction Commit Timeouts

- External Transaction Commit Exceptions

External Transaction Commit Timeouts

When an external transaction is committed, the external transaction service expects each transaction owner to commit its portion of the overall transaction within a finite amount of time. If any individual transaction exceeds this timeout interval, the external transaction service will fail the specific transaction and roll it back (or mark it rollback only).

If your transaction is large and its commit transaction may exceed the external transaction service timeout interval, use `UnitOfWork` method `writeChanges` to write changes to the data source before committing the external transaction. This will reduce the time it takes for your part of the global transaction to commit.

For more information about the `UnitOfWork` method `writeChanges`, including restrictions and warnings, see "Writing Changes Before Commit Time" on page 99-7.

External Transaction Commit Exceptions

When you use the unit of work with an external transaction service, commit exceptions may not be thrown until long after your application thread calls its `UnitOfWork` method `commit` and returns. In this case, commit exceptions are thrown to the client of the container-managed transaction (CMT) call, forcing the client to handle this server-side failure.

You can use the `UnitOfWork` method `writeChanges` to write changes to the data source before the external transaction commits. This allows your application thread to catch and handle most exceptions that could be thrown at the time the external transaction service commits the global transaction.

For more information about the `UnitOfWork` method `writeChanges`, including restrictions and warnings, see "Writing Changes Before Commit Time" on page 99-7.

For more information on handling unit of work exceptions in general, see "Handling Exceptions" on page 99-33.

Integrating the Unit of Work With CMP

All modifications to persistent beans should be carried out in the context of a transaction.

Modifying entity beans without a transaction can lead to an inconsistent state, potentially corrupting the values in the TopLink cache. Because of this, TopLink does not support modifying a bean through its remote interface when no transaction is active. If you attempt to do so, TopLink simply does not write changes to the database.

Although TopLink does not let you modify an entity bean through its remote interface without a transaction, TopLink *does* let you invoke methods on its home interface that change the state in the underlying database without a transaction. For example, you may invoke remove and create methods on the home interface of an entity bean without a transaction.

To integrate TopLink transactions and the unit of work with container-managed persistence (CMP), you must consider the following:

- CMP Transaction Attribute
- Local Transactions
- Nondeferred Changes

CMP Transaction Attribute

To ensure that all modifications to persistent beans are carried out in the context of a transaction, transactional attributes must be properly specified in the bean deployment descriptors.

The transaction may be either client-controlled or container-controlled.

Client-controlled transactions are started explicitly by your application by way of the `javax.transaction.UserTransaction` interface.

Container-controlled transactions are started implicitly by the container to satisfy the transaction attribute configuration when a bean method is invoked in the absence of a client-controlled transaction.

Table 99–2 shows what transaction (if any) an EJB method invocation uses depending on how its transaction attribute is configured and whether or not a client-controlled transaction exists at the time the method is invoked.

Oracle recommends that you do not make modifications to entity beans under conditions identified as "Use no transaction" in Table 99–2. Oracle also recommends that you avoid using the `Supports` transaction attribute because it leads to a nontransactional state whenever the client does not explicitly provide a transaction.

Table 99–2 EJB Transaction State by Transaction Attribute

Transaction Attribute	Client-Controlled Transaction Exists	Client-Controlled Transaction Does Not Exist
`NotSupported`	Use no transaction	Use no transaction
`Supports`	Use client-controlled transaction	Use no transaction
`Required`	Use client-controlled transaction	Use container-controlled transaction
`RequiresNew`	Use client-controlled transaction	Use container-controlled transaction
`Mandatory`	Use client-controlled transaction	Exception raised
`Never`	Exception raised	Use no transaction

Depending on the EJB container you use, you may be able to write without a container-controlled transaction (see "Integrating the Unit of Work With an External Transaction Service" on page 99-21). In this case, TopLink automatically uses a transaction of its own, referred to as a local transaction (see "Local Transactions" on page 99-25).

Local Transactions

Some EJB containers (such as OC4J) support writing without an active JTA transaction.

If you execute a bean method outside a JTA transaction while the transaction attribute (see "CMP Transaction Attribute" on page 99-25) is set to `Supports`, `NotSupported`, or `Never`, TopLink performs the operation within a local unit of work and commits the unit of work at the end of the method. This unit of work is referred to as a local transaction.

The reason for this is because the update semantics in the EJB specification are left undefined for these scenarios, and a proper transactional model demands that a transaction be active before being able to modify data. TopLink also requires change operations to occur within a unit of work to ensure that the session cache remains consistent.

Nondeferred Changes

Some EJB containers (such as OC4J) support nondeferred changes: the ability to modify the data source immediately as you change the persistent fields of an entity bean.

Using nondeferred changes, you can achieve backwards compatibility with the native behavior of some EJB containers (such as OC4J) and you can accommodate advanced applications that rely on the database and entity changes being synchronized for such things as triggers or stored procedures based on transient state within the transaction, deletion and creation of rows with the same primary key, or other complex queries that depend on transient transaction state.

Nondeferred changes have the disadvantage of being the least efficient approach: they produce the greatest number of data source interactions.

By default, TopLink defers all changes until commit time. This is the most efficient approach that produces the least number of data source interactions.

For more information, see "Nondeferred Changes" on page 23-3.

Database Transaction Isolation Levels

Achieving a particular database transaction isolation level in a TopLink application is more involved than simply using the `DatabaseLogin` method `setTransactionIsolation`.

In a typical TopLink application and in J2EE applications that require persistence in general, a variety of factors affect when database transaction isolation levels apply and to what extent a particular database transaction isolation can be achieved.

This section describes these factors and provides guidelines on configuring and using TopLink to achieve each database transaction isolation level to the extent possible given these factors.

This section includes the following:

- General Factors Affecting Transaction Isolation Level
- Read Uncommitted Level
- Read Committed Level
- Repeatable Read Levels
- Serializable Read Levels

General Factors Affecting Transaction Isolation Level

This section describes some of the important factors and variables that may affect the degree to which your TopLink application can achieve a particular database transaction isolation level. These factors include the following:

- External Applications
- TopLink Coordinated Cache
- DatabaseLogin Method setTransactionIsolation
- Reading Through the Write Connection
- Managing Cache Access
- CMP and External Transactions

External Applications

In many cases, your TopLink application is not the only application that can update to the database. External, non-TopLink applications, can also update the database at any time.

In this case, your TopLink application must use the `ObjectLevelReadQuery` method `refreshIdentityMapResult` (see "Refreshing the Cache" on page 93-35) or `Descriptor` methods `alwaysRefreshCache` and `disableCacheHits` (see "Configuring Cache Refreshing" on page 25-27).

For more information, see "Managing Cache Access" on page 99-29.

If the external application can update a version field in the database, your TopLink application could use `alwaysRefreshCache` in conjunction with `Descriptor` method `onlyRefreshCacheIfNewerVersion` to ensure that refresh operations are performed only when required. Another, recommended way to achieve this, is to use the descriptor isolated cache option (see "Cache Isolation" on page 87-9), as well as cache invalidation (see "Cache Invalidation" on page 87-8).

TopLink Coordinated Cache

Consider multiple TopLink applications (each running on its own application server instance) configured to use a distributed, coordinated cache (as described in "Understanding Cache Coordination" on page 87-10). A TopLink application instance first commits changes to its own cache before the change is distributed to other caches. Because cache coordination is not instantaneous, there is a possibility that one TopLink application instance may read an older version of an object from its cache before a cache coordination message is received.

To provide your TopLink application with the most up-to-date version of an object use the descriptor isolated cache option (see "Cache Isolation" on page 87-9), as well as cache invalidation (see "Cache Invalidation" on page 87-8).

You can also avoid stale data by using `Descriptor` methods `alwaysRefreshCache` and `disableCacheHits`. For more information on the `disableCacheHits` method, see "Managing Cache Access" on page 99-29.

Caution: Using `Descriptor` methods `alwaysRefreshCache` and `disableCacheHits` will result in frequent database hits. Use only when absolutely necessary.

DatabaseLogin Method setTransactionIsolation

Use the `DatabaseLogin` method `setTransActionIsolation` to configure the database transaction isolation level that TopLink applies to any database connection it obtains, for example:

```
databaseLogin.setTransactionIsolation(DatabaseLogin.TRANSACTION_SERIALIZABLE);
```

This method sets the transaction isolation level used for both database read and write operations on the database connections obtained from either an internal or external connection pool (see "Connection Pools" on page 72-3), for both internal transactions and external transactions as in the case of CMP.

However, with TopLink, by default read operations use a different database connection than write operations, typically obtained from an external connection pool, or may use the cache, bypassing the database entirely. Thus, with TopLink, by default, read operations are always performed outside the transaction or unit of work, even if

you perform the read operation within a transaction or unit of work. Although database transaction isolation applies to both read and write connections, the read is not performed as part of the transaction. Therefore, the read operation overrides the transaction isolation set on the database.

Depending on the level of transaction isolation you are trying to achieve, you may require that the same transaction isolation be applied to both read and write operations. You must take special action to make TopLink use the same connection for both read and write operations. For more information, see "Reading Through the Write Connection" on page 99-28.

Reading Through the Write Connection

Recall that TopLink, by default, performs read operations with a different database connection than used for write operations ("DatabaseLogin Method setTransactionIsolation" on page 99-27). However, from the perspective of database transaction isolation, there is a one-to-one relationship between transaction and database connection: that is, all database operations (including read operations) must use the same database connection in order to achieve a particular database transaction isolation level.

In general, when TopLink performs a read operation, if a write connection already exists, TopLink will use the write connection for the read operation. This is called "reading through the write connection." If a write connection does not yet exist, TopLink will acquire another connection and use that for the read operation.

You can configure TopLink to allocate a write connection early using any of the following:

- Pessimistic Locking Query
- Unit of Work Method beginTransactionEarly
- ConnectionPolicy Method setShouldUseExclusiveConnection

> **Caution:** Depending on the database transaction isolated level reading through the write connection may lock the object being read. This will affect performance and reduce concurrency. Oracle recommends that you do not use these advanced techniques unless strict database transaction isolation is absolutely necessary.

For more information, see "CMP and External Transactions" on page 99-30.

Pessimistic Locking Query When you use pessimistic locking (`ObjectLevelReadQuery` methods `acquireLocks` or `acquireLocksWithoutWaiting` or `Session` method `refreshAndLockObject`), TopLink does the following:

- Allocates a write connection used for both read and write operations.
- Always reads from the database.
- Always updates the cache with the database version.

Unit of Work Method beginTransactionEarly This method is advanced API. If you call `beginTransactionEarly` on an instance of a unit of work, all read operations should be performed through that instance of the unit of work.

This method starts a database transaction immediately: any objects you read will lock data in the database before commit time, reducing concurrency.

ConnectionPolicy Method setShouldUseExclusiveConnection Client sessions can access the data source using a connection pool or an exclusive connection. To use an exclusive connection, acquire your client session using a `ConnectionPolicy` (see "Acquiring a Client Session That Uses Exclusive Connections" on page 75-7).

If you are using isolated client sessions (see "Isolated Client Sessions" on page 72-19), you can use exclusive connections for reading isolated data. In this case, you can configure TopLink to acquire an exclusive connection from the write connection pool and use it for both writing and reading isolated data. However, TopLink still acquires a shared connection from the read connection pool for reading nonisolated data.

For more information, see "Exclusive Write Connections" on page 74-19.

Managing Cache Access

By default, TopLink uses the shared session cache as much as possible. Doing so increases concurrency and improves performance. However, to achieve a particular transaction isolation level, you may need to avoid the cache using some or all the following:

- Isolated Client Session Cache
- ReadObjectQuery
- ReadAllQuery
- Descriptor Method disableCacheHits
- DatabaseQuery Method dontMaintainCache

Isolated Client Session Cache This method always goes to the database for the initial read operation of an object whose descriptor is configured as isolated. By avoiding the shared session cache, you do not need to use the more complicated descriptor and query APIs to disable cache hits or always refresh. For more information about isolated client sessions, see "Isolated Client Sessions" on page 72-19. This is particularly useful for achieving serializable transaction isolation (see "Serializable Read Levels" on page 99-30).

ReadObjectQuery This API goes to the database unless it is a primary key-based query, in which case it will go to the cache first. For information on how to avoid the cache entirely in this case, see "Descriptor Method disableCacheHits" on page 99-29.

ReadAllQuery This API always goes to the databases. For information on how to avoid the cache entirely in this case, see the description of the `Descriptor` method `alwaysRefreshCache` in "Cache Refresh API" on page 87-14.

Descriptor Method disableCacheHits This API allows for cache hits on primary key, read-object queries to be disabled. This can be used with the `Descriptor` method `alwaysRefreshCache` to ensure queries always go to the database.

DatabaseQuery Method dontMaintainCache This is a query-level means of preventing objects from being added to the shared session cache. Using an isolated client session (see "Isolated Client Session Cache" on page 99-29) is a simpler approach to achieving the same ends.

CMP and External Transactions

In general, the transaction isolation information in this section applies to both CMP and non-CMP applications, with the following exception:

For application servers other than OC4J, when using a TopLink application with CMP, Oracle recommends that you configure your container to use separate read and write connection pools, and to associate only the write connections with an external transaction. This means the read connections do not participate in the transaction.

> **Note:** OC4J always uses the same connection pool for reading and writing: it uses JTA connections from that pool for writing, and non-JTA connections from the pool for reading.

However, because TopLink treats EJB finders as just another type of query, you can use your descriptor configuration to exploit the options described in "Reading Through the Write Connection" on page 99-28. For example, if you configure a descriptor to use pessimistic locking (see "Configuring Locking Policy" on page 25-64), then when its finder is invoked it will allocate a write connection early and both read and write operations will use the same connection.

Refer to "Externally Managed Transactional Data Sources" on page 81-1 for more information on external transactions with transactional data sources.

Read Uncommitted Level

Oracle does not recommend using this transaction isolation level.

In general, a read uncommitted operation is not necessary. Using TopLink, a transaction isolation of read committed gives you better performance than read uncommitted but with greatly improved data integrity.

Read Committed Level

Using the unit of work guarantees that you will read only committed data in the shared session cache or committed data in the database.

Repeatable Read Levels

To achieve repeatable read operations, you must use a unit of work, you must register all objects in the unit of work (both objects you intend to modify and objects you intend only to read), and you must use `ObjectLevelReadQuery` method `conformResultsInUnitOfWork` or `Descriptor` method `alwaysConformResultsInUnitOfWork`.

By doing so, each time you query a registered object, you will get the version of the object as it currently is in your unit of work.

Serializable Read Levels

To achieve serializable transaction isolation with TopLink, Oracle recommends that you use an isolated client session (see "Isolated Client Sessions" on page 72-19) as follows:

1. Configure the database transaction isolation as serializable.

2. Configure objects as isolated (see "Configuring Cache Isolation at the Project Level" on page 19-16 or "Configuring Cache Isolation at the Descriptor Level" on page 25-37).
3. Use the `UnitOfWork` method `beginTransactionEarly` (see "Unit of Work Method beginTransactionEarly" on page 99-28).

If you are only concerned about the write aspect of serializable, optimistic locking is sufficient.

To prevent phantom read transactions (that is, when a transaction detects that new records that have been added to the database after the transaction started), use the `ReadQuery` method `cacheQueryResults`.

Troubleshooting a Unit of Work

This section examines common unit of work problems and debugging techniques, including the following:

- 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
- Validating a Unit of Work

Avoiding the Use of Post-Commit Clones

A common unit of work error is holding on to clones after commit time. Typically the clones are stored in a static variable and the developer incorrectly believes 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 believes is the cache copy is not updated (because a unit of work updates only the cache copy, not old clones).

Consider the error in Example 99–16. In this example you 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 clone 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 99–16 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 is 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 98-2, it was 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 code is equivalent to the code in the subsequent example:

```
session.readObject(pet);
```

The following is equivalent to the preceding code:

```
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. As a result, you can use very quick and simple method for accessing the cache copy of an object as shown in Example 99–17.

Example 99–17 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 known objects 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 99–18.

Example 99–18 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

This section explains how to handle the following:

- Exceptions at Commit Time
- Exceptions During Conforming

Exceptions at Commit Time

TopLink exceptions are instances of `RuntimeException`, which means that methods that throw them do not have to be placed in a `try-catch` block.

However, the unit of work `commit` method is one that should be called within a `try-catch` block to deal with problems that may arise.

Example 99–19 shows one way to handle unit of work exceptions:

Example 99–19 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
```

See "Database Exceptions (4002 – 4018)" on page 13-27 for more information on `DatabaseException`.

If you use optimistic locking, you must catch exceptions at commit time 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 and the Unit of Work" on page 97-12.

If you are using an external transaction service, exceptions may be thrown long after your `UnitOfWork` code has returned. Using `UnitOfWork` method `writeChanges`, you can catch and handle most exceptions before the external transaction is committed. For more information, see "External Transaction Commit Exceptions" on page 99-24.

Exceptions During Conforming

You can conform query results in a unit of work across one-to-many relationships and a combination of both one-to-one and one-to-many relationships. Example 99–20 illustrates a query across two levels of relationships, one-to-many and one-to-one.

Example 99–20 Querying Across Two Levels of Relationship

```
Expression exp =
  bldr.anyOf("managedEmployees").get("address").get("city").equal("Perth");
```

By default, any exceptions thrown during conforming are suppressed. However, you can use the `UnitOfWork` method `setShouldThrowConformExceptions` to make the unit of work throw all conforming exceptions. This method takes one `int` argument with the following values:

- 0–do not throw conform exceptions (default)
- 1–throw all conform exceptions

For more information on customizing exception handling when using conforming and in-memory queries, see "Handling Exceptions Resulting From In-Memory Queries" on page 93-33.

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 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. TopLink offers the following APIs to toggle validation:

- `dontPerformValidation`: disables validation

- `performFullValidation`: enables validation

Validating the Unit of Work Before Commit Time

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 trace of all traversed objects. This may help you more easily find the problem. You can call this method at any time on a unit of work.

Glossary

This glossary contains terms and abbreviations that you should be familiar with when using Oracle TopLink.

attribute

A variable of a class or object. In TopLink, an *attribute* describes all instance variables of a class. Every attribute contains a single mapping. Attributes store primitive data such as integers, and simple Java types such as `String` or `Date`.

authentication

The means by which a data source validates a user's identity and determines whether or not the user has sufficient privileges to perform a given action.

bean class

The implementation of the bean. The bean is accessed from the client using the home and remote interfaces.

bean-managed persistence (BMP)

A scheme for persisting entity beans that requires the developer to manually code the methods that perform the persistence.

Compare to container-managed persistence (CMP).

branch class

Has a persistent superclass and also has subclasses. By default, queries performed on the branch class return instances of the branch class and any of its subclasses. However, the branch class can be configured so that queries on it return only instances of itself without instances of its subclasses.

Compare to leaf class.

class

A category of objects. Classes allow data and method to be grouped together.

class indicator field

A field in the table of the root class that indicates which subclass should be instantiated

client session broker

A collection of client sessions, one from each server session associated with the session broker.

connection pool

A collection of reusable connections to a single data source.

container-managed persistence (CMP)

A scheme for persisting entity beans that uses information supplied by the developer or deployer to perform the persistence

Compare to bean-managed persistence (BMP).

custom SQL

Refers to any non-TopLink-generated SQL used through TopLink. This includes hard-coded SQL and stored procedure calls.

data definition language (DDL)

The data definition part of the structured query language (SQL). TopLink Workbench can generate DDL creation scripts that can be used to create tables on the desired database.

database session

A database session provides a client application with a single data store connection, for simple, standalone applications in which a single connection services all data store requests for one user.

default mapping

A relational persistence framework term that refers to making the framework automatically generate the object descriptor metadata (including such things as mappings, login data, database platform, locking, and foreign keys). Default mapping is available for TopLink projects using EJB 2.0 CMP applications with OC4J.

dependent class path (IBM WebSphere)

Location where nonbean classes are specified. TopLink requires that the bean classes be included here since they are referenced by the project.

deployment descriptor

A set of XML files that provide the additional required information to install an EJB within its server. Typically, this incudes security, transaction, relationship, and persistence information.

Compare with TopLink descriptors.

descriptors

An TopLink object that describes how an object's attributes and relationships are to be represented in relational database table(s). An "TopLink descriptor" is not the same as a deployment descriptor, although it plays a similar role.

direct access

By default, TopLink accesses public attributes directly when writing the attributes of the object to the database or reading the attributes of the object from the database.

Compare to method access.

direct mapping

There are two basic ways of storing object attributes directly in a table:

- The information can be stored directly if the attribute type is comparable to a database type.
- If there is no database primitive type that is logically comparable to the attribute's type, it must be transformed on its way to and from the database

TopLink provides five classes of direct mappings.

Compare to relationship mapping.

Enterprise Java Beans (EJB)

EJB are server-side domain objects that fit into a standard component-based architecture for building enterprise applications with Java. They are objects that become distributed, transactional, and secure components. TopLink Workbench uses three types of EJB: session beans, entity beans, and message-driven beans.

expressions

The TopLink equivalent of an SQL conditional clause. TopLink expressions are specified using the `Expression` and `ExpressionBuilder` classes.

entity beans

EJB that represent a persistent data object. TopLink uses two schemes for persisting entity beans: bean-managed persistence (BMP) and container-managed persistence (CMP).

fetch group

A performance enhancement that defines a subset of object attributes to be loaded initially and ensures that all other attributes are loaded on demand.

hub

A common connection point for devices in a network.

identity map

Used to cache objects for performance and to maintain object identity.

See also object identity.

independent relationship

A relationship in which the source and target are public objects that exist independently; the destruction of one object does not necessarily imply the destruction of the other.

Compare to private relationship.

indirection

An indirection object is one that acts as a stand-in for another object. In TopLink, indirection is implemented through value holders, which delay database access through acting as substitute for any object relationships.

inheritance

Describes how a child class inherits the characteristics of its parent class. TopLink supports multiple approaches to database implementations that preserve the inheritance relationship.

in-memory query

A query that is run against the shared session cache.

instantiate

Create an instance of a Java class.

J2C

The J2EE Connector architecture (J2C) adapter is a way to persist Java objects to a nonrelational data source, such as XML.

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 multitiered web-based applications.

J2EE Containers

A J2EE container is a run-time environment for Enterprise Java Beans (EJB) 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 Containers for J2EE.

Java Messaging Service (JMS)

The JMS API is a protocol for communication that provides asynchronous communication between components in a distributed computing environment.

Java Naming and Directory Interface (JNDI)

The JDBC specification recommends using a JNDI naming service to acquire a connection to a database. TopLink supports acquiring a database connection in this fashion. To take advantage of this feature, construct and configure an instance of `oracle.toplink.jndi.JNDIConnector` and pass it to the project login object using the `setConnector` method.

Java Transaction API (JTA)

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.

leaf class

Has a persistent superclass in the hierarchy but does not have subclasses; queries performed on the leaf class can return only instances of the leaf class.

Compare to branch class.

locking policy

A mechanism that ensures one user does not overwrite another users's work. TopLink descriptors support optimistic and pessimistic locking policies.

mappings

Describe how individual Java objects and attributes relate to a data source.

message-driven beans

An EJB that processes asynchronous Java Messaging Service (JMS) messages. For TopLink clients, a message-driven bean is simply a JMS consumer with no conversational state and no home or remote interfaces.

method access

The application registers accessor methods for the attribute.

Compare to direct access.

named query

A TopLink query that is created and stored, by name, in a session for later retrieval and execution

object identity

Ensures that each object is represented by one and only one instance in the application; that is, multiple retrievals of the same object return references to the same object instance, not multiple copies of the same object. Violating object identity can corrupt the object model.

See also identity map.

optimistic locking

Also known as write locking; allows unlimited read access to objects. A client can write an object to the database only if the object has not changed since it was last read.

Compare to pessimistic locking.

packet

A piece of a message transmitted over a packet-switching network. One of the key features of a packet is that it contains the destination address in addition to the data.

packet time-to-live

A number of hops that session data packets can take before expiring. The default is 2.

See also packet.

persist

In object technology, the storage of an Java object by a data source.

pessimistic locking

Objects are locked before they are edited, which ensures that only one client is editing the object at any given time.

Compare to optimistic locking.

primary key

A field (or combination of fields) that uniquely identifies a record in the data source.

private relationship

A relationship in which the target object is considered to be a private component of the source object; the target object cannot exist without the source and is accessible only through the source object; furthermore, if the source object is destroyed, the target object is destroyed as well.

Compare to independent relationship.

query manager

An object, owned by a descriptor, that controls the way the descriptor accesses the database. The query manager generates its own default SQL to access the database in a transparent manner.

query optimization

TopLink supports two forms of query optimization: joining and batch reading. Their purpose is to optimize database access through reducing the number of database calls required to read a group of objects.

relationship

In TopLink, a reference between two TopLink-enabled objects.

relationship mapping

Persistent objects use relationship mappings to store references to instances of other persistent classes. The appropriate mapping class is chosen primarily by the cardinality of the relationship. TopLink provides five classes of relationship mappings.

Compare to direct mapping.

Remote Method Invocation (RMI)

A set of protocols that enable Java objects to communicate remotely with other Java objects.

remote session

A remote session is a client-side session that communicates over RMI with a corresponding client session and server session on the server side. Remote sessions handle object identity and marshalling and unmarshalling between client side and server side.

service channel

A name of the TopLink coordinated cache channel to which sessions subscribe in order to participate in the same coordinated cache.

session beans

EJB that represent a business operation, task, or process. TopLink can use session beans to make the regular Java objects they access persistent, or to wrap other legacy applications.

stale data

An artifact of caching, in which an object in the cache is not the most recent version committed to the data source.

TopLink session broker

A mechanism that enables client applications to transparently access multiple databases through a single TopLink session.

unit of work

A transactional TopLink session that allows for a transaction to occur at the object level not only the database level. Changes to objects are not visible globally until the unit of work is committed.

value holder

A wrapping object used by TopLink to delay database access.

Index

A

access
  data access, 81-1
  direct, 32-32
  modifiers, 4-42
  optimizing data access, 11-14
  remote sessions, 72-32
access method
  direct, 32-14
  generating, 4-30
  mappings, 32-15
  method, 32-14
  specifying, 32-32
access modifiers, classes, 4-42
acquiring
  client sessions, 75-6
  sessions, at runtime, 72-5
  unit of work, 98-1
activating descriptors, 4-10
Add Named Query dialog, 25-12, 25-18, 25-20
Add New Class dialog, 4-40, 4-53
Add New Table button, 4-22
Add or Refresh Class button, 4-50
`addConstraingDependencies()`, 99-17
address
  multicast group, 88-4
  multicast port, 88-5
Add/Update Existing Tables from Database button, 4-23
advanced properties for descriptors, 19-7
After Load tab, 25-81
aggregate collection relational mappings
  and EJB, 33-11
  configuring, 41-1
  understanding, 33-10
aggregate descriptors
  about, 23-5
  aggregate object mapping, 23-5
  EIS projects, 23-8
  EJB 3.0, 23-6
  inheritance, 23-17
  relational projects, 23-5
  XML projects, 23-8
aggregate object relational mappings
  aggregate descriptors, and, 23-5
  configuring, 43-1
  understanding, 33-12
aggregation, isolated client sessions, 72-24
`AllFieldsLockingPolicy`, 23-20
allows none, 25-18, 95-15
allows null, 25-18, 95-15
amending descriptors, 2-21, 23-5, 25-81
  see also after load
announcement delay, 88-11
Ant, integrating with Oracle TopLink Workbench, 4-53
any collection XML mappings
  configuring, 69-1
  understanding, 62-29
any object XML mappings
  configuring, 68-1
  understanding, 62-27
application development
  deploying, 10-1
  mapping, 2-21
  querying, 2-13, 71-2
  troubleshooting, 15-1
application layer, remote sessions, 72-31
application servers
  EJB support, customizing, 7-25
  integrating with Oracle TopLink, 7-1
  logging, 72-9
  optimization, 11-30
  setter parameter type checking, 7-25
  single-object finder return type checking, 7-25, 7-26
  software requirements, 7-2
  target platforms, 2-5
  unknown primary key class support, 7-25
architectures
  application, 1-4
  BMP, 1-6, 2-31
  cache, 87-1
  choosing, 2-3
  CMP, 1-5, 2-27
  EIS, 2-4, 53-5
  EJB entity beans, 1-5, 1-6, 2-27, 2-31
  EJB session bean facade, 1-5, 2-25
  locking, 2-11
  optimistic locking, 2-11
  Oracle TopLink, 1-1

pessimistic locking, 2-11
selecting, 2-5
session brokers, 72-26
sessions, 72-2
three-tier, 1-4, 2-22
two-tier, 1-6, 2-24
unit of work, 97-1
web services, 1-6, 2-35
arguments, binding in query, 26-6
array
dimensionality, 4-45, 4-49
object-relational mappings, 46-2
`AsOfClause`, 96-2
asynchronous change propagation, 88-2
`AttributeChangeTrackingPolicy`
about, 97-8
OC4J CMP integration, 97-8
OC4J EJB 3.0 CMP integration, 97-8
other application servers, 97-8
attributes
adding to descriptors, 4-44
array dimensionality, 4-45, 4-49
changes, tracking, 25-75
final, 4-45, 4-48
in TopLink Workbench, Navigator window, 4-9
lazy loading. *see* fetch groups
static, 4-45, 4-48
transforming, 32-29, 32-31, 33-15
transient, 4-45, 4-48
unmapping, 31-6
volatile, 4-45, 4-48
Attributes tab, 4-44, 4-48
Attunity Connect platform, 81-3
auditing
authentication, 81-6
unit of work, 99-21
authentication
about, 81-5
auditing, 81-6
proxy authentication, 81-5
simple JDBC authentication, 81-5
three-tier architecture, 81-5
two-tier architecture, 81-5
Automap, 31-2
automapping descriptors
about, 31-2
see also mappings
automatic table generation
about, 30-4
configuring, 8-12

B

Base64 encoded strings, 32-25
batch options
mappings, 34-7
writing, 11-15
batch reading
in query objects, 94-10
read optimization, and, 11-19
batch writing
about, 11-15, 11-26, 83-11
dynamic, 11-15
dynamic, `setMaxBatchWritingSize()`, 11-15
MySQL4 platform, 83-9
non-parameterized, 11-15
parameterized, 11-15
`setMaxBatchWritingSize()`, 11-15
BEA WebLogic
deploying to, 10-2
deployment exceptions, 15-3
modifying persistence descriptor, 8-14
setting classpath, 7-15
setting shared library, 7-15
transport layer, 72-31
using a security manager, 7-20
@Bean fetch=lazy, 30-9
beans
session beans, 2-26
stateful beans, 2-26
stateless beans, 2-26
bidirectional relationships
about, 33-2
generating, 4-31
in one-to-one mappings, 32-34
target keys, 33-5
with indirection, 32-35
`bindAllParameters()` method, 20-7
bindings
arguments, 26-6
input paramters, 94-20
JAXB, 17-10
LOB mappings, 31-3
see parameter binding
BLOB
mapping to, 31-2
BMP
and EJB 1.1, 2-32
and EJB 2.0, 2-32
and TopLink, 2-31
deployment files, 8-9
descriptors, 23-3, 25-46
packaging for deployment, 9-5
boolean logic in expressions, 95-3
branch classes, 23-13
buttons. *see* toolbars
bye array binding, LOB mappings, 31-3
Byte array Base64, 32-25

C

cache
about, 2-13, 2-18, 87-1
architecture, 87-1
configuring, 87-6, 96-20
coordination, 87-9, 88-1, 89-1, 90-1
descriptor level, 25-35
disabling during read query, 93-34
distributed, 87-9
expiration, 19-19, 25-42, 96-21

expression limitations, 93-31
identity maps, using, 72-29
in-memory queries, 93-30, 93-31
internal query object cache, 93-36, 93-37, 96-20, 96-21
invalidation, 19-19, 25-42, 87-7, 87-8
isolated client sessions, 72-24
isolation, 87-6, 87-9
object cache, 93-36
object cascading refresh, 93-35
object refresh, 93-35
optimizing, 11-13
project level, 19-13
queries, 87-6, 93-29
query cache, 93-36
`readObject` method, and, 94-2
refreshing, 25-27, 87-8, 93-35
restrictions, 93-37
service channel, 88-3
sessions, 72-3, 72-33, 87-2
stale data, 87-6
storing query results, 93-36, 96-20
type and size, 19-13, 25-35
unit of work cache, 87-2
cache coordination
about, 87-9
application server clustering, and, 7-4
avoiding stale data, 87-7
CMP projects, 8-11
EJB Entity Beans with BMP architecture, and, 2-33
explicit query refreshes, 87-7
JMS, 89-1
`orion-ejb-jar.xml`, 8-11
packet time-to-live, 88-15
permissions, 7-23
RMI, 90-1
cache invalidation, avoiding stale data, 87-7
cache synchronization. *see* cache coordination
`cacheAllStatements()`, 20-7
`cacheQueryResults()`, 25-25
cache-synchronization property, 8-11
Caching tab, 25-36, 25-38, 25-40, 25-44
calendar, 34-3
call
call queries, 93-17
EIS, 94-24
EJBQLCall, 94-24
SQLCall, 94-19
StoredFunctionCall, 94-23
StoredProcedureCall, 94-21
`Call` object, queries, 93-3
cascading
object refresh, 93-35
optimistic version locking, 23-19
write queries, compared to non-cascading, 93-15, 94-14
catalog, database, 4-22
`catchExcpetions()`, 74-19
change policy
about, 25-73
attribute change tracking, configuring, 25-75
deferred change detection, configuring, 25-73
empty transaction, 97-6
object change tracking, configuring, 25-73
unit of work, 97-6
change tracking
attribute, configuring, 25-75
deferred, configuring, 25-73
object, configuring, 25-73
changed items, displaying in TopLink Workbench Navigator window, 4-9
`ChangedFieldsLockingPolicy`, 23-21
changing package names, 4-52
`checkDatabase()`, 74-18
checking in/out projects, 6-3
`checkInstantiationPolicy()`, 74-19
Choose a Schema Context dialog box, 28-3, 29-2
Choose Query Key dialog box, 95-14
Choose Relationships to Generate dialog box, 4-30
Choose Root Element dialog box, 28-4, 29-5
class extraction method
about, 23-15
inheritance, 23-15
class indicator
about, 23-14
class extraction method, 23-15
class indicator field, 23-14, 38-1
class loader
host application, 2-5
loading session, 75-4
class modifiers, 4-42
Class tab, 4-41, 4-42, 4-43
classes
access modifiers, 4-42
adding and refreshing, 4-50
branch, 23-13
creating, 4-40, 4-52
CursoredStream, optimizing, 96-18
Database Exception, 94-25
`DatabaseMapping`, 30-25
default null values, 32-36
DeleteObjectQuery, 94-13
ExpressionBuilder, 95-16
generating from database, 4-30
InsertObjectQuery, 94-13
`InsertObjectQuery`, 97-12
interfaces, 4-43
leaf, 23-13
merging files, 6-5
methods, adding, 4-47
naming, 4-41
non-descriptor classes, 4-51
object model, 2-16
Performance Profiler, 11-3
persistent requirements, 2-12
preferences, 4-16
refreshing, 4-50
removing, 4-51
root, 23-13

troubleshooting, 14-25
unit of work, 97-12
UpdateObjectQuery, 94-13, 94-14
ValueHolderInterface, 2-12, 30-7, 33-9
VariableOneToOneMapping, 33-6
see also specific class name
classpath
adding, 19-3
BEA WebLogic, 7-15
configuring, 4-2, 7-15, 7-21
connector.jar, 4-2
custom Collection class, 4-3
DRIVER_CLASSPATH, Oracle TopLink Workbench, 4-2
IBM WebSphere, 7-21
J2C adapter, 4-2
JDBC driver, 4-2, 4-23
JDBC_CLASSPATH, 4-2
Oracle TopLink Workbench DRIVER_CLASSPATH, 4-2
relative, 19-3
setting for BEA WebLogic, 7-15
troubleshooting, 14-25
xdb.jar file, 4-3
client sessions
about, 72-1, 72-14, 72-26
acquiring, 75-6
configuration, 79-1
example, 72-16
shared resources, 72-15
client-controlled transactions, 99-25
client-server architecture. *See* two-tier
CLOB
mapping to, 31-2
clones
copying methods, 25-72
merging changes, 99-13
post-commit, avoiding, 99-31
unit of work, 97-2, 97-9
Cloudscape platform, 81-3
clustering, integrating TopLink with, 7-4
CMP
and EJB 1.1, 2-27
and EJB 2.x, 2-27
and EJB 3.0, 2-27
and TopLink, 2-27
CMPPolicy, 23-3, 25-46
deploying, 8-9, 10-2
descriptors, inheritance, 23-4
external transactions, 97-3
isolated client sessions, 72-25
local transaction, 99-25
non-deferred write, 99-26
OC4J persistence, 7-5
packaging for deployment, 9-4
setter parameter type checking, 7-25
single-object finder return type checking, 7-26
transaction attribute, 99-25
unit of work, 97-3, 99-24
unknown primary key class support, 7-25
code generation, optimizing, 11-8
collapsing items in Navigator window, 4-9
collection class
specifying, 94-9
specifying in query objects, 94-9
collections
persistent requirements for mappings, 2-12
query results, 93-8
comments
descriptors, 25-8, 25-9
mappings, 32-18
projects, 19-21
commit
and Java Transaction API, 97-10
failure, resuming after, 99-14
resuming unit of work after, 99-14
Communication Exceptions, 13-65
composite collection EIS mappings
configuring, 58-1
example, 53-8
understanding, 53-7
composite collection XML mappings
configuration, 67-1
configuring, 67-1
understanding, 62-25
composite descriptors
about, 23-5
composite object mapping, 23-8
EIS projects, 23-8
XML projects, 23-8
composite direct collection EIS mappings
configuring, 56-1
understanding, 53-6
composite direct collection XML mappings
configuring, 65-1
understanding, 62-14
composite EIS descriptors, 24-5
composite object EIS mappings
composite descriptors, and, 23-8
configuring, 57-1
understanding, 53-7
composite object mappings
composite descriptors, and, 23-8
composite object XML mappings
configuration, 66-1
configuring, 66-1
understanding, 62-21
composite primary key, 33-9
concrete class. *see* container policy
concurrency
about, 2-18
exceptions, 13-25
server session, 72-18
Concurrency Exceptions, 13-25
configurations
about, 73-2
creating, 73-2
development environment, 6-1
new, 73-2
Oracle JDeveloper, 6-1

session, 73-2, 73-4, 73-6, 73-9
conforming
about, 99-8
alternatives, UnitOfWork method writeChanges, 99-12
alternatives, UnitOfWork properties, 99-13
descriptors, and, 99-12
queries, alternatives to, 99-12
`ConnectBy`, 96-7
connection policy
configuring, 74-19
exclusive connections, 74-19
lazy connection acquisition, 74-20
connection pool
about, 81-7
connection count, 86-1
ConnectionPolicy, 72-20
external, 81-8
internal, 81-7, 81-8, 81-9
lazy connection allocation, 72-19, 75-9
named, 81-9
parameter binding, 11-16
prepared statement caching, 11-16
read, 81-8
sequence, 17-5, 81-8
server session, 72-18
sessions, and, 72-3
size, 86-1
write, 81-8
Connection Specifications tab, 21-3
Connection tab, 21-3
connections
about, 81-6
connection pool, 81-7
exclusive write connection, 74-19
lazy acquisition, 74-20
reading through the write connection, 99-28
`connector.jar`, 4-2, 17-7
container configuration file, 8-5
container policy
about, 32-26
custom Collection class, 4-3
sorting, in memory, 32-26
container-controlled transactions, 99-25
context
JAXB path, 17-14
menus, 4-5
schema, 28-2, 29-2
`Context.SECURITY_CREDENTIALS`, 88-9
`Context.SECURITY_PRINCIPAL`, 88-9
Conversion Exception, 13-26
Converter tab
object type mappings, 32-22
converters
custom, 33-3
object type, 32-22
coordinated announcement delay, 88-11
coordinated cache, 8-11
configuring, 88-1
naming service, 88-7
service channel, 88-3
copy policy
about, 25-71
method, 25-72
setting, 25-72
copying project objects, 6-6
Copying tab, 25-72
CORBA
Oracle TopLink transport layer support, 72-31
Transaction Service *see* OTS
Create New Project button, 18-2
Create New Project dialog box, 18-2
Create New Session dialog, 73-4, 73-6, 73-8
Create Project from JAXB dialog, 18-6
Create Project from OC4J dialog, 7-9
creating
configurations, 73-2
expressions, 95-13
sessions, 73-4, 73-6, 73-8, 85-1
Crimson XML parser, 7-3
cursored streams
example, 96-18
optimizing, 96-18
remote sessions, 72-32
usage example, 72-32
cursors
as query results, 93-8
traversing scrollable, 96-16
Custom Calls tab, 28-6
custom SQL
Custom SQL tab, 26-7
unit of work, 99-16
customization
about, 12-1
data types, 12-1
EIS, 12-1
mapping extensions, 12-1
overview, 2-15
XML, 12-1

D

data access
about, 81-1
authentication, 81-5
connection pool, 81-7
connections, 81-6
optimizing, 11-14, 11-15
platforms, 81-3
data level queries
example, 95-13
in expressions, 95-12
data source platform
about, 81-3
Attunity Connect database, 81-3
Cloudscape database, 81-3
databases, 81-3
DB2 database, 81-3
EIS, 81-4
HSQL database, 81-3

Informix database, 81-3
J2C adapter, 81-4
JDBC drivers, 81-3
Microsoft Access database, 81-3
MySQL4 database, 81-3
Oracle database, 81-3
Oracle8 database, 81-3
Oracle9 database, 81-3
PointBase database, 81-3
SQLAnyWhere database, 81-3
SQLServer database, 81-3
SybasePlatform database, 81-3
data sources
configuring, 82-1
nontransactional, 81-1
transactional, 81-1
troubleshooting, 14-25
Database Exceptions, 13-27, 94-25
database fields, configuring, 34-2
database functions, in expressions, 95-3
database login
parameter binding, 83-9
prepared statement caching, 83-9
Database Preferences, 4-17
database queries
about, 93-10, 94-4
fetch groups, 93-13
join reading, 93-12
object level modify query, 93-14, 93-15
object level read query, 93-11, 93-13
partial object query, 93-11
read all query, 93-11
read object query, 93-11
report query, 93-15
database schema
tables, 4-22
database sessions
about, 72-2, 72-29
cache, 72-33
configuration, 80-1
creating, 73-8
database tables
about, 4-21
adding to database, 4-22
creating, 4-22
descriptors and classes, generating, 4-30
EJB entity generation, 4-31
fields, 4-25
generating, 4-29, 4-30, 4-32
importing, 4-22
Java source generation, 18-15
JDBC driver classpath, 4-23
properties, 4-25
references, 4-26, 4-28
removing, 4-24
renaming, 4-24
schema, 4-22
SQL generation, 4-29
TopLink Workbench, Navigator window, 4-9
`DatabaseException` class, 94-25
`DatabaseLogin`, 81-2
`DatabaseMapping` class, 30-25
`DatabaseQuery`, 93-3
`DatabaseRow`, 33-15
databases
catalog, 4-22
common problems, 14-25
connect to, 4-21
creating reference tables on, 4-27
custom drivers, 4-18
disconnect from, 4-21
drivers, 4-18
exceptions, 94-25
fields, configuring, 34-2
for project, 18-3
Java type conversion, 5-3
linking, 72-28
log out of, 4-22
logging into, 4-22, 20-7
logins, 83-1
mapping. *See* mappings
platform, 18-3, 20-2, 21-2, 83-1, 84-1
preferences, 4-17
schema, 4-22
schema manager, 5-1
tables, 4-21
TopLink Workbench, Navigator window, 4-9
troubleshooting, 14-25
type conversion, schema manager, 5-3
using with Oracle TopLink Workbench, 4-21
DatabaseSession class
logging SQL and messages, 72-10
DB2
platform, 81-3
schema manager type conversion, 5-3
DBase platform, 81-3
default mapping
about, 30-4
automatic table generation, and, 30-4
configuring, 8-12
default table generator, 5-6
Default Mapping Exception, 13-69
default table generator
default mapping, 5-6
table creator, creating, 5-4
defaults
login level null values, 82-5
mapping level null values, 32-12
null values, 32-12, 32-36, 82-5
optimization, 11-8
root, 29-5
see also preferences
DefaultSequence, 83-6
deferred change detection
configuring, 25-73
`DeferredChangeDetectionPolicy`, 97-7
Delete All Interaction tab, 60-3
`deleteObject()`, 26-8
DeleteObjectQuery, 94-13
deletes

controlling order, 99-16
delete operation, 94-4
queries, EIS mappings, 60-3
demarcation of unit of work transactions, 97-2
dependent objects
non-deferred write, 23-4
deploy tool
about, 10-5
troubleshooting, 10-6
using with WebSphere Studio Application Developer, 10-5
deploying
about, 2-15, 10-1
application server requirements, 7-1
BEA WebLogic, 10-2
CMP applications, 10-2, 10-4
database login, 20-7
entity beans overview, 2-15
generating XML for, 8-3
hot deployment, 10-4
IBM WebSphere, 10-3
Java applications, 10-1
JSP and Servlet applications, 10-1
modifying BEA WebLogic persistence descriptor, 8-14
non-CMP applications, 10-4
packaging, 9-1
Session Bean applications, 10-1
troubleshooting, 15-1
deployment descriptors, 23-3
deployment exceptions
BEA WebLogic deployment, 15-3
IBM WebSphere deployment, 15-12
deployment files
BMP applications, 8-9
CMP applications, 8-9
creating, 8-1
descriptors, 23-3
EJB 3.0, 8-2, 9-1
JARs, troubleshooting, 15-1
Java applications, 8-8
JSP and Servlet applications, 8-8
Session Bean applications, 8-9
XML, generating, 8-3
deployment XML, exporting, 18-14
`DeploymentXMLGenerator`, 8-3
Descriptor Event Listener, 25-62
Descriptor Event Manager
about, 23-8
Descriptor Event Listener, 25-62
domain object methods, 25-59
event types, 25-59
handlers, 23-8, 25-59, 25-62
handlers, Descriptor Event Listener, 25-62
understanding, 23-8
descriptor events
about, 23-8
Descriptor Event Listener, 25-62
domain object methods, 25-59
handlers, 23-8, 25-59, 25-62
types of, 25-59
understanding, 23-8
Descriptor Exceptions, 13-2, 13-29
Descriptor Info tab, 25-3, 25-6, 25-7, 26-2, 26-4, 28-2, 28-4, 29-2, 29-4, 29-5, 29-6
DescriptorEventListener, 25-62
descriptors
about, 16-2, 23-1
advanced properties, default, 19-7
aggregate, 23-5, 24-2
aggregate, EJB 3.0 and, 23-6
aggregate, relational projects and, 23-5
amending, 2-21, 23-5, 25-81
API, 23-23
architecture, 23-2
attributes, adding, 4-44
automapping, 31-2
automatically mapping, 31-2
BMP, 23-3, 25-46
cache refreshing, 25-27
change policy, 25-73
child inheritance, 25-51
class, 24-2
CMP, 23-3, 25-46
CMPPolicy, 23-3, 25-46
comments, 25-8, 25-9
composite, 23-5, 23-8, 24-5
composite EIS, 24-5
configuring, 25-1
conforming, 99-8
creating, 24-1, 24-2, 24-5
custom EIS interactions for basic persistence, 28-6
custom SQL queries for basic persistence, 26-6
deactivating, 4-10
default mappings, 30-4
default root, 28-4
deployment information, 23-3
Descriptor Event Listener, 25-62
domain object methods, 25-59
EIS, 23-12, 24-5, 28-1
EIS projects, 23-8
EJB, 23-3, 25-46
EJB information, 23-3, 25-46
errors, 4-10, 4-11, 14-3
event handlers, 23-8, 25-59, 25-62
events, 23-8, 25-59
existence checking, 11-13, 25-43
fetch groups, 25-79
files, merging, 6-5
generating from database, 4-30
hierarchy, inheritance, 23-23
history policy, 25-76
identity maps, 19-14, 25-35, 25-38, 25-40
in Java, 23-23
inactive, 4-10
inheritance, 23-3, 23-12, 25-51, 25-52
instantiation, 11-13
interface, 24-2, 25-31, 25-33
mapping, 25-3, 26-2, 30-4, 31-1, 31-2
merging, 6-5

named queries, 25-10
nondescriptor classes, 4-51
object-relational, 23-11, 27-1
optimizing, 11-13
parent inheritance, 25-52
projects, 16-2, 23-1
query key interfaces, 25-31
query timeout, 25-24, 25-26
read only, 25-5
registering with sessions, 72-13, 74-2, 74-3, 74-10
relational, 23-11, 24-2, 26-1
removing, 4-51
returning policy, 25-67
root EIS, 24-5
root element, 29-5
schema context, 28-2, 29-2, 29-3
sequencing, 23-9, 26-3
TopLink Workbench, Navigator window, 4-9
types of, 23-1
validating, 24-6
XML, 23-12, 29-1
XML projects, 23-8
detachment indirection, 30-10
developing applications with Oracle TopLink, 2-1
development environments
about, 3-2
configuring, 6-1
database logins, 20-7
development process
about, 2-1
additional support, 2-3
stages of, 2-2
with Oracle TopLink, 2-1
development tools
about, 3-1
profiler, 11-2
schema manager, 5-1
dimensionality, array, 4-45, 4-49
direct access
about, 19-4, 32-14
specifying, 32-32
direct collection relational mappings
configuring, 42-1
example, 33-11
understanding, 33-11
direct collections
session broker limitations, 72-28
direct EIS mappings
configuring, 55-1
understanding, 53-5
direct field
in direct collection mappings, 42-2
direct key fields, 44-2
direct map relational mappings
configuring, 44-1
direct keys, 44-2
direct value, 44-1
understanding, 33-12
direct mappings
generating deprecated, 19-12
with EJB, 33-4
direct value fields, 44-1
direct XML mappings
configuring, 64-1
understanding, 62-5
directionality in mappings, 33-2
direct-to-field mappings
ObjectTypeMapping deprecated, 33-3
SerializedObjectMapping deprecated, 33-3
type conversions, 34-3
TypeConversionMapping deprecated, 33-3
direct-to-field relational mappings
configuring, 35-1
options, 35-1
timestamp support, 34-3
understanding, 33-4
direct-to-XMLType relational mappings
configuring, 36-1
understanding, 33-4
Discovery Exception, 13-70
DMS profiler
about, 11-4, 72-12
accessing with JMX, 11-7
and JMX
nouns, 11-4, 72-12
selecting, 74-11, 74-14, 74-17, 74-20
sensors, 11-4, 72-12
document information in XML schemas, 4-34, 4-36, 4-38
documentation
hosted, 4-14
See also Help
does exist write object, 11-27
`dontOptimizeDataConversion()`, 11-15
`doPrivileged()`, 7-4
`DRIVER_CLASSPATH`
Oracle TopLink Workbench environment, 4-2
drivers, custom database, 4-18
dynamic batch writing
about, 11-15
`setMaxBatchWritingSize()`, 11-15
dynamic fetch groups, querying with, 96-4

E

Editor window, about, 4-4, 4-10
EIS
about, 17-8
architecture, 2-4
call, 94-24
custom interactions for basic persistence, per descriptor, 28-6
indexed records, configuring, 28-5
interactions, 28-6, 93-5, 94-24
mapped records, configuring, 28-5
mappings, 17-8, 53-2
projects, 18-3
queries, 93-5
record format, configuring, 28-5
XML records, configuring, 28-5

EIS descriptors
composite, 24-5
configuring, 28-1
default root, 28-4
locking policy, 25-64
root descriptor, 24-5
schema context, 28-2
`setDataTypeName`, 28-5
understanding, 23-12
EIS mappings
about, 53-1, 53-2
architecture, 53-5
composite collection, 53-7, 58-1
composite direct collection, 53-6, 56-1
composite object, 53-7, 57-1
configuring, 54-1
direct, 53-5, 55-1
jaxb:class support, 53-3
list support, 53-3
one-to-many, 53-12, 60-1
one-to-many, key on source, 53-13
one-to-many, key on target, 53-15
one-to-one, 53-8, 59-1
one-to-one, key on source, 53-9
one-to-one, key on target, 53-10
transformation, 53-17, 61-1
types of, 53-1
union support, 53-3
xsd:list, 53-3
xsd:union, 53-3
EIS projects
configuring, 21-1
`connector.jar`, 17-7
indexed records, 17-9
mapped records, 17-9
sequencing, 17-5
understanding, 17-7
XML records, 17-9
EIS queries, 93-5
EIS record types, supported, 53-2
`EISLogin`, 81-2
EJB
descriptors, 18-10
isolated client sessions, 72-25
setter parameter type checking, 7-25
single-object finder return type checking, 7-25
unknown primary key class support, 7-25
EJB 1.1
and BMP, 2-32
and CMP, 2-27
EJB 1.x
indirection, 30-9
serialization, 30-9
EJB 2.x
and BMP, 2-32
and CMP, 2-27
default mapping, 30-4
indirection, 30-9
serialization, 30-9
EJB 3.0
`<J2EE-Container>-jar.xml` file, 8-6
and CMP, 2-27
attribute change tracking policy, OC4J CMP integration, 7-5, 97-8
Bean annotation, fetch=lazy, 30-9
default mapping, 30-4
deployment files, 8-2, 9-1
deployment files, OC4J CMP integration, 7-5
detachment, 30-10
Embedded annotation, 23-6
packaging for deployment, 9-1
packaging, OC4J CMP integration, 7-5
`projects.xml` file, 8-3, 8-5
serialization, 30-10
`sessions.xml` file, 8-5
`toplink-ejb.xml` file, 8-7
value holder indirection, 30-9
EJB descriptors, opening projects with, 18-10
EJB entities
CMP hot deployment, 10-4
deployment overview, 2-15
EJB 2.x indirection, 30-9
EJB 3.0 indirection, 30-9
generating, 4-31
hot deployment, 10-4
inheritance, 23-4, 23-17
inserting after `ejbCreate`, 23-4
inserting after `ejbPostCreate`, 23-4
mapping, 19-6
non-CMP hot deployment, 10-4
non-deferred write, understanding, 23-3, 99-26
sequencing, 17-21
EJB entity beans
and EJB 1.1, 2-27, 2-32
and EJB 2.x, 2-27, 2-32
and EJB 3.0, 2-27
with BMP architecture, 1-6, 2-31
with CMP architecture, 1-5, 2-27
EJB finders
about, 93-24
Call finders, 93-26
creating, 96-8
DatabaseQuery finders, about, 93-26
default finders, about, 93-26
default finders, creating, 96-8
EJB QL finders, about, 93-27
`ejb-jar.xml` options, 96-9
ejbSelect method, 93-28
ejbSelect, creating, 96-15
ejbSelect, using, 96-15
expression finders, about, 93-27
named query finders, about, 93-27
predefined, about, 93-24
primary key finders, about, 93-27
redirect finders, about, 93-28
redirect finders, using, 96-12
single-object finder return type checking, 7-25
SQL finders, about, 93-28
EJB Info tab, 25-46
EJB JAR XML Exception, 13-77

EJB Preferences, 4-16
EJB QL
  exceptions, 13-63
  queries, 26-6, 93-5
  query language, 93-5
EJB session bean facade architecture
  about, 1-5, 2-25
  understanding, 2-25
EJB Session Beans, 72-31
ejbc
  about, 10-2
  troubleshooting, 10-3
ejbCreate, 23-4
ejb-jar.xml file
  about, 8-5, 18-15
  configuring, 8-5
  corresponding to Oracle TopLink Workbench functions, 18-15
  EJB finder options, 96-9
  location, 19-7
  managing, 6-6
  preferences, 4-16
  synchronization under EJB 2.0, 8-5
  updating from, 18-16
  writing, 18-16
ejbPostCreate, 23-4
@Embeddable, 23-6
@Embedded, 23-6
empty unit of work transactions, 97-6
encrypting login passwords, 18-13
enhanced validation exceptions, 8-4
Enterprise Information Systems. *see* EIS
entity beans
  deployment, 2-15
  descriptor information, 23-3
  direct mappings, 33-4
  indirection, EJB 2.x, 30-9
  indirection, EJB 3.0, 30-9
  sequencing with, 17-21
Entity Manager Setup Exception, 13-80, 13-81
EntityManagerSetupException, 13-80, 13-81, 13-83
environment
  configuring, 4-2
  JAVA_HOME, 4-2
  JDBC_CLASSPATH, 4-2
  proxy, 4-13, 4-14, 18-16
error codes
  1-176, 13-2
  1-99, 14-1
  100-199, 14-3
  200-399, 14-3
  400-599, 14-13
  500-699, 14-17
  700-799, 14-23
  800-899, 14-23
  12000-12004, 13-65
  18001-18002, 13-69
  22001-22004, 13-70
  22101-22105, 13-71
  3001-3007, 13-26
  4002-4018, 13-27
  5001-5008, 13-29
  6001-6098, 13-31
  7001-7104, 13-43
  72000-72023, 13-77, 13-80, 13-81, 13-83
  8001-8010, 13-63
  9000-9009, 13-64
errors
  about, 13-1
  codes and descriptions, 13-1, 14-1
  descriptors, 4-10, 4-11
  migration, 7-13
  Oracle TopLink Workbench, 14-1
Event Manager, 72-5
events
  about, 23-8
  client session, 72-6
  database access, 72-6
  Descriptor Event Listener, 25-62
  domain object methods, 25-59
  handlers, 23-8, 25-59, 25-62
  listeners, sessions, 72-7
  session, 72-5
  session manager, 72-6
  sever session, 72-6
  types of, 25-59
  unit of work, 72-6
examples
  composite collection EIS mapping, 53-8
  context menu, 4-5
  cursored streams, 96-17
  direct collection mappings, 33-11
  direct-to-field mappings, 33-4
  exception handler, 74-13
  indirection, 30-6
  inheritance, 23-12
  Oracle TopLink Workbench, 4-3
  performance optimization, 11-23, 11-25
  proxy indirection in code, 32-9
  READALL finders, 96-10
  report query, 94-6
  scrollable cursors, 96-17
  serialized mapping, 30-10
  stored procedure call, 94-21, 94-22
  transformation mapping, 33-15
  transformation XML mapping, 53-18, 62-31
  Unit of Work, 97-6, 98-7
  write, write all, 94-3
exception handler
  about, 72-12
  example, 74-13
  selecting, 74-12
exceptions
  chained, 72-11
  communication exceptions, 13-65
  conversion exceptions, 13-26
  database exceptions, 13-27, 94-25
  Default mapping exception, 13-69
  descriptor exceptions, 13-2
  discovery exceptions, 13-70

EJB JAR XML exceptions, 13-77, 13-80, 13-81, 13-83
EJB QL exceptions, 13-63
enhanced validation, 8-4
`java.security.AccessControlException`, 15-12, `15-13`
JMS processing exceptions, 13-69
Migration utility exception, 13-74
optimistic locking, 13-29
query exceptions, 13-31
remote command manager exceptions, 13-71
selecting exception handler, 74-12
session loader exceptions, 13-64
Transaction exception, 13-73
validation exceptions, 13-43
XML conversion exception, 13-74
exclusive connections
about, 99-29
internal read connection pool, 86-6
isolated sessions, 72-20, 74-19
named queries, 25-24
existence checking, 99-5
descriptors, 25-43
projects, 19-8
expanding items in Navigator window, 4-9
expiration of objects in the cache, 19-19, 25-42
explicit query refreshes, cache coordination, 87-7
exporting
deployment XML, 18-14
Java model, 18-14
Java source, 18-14
preferences, 4-12
projects, 18-13
Expression Builder, 95-13, 95-15
Expression Builder dialog box, 95-13
`Expression` class, 95-1
`ExpressionMath` class, 95-1
expressions
about, 93-3, 95-1
allows none, 25-18, 95-15
allows null, 25-18, 95-15
building, 95-13
comparing with SQL, 95-1
components, 95-2
creating, 95-13
data level queries, 95-12
database functions, 95-3
in relationships, 95-6
in-memory queries, limitations, 93-31
mathematical functions, 95-4
multiple, 95-10
one-to-one mappings, 95-5
outer joins, 94-11
parallel expressions, 95-11
parameterized, 95-8
platform functions, 95-16
query keys, 95-10
subqueries and subselects, 95-10
user-defined functions, 95-5, 95-16
using Boolean logic, 95-3
XML Type functions, 95-5
see also queries
external
applications, 99-27
connection pools, 81-8
controller, transaction, 97-1
JDBC pools, 2-33
transactions, 97-1
external transaction controller
configuration, sessions, 74-14
session, 97-2

F

factory name, JMS connection, 89-2
failure, resuming unit of work after commit, 99-14
features of Oracle TopLink, 1-4
fetch groups
about, 23-5, 25-79
configuring, 25-79, 96-3
default, 25-79, 96-3
disabling, 96-3
dynamic, 96-4
object level read query, 93-13
read optimization, and, 25-79
size, 11-18
static, 96-3
field references, 34-8
Field uses XML Schema "type" attribute option, 30-12
fields in database tables, 4-25
Fields tab, 43-2
field-to-object attribute transformation, 32-29, 32-31
files
JAXB-specific, 17-10
TopLink-specific, 17-11
see also specific file name
final attributes, 4-45, 4-48
`findAll`, using, 93-26
finders
caching options, 93-37
disabling cache, 93-38
managing large result sets, 96-18
refreshing results, 93-38
see also EJB finders
flashback queries
about, 93-18
historical client sessions, 78-1
`forceUpdateToVersionField()`, 99-18
foreign keys
about, 2-17, 34-8
configuring in EIS mappings, 59-1
EIS mappings, 60-1
multiple tables, 26-14
one-to-many mappings, 33-7
one-to-one mappings, 32-34, 33-5
parameterized expressions, 95-8
references, 14-15, 14-23
target, 33-5, 34-9
troubleshooting, 14-15, 14-23

full identity map, 87-3

G

garbage collection, managing, 98-8
General Preferences dialog, 4-13
Generate Classes and Descriptors dialog, 4-30
Generate EJB Entity Classes and Descriptors dialog, 4-31
generating
  access method, 4-30
  deployment JARs, troubleshooting, 15-1
  deprecated direct mappings, 19-12
  see also exporting
`getCatalogs()`, 4-22
`getField()`, 95-12
`getImportedKeys()`, 4-22
`getParameter`, 95-8
`getPrimaryKeys()`, 4-22
`getTable()`, 95-12
`getTables()`, 4-22
`getTableTypes()`, 4-22
`getValue()`, 30-7
`getValue()` method, 30-7

H

hard cache weak identity map
  about, 87-4
  when to use, 87-5
help
  about, 4-12
  displaying, 4-12
Help button., 4-12
Help Preferences, 4-14, 4-16
hierarchical queries
  about, 93-18
  described, 96-6
hints, Oracle Hints in queries, 93-18
historical client sessions
  about, 72-1, 72-25
  cache, 72-33
  limitations of, 72-25
historical queries, 96-2
  about, 93-21
  see also AsOfClause
history policy, configuring, 25-76
holders, value, 30-7
host URL, JMS topic, 89-3
hosted
  documentation, 4-14
  XSD files, 8-2, 8-4
hot deployment
  about, 10-4
  CMP applications, 10-4
  non-CMP applications, 10-4
HSQL platform, 81-3

I

IBM Informix Database native sequencing, 17-19
IBM WebSphere
  deploy tool, 10-5
  deploying to, 10-3
  deployment exceptions, 15-12
  setting classpath, 7-21
identity
  about, 2-17, 87-3
  cache, and, 87-3
  using cache to preserve, 87-3
  see also identity map
identity map cache
  disabling during a write query, 94-15
  refresh in read query, 93-35
identity maps
  about, 25-35, 25-38, 25-40, 72-29
  cascading refresh during read query, 93-36
  descriptors, 19-14, 25-35
  example, 93-36
  full, 87-3
  guidelines for choosing type, 87-4
  hard cache weak identity map, 87-4, 87-5
  isolated client sessions, 72-24
  no identity map, 87-4
  refreshing during read query, 93-35
  soft cache weak identity map, 87-4, 87-5
  soft cache weak identity map and read optimization, 11-19
  specifying, 25-35, 25-38, 25-40
  weak, 87-3
  weak identity map and read optimization, 11-19
Identity tab. *see* Caching tab
impedance mismatch, solving, 1-2
Implementors tab, 25-34
Import Tables from Database dialog, 4-23
importing
  classes, 4-16
  preferences, 4-12
inactive descriptors
  about, 4-10
  mapping to, 34-6, 54-3
independent relationships, 32-16
indexed records, 53-2
indirection
  about, 2-19, 30-5
  bidirectional relationships, 32-35
  choosing the correct type, 32-4
  configuring, 32-3
  EJB, 30-9
  EJB 2.x CMP, 30-9
  EJB 3.0 CMP, 30-9
  example, 30-6
  many-to-many mappings, 33-9
  nontransparent, 2-12
  one-to-many mappings, 32-34
  proxy indirection, 30-8
  remote sessions, 72-32
  serialization, 30-9
  transparent, 2-12, 30-8
  value holder, 30-7
  `ValueHolderInterface`, 2-12

see also proxy indirection, transparent indirection
Informix platform, 81-3
inheritance
about, 2-18, 23-3, 23-12
aggregate classes, 23-17
aggregate collection mappings, 33-10
branch classes, 23-13
child descriptors, 25-51
class extraction, 23-15
class indicator, 23-14, 23-15
descriptors, 23-3, 23-12, 25-51, 25-52
finding subclasses, 23-14
instantiating subclasses, 23-14
isolated client sessions, 72-24
leaf classes, 23-13, 96-4
primary keys, 23-16
queries, 93-22
querying on hierarchy, 96-4
relational parent, 25-52
root class, 23-13, 25-51, 25-53
root class subclasses, finding in inheritance, 23-14
supporting with multiple tables, 23-16
supporting with one table, 23-16
transformed to relational model, 11-11
using with EJB, 23-4, 23-17
inheritance hierarchies
descriptors, 23-23
querying on, 96-4
Inheritance tab, 25-51, 25-53
inherited subclasses, mapping, 25-58
in-memory query
about, 87-6
check cache using exact primary key, 93-30
check cache using primary key, 93-30
check database if not in cache, 93-30
conform results in unit of work, 93-31
expression limitations, 93-31
supported, 93-31
using, 93-30
inner join, 95-6
insert operation, 94-3, 94-4
`insertObject()`, 26-7
instantiation policy
about, 25-70
setting, 25-70
Instantiation tab, 25-70
integrity checker, 24-6
about, 72-12
configuring, 74-18
interactions
about, 94-24
creating, 28-6
interface alias
about, 26-10
creating, 26-11
Interface Alias tab, 26-11
interfaces
classes, implementing, 4-43
customizing, 4-13
descriptors, 24-2, 25-31, 25-33
queries, 93-22
query keys, 25-31
querying on, 96-4
internal connection pool
about, 81-7
named, 81-9
read, 81-8
sequence, 81-8
write, 81-8
internal query object cache
about, 93-36
configuring, 96-20, 96-21
expiration, 96-21
restrictions, 93-37
internal transactions, 97-1
invalidation of objects in the cache, 19-19, 25-42, 87-8
IP address for multicast group, 88-4
isolated client sessions
about, 72-1, 72-19, 77-1
configuration, 77-1
life cycle, 72-22
limitations of, 72-24
session event handlers, 72-21
with Oracle Virtual Private Database (VPD), 72-21
isolated session
cache, 72-33
ConnectionPolicy, 72-20
exclusive connections, 72-20
supported databases, 72-21
isolation
cache, 87-6
transaction levels, 97-2
unit of work transactions, 97-4
Iterator interface, 96-16

J

J2C adapters
about, 81-4
configuring for Oracle TopLink Workbench, 4-2
`EISLogin`, 81-2
selecting, 81-4
with EIS, 18-3
`<J2EE-Container>-ejb-jar.xml` file, 8-5
`<J2EE-Container>-jar.xml` file
EJB 3.0, 8-6
J2EE
parameter binding, 11-16
prepared statement caching, 11-16
web applications, 1-4
Java
database tables, 18-15
exporting to, 18-14
integration with any datasource, 1-2
iterators, 96-16
object model, 2-11
Java applications
deploying, 10-1
deployment files, 8-8

packaging for deployment, 9-1
Java Cryptography Extension, 18-13
Java Management Extensions. *see* JMX
Java Naming and Directory Interface. *See* JNDI
Java Object Builder, 72-4
Java streams
described, 96-17
optimizing, 96-18
support for, 96-17
Java Transaction API
and unit of work commit, 97-10
and unit of work rollback, 97-11
see also JTA
Java Transaction Service *see* JTA
`JAVA_HOME`, 4-2
`java.security.AccessControlException`, 15-12, `15-13`
`java.util.Collection` interface, 32-26
`java.util.Map` interface, 32-26
`java.util.Set` interface, 32-26
`javax.ejbEntityBean` interface, 4-31
JAXB
creating projects from, 18-6
files, 17-10
generating project from the command line, 18-8
jaxb:class, and EIS mappings, 53-3
jaxb:class, and XML mappings, 62-4
proxy configuration, 18-8
`tljaxb.cmd`, `18-8`
`tljaxb.sh`, `18-8`
typesafe enumeration converter, 32-25
understanding, 17-10
validation, 17-14
XML projects, 17-10
JAXB typesafe enumeration converter
configuring, 32-25
understanding, 30-24
`JAXBContext`, 17-14
JCE. *see* Java Cryptography Extension
JConnect, 11-14
JDBC
adaptor for EIS, 53-5
database gateway for EIS, 53-5
driver classpath, 4-23
JConnect, 11-14
Sybase JConnect, 11-14
JDBC drivers
about, 81-3
configuring for Oracle TopLink Workbench, 4-2
fetch size, 11-14, 11-18
general properties, 11-14
mapping LOBs, 31-3
selecting, 81-3
JDBC pools
external with EJB Entity Beans with BMP architecture, 2-33
`JDBC_CLASSPATH`, 4-2
JDeveloper. *See* Oracle JDeveloper
JMS
connection factory name, 89-2
coordinated cache, 89-1
Processing Exceptions, 13-66, 13-69
topic host URL, 89-3
topic name, 89-1
JMX
about
and DMS profiler
DMS profiler, 11-7
JNDI naming service, 88-7
joining
about, 93-12
expressions, and, 95-6
mappings, and, 37-1
one-to-many, about, 94-12
one-to-many, when not to use, 93-12
one-to-one mappings, 37-1, 94-12
optimizing reads, 11-19
queries, and, 94-11, 94-12
QueryManager expressions, 96-4
read queries, 11-19, 94-11
JSP and Servlet applications
deploying, 10-1
deployment files, 8-8
packaging for deployment, 9-2
JTA
about, 97-3
and unit of work, 97-3
isolated client sessions, 72-25
unit of work, 97-1
JTA/JTS
using with EJB Entity Beans with BMP architecture, 2-33
just-in-time reading. *see* indirection

K

Key Converter tab, 44-3
key pairs
database table reference, 4-28
troubleshooting, 14-15, 14-23
keys
about, 2-17
foreign, 2-17, 32-34, 33-5
foreign, target, 33-5
inheritance, 23-16
multiple tables, 26-13
primary, 23-16, 26-13, 32-2, 33-6, 33-9, 38-3
read-only settings, 32-2
reference key field, 33-11, 33-12
variable class relationships, 33-6, 38-3

L

language, specifying, 4-3
large result sets, managing in finders, 96-18
lazy attribute loading and read optimization, 25-79
lazy connection
acquisition, 74-20
allocation, 72-19, 75-9
lazy loading. *see* indirection

lazy reading. *see* indirection
leaf classes, 23-13, 96-4
life cycle of unit of work, 97-5
LOB
  mapping to, 31-2
local
  documentation, 4-14
  transactions, 99-25
locked files, 6-6
locking policy
  `AllFieldsLockingPolicy`, 23-20
  `ChangedFieldsLockingPolicy`, 23-21
  configuring, 25-64
  field locking, 23-20
  optimistic, 23-9, 23-19, 23-20
  optimistic version locking, 23-18
  `OptimisticLockException`, 23-19, 23-21
  pessimistic locking policy, 23-9, 23-22
  `SelectedFieldsLockingPolicy`, 23-21
  stale data, and, 87-6
  three-tier architectures, optimistic locking and, 23-22
  three-tier architectures, pessimistic locking and, 23-23
  `TimestampLockingPolicy`, 23-18
  understanding, 23-9, 23-18
  version locking, 23-19
  `VersionLockingPolicy`, 23-18
Locking tab, 25-65
log into database, 4-22
Log Out of Database, 4-22
log out of database, 4-22
logging
  application server, 72-9
  chained exceptions, 72-11
  java.util.logging, 72-9
  log level, 72-10
  Oracle Enterprise Manager 10*g*, *72-11*
  output, 72-10
  permissions, 7-24
  sessions, 72-7, 74-4, 74-5
  TopLink native logging, 72-8
  types, 72-8
Logging tab, 74-5
login
  CMP deployment, 17-3
  database, 20-5, 20-7, 83-1
  deployment, 17-3
  development, 17-4
  platforms, and, 17-4
  projects, and, 17-3, 71-1, 81-2
  role in project, 17-3, 81-2
  session, 17-3, 74-4
  session role, non-CMP, 17-3
`logMessages` method, *72-7*
look and feel, specifying, 4-13

M

Manage Non-Descriptor Classes dialog, 4-51
management, source control, 6-3
manager, session events, 72-5
many-to-many mappings
  relation table, 33-9
  relation tables, 40-1
  session broker limitations, 72-28
many-to-many relational mappings
  configuring, 40-1
  EJB, 33-9
  understanding, 33-8
mapped records, 53-3
mapping extensions
  custom data types, 12-1
  JAXB typesafe enumeration converter, 30-24, 32-25
  object type converter, 30-12, 32-22
  serialized object converter, 30-10, 32-18
  simple type translator, 30-12, 32-23
  transformation mappings, 30-14
  type conversion converter, 30-11, 32-20
mappings
  about, 2-16, 2-17, 16-1, 16-2, 30-1, 31-1
  access types, 32-15
  aggregate collection mappings and EJB, 33-11
  anyType mapping, 63-3
  as part of the application development process, 2-21
  automatic, 31-2
  batch options, 34-7
  class hierarchy, 30-25
  comments, 32-18
  configuring, 31-1, 32-1
  database field, 34-4
  default mapping, 30-4
  deprecated, generating, 19-12
  direct access, 11-13, 32-14
  directionality, 33-2
  EIS mappings, 17-8, 30-27, 53-1, 54-1
  EJB 2.0 entities, 19-6
  errors, 14-13
  example, 30-3
  extensions, about, 30-10
  hierarchy, 30-25
  inactive descriptors, 34-6, 54-3
  indirection, 11-13, 30-5, 32-3
  isolated client sessions, 72-24
  manually configuring, 31-1
  many-to-many, 33-9
  many-to-many, with EJB, 33-9
  method access, 32-14, 32-32
  null values, 32-12, 32-36
  object-relational, 46-1, 47-1
  one-to-many object, with EJB, 33-8
  one-to-one with EJB, 33-6
  optimizing, 11-13
  OX mappings, 30-27
  projects, and, 16-2
  read only, 32-2, 32-3
  relation tables, 40-1, 40-2
  relational, 33-1, 34-1

removing, 31-6
to tables, 25-3, 26-2
TopLink Workbench, Navigator window, 4-9
types of, 30-1
XML mappings, 62-1, 63-1
mathematical functions, in TopLink expressions, 95-4
menu bar, 4-5
menus
about, 4-4, 4-5
context menus, 4-5
menu bar, 4-5
merging
changes in clones, 99-13
Oracle TopLink Workbench project files, 6-4
project files, 6-4
messages, error, 13-1, 14-1
metadata
about, 2-13, 2-19
advantages, 2-20
creating, 2-20, 2-21
mapping and configuration, 16-1
project metadata, 2-20
session metadata, 2-21
Metalink, 2-3
method access
about, 19-4, 32-14
setting, 32-32
methods
adding, 4-47
`getValue()`, 30-7
`setValue()`, 30-7
wrapper policy, 25-79
see also specific method name
Microsoft Access
platform, 81-3
schema manager type conversion, 5-3
Microsoft SQL Database native sequencing, 17-19
migrating
error messages, 7-13
OC4J persistence to TopLink, 7-5
Oracle TopLink Workbench projects, 18-10
troubleshooting, 7-13
Migration Utility Exception, 13-74
model source, exporting, 18-14
modifiers, class, 4-42
multicast group address, coordinated cache, 88-4
multicast port, coordinated cache, 88-5
multiple sessions, 72-28, 75-2
multiple tables
about, 26-13
specifying for descriptors, 26-13
multiplicity in relationships, resolving, 7-14
multi-processing, 11-29
Multitable Info tab, 25-50, 26-13
mutable mappings, 32-33
`.mwp` file, 4-1, 18-2
MySQL4
batch writing, 83-9
platform, 81-3
primary key restrictions, 17-15
schema manager type conversion, 5-3

N

named connection pools, 81-9
named queries
about, 25-10, 74-21, 93-16
configuring, 25-10, 74-21
descriptor level, 25-10
exclusive connections, 25-24
options, advanced, 25-24
parameter binding, 25-22
prepared statement caching, 25-22
redirect query, 93-16
session level, 74-21
using, 94-18
when not to use, 93-17
when to use, 93-16
namespaces
about, 17-5, 17-22
configuring, 4-37
naming service
coordinated cache, 88-7
JNDI, 88-7
RMI, 88-9
native sequencing
IBM Informix Database, 17-19
Microsoft SQL Database, 17-19
Microsoft SQL Server, 5-6
non-Oracle database, 17-19
Oracle Database, 17-18, 17-20
Oracle Database SEQUENCE object, 17-19
Sybase Database, 5-6, 17-19
Navigator window
about, 4-4, 4-9
attribute and mapping, 4-9
database, 4-9
database tables in, 4-21
descriptor, 4-9
example, 4-9
package, 4-9
project, 4-9
unsaved or changed item, 4-9
NCHAR, 30-12
NCLOB, 30-12
neediness warnings. *See* troubleshooting
nested table object-relational mappings
configuring, 52-1
understanding, 46-3
nested unit of work, 97-9, 99-15
new projects, 18-2
New Reference dialog box, 4-27
New Session button, 73-4, 73-6, 73-8
New Sessions Configuration, 73-2
New Table dialog box, 4-22
`newInstance method`, 99-2
no identity map, 87-4
non-cascading write queries
compared to cascading, 93-15, 94-14

creating using `dontCascadeParts ()` method, 93-15, 94-14
non-deferred write
configuring, 99-12
dependent objects, 23-4
understanding, 23-3, 99-7, 99-26
nonintrusive persistence, 2-19
nonpersistent projects, 17-2
nonrelational projects, 17-2
nontransactional data sources, 81-1
nontransparent indirection, 2-12
nouns
DMS profiler, 11-4, 72-12
null values
default, 32-12, 32-36, 82-5
in expressions, 95-14
login level, 82-5
mapping level, 32-12
NVARCHAR2, 30-12

O

object array object-relational mappings
configuring, 51-1
understanding, 46-2
object cache, 93-36
object cache, sessions, 72-3
object change tracking
configuring, 25-73
object identity, 72-29
about, 2-17, 87-3
cache, and, 87-3
using cache to preserve, 87-3
see also identity map
object indirection
read optimization, as, 11-19
object level modify query
about, 93-14, 93-15
object level read query
about, 93-11, 93-13
fetch groups, 93-13
join reading, 93-12
partial object query, 93-11
read all query, 93-11
read object query, 93-11
object model
about, 2-16
generating with `tljaxb.cmd`, 18-8
optimization, 11-8
Oracle TopLink requirements, 2-11
object type converter
about, 12-1, 30-12
configuring, 32-22
object type mappings
configuring, 32-22
`ObjectLevelChangeTrackingPolicy`, 97-7
object-relational descriptors
configuring, 27-1
locking policy, 25-64
understanding, 23-11
object-relational mappings
about, 46-1
array, understanding, 46-2
configuring, 47-1
nested table, 46-3, 52-1
object array, 46-2, 51-1
overview, 2-16
reference, 46-2, 49-1, 50-1
structure, 46-2, 48-1
object-relational projects
about, 17-6
sequencing, 17-5
objects
cascading refresh in cache, 93-35
creating and registering, 99-2
query, 94-6
refreshing in cache, 93-35
registering and unregistering, 99-1
ObjectTypeMapping
see ObjectTypeConverter
OC4J. *See* Oracle Containers for J2EE
one-to-many EIS mappings
configuring, 60-1
key on source, 53-13
key on target, 53-15
understanding, 53-12
one-to-many relational mappings
configuring, 39-1
understanding, 33-7
one-to-one EIS mappings
configuring, 59-1
key on source, 53-9
key on target, 53-10
understanding, 53-8
one-to-one relational mappings
configuring, 37-1
expressions, 95-5
joining, 37-1
understanding, 33-5
with EJB, 33-6
online help, 4-14
Open Project button, 18-10
opening projects, 18-10
operators
boolean logic, 95-3
optimistic locking
about, 23-9
application architecture, 2-11
cascading locking policy, 23-19, 25-67
database exception, 94-25
exceptions, 13-29
field locking policy, about, 23-20
rollbacks, 23-20
version locking policy, 23-18, 23-19, 25-67
with `forceUpdateToVersionField()` method, 99-18
optimistic locking policy
field locking, about, 23-20
version locking, 23-18, 23-19, 25-67
`OptimisticLockException`, 23-19, 23-21

optimization
about, 11-1
application bottlenecks, 11-2
application server, 11-30
batch reading, 11-17
batch writing, 11-15
CMP partial object queries, 11-17
code generation, 11-8
data access, 11-14, 11-15
data format, 11-15
database, 11-30
descriptors, 11-13
DMS profiler, 11-4, 72-12
existence checking, 11-13
fetch groups, 11-17
fetch size, JDBC, 11-18
general, 11-8
inheritance, 23-16
instantiation, 11-13
JDBC driver, 11-14, 11-18
join reading, 11-17
mappings, 11-13
named queries, 11-17
object model, 11-8
overview, 2-15
parameter binding, 11-15
partial object queries, 11-17
prepared statement caching, 11-15
profiler, 11-4, 72-12
optimization
TopLink Profiler, 11-2
queries, 11-16
reading, 11-19
ReadQuery method setMaxRows, 11-18
schema, 11-8
setMaxRows, 11-18
understanding, 11-1
unit of work, 11-30
writing, 11-26
Oracle
development support, 2-3
remote session support, 72-31
Oracle Containers for J2EE
creating projects from, 7-9
migrating to TopLink, 7-5
Oracle Database
date and timestamp mappings, 34-3
native sequencing, 17-20
Oracle Database
SEQUENCE object, 17-19
platform, 81-3
schema manager type conversion, 5-3
Oracle extensions
hierarchical queries, 96-6
Oracle Hints, 96-6
Oracle Hints, using with TopLink queries, 96-6
Oracle JDeveloper
configuring with Oracle TopLink, 6-1
TopLink sessions, 6-3
Oracle TopLink
about, 1-1, 4-1
application architectures, 1-4
architectures, 1-1
deploy tool for IBM WebSphere, 10-5
development, 2-1, 3-1
features, 1-4
integrating with application server, 7-1, 7-2
mapping types, 30-1
optimization, 11-1
packaging your application, 9-1
public source, 12-2
runtime components, 3-2
understanding, 1-1
Oracle TopLink Sessions Editor. *see* sessions
Oracle TopLink Workbench
about, 4-1
Ant integration, 4-53
classpath, 4-2
creating projects, 18-1
development process, 4-1
`DRIVER_CLASSPATH`, 4-2
environment, 4-2
error messages, 14-1, 14-3
`JDBC_CLASSPATH`, 4-2
parts of, 4-3
preferences, 4-12
project, 4-1, 18-2
proxy, 4-13, 4-14, 18-16
sample, 4-3
table creator, creating, 5-4
upgrading projects, 18-10
Oracle Virtual Private Database (VPD)
isolated client sessions, 72-21
proxy authentication, 72-22, 81-6, 83-13
`oracle.sql.TimeStamp`, 34-3
order
query keys, 34-8
relational mappings, 34-8
`OrderSibling`, 96-7
`orion-ejb-jar.xml` file
about, 8-9
`entity-deployment` attribute `pm-name`, 8-10
modifying for Oracle TopLink, 8-9
`persistence-manager` attribute `class-name`, 8-10
`persistence-manager` attribute `descriptor`, 8-10
`persistence-manager` attribute `name`, 8-10
`persistence-manager` subentry for`pm-properties`
`cache-synchronizations`, 8-11
`customization-class`, 8-11
`db-platform-class`, 8-11
`default-mapping`, 8-11
`project-class`, 8-11
`remote-relationships`, 8-11
`session-name`, 8-11
OTN (Oracle Technology Network), 1-4, 2-3
OTS (Object Transaction Service)
about, 97-3

unit of work, 97-3
outer joins
in expressions, 94-11
inheritance, 25-49
output parameter event in stored procedures, 94-23
OX mappings
about, 30-27
extensions, simple type translator, 30-13, 30-14
read conversions, 30-13
write conversions, 30-14

P

package names
generating, 4-30
renaming, 4-52
TopLink Workbench, Navigator window, 4-9
see also classes
packaging for deployment
about, 9-1
BMP applications, 9-5
CMP applications, 9-4
EJB 3.0, 9-1
Java applications, 9-1
JSP and Servlet applications, 9-2
Session Bean applications, 9-3
packet time-to-live cache coordination, 88-15
parallel
expressions, 95-11
unit of work, 97-9
parameter binding
about, 11-15
byte arrays, 11-15
configuring, 20-7, 25-22, 83-9, 94-17
database login level, 83-9
descriptor level, 25-22
external connection pools, 11-16
internal connection pools, 11-16
J2EE, 11-16
named queries, 25-22
optimizing, 11-15
project level, 20-7
queries, 94-17
streams, 11-15
strings, 11-15
trouble shooting, 11-15
parameterized batch writing
about, 11-15
`setMaxBatchWritingSize()`, 11-15
parameterized expressions
about, 95-8
example, 95-9
parameterized SQL
enabling on queries, 94-17
Oracle TopLink optimization features, 11-26
See also parameter binding
parser conflicts, XML, 7-3
partial object reading optimization, 11-20
passwords, encryption, 18-13
performance optimization
about, 11-1
application bottleneck, 11-2
examples, 11-20
JConnect method isClosed, 11-14
using Performance Profiler, 11-2
Performance Profiler
about, 11-2
class, 11-3
persistence
about, 2-19
BEA WebLogic deployment, 8-14
by reachability, 98-6
components of, 2-12
descriptor, 8-13
implementation options, 2-12
manager, 7-4
OC4J, 7-5
projects, 17-2
types, 19-5
using a persistence layer, 2-14
persistence manager
default, 7-4
migration, 7-4
restrictions, 7-4
persistent classes
project, 4-32
requirements, 2-11, 2-12
types, 19-7
pessimistic locking
about, 23-9, 23-22
application architecture, 2-11
policy, 23-22
phantom reads, preventing, 99-31
platforms
Attunity Connect database, 81-3
Cloudscape database, 81-3
data source, 81-3
database, 18-3, 20-2, 21-2, 81-3, 83-1, 84-1
DB2 database, 81-3
EIS, 81-4
functions in expressions, 95-16
HSQL database, 81-3
Informix database, 81-3
J2C adapter, 81-4
JDBC drivers, 81-3
Microsoft Access database, 81-3
MySQL4 database, 81-3
Oracle database, 81-3
Oracle8 database, 81-3
Oracle9 database, 81-3
parser, XML, 7-2
PointBase database, 81-3
projects, and, 17-4
server, 74-15
session configuration, 74-15
SQLAnyWhere database, 81-3
SQLServer database, 81-3
SybasePlatform database, 81-3
XML parser, 7-2
see also target platforms

PointBase platform, 81-3
pop-up menus. *see* context menus
ports
  multicast group, 88-5
  permissions, 7-23, 7-24
post-commit clones, avoiding, 99-31
Potential EJB Descriptors dialog box, 18-10
pre-allocating sequence numbers, 17-20, 20-4, 83-5, 84-2
preferences
  class import, 4-16
  database, 4-17
  EJB, 4-16
  general, 4-13
  help, 4-14
  importing and exporting, 4-13
  Oracle TopLink Workbench, 4-12
  sessions, 4-18, 4-19
Preferences - Class dialog, 4-16
Preferences - EJB dialog, 4-16
Preferences - General dialog, 4-13
Preferences - Help dialog, 4-14
Preferences - Mappings dialog, 4-15
Preferences dialog, 4-13
prepared statement caching
  about, 11-15
  configuring, 20-7, 25-22, 83-9, 94-17
  database login level, 83-9
  descriptor level, 25-22
  external connection pools, 11-16
  internal connection pools, 11-16
  J2EE, 11-16
  named queries, 25-22
  optimizing, 11-15
  project level, 20-7
  queries, 94-17
  query level, 94-17
preserving XML documents, 29-6
primary key
  about, 2-17
  cache, 93-34
  composite, 33-9
  inheritance, 23-16
  multiple tables, 26-13
  `primkey` in `ejb-jar.xml` file, 18-15
  queries with compound, 93-34
  read-only settings, 32-2
  restrictions, 17-15
  setting, 4-26, 25-3, 26-2
  unit of work, 97-11
  unknown, 25-47
  variable class relationships, 33-6, 38-3
private relationships, 32-16
Problems window
  about, 4-4, 4-11
  sample, 4-11
  see also error messages
profiler
  about, 74-10
  development tool, 11-2
  DMS, 11-4, 72-12
  Oracle TopLink, 11-2, 11-3, 72-12
  selecting, 74-11, 74-14, 74-17, 74-20
Project - Multiple Projects tab, 74-10
Project Status Report dialog box, 18-13
projects
  about, 16-1, 17-1, 17-2, 18-2
  architecture, 17-2
  cache type and size, 19-13, 25-35
  comments, 19-21
  configuring, 19-1
  copying objects, 6-6
  creating, 7-9, 18-1, 18-2, 18-3, 18-6
  deployment login, and, 17-3
  deployment overview, 2-15
  descriptors, 16-2
  development login, and, 17-4
  direct access to mapped fields, 19-4
  EIS, 17-7, 17-9, 21-1
  errors, 14-3
  existence checking, 19-8
  exporting, 18-13, 18-14
  for sessions, 74-9
  indexed records, 17-9
  Java, 18-3
  JAXB, 18-6
  locked, 6-6
  login, 17-3, 71-1, 81-2
  login, and, 17-3
  mapped field access, default, 19-4
  mapped records, 17-9
  mapping projects, creating, 18-2
  mappings, 16-2, 18-2
  merging, 6-4
  metadata, 2-20
  method access to mapped fields, 19-4
  model, exporting, 18-14
  `.mwp` file, 4-1
  nondescriptor classes, 4-51
  non-persistent, 17-2
  nonrelational, 17-2
  object-relational, types supported, 17-6
  OC4J, creating from, 7-9
  open, 18-10
  Oracle TopLink Workbench, 18-2
  packages, renaming, 4-52
  persistence type, 17-2, 19-7
  platforms, and, 17-4
  prior TopLink versions, 18-10
  recently opened, 18-10
  relational, 17-2, 17-6, 20-1
  renaming, 18-12
  reopening, 18-10
  saving, 18-11
  sequencing, 17-4, 17-5, 20-3, 83-4
  session login, and, 17-3
  sharing, 6-6
  status report, 18-12
  team development, 6-3
  TopLink Workbench, Navigator window, 4-9

types of, 17-1, 18-2, 23-1
updating from `ejb-jar.xml`, 18-16
upgrading from 2.x or 3.x, 18-10
writing `ejb-jar.xml` file, 18-16
XML, 17-9, 22-1
XML records, 17-9
`projects.xml` file
about, 8-2
EJB 3.0, 8-3, 8-5
schema, 8-2
XSD file, 8-2
propagation mode, cache, 88-2
proxies. *see* wrapper policy
proxy authentication
about, 81-5
applications, 81-5
Oracle Virtual Private Database, 72-22, 81-6, 83-13
session events, 72-6
use cases, 81-5
proxy indirection
about, 30-8
example, 32-9
restrictions, 30-8
proxy settings, preferences, 4-13
public source code, 12-2

Q

qualified names, database tables, 4-23, 4-24
queries
about, 93-1
application development process, 2-13, 71-2
building, 93-6
cache, 93-29
`Call` queries, 93-3, 93-17
cascading, 93-15, 94-14
concepts, 93-2
conforming, 99-8
database queries, 93-10, 94-4
DatabaseQuery, 93-3
descriptor query manager, 93-3
EIS interactions, 93-5
EJB finders, 93-24
EJB QL query language, 93-5
`ejb-jar.xml` file, 25-10, 26-6, 28-6
executing, 93-6
expressions, 93-3
fetch groups, 93-13
flashback queries, 78-1, 93-18
hierarchical queries, Oracle extensions, 93-18
hints, Oracle extensions, 93-18
historical, 93-21, 96-2
interface and inheritance queries, 93-22
joining, 93-12
languages, about, 93-4
named queries, 93-16
object level modify query, 93-14, 93-15
object level read query, 93-11, 93-13
on inheritance hierarchies, 96-4
on interfaces, 96-4
optimizing, 11-16
Oracle database features, 96-5
Oracle extensions, 93-18, 93-19
parameter binding, 94-17
partial object query, 93-11
performance, 11-16
prepared statement caching, 94-17
query keys, 93-4
read all query, 93-11
read object query, 93-11
redirect queries, 93-20
remote sessions, 72-32
report, 11-21, 93-8
report query, 93-15
results, 93-8
returning policy, 93-17
session queries, 93-9, 94-1
SQL query language, 93-4
stored functions, Oracle extensions, 93-19
subqueries, 95-10
summary queries, 93-3
timeout, 25-24, 25-26, 94-10
types, 93-1
UpdateAll, 93-14
XML query language, 93-5
Queries tab, 25-26, 25-28
query by example, 94-6
query cache, 93-36
Query Exception, 13-31
query keys
about, 25-30, 25-33, 93-4
adding, 25-31, 25-34
and expressions, 95-10
creating, 25-31, 25-34
direct mappings, 25-30
generating, 25-30
in expressions, 95-10
interface descriptors, 25-31, 25-33
Java implementation, 25-31
modifying, 25-31
order, 34-8
relationship mappings, 25-31
specifying, 25-31, 25-34
unmapped attributes, 25-30, 25-31
Query Keys tab, 25-31
query object query. *See* `DatabaseQuery`
query objects
batch reading, 94-10
cache expiration, 96-21
caching results, 93-36, 96-20
examples, 94-5
ordering for `ReadAll` queries, 94-9
report query, 11-21
specifying collection class, 94-9
query results
about, 93-8
caching, 93-36
collections, 93-8
cursors, 93-8

reports, 93-8
streams, 93-8
query timeout example, 94-10
QueryManager
about, 23-8
joining expressions, 96-4
QuerySequence, 83-8

R

read access
providing in sessions, 72-16
read all operation, 94-2
read conversions
simple type translator, 30-13
read only
descriptors, 25-5
files, 6-6
mappings, 32-2, 32-3
read operation, 94-2
read optimization
about, 11-19
batch reading, 11-19
fetch groups, 25-79
joining, 11-19
lazy attribute loading, 25-79
object indirection, 11-19
partial object reading, 11-20
report query, 11-20
soft cache weak identity map, 11-19
unit of work, 11-19
weak identity map, 11-19
read queries
cascading refresh of identity maps, 93-36
identity map cache refresh, 93-35
refreshing identity maps, 93-35
`ReadAll finders`
executing, 96-10
`ReadAll` queries
ordering in query objects, 94-9
`readAllObjects()`
about, 26-8
example, 94-2
reading
`ejb-jar.xml`, 18-16
just-in-time reading, 30-8
whole XML documents, 36-1
reading through the write connection, 99-28
read-locking, 23-9
`readObject()`
example, 94-2
recently opened projects, 18-10
redirect queries
about, 93-20, 96-1
creating, 96-1
finders, 96-13
using, 96-12
reference key field, 33-11, 33-12
reference object-relational mappings
configuring, 49-1, 50-1
understanding, 46-2
`ReferenceMapping` class, 46-2
references
database tables, 4-26, 4-28
foreign keys, 14-15, 14-23
Refresh from Database button, 4-24
refresh policy
cache, 87-8
EJB finders, 87-8
refreshing
cache, 25-27
classes, 4-50
`refreshObject()`, 94-2
remote sessions, 72-32
sessions, 75-5
registering objects, 99-1
`registerNewObject method`, 99-2
reimporting schemas, 4-35
relation tables
about, 33-9
many-to-many mappings, 40-1
mappings, 40-2
relational descriptors
associated table, 26-2
configuring, 26-1
locking policy, 25-64
understanding, 23-11
relational mappings
about, 33-1
aggregate collection, 33-10, 41-1
aggregate object, 33-12, 43-1
configuring, 34-1
converters, 33-3
direct collection, 33-11, 42-1
direct map, 33-12, 44-1
direct-to-field, 33-4, 35-1
direct-to-XMLType, 33-4, 36-1
many-to-many, 33-8, 40-1
one-to-many, 33-7, 39-1
one-to-one, 33-5, 37-1
options, 34-1
order, 34-8
transformation, 33-3, 33-15, 45-1
variable one-to-one, 33-6, 38-1
relational projects
about, 17-2
configuring, 20-1
object-relational databases, 17-6
relational databases, 17-6
understanding, 17-6
relationships
about, 2-17
bidirectional, 4-31, 32-34, 33-5
expressions, 95-6
in `ejb-jar.xml` file, 18-16
unexpected multiplicity, 7-14
relative locations
about, 19-2
classpath, 19-3
Remote Command Manager Exception, 13-71

remote connection using RMI, example, 73-10
remote sessions
about, 72-2, 72-30
application layer, 72-31
creating, 73-10
limitations of, 72-31
securing access, 72-32
server layer, 72-31
transport layer, 72-31
unit of work, 72-32
Remove Class button, 4-51
Remove Table button, 4-24
removing sessions from brokers, 79-1
Rename dialog, 4-24
renaming
packages, 4-52
projects, 18-10, 18-12
reopening projects, 18-10
report query
about, 93-15
query objects, 11-21
read optimization, and, 11-20
using, 11-21, 93-8
reports
project status, 18-12
query results, 93-8
see also status reports
resuming unit of work
after commit, 99-14
after commit failure, 99-14
returning policy
configuring, 25-67
SQLCall, 93-17
Returning tab, 25-68
RMI
coordinated cache, 90-1
naming service, 88-9
remote session support, 72-31
rollback
and Java Transaction API, 97-11
overview, 97-11
with optimistic locking, 23-20
root class
about, 23-13
inheritance, 23-13
root EIS descriptor, 24-5
root element, descriptor, 29-5
runtime
acquiring sessions, 72-5
components, 3-2
services, configuring sessions, 74-14
troubleshooting, 15-14

S

Save All button, 18-11
Save button, 18-11
saving projects, 19-2
schema context
descriptors without, 29-3
EIS descriptors, 28-2
XML descriptors, 29-2
Schema Document Info tab, 4-36, 4-38
schema manager
about, 5-1
automatic table creation, 5-6
creating a table creator, 5-4
DB2, 5-3
default table generator, 5-4
Java table creator, 5-4
MS Access, 5-3
MySQL, 5-3
Oracle, 5-3
Oracle TopLink Workbench table creator, 5-4
sequencing, 5-3, 5-6
Sybase, 5-3
table creator, 5-2, 5-4, 5-6
table definition, 5-2
type conversion, 5-3
usage, 5-1
Schema Structure tab, 4-34
schemas
about, 2-17
context for EIS descriptors, 28-2
context for XML descriptor, 29-2
data storage, 2-17
database, 4-22
default root for EIS descriptors, 28-4
details, 4-34
document information, 4-36, 4-38
errors, 14-23
importing, 4-35
optimizing, 11-8
properties, 4-35
reimporting, 4-35
schema manager, 5-1
structure, 4-34
XML schemas, 4-33
SCM. *see* source control management
scripts
SQL, generating, 4-29
see also SQL
scrollable cursor
traversing, 96-16
using for ReadAllQuery, 96-16
security
cache coordination, permissions, 7-23
data source access, permissions, 7-24
disabling `doPrivileged()`, 7-25
`doPrivileged()`, 7-4
EJB, permissions, 7-24
enabling `doPrivileged()`, 7-25
J2EE application, permissions, 7-24
java.util.logging, permissions, 7-24
loading project.xml, permissions, 7-23
loading sessions.xml, permissions, 7-23
logging, permissions, 7-24
password encryption, 18-13
permissions by feature, 7-23
permissions when `doPrivileged()`

disabled, 7-24
port permissions, 7-23, 7-24
`SecurityManager` integration, 7-4
system properties, permissions, 7-23
understanding permissions, 7-22
with BEA WebLogic, 7-20
Select Classes dialog box, 4-50
`SelectedFieldsLockingPolicy`, 23-21
sensors, DMS profiler, 11-4, 72-12
SEQ_COUNT column in sequence table, 17-20
sequence connection pools, 81-8
sequence numbers, write optimization features, 11-26
SEQUENCE objects in Oracle native sequencing, 17-19
sequencing
and relational projects, 17-14
BEA WebLogic, 17-22
BEA WebLogic single column sequence table, 17-17, 20-3, 83-4
configuring, 17-15
connection pools, 81-8
default, 83-6
DefaultSequence class, 83-6
descriptors, 23-9, 26-3
entity beans, 17-21
IBM WebSphere, 17-22
isolated client sessions, 72-24
Java configuration, 17-15
Microsoft SQL Server, 5-6
native, 17-18, 17-19, 17-20
non-Oracle database native, 17-19
Oracle Database native, 17-18, 17-20
Oracle TopLink Workbench configuration, 17-15
overriding default, 83-6
platform default, 83-6
preallocation, 11-29, 17-20
QuerySequence class, 83-8
schema manager, 5-6
SEQ_COUNT, 17-20
sequence type, configuring, 20-3, 83-4
sessions, and, 72-13
stored procedures, 17-18, 83-7
Sybase Database, 5-6
table, 17-16
table, default column and table names, 17-17
unary table, 17-17
with stored procedures, 83-7
serialization
descriptor exceptions, 99-13
EJB 1.x, 30-9
EJB 2.x, 30-9
EJB 3.0, 30-10
indirection, 30-9
merging into session cache with unit of work, 99-13
serialized object converter
about, 12-2, 30-10
configuring, 32-18
serialized object mappings, 32-19
SerializedObjectMapping. *see* SerializedObjectConverter
server layer, 72-31
server platform
external transaction controller, 74-14
runtime services, 74-14
session configuration, 74-14
session event listener, 74-17
Server Platform tab, 74-15
server sessions
about, 72-1, 72-14, 72-15
cache, 72-33
connection options, 72-18
service channel, coordinated cache, 88-3
session beans
about, 2-26
deploying, 8-9, 9-3, 10-1
model, 2-24, 2-26
remote session support for, 72-31
session brokers
about, 72-2, 72-26
adding sessions to, 79-1
alternatives, 72-28
architecture, 72-26
configuration, 79-1
limitations of, 72-28
renaming, 79-1
two-phase commits, 72-27
two-stage commits, 72-27
session configuration file
about, 73-1
loading alternative, 75-4
preferences, 4-18, 4-19
see also `sessions.xml` file
session customizer, 72-4
Session Event Manager, 72-5
session events
example, 72-5, 72-6
isolated sessions, and, 72-5
manager, 72-5
proxy authentication, and, 72-6
Session Loader Exceptions, 13-64
Session Manager
about, 75-1, 75-2
acquiring, 75-2, 75-3
defaults, 75-3
destroying sessions, 75-11
J2EE defaults, 75-3
sessions, acquiring, 72-5
storing sessions, 75-10
sessions
about, 2-13, 71-1, 72-1, 72-2, 73-1
acquiring at runtime, 72-5, 75-1
adding to session brokers, 79-1
additional mapping projects, 74-9
API, 72-33
application server logging, 72-9
architecture, 72-2
cache, 72-33, 87-2
client, 72-14

configuring, 74-1, 74-17
connection policy, 74-19
creating, 2-21, 73-1, 73-4, 73-6, 73-8, 85-1
customization, 72-4
database sessions, 72-29, 73-8
destroying in Session Manager, 75-11
errors, 14-23
event listeners, 72-7, 74-17
events, 72-6
external transaction controller, 74-14, 97-2
historical client sessions, 72-25
in `sessions.xml` file, 72-4
isolated client sessions, 72-19, 77-1
loading with alternative class loader, 75-4
logging, 72-7, 72-9, 72-10, 74-4, 74-5
logging into, 75-10
logging out of, 75-10
logins, 74-4
management of, 75-1
metadata, about, 2-21
multiple sessions, 72-28, 75-2
named queries, 74-21
object cache, 72-3
optimizing, 11-13
preferences, 4-18, 4-19
queries, 93-6, 93-9, 94-1
refreshing, 75-5
registering descriptors, 72-13, 74-2, 74-3, 74-10
remote sessions, 72-30, 73-10
sequencing, about, 72-13
server, 72-14
server platform, 74-14, 74-15
SQL and messages, 72-10
storing in Session Manager, 75-10
three-tier architecture, 72-14
transformation mappings, 33-15
types, 72-1
unit of work, 72-1, 72-19
`sessions.xml` file
about, 8-4
acquiring, 72-5
CMP applications, 8-4, 72-4
configuring with JDeveloper, 6-3
creating, 73-1
default location, 72-5
EJB 3.0, 8-5
loading alternative configuration file, 75-4
non-CMP applications, 8-4
schema, 8-4
sessions, 72-4
XSD file, 8-4
`setAdditionalJoinExpression()`, 96-5
`setenv.cmd` file, 4-2
`setenv.sh` file, 4-2
`setMaxBatchWritingSize()`, 11-15
`setMultipleTableJoinExpression()`, 96-5
`setShouldPerformDeletesFirst()`, 99-17
Settings tab, 25-26, 25-28
`setValue()` method, 30-7
shared library
setting for BEA WebLogic, 7-15
shared library for BEA WebLogic, 7-15
simple type translators
about, 12-2, 30-12
configuring, 32-23
in Java, 32-25
read conversions, 30-13
write conversions, 30-14
soft cache weak identity map
about, 87-4
when to use, 87-5
sorting, in memory, 32-26
source code, public, 12-2
source control management
projects, 6-3
with Oracle TopLink Workbench, 6-3
see also team development
source table, reference, 4-28
`source.jar`, 12-2
splash screen, 4-13
SQL
call, 94-19
comparing with expressions, 95-1
custom queries for basic persistence, per descriptor, 26-6
EJBQLCall, 94-24
generating from database tables, 4-29
parameter binding, 11-15
parameterized, 11-26, 94-17
prepared statement caching, 11-15
queries, 93-4
scripts with binding arguments, 26-6
SQLCall, 94-19
`StoredFunctionCall`, 94-23
StoredProcedureCall, 94-21
unit of work, 99-16
SQL Creation Script dialog box, 4-29
SQL DISTINCT, 11-24
SQL Exception, 13-27
SQLAnyWhere platform, 81-3
SQLCall
about, 93-17
binding input parameters, 94-20
input parameters, 94-20
input-output parameters, specifying, 94-20
output parameters, 94-20
Returning Policy, 93-17
using, 94-19
SQLServer platform, 81-3
stages of development with Oracle TopLink, 2-2
stale data
cache, 87-6, 87-7
coordination, cache, 87-7
invalidating the cache, 87-7
locking policy, and, 87-6
per-class cache configuration, 87-7
per-query cache refresh, 87-7
`StartWith`, 96-7
stateful
beans, 2-26

comparing with stateless, 2-26
stateless
comparing with stateful, 2-26
static attributes, 4-45, 4-48
static fetch groups, querying with, 96-3
status report, generating, 18-12
stored functions
about, 93-19
using, 94-23
stored procedures
output parameter event, 94-23
queries, 94-21
sequencing, and, 17-18, 83-7
streams
as query results, 93-8
cursored, 72-32, 96-18
string bindings, CLOB mappings, 31-4
structure object-relational mappings
configuring, 48-1
understanding, 46-2
`StructureMapping` class, 46-2
subqueries
multiple expressions, 95-10
subselects in expressions, 95-10
subselects, in expressions, 95-10
summary queries, 93-3
superclass, 4-41
Sybase
database schema manager type conversion, 5-3
native sequencing, 17-19
platform, 81-3
synchronous change propagation, 88-2
system properties
oracle.j2ee.security.usedoprivileged, 7-25
oracle.j2ee.toplink.security.usedoprivileged, 7-25
`toplink.cts.collection.checkParameters`, 7-25
toplink.xml.platform, 7-3

T

table creator
about, 5-2
creating, 5-4
using, 5-6
Table Creator dialog, 5-4
table generation. *see* automatic table generation
table sequence
about, 17-16
default column and table names, 17-17
tables
adding database, 4-22
associating with relational descriptors, 26-2
database, 4-21
defining schema, 5-2
errors, 14-17
generating, automatic, 30-4
import filter, 4-23
mapping to descriptors, 25-3, 26-2
merging files, 6-5
multiple, 26-13
primary key, 4-26
references, 34-8
relation tables for mappings, 40-2
`TableDefinition` class, 5-2
see also database tables
target foreign keys
about, 33-5, 34-9
configuring, 34-9
target platforms
about, 2-5
choosing, 2-3
target tables
in direct collection mappings, 42-1
reference, 4-28
team development, 6-3
technical support, 2-3
three-tier architecture
about, 1-4, 2-22
authentication, 81-5
migrating to scalable architecture, 72-29
overview, 2-22
sessions, 72-14
timestamp support
about, 34-3
direct to field mappings, 34-3
Oracle Database, 34-3
TIMESTAMP timezone, 34-4
`TimestampLockingPolicy`, 23-18
timezone, with TIMESTAMP, 34-4
`tljaxb.cmd` file, 18-8
toolbars, 4-4, 4-6
topic name, 89-1
TopLink. *see* Oracle TopLink
TopLink expressions. *see* expressions
TopLink profiler
about, 11-2
selecting, 74-11, 74-14, 74-17, 74-20
TopLink Workbench. *see* Oracle TopLink Workbench
`toplink-ejb-jar.xml` file, 8-6, 16-2
`toplink-ejb.xml` file, 8-7
Transaction Exception, 13-73
transactional data sources, 81-1
transactions
client-controlled, 99-25
CMP, 97-3, 99-25
container-controlled, 99-25
demarcation, 97-2
external transaction controller, 74-14, 97-2
external, integrating, 97-2, 99-21
isolated client sessions, 72-25
isolation, 97-2, 97-4
JTA, 97-3
JTS, 97-3
local, 99-25
local, CMP, 99-25
OTS, 97-3
overview, 2-14, 71-3, 97-1
see also unit of work
transformation EIS mappings

configuring, 61-1
understanding, 53-17
transformation mappings
about, 30-14
attribute transformation, 32-29, 32-31
mutable, 32-33
transformation relational mappings
configuring, 45-1
understanding, 33-15
transformation XML mappings
configuring, 70-1
understanding, 62-31
transient attributes, 4-45, 4-48
transparent indirection
about, 30-8
persistent class requirements, 2-12
transport layer, 72-31
troubleshooting
BEA WebLogic deployment, 15-3
data sources, 14-25
deploy tool, 10-6
deployment, 15-1
`ejbc`, 10-3
IBM WebSphere deployment, 15-12
migration from OC4J persistence, 7-13
Oracle TopLink Workbench, 14-1
unit of work, 99-31, 99-35
two-phase commits, 72-27
two-stage commits, 72-27
two-tier architecture
about, 1-6, 2-24
authentication, 81-5
understanding, 2-24
type conversion
automatic, 34-2
NCHAR, 30-12
NCLOB, 30-12
NVARCHAR2, 30-12
`oracle.sql.TimeStamp`, 34-3
schema manager, 5-3
String to TIMESTAMP, 30-12
TIMESTAMP to String, 30-12
type conversion converter
about, 12-2, 30-11
configuring, 32-20
provided by direct-to-field mappings, 34-3
TypeConversionMapping
see TypeConversionConverter
types of mappings, 30-1
typesafe enumeration, in EIS mappings, 53-3

U

unary table sequence
about, 17-17
BEA WebLogic single column sequence table, 17-17, 20-3, 83-4
undeployment, 10-4
unexpected relationship multiplicity, 7-14
unidirectional relationships, 33-2
unit of work
about, 97-1, 97-2, 97-4
acquiring, 98-1
API, 97-11
architecture, 97-1
auditing, 99-21
benefits of, 97-4
cache, 87-2
change policy, 97-6
clones, 97-9
CMP integration, 97-3
commit and Java Transaction API, 97-10
commit, writing changes before, 99-7, 99-12, 99-24
conform results of in-memory query, 93-31
creating objects, 98-2
deleting objects, 98-7
example, 97-6
external transaction controller, 74-14, 97-2
external transactions, 97-2, 99-21
integrating with CMP, 99-24
isolation, 97-4
JTA integration, 97-3
JTS integration, 97-3
life cycle, 97-5
modifying objects, 98-2
mutable mappings, 32-33
nested, 97-9, 99-15
`newInstance method`, 99-2
optimization, 11-30, 97-11
OTS integration, 97-3
parallel, 97-9, 99-15
pre-commit validation, 99-35
primary keys, 97-11
proxy indirection, 30-8
queries, 97-12
read optimization, 11-19
read-only classes, 99-6, 99-7
`registerNewObject method`, 99-2
remote sessions, 72-32
resuming, 99-14
reverting, 99-15
rollback, 97-11
sessions, 72-1, 72-19
transaction demarcation, 97-2
transactions, 97-4
unit of work demarcation, 97-2
troubleshooting, 99-31
validating objects, 99-34
with custom SQL, 99-16
write optimization, 11-26
writing changes before commit, about, 99-7
writing changes before commit, and external transaction exceptions, 99-24
writing changes before commit, and external transaction timeouts, 99-24
writing changes before commit, as alternative to conforming, 99-12
unmapping, 31-6

unregistering objects, 99-1
unsaved items, displaying in TopLink Workbench Navigator window, 4-9
update
operation, 94-3, 94-4
projects from prior versions, 18-10
UpdateAll query, 93-14
`updateObject()`, 26-8
Use XML Schema "type" attribute, configuring, 32-23
`useBatchWriting()`, 83-11
user-defined functions, in expressions, 95-16

V

validating
descriptors, 24-6
JAXB, 17-14
projects, 18-12
unit of work, 99-34
Validation Exceptions, 13-43
Value Converter tab, 44-4
value holders
about, 30-7
`ValueHolder` class, 30-7
`ValueHolderInterface` class, 2-12, 30-7, 33-9
variable one-to-one relational mappings
class indicator, 38-1
configuring, 38-1
primary key, unique, 38-3
understanding, 33-6
`VariableOneToOneMapping` class, 33-6
Varray in Oracle database. *see* array mappings
version control, 6-6
Version Control Assistance dialog box, 6-6
`VersionLockingPolicy`, 23-18
volatile attributes, 4-45, 4-48
VPD. *see* Oracle Virtual Private Database

W

warning icon, 4-10, 4-11
weak identity map, 87-3
web browser, specifying, 4-14
web services architecture, 1-6, 2-35
WebLogic. *see* BEA WebLogic
`weblogic-ejb-jar.xml` file
described, 8-5
modifying for Oracle TopLink, 8-13
unsupported tags, 8-15
WebSphere. *see* IBM WebSphere
welcome screen, 4-14
wildcard, 63-3
wrapper policy
about, 25-78
implementing in Java, 25-79
write
conversions, simple type translator, 30-14
write all operation, 94-3
write query
disabling identity map cache, 94-15
non-cascading, 94-14
objects, 94-13
overview, 94-13
write-locking, 23-9
writing
batch, 11-26, 83-11
`ejb-jar.xml` file, 18-16
optimization, 11-26
sessions write access, 72-17

X

`xdb.jar` file, 4-3
XML
descriptor, schema context, 29-2
generating deployment, 8-3
mappings, concepts, 62-2
preserving data, 29-6
projects, 18-3
query language, 93-5
reading whole documents, 36-1
records, 53-3
schemas, 4-33
XML Conversion Exception, 13-74
XML descriptors
configuring, 29-1
schema context, 29-2
understanding, 23-12
XML mappings
about, 62-1
any collection, 62-29, 69-1
any object, 62-27, 68-1
any type support, 62-4
composite collection, 62-25, 67-1
composite direct collection, 62-14, 65-1
composite object, 62-21, 66-1
concepts, 62-2
configuring, 63-1
default conversion pairs, customizing, 62-11
direct, 62-5, 64-1
extensions, 62-4
jaxb:class support, 62-4
list support, 62-3
reference descriptor, configuring, 63-2
transformation, 62-31, 70-1
types of, 62-1
union support, 62-3
xsd:list, 62-3
xsd:union, 62-3
XML parser platform
about, 7-2
configuring, 7-3
creating, 7-3
Crimson, 7-3
default, 7-3
limitations, 7-3
parser conflicts, 7-3
toplink.xml.platform system property, 7-3
used by, application server, 7-2

used by, Oracle TopLink, 7-2
XML Platform Exception, 13-83
XML projects
configuring, 22-1
JAXB support, 17-10
sequencing, 17-5
understanding, 17-9
XML queries, 93-5
XML schema
jaxb:class, and EIS mappings, 53-3
jaxb:class, and XML mappings, 62-4
jaxb:class, understanding, 30-20
type, 32-23
xs:any, understanding, 62-4
xs:anyType, understanding, 62-4
xsd:list, understanding, 30-17
xsd:union, understanding, 30-17
see also schemas
XML Type functions, 95-5
XMLMarshalException, 13-81
XMLPlatformException, 13-83
XPath
by name, 30-16
by position, 30-15
mapping Java attributes, 32-10
support in OX mappings, 30-15
support in XML mappings, 53-3, 62-3
XSD file
`projects.xml` file, 8-2
`sessions.xml` file, 8-4

Z

zero-argument constructors
editing, 4-16