

Oracle TopLink

Developer's Guide, Volume 2

10g Release 3 (10.1.3)

B13593-02

February 2006

Understanding Mappings

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 Corporation, 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 and PeopleSoft 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	li

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	2-1
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	2-4
Selecting an Architecture With TopLink	2-5
Tiers.....	2-5
Three Tier	2-5
Two Tier	2-6
Service Layer.....	2-6
EJB Session Beans.....	2-7
EJB Entity Beans	2-7
Plain Old Java Objects (POJO)	2-8
Data Access	2-8
Data Type	2-8
Multiple Data Sources	2-8
Isolating Data Access.....	2-8
Historical Data Access.....	2-9

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

Advantages and Disadvantages	2-24
Technical Challenges	2-24
Understanding the EJB Session Bean Facade Architecture.....	2-24
Example Implementation.....	2-25
Advantages and Disadvantages	2-25
Understanding Session Beans	2-25
Technical Challenges	2-26
Unit of Work Merge.....	2-26
Understanding the EJB Entity Beans with CMP Architecture	2-27
Example Implementation.....	2-28
Advantages and Disadvantages	2-28
Technical Challenges	2-28
External JDBC Pools	2-29
JTA/JTS Integration.....	2-29
Cache Coordination.....	2-29
Maintaining Bidirectional Relationships.....	2-29
Managing Dependent Objects in EJB 1.1	2-30
Managing Dependent Objects in EJB 2.0.....	2-32
Managing Collections of EJBObjects in EJB 1.1	2-32
Managing Collections of EJBObjects in EJB 2.0	2-33
Understanding the EJB Entity Beans With BMP Architecture	2-33
Example Implementations.....	2-35
Advantages and Disadvantages	2-35
Technical Challenges	2-35
External JDBC Pools	2-35
JTA/JTS Integration.....	2-36
Cache Coordination.....	2-36
Understanding the Web Services Architecture.....	2-36
Example Implementations.....	2-36
Advantages and Disadvantages	2-36
Technical Challenges	2-37

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	4-27
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 Entities 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.....	4-38
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 Nondesoriptor 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 an Integrated Development Environment

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

6 Using the Schema Manager

Understanding the Schema Manager	6-1
Schema Manager Java and Database Type Conversion	6-3
Sequencing	6-3
Creating a Table Creator.....	6-4
Using TopLink Workbench During Development.....	6-4
Using the Default Table Generator at Run Time	6-4
Using Java.....	6-4
Creating a TableCreator Class.....	6-5
Creating a TableDefinition Class	6-5
Adding Fields to a TableDefinition.....	6-5
Defining Sybase and Microsoft SQL Server Native Sequencing.....	6-6
Creating Tables With a Table Creator.....	6-6
Automatic Database Table Creation	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
EJB 3.0 and the project.xml File	8-3
sessions.xml File	8-3
XSD File Format	8-4
Non-CMP Applications and Session Metadata	8-4
CMP Applications and Session Metadata	8-4
EJB 3.0 and the sessions.xml File	8-5
ejb-jar.xml File	8-5
EJB 3.0 and the ejb-jar.xml File	8-5
<J2EE-Container>-ejb-jar.xml File	8-5
OC4J and the orion-ejb-jar.xml File	8-5
BEA WebLogic Server and the weblogic-ejb-jar.xml File	8-6
EJB 3.0 and the <J2EE-Container>-jar.xml File	8-6
toplink-ejb-jar.xml File	8-6
OC4J and the toplink-ejb-jar.xml File	8-6
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
EJB 3.0 and the toplink-ejb-jar.xml File	8-8
Java Applications	8-8
JavaServer Pages and Servlet Applications	8-8
Session Bean Applications	8-8
CMP Applications	8-9
BMP Applications	8-9
Configuring the orion-ejb-jar.xml File for OC4J	8-9
Configuring persistence-manager Entries	8-9
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	8-13
Unsupported weblogic-ejb-jar.xml File Tags	8-14

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	9-4
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 Performance With the TopLink Profiler	11-2
Configuring the TopLink Performance Profiler	11-3
Accessing the TopLink Profiler Results	11-3
Measuring 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	11-17
Cursored Streams and Scrollable Cursors.....	11-18
Read Optimization Examples.....	11-19
Reading Case 1: Displaying Names in a List	11-20
Reading Case 2: Batch Reading Objects.....	11-22
Reading Case 3: Using Complex Custom SQL Queries	11-24
Reading Case 4: Using View Objects	11-24
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 – 200)	13-2
Concurrency Exceptions (2001 – 2006)	13-25
Conversion Exceptions (3001– 3008)	13-26
Database Exceptions (4002 – 4018).....	13-27
Optimistic Lock Exceptions (5001 – 5008)	13-29
Query Exceptions (6001 – 6121)	13-30
Validation Exceptions (7001 – 7147)	13-42
EJB QL Exceptions (8001 – 8010)	13-55
Session Loader Exceptions (9000 - 9010)	13-56
EJB Exception Factory Exceptions (10001 - 10069)	13-58
Communication Exceptions (12000 - 12003).....	13-64
XML Data Store Exceptions (13000 - 13020)	13-64
Deployment Exceptions (14001 - 14033)	13-67
Synchronization Exceptions (15001 - 15025)	13-70
SDK Data Store Exceptions (17001 - 17006)	13-71
EIS Exceptions (17007 – 17025)	13-72
JMS Processing Exceptions (18001 - 18002).....	13-74
SDK Descriptor Exceptions (19001 - 19003)	13-75
Default Mapping Exceptions (20001 - 20008)	13-75

Discovery Exceptions (22001 - 22004).....	13-77
Remote Command Manager Exceptions (22101 - 22111)	13-77
Transaction Exceptions (23001 - 23011)	13-79
XML Conversion Exceptions (25001 - 25016)	13-80
Migration Utility Exceptions (26001 - 26017).....	13-82
EJB JAR XML Exceptions (72000 – 72022)	13-84

14 TopLink Workbench Error Reference

Miscellaneous Errors (1 – 89).....	14-1
Project Errors (100 – 126)	14-2
Descriptor Errors (200 – 350)	14-4
Mapping Errors (400 – 483)	14-11
Table Errors (500 – 610)	14-15
XML Schema Errors (700 – 706)	14-20
Session Errors (800 – 812)	14-21
Common Classpath Problems.....	14-22
Data Source Problems	14-23
Database Connection Problems	14-23

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.....	15-14

Part VI TopLink Tutorial

16 Planning the Tutorial Project

Overview	16-1
Before you Begin	16-1
Understanding the Tutorial Project.....	16-2
Source Files.....	16-2

Object Model Classes.....	16-2
Application Classes	16-3
Data Model.....	16-4
Creating and Populating Database Tables	16-7

17 Building the Tutorial Project

Step 1: Creating a Project	17-1
TopLink Workbench Project.....	17-1
Adding Java Classes	17-3
Step 2: Adding a Data Source.....	17-5
Creating a Database Development Login.....	17-5
Importing Database Tables	17-6
Project-Level Sequencing	17-8
Step 3: Associating Descriptors to Tables	17-9
Associating With Multiple Tables	17-10
Descriptor-Level Sequencing.....	17-11
Step 4: Mapping Attributes	17-12
Direct-to-Field Mappings.....	17-12
Using Automap	17-13
Additional Direct-to-Field Mappings	17-14
One-to-One Mappings.....	17-14
Additional One-to-One Mappings	17-16
One-to-Many Mappings.....	17-16
Additional One-to-Many Mappings	17-18
Object Type Mappings	17-18
Aggregate Object Mappings.....	17-19
Aggregate Descriptor	17-20
Aggregate Mapping.....	17-20
Direct Collection Mappings.....	17-21
Many-to-Many Mappings.....	17-21
Transformation Mappings.....	17-23
Step 5: Using Advanced Descriptor Properties	17-25
Inheritance.....	17-25
Root Parent Class	17-25
Child Class	17-26
Indicator Values	17-27
Locking Policy	17-28
Named Queries.....	17-29
Step 6: Verifying the Project	17-32

18 Deploying the Tutorial Project

Step 1: Generating Deployment Information	18-1
Creating the project.xml File	18-1
Step 2: Creating Sessions Configuration	18-3
Creating the sessions.xml File	18-3
Creating a Server Session.....	18-3
Associating the Project With the Session.....	18-4

Updating the data-sources.xml File.....	18-5
Creating a Database Deployment Login.....	18-6
Configuring Logging Options.....	18-7
Step 3: Packaging for Deployment.....	18-8
Domain JAR File.....	18-8
WAR File	18-8
EAR File.....	18-8
Step 4: Deploying to an Application Server.....	18-9
Step 5: Running the Tutorial Application.....	18-9

Part VII Mapping and Configuration Overview

19 Understanding TopLink Mapping and Configuration Concepts

Mapping and Configuration Concepts	19-1
Projects.....	19-1
Descriptors	19-2
Mappings.....	19-2

Part VIII Projects

20 Understanding Projects

TopLink Project Types	20-1
Project Concepts	20-2
Project Architecture	20-2
Relational and Nonrelational Projects.....	20-2
Persistent and Nonpersistent Projects.....	20-2
Projects and Login.....	20-3
Non-CMP Session Role: Session Login.....	20-3
CMP Deployment Role: Deployment Login	20-3
Development Role: Development Login	20-4
Projects and Platforms.....	20-4
Projects and Sequencing.....	20-4
Configuring how to Obtain Sequence Values.....	20-5
Configuring Where to Write Sequence Values.....	20-5
XML Namespaces.....	20-5
Relational Projects.....	20-6
Building Relational Projects for a Relational Database	20-6
Building Relational Projects for an Object-Relational Database	20-6
EIS Projects	20-7
Building EIS Projects with XML Records	20-9
Building EIS Projects With Indexed or Mapped Records	20-9
XML Projects	20-9
TopLink Support for Java Architecture for XML Binding (JAXB)	20-10
Understanding JAXB-Specific Generated Files	20-10
Understanding TopLink-Specific Generated Files.....	20-11
Using TopLink JAXB Compiler Generated Files at Run Time	20-13

JAXB Validation	20-14
Understanding the Project API	20-14
Project Inheritance Hierarchy.....	20-14
Understanding Sequencing in Relational Projects	20-14
Sequencing Configuration Options	20-15
Sequencing Types.....	20-16
Table Sequencing	20-16
Unary Table Sequencing	20-17
Query Sequencing.....	20-18
Default Sequencing.....	20-18
Native Sequencing With an Oracle Database Platform.....	20-18
Native Sequencing With a Non-Oracle Database Platform.....	20-19
Sequencing and Preallocation Size	20-20
Sequencing With CMP Entity Beans	20-21
Understanding XML Namespaces	20-22
TopLink Workbench Namespace Resolution	20-22
Element and Attribute Form Options	20-22
Element Form Default Qualified and Attribute Form Default Unqualified	20-23
Element and Attribute Form Default Unqualified.....	20-24
Element and Attribute Form Default Qualified	20-25
TopLink Runtime Namespace Resolution.....	20-26

21 Creating a Project

Project Creation Overview	21-1
Using TopLink Workbench	21-2
Creating New TopLink Workbench Projects.....	21-2
Using Java.....	21-3
Creating a Project for an Existing Object and Data Model	21-5
Using TopLink Workbench	21-5
Creating a Project From an Existing Object Model	21-5
Using TopLink Workbench	21-5
Creating a Project From an Existing Data Model	21-5
Using TopLink Workbench	21-6
Creating an XML Project From an XML Schema	21-6
Using TopLink Workbench	21-6
Using the Command Line.....	21-8
Creating a Project by Migrating an EAR to OC4J	21-9
Creating a Project From an OC4J EJB 2.0 CMP EAR at Deployment Time	21-10
Working With Projects	21-10
Opening Existing Projects	21-10
Saving Projects.....	21-11
Saving Projects With a New Name or Location	21-12
Generating the Project Status Report	21-12
Exporting Project Information	21-13
Exporting Deployment XML Information.....	21-14
Exporting Model Java Source.....	21-14
Exporting Project Java Source	21-14

Exporting Table Creator Files.....	21-15
Working With the ejb-jar.xml File.....	21-15
Writing to the ejb-jar.xml File.....	21-16
Reading From the ejb-jar.xml File.....	21-16

22 Configuring a Project

Configuring Common Project Options.....	22-1
Configuring Project Save Location	22-2
Using TopLink Workbench	22-2
Configuring Project Classpath.....	22-3
Using TopLink Workbench	22-3
Configuring Mapped Field Access at the Project Level.....	22-4
Using TopLink Workbench	22-4
Configuring Persistence Type.....	22-5
Using TopLink Workbench	22-6
Configuring Default Descriptor Advanced Properties.....	22-7
Using TopLink Workbench	22-8
Configuring Existence Checking at the Project Level.....	22-8
Using TopLink Workbench	22-9
Configuring Project Deployment XML Options.....	22-10
Using TopLink Workbench	22-10
Configuring Model Java Source Code Options.....	22-11
Using TopLink Workbench	22-11
Configuring Deprecated Direct Mappings	22-12
Using TopLink Workbench	22-13
Configuring Cache Type and Size at the Project Level.....	22-13
Using TopLink Workbench	22-14
Configuring Cache Isolation at the Project Level	22-16
Using TopLink Workbench	22-16
Configuring Cache Coordination Change Propagation at the Project Level.....	22-17
Using TopLink Workbench	22-18
Configuring Cache Expiration at the Project Level	22-19
Using TopLink Workbench	22-20
Configuring Project Comments.....	22-21
Using TopLink Workbench	22-21

23 Configuring a Relational Project

Relational Project Configuration Overview	23-1
Configuring Relational Database Platform at the Project Level.....	23-2
Using TopLink Workbench	23-2
Configuring Sequencing at the Project Level.....	23-3
Using TopLink Workbench	23-3
Using Java.....	23-4
Configuring Login Information	23-5
Using TopLink Workbench	23-5
Configuring Development and Deployment Logins	23-6

Using TopLink Workbench	23-6
Logging in to the Database	23-7
Configuring Named Query Parameterized SQL and Statement Caching at the Project Level.....	23-7
Using TopLink Workbench	23-8
Configuring Table Generation Options.....	23-9
Using TopLink Workbench	23-9
Configuring Table Creator Java Source Options.....	23-10
Using TopLink Workbench	23-10
Configuring Project Java Source Code Options.....	23-11
Using TopLink Workbench	23-11

24 Configuring an EIS Project

EIS Project Configuration Overview.....	24-1
Configuring EIS Data Source Platform at the Project Level.....	24-2
Using TopLink Workbench	24-2
Configuring EIS Connection Specification Options at the Project Level.....	24-2
Using TopLink Workbench	24-3

25 Configuring an XML Project

XML Project Configuration Overview	25-1

Part IX Descriptors

26 Understanding Descriptors

Descriptor Types	26-1
Descriptor Concepts.....	26-2
Descriptor Architecture.....	26-2
Descriptors and Inheritance.....	26-3
Descriptors and EJB	26-3
Nondeferred Changes	26-3
Creating a New Entity Bean and ejbCreate / ejbPostCreate Methods	26-4
Inheritance	26-4
Fetch Groups	26-4
Amendment and After-Load Methods	26-5
Descriptors and Aggregation	26-5
Aggregate and Composite Descriptors in Relational Projects	26-5
Root and Composite Descriptors in EIS Projects.....	26-8
Composite Descriptors in XML Projects.....	26-8
Descriptor Event Manager.....	26-8
Descriptor Query Manager.....	26-8
Descriptors and Sequencing	26-9
Descriptors and Locking	26-9
Default Root Element.....	26-9
Relational Descriptors.....	26-11
Object-Relational Descriptors	26-11

EIS Descriptors	26-11
XML Descriptors.....	26-12
Understanding Descriptors and Inheritance	26-12
Specifying a Class Indicator.....	26-13
Using Class Indicator Fields.....	26-13
Using Class Extraction Methods.....	26-14
Inheritance and Primary Keys (Relational and EIS Only).....	26-15
Single and Multi-Table Inheritance (Relational Only).....	26-15
Single Table Inheritance.....	26-15
Multitable Inheritance.....	26-16
Aggregate and Composite Descriptors and Inheritance	26-17
Inheritance and EJB.....	26-17
Understanding Descriptors and Locking	26-17
Optimistic Version Locking Policies	26-17
Optimistic Version Locking Policies and Cascading	26-18
Optimistic Locking and Rollbacks.....	26-20
Optimistic Field Locking Policies	26-20
Pessimistic Locking Policy.....	26-21
Locking in a Three-Tier Application	26-21
Optimistic Locking in a Three-Tier Application	26-21
Pessimistic Locking in a Three-Tier Application	26-22
Understanding the Descriptor API	26-22
Descriptor Inheritance Hierarchy	26-23

27 Creating a Descriptor

Descriptor Creation Overview	27-1
Creating a Relational Descriptor.....	27-1
Using TopLink Workbench	27-2
Relational Class Descriptors.....	27-2
Relational Aggregate Descriptors.....	27-2
Relational Interface Descriptors.....	27-2
Using Java.....	27-2
Creating an Object-Relational Descriptor.....	27-3
Using Java.....	27-3
Creating an EIS Descriptor.....	27-4
Using TopLink Workbench	27-4
EIS Root Descriptors.....	27-5
EIS Composite Descriptors	27-5
Using Java.....	27-5
Creating an XML Descriptor	27-5
Using TopLink Workbench	27-5
Using Java.....	27-5
Validating Descriptors.....	27-6
Generating Java Code for Descriptors	27-6

28 Configuring a Descriptor

Configuring Common Descriptor Options	28-1
Configuring Primary Keys	28-2
Using TopLink Workbench	28-3
Using Java.....	28-4
Relational Projects.....	28-4
EIS Projects.....	28-4
Configuring Read-Only Descriptors	28-4
Using Read-Only Entity Beans.....	28-5
Using TopLink Workbench	28-5
Using Java.....	28-6
Configuring Unit of Work Conforming at the Descriptor Level	28-6
Using TopLink Workbench	28-7
Using Java.....	28-7
Configuring Descriptor Alias	28-7
Using TopLink Workbench	28-8
Using Java.....	28-9
Configuring Descriptor Comments	28-9
Using TopLink Workbench	28-9
Configuring Named Queries at the Descriptor Level	28-10
Using TopLink Workbench	28-10
Adding Named Queries.....	28-12
Configuring Named Query Type and Parameters	28-13
Configuring Named Query Selection Criteria.....	28-14
Configuring Read All Query Order	28-15
Configuring Named Query Optimization.....	28-16
Configuring Named Query Attributes	28-17
Configuring Named Query Group/Order Options	28-19
Creating an EIS Interaction for a Named Query	28-20
Configuring Named Query Options.....	28-22
Configuring Named Query Advanced Options.....	28-24
Using Java.....	28-25
Configuring Query Timeout at the Descriptor Level	28-26
Using TopLink Workbench	28-26
Using Java.....	28-27
Configuring Cache Refreshing	28-27
Using TopLink Workbench	28-28
Using Java.....	28-29
Configuring Query Keys.....	28-29
Using TopLink Workbench	28-31
Using Java.....	28-31
Configuring Interface Query Keys	28-33
Using TopLink Workbench	28-34
Using Java.....	28-34
Configuring Cache Type and Size at the Descriptor Level	28-35
Using TopLink Workbench	28-35
Using Java.....	28-37

Configuring Cache Isolation at the Descriptor Level	28-37
Using TopLink Workbench	28-37
Using Java.....	28-38
Configuring Cache Coordination Change Propagation at the Descriptor Level	28-38
Using TopLink Workbench	28-39
Using Java.....	28-40
Configuring Cache Expiration at the Descriptor Level.....	28-40
Using TopLink Workbench	28-41
Using Java.....	28-42
Configuring Cache Existence Checking at the Descriptor Level	28-42
Using TopLink Workbench	28-42
Using Java.....	28-43
Configuring a Descriptor With EJB Information	28-44
Using TopLink Workbench	28-44
Using Java.....	28-46
Configuring CMP Information	28-46
Configuring BMP Information.....	28-47
Configuring Reading Subclasses on Queries	28-47
Using TopLink Workbench	28-48
Using Java.....	28-48
Configuring Inheritance for a Child (Branch or Leaf) Class Descriptor	28-49
Using TopLink Workbench	28-49
Using Java.....	28-50
Configuring Inheritance for a Parent (Root) Descriptor.....	28-50
Using TopLink Workbench	28-51
Using Java.....	28-52
Configuring Inheritance Expressions for a Parent (Root) Class Descriptor	28-53
Using Java.....	28-55
Configuring Inherited Attribute Mapping in a Subclass.....	28-56
Using TopLink Workbench	28-56
Using Java.....	28-57
Configuring a Domain Object Method as an Event Handler.....	28-57
Using TopLink Workbench	28-58
Using Java.....	28-60
Configuring a Descriptor Event Listener as an Event Handler.....	28-60
Using Java.....	28-62
Configuring Locking Policy	28-62
Using TopLink Workbench	28-63
Using Java.....	28-65
Configuring an Optimistic Locking Policy	28-65
Configuring Optimistic Locking Policy Cascading	28-65
Configuring a Pessimistic Locking Policy	28-65
Configuring Returning Policy	28-65
Using TopLink Workbench	28-66
Using Java.....	28-67
Configuring Instantiation Policy	28-68
Using TopLink Workbench	28-68

Using Java.....	28-69
Configuring Copy Policy	28-69
Using TopLink Workbench	28-70
Using Java.....	28-70
Configuring Change Policy	28-70
Using Java.....	28-71
Configuring Deferred Change Detection Policy	28-71
Configuring Object Change Tracking Policy	28-71
Configuring Attribute Change Tracking Policy	28-72
Configuring a History Policy	28-73
Using Java.....	28-75
Configuring Write Responsibility	28-75
Configuring Wrapper Policy	28-75
Using Java.....	28-76
Configuring Fetch Groups.....	28-77
Using Java.....	28-78
Configuring Amendment Methods	28-78
Using TopLink Workbench	28-79

29 Configuring a Relational Descriptor

Relational Descriptor Configuration Overview.....	29-1
Configuring Associated Tables.....	29-2
Using TopLink Workbench	29-2
Using Java.....	29-3
Configuring Sequencing at the Descriptor Level	29-3
Using TopLink Workbench	29-4
Using Java.....	29-5
Configuring a Sequence by Name	29-5
Configuring the Same Sequence for Multiple Descriptors	29-5
Configuring the Platform Default Sequence	29-6
Configuring Custom SQL Queries for Basic Persistence Operations.....	29-6
Using TopLink Workbench	29-7
Using Java.....	29-8
Configuring Interface Alias	29-10
Using TopLink Workbench	29-11
Using Java.....	29-11
Configuring a Relational Descriptor as a Class or Aggregate Type	29-11
Using TopLink Workbench	29-12
Using Java.....	29-12
Configuring Multitable Information	29-13
Using TopLink Workbench	29-13
Using Java.....	29-14

30 Configuring an Object-Relational Descriptor

Object-Relational Descriptor Configuration Overview	30-1
Configuring Field Ordering	30-2
Using Java.....	30-2

31 Configuring an EIS Descriptor

EIS Descriptor Configuration Overview	31-1
Configuring Schema Context for an EIS Descriptor	31-2
Using TopLink Workbench	31-2
Choosing a Schema Context	31-3
Using Java	31-3
Configuring Default Root Element	31-3
Using TopLink Workbench	31-4
Choosing a Root Element	31-4
Using Java	31-5
Configuring Record Format	31-5
Using Java	31-6
Configuring Custom EIS Interactions for Basic Persistence Operations	31-6
Using TopLink Workbench	31-6
Using Java	31-8
Configuring an EIS Descriptor as a Root or Composite Type	31-8
Using TopLink Workbench	31-9
Using Java	31-9

32 Configuring an XML Descriptor

XML Descriptor Configuration Overview	32-1
Configuring Schema Context for an XML Descriptor	32-1
Using TopLink Workbench	32-2
Choosing a Schema Context	32-2
Using Java	32-3
Configuring for Complex Type of anyType	32-3
Using TopLink Workbench	32-4
Configuring Default Root Element	32-5
Using TopLink Workbench	32-5
Choosing a Root Element	32-5
Configuring Document Preservation	32-6
Using TopLink Workbench	32-6
Using Java	32-6

Part X Mappings

33 Understanding Mappings

Mapping Types	33-1
Mapping Concepts	33-2
Mapping Architecture	33-2
Example Mapping	33-3
Automatic Mappings	33-4
Automapping With TopLink Workbench at Development Time	33-4
Default Mapping in EJB 2.0 or 3.0 CMP Projects Using OC4J at Run Time	33-4
JAXB Project Generation at Development Time	33-5
Indirection	33-5

Value Holder Indirection.....	33-6
Transparent Indirect Container Indirection.....	33-8
Proxy Indirection	33-8
Indirection and EJB.....	33-9
Indirection, Serialization, and Detachment.....	33-9
Method Accessors and Attribute Accessors.....	33-10
Mapping Converters and Transformers	33-10
Serialized Object Converter.....	33-10
Type Conversion Converter	33-11
Object Type Converter	33-12
Simple Type Translator.....	33-12
Transformation Mappings.....	33-14
Mappings and XPath	33-15
XPath by Position.....	33-15
XPath by Path and Name.....	33-15
XPath by Name.....	33-16
Self XPath	33-16
Mappings and xsd:list and xsd:union Types	33-17
Mapping an xsd:union Type	33-17
Mapping an xsd:list Type	33-17
Mapping a List of Unions	33-18
Mapping a Union of Lists	33-19
Mapping a Union of Unions.....	33-19
Mappings and the jaxb:class Customization.....	33-20
all, choice, or sequence Structure.....	33-20
group Structure	33-21
sequence or choice Structure Containing a group	33-21
group Structure Containing a sequence or choice	33-22
group Structure Containing a group.....	33-23
Limitations of jaxb:class Customization Support.....	33-23
Mappings and JAXB Typesafe Enumerations.....	33-24
Understanding the Mapping API.....	33-25
Relational Mappings	33-25
Object-Relational Mappings.....	33-26
XML Mappings	33-27
EIS Mappings.....	33-27

34 Creating a Mapping

Mapping Creation Overview	34-1
Creating Mappings Manually During Development	34-1
Using TopLink Workbench	34-1
Creating Mappings Automatically During Development	34-2
Using TopLink Workbench	34-2
Creating Mappings Automatically During Deployment	34-2
Creating Mappings to Oracle LOB Database Objects.....	34-2
Using the Oracle JDBC OCI Driver or Server Driver.....	34-3
Using the Oracle JDBC Thin Driver	34-5

Removing Mappings	34-6
Using TopLink Workbench	34-6

35 Configuring a Mapping

Configuring Common Mapping Options.....	35-1
Configuring Read-Only Mappings.....	35-2
Using TopLink Workbench	35-3
Using Java.....	35-3
Configuring Indirection.....	35-3
Using TopLink Workbench	35-4
Using Java.....	35-5
Configuring ValueHolder Indirection	35-6
Configuring ValueHolder Indirection With Method Accessing.....	35-7
Configuring ValueHolder Indirection With EJB 3.0 on OC4J	35-8
Configuring IndirectContainer Indirection	35-8
Configuring Proxy Indirection.....	35-9
Configuring XPath	35-10
Using TopLink Workbench	35-11
Choosing the XPath	35-12
Configuring a Default Null Value at the Mapping Level	35-12
Using TopLink Workbench	35-13
Using Java.....	35-13
Configuring Method Accessing.....	35-14
Using TopLink Workbench	35-14
Using Java.....	35-15
Configuring Private or Independent Relationships.....	35-16
Using TopLink Workbench	35-17
Using Java.....	35-17
Configuring Mapping Comments.....	35-18
Using TopLink Workbench	35-18
Configuring a Serialized Object Converter	35-18
Using TopLink Workbench	35-19
Using Java.....	35-19
Configuring a Type Conversion Converter.....	35-20
Using TopLink Workbench	35-20
Using Java.....	35-21
Configuring an Object Type Converter	35-22
Using TopLink Workbench	35-22
Using Java.....	35-23
Configuring a Simple Type Translator.....	35-23
Using TopLink Workbench	35-24
Using Java.....	35-24
Configuring a JAXB Typesafe Enumeration Converter.....	35-25
Using Java.....	35-26
Configuring Container Policy.....	35-26
Using TopLink Workbench	35-27
Using Java.....	35-28

Configuring Attribute Transformer	35-29
Using TopLink Workbench	35-30
Using Java.....	35-31
Configuring Field Transformer Associations	35-31
Using TopLink Workbench	35-32
Specifying Field-to-Transformer Associations	35-32
Using Java.....	35-33
Configuring Mutable Mappings	35-33
Using TopLink Workbench	35-34
Using Java.....	35-34
Configuring Bidirectional Relationship	35-34
Using TopLink Workbench	35-35
Configuring the Use of a Single Node	35-36
Using TopLink Workbench	35-36
Using Java.....	35-37

Part XI Relational Mappings

36 Understanding Relational Mappings

Relational Mapping Types	36-1
Relational Mapping Concepts	36-2
Directionality	36-2
Converters and Transformers	36-3
Using a Direct Mapping.....	36-3
Using a Converter Mapping.....	36-3
Using a Transformation Mapping.....	36-3
Relational Mappings and EJB.....	36-3
Direct-to-Field Mapping	36-4
Direct to XMLType Mapping	36-4
One-to-One Mapping	36-5
One-to-One Mappings and EJB.....	36-6
Variable One-to-One Mapping	36-6
One-to-Many Mapping	36-7
One-to-Many Mappings and EJB.....	36-8
Many-to-Many Mapping	36-8
Many-to-Many Mappings and EJB.....	36-9
Aggregate Collection Mapping	36-10
Aggregate Collection Mappings and Inheritance	36-10
Aggregate Collection Mappings and EJB	36-11
Implementing Aggregate Collection Mappings.....	36-11
Direct Collection Mapping	36-11
Direct Map Mapping	36-12
Aggregate Object Mapping	36-12
Aggregate Object Mappings with a Single Source Object.....	36-13
Aggregate Object Mappings With Multiple Source Objects.....	36-14
Implementing an Aggregate Object Relationship Mapping.....	36-14
Transformation Mapping	36-15

37	Configuring a Relational Mapping	
	Configuring Common Relational Mapping Options.....	37-1
	Configuring a Database Field.....	37-2
	Using TopLink Workbench	37-4
	Configuring Reference Descriptor.....	37-5
	Using TopLink Workbench	37-5
	Configuring Batch Reading.....	37-6
	Using TopLink Workbench	37-7
	Using Java.....	37-7
	Configuring Query Key Order	37-8
	Using TopLink Workbench	37-8
	Configuring Table and Field References (Foreign and Target Foreign Keys).....	37-8
	Using TopLink Workbench	37-9
38	Configuring a Relational Direct-to-Field Mapping	
	Relational Direct-to-Field Mapping Configuration Overview	38-1
39	Configuring a Relational Direct-to-XMLType Mapping	
	Relational Direct-to-XMLType Mapping Overview.....	39-1
	Configuring Read Whole Document.....	39-1
	Using TopLink Workbench	39-1
40	Configuring a Relational One-to-One Mapping	
	Relational One-to-One Mapping Configuration Overview	40-1
	Configuring Joining at the Mapping Level.....	40-1
	Using TopLink Workbench	40-2
41	Configuring a Relational Variable One-to-One Mapping	
	Relational Variable One-to-One Mapping Configuration Overview	41-1
	Configuring Class Indicator.....	41-1
	Using TopLink Workbench	41-2
	Configuring Unique Primary Key	41-3
	Understanding Unique Primary Key	41-3
	Using TopLink Workbench	41-3
	Using Java.....	41-4
	Configuring Query Key Association.....	41-4
	Using TopLink Workbench	41-4
42	Configuring a Relational One-to-Many Mapping	
	Relational One-to-Many Mapping Configuration Overview	42-1
43	Configuring a Relational Many-to-Many Mapping	
	Relational Many-to-Many Mapping Configuration Overview.....	43-1
	Configuring a Relation Table.....	43-1

Using TopLink Workbench	43-2
44 Configuring a Relational Aggregate Collection Mapping	
Relational Aggregate Collection Mapping Configuration Overview.....	44-1
45 Configuring a Relational Direct Collection Mapping	
Relational Direct Collection Mapping Configuration Overview	45-1
Configuring Target Table	45-1
Using TopLink Workbench	45-2
Configuring Direct Value Field	45-2
Using TopLink Workbench	45-2
46 Configuring a Relational Direct Map Mapping	
Relational Direct Map Mapping Configuration Overview	46-1
Configuring Direct Value Field	46-1
Using TopLink Workbench	46-1
Configuring Direct Key Field	46-2
Using TopLink Workbench	46-2
Configuring Key Converters.....	46-3
Using TopLink Workbench	46-3
Configuring Value Converters.....	46-4
Using TopLink Workbench	46-4
47 Configuring a Relational Aggregate Object Mapping	
Relational Aggregate Object Mapping Configuration Overview	47-1
Configuring Aggregate Fields	47-1
Using TopLink Workbench	47-2
Configuring Allowing Null Values	47-2
Using TopLink Workbench	47-2
48 Configuring a Relational Transformation Mapping	
Relational Transformation Mapping Configuration Overview.....	48-1
Part XII Object-Relational Mappings	
49 Understanding Object-Relational Mappings	
Object-Relational Mapping Types.....	49-1
Object-Relational Structure Mapping.....	49-2
Object-Relational Reference Mapping	49-2
Object-Relational Array Mapping	49-2
Object-Relational Object Array Mapping.....	49-2
Object-Relational Nested Table Mapping.....	49-3

50	Configuring an Object-Relational Mapping	
	Configuring Common Object-Relational Mapping Options	50-1
	Configuring Reference Class	50-2
	Using Java.....	50-2
	Configuring Attribute Name	50-2
	Using Java.....	50-3
	Configuring Field Name	50-3
	Using Java.....	50-3
	Configuring Structure Name	50-4
	Using Java.....	50-4
51	Configuring an Object-Relational Structure Mapping	
	Object-Relational Structure Mapping Configuration Overview.....	51-1
52	Configuring an Object-Relational Reference Mapping	
	Object-Relational Reference Mapping Configuration Overview	52-1
53	Configuring an Object-Relational Array Mapping	
	Object-Relational Array Mapping Configuration Overview	53-1
54	Configuring an Object-Relational Object Array Mapping	
	Object-Relational Object Array Mapping Configuration Overview.....	54-1
55	Configuring an Object-Relational Nested Table Mapping	
	Object-Relational Nested Table Mapping Configuration Overview.....	55-1
Part XIII EIS Mappings		
56	Understanding EIS Mappings	
	EIS Mapping Types.....	56-1
	EIS Mapping Concepts.....	56-2
	EIS Record Type	56-2
	Indexed Records.....	56-2
	Mapped Records	56-3
	XML Records	56-3
	XPath Support.....	56-3
	xsd:list and xsd:union Support	56-3
	jaxb:class Support.....	56-3
	Typesafe Enumeration Support	56-3
	Composite and Reference EIS Mappings	56-4
	Composite EIS Mappings	56-4
	Reference EIS Mappings	56-4
	EIS Mapping Architecture	56-5
	EIS Direct Mapping	56-5

EIS Composite Direct Collection Mapping.....	56-6
EIS Composite Object Mapping.....	56-7
EIS Composite Collection Mapping.....	56-7
EIS One-to-One Mapping.....	56-8
EIS One-to-One Mappings With Key on Source.....	56-9
EIS One-to-One Mappings With Key on Target.....	56-10
EIS One-to-Many Mapping.....	56-12
EIS One-to-Many Mappings With Key on Source.....	56-13
EIS One-to-Many Mappings With Key on Target.....	56-15
EIS Transformation Mapping.....	56-17
 57 Configuring an EIS Mapping	
Configuring Common EIS Mapping Options.....	57-1
Configuring Reference Descriptors.....	57-2
Using TopLink Workbench.....	57-2
Configuring Selection Interaction.....	57-3
Using TopLink Workbench.....	57-4
 58 Configuring an EIS Direct Mapping	
EIS Direct Mapping Configuration Overview.....	58-1
 59 Configuring an EIS Composite Direct Collection Mapping	
EIS Composite Direct Collection Mapping Configuration Overview.....	59-1
 60 Configuring an EIS Composite Object Mapping	
EIS Composite Object Mapping Configuration Overview.....	60-1
 61 Configuring an EIS Composite Collection Mapping	
EIS Composite Collection Mapping Configuration Overview.....	61-1
 62 Configuring an EIS One-to-One Mapping	
EIS One-to-One Mapping Configuration Overview.....	62-1
Configuring Foreign Key Pairs.....	62-1
Using TopLink Workbench.....	62-2
 63 Configuring an EIS One-to-Many Mapping	
EIS One-to-Many Mapping Configuration Overview.....	63-1
Configuring Foreign Key Pairs.....	63-1
Using TopLink Workbench.....	63-2
Configuring Delete All Interactions.....	63-3
Using TopLink Workbench.....	63-4
 64 Configuring an EIS Transformation Mapping	
EIS Transformation Mapping Configuration Overview.....	64-1

Part XIV XML Mappings

65 Understanding XML Mappings

XML Mapping Types	65-1
XML Mapping Concepts	65-2
Mapping to Simple and Complex Types	65-2
Mapping Order.....	65-3
XPath Support.....	65-3
xsd:list and xsd:union Support	65-3
xs:any and xs:anyType Support.....	65-4
jaxb:class Support.....	65-4
Typesafe Enumeration Support	65-4
Mapping Extensions	65-4
XML Direct Mapping.....	65-5
Mapping to a Text Node	65-5
Mapping to a Simple Text Node.....	65-5
Mapping to a Text Node in a Simple Sequence.....	65-6
Mapping to a Text Node in a Subelement.....	65-6
Mapping to a Text Node by Position	65-7
Mapping to an Attribute	65-8
Mapping to a Specified Schema Type.....	65-9
Mapping to a List Field With an XML Direct Mapping	65-10
Mapping to a Union Field With an XML Direct Mapping.....	65-10
Mapping to a Union of Lists With an XML Direct Mapping.....	65-12
Mapping to a Union of Unions With an XML Direct Mapping	65-12
Mapping With a Simple Type Translator	65-13
XML Composite Direct Collection Mapping.....	65-14
Mapping to Multiple Text Nodes	65-15
Mapping to a Simple Sequence.....	65-15
Mapping to a Sequence in a Subelement.....	65-15
Mapping to Multiple Attributes.....	65-16
Mapping to a Single Text Node With an XML Composite Direct Collection Mapping	65-17
Mapping to a Single Attribute With an XML Composite Direct Collection Mapping	65-18
Mapping to a List of Unions With an XML Composite Direct Collection Mapping.....	65-18
Mapping to a Union of Lists With an XML Composite Direct Collection Mapping.....	65-19
Specifying the Content Type of a Collection With an XML Composite Direct Collection Mapping	65-20
XML Composite Object Mapping.....	65-21
Mapping Into the Parent Record.....	65-21
Mapping to an Element.....	65-22
Mapping to Different Elements by Element Name.....	65-23
Mapping to Different Elements by Element Position	65-24
XML Composite Collection Mapping	65-25
XML Any Object Mapping.....	65-27
XML Any Collection Mapping	65-29
XML Transformation Mapping.....	65-31

66	Configuring an XML Mapping	
	Configuring Common XML Mapping Options	66-1
	Configuring Reference Descriptor.....	66-2
	Using TopLink Workbench	66-2
	Configuring Maps to Wildcard.....	66-3
	Using TopLink Workbench	66-3
67	Configuring an XML Direct Mapping	
	XML Direct Mapping Configuration Overview.....	67-1
68	Configuring an XML Composite Direct Collection Mapping	
	XML Composite Direct Collection Mapping Configuration Overview.....	68-1
69	Configuring an XML Composite Object Mapping	
	XML Composite Object Mapping Configuration Overview.....	69-1
70	Configuring an XML Composite Collection Mapping	
	XML Composite Collection Mapping Configuration Overview	70-1
71	Configuring an XML Any Object Mapping	
	XML Any Object Mapping Configuration Overview.....	71-1
72	Configuring an XML Any Collection Mapping	
	XML Any Collection Mapping Configuration Overview	72-1
73	Configuring an XML Transformation Mapping	
	XML Transformation Mapping Configuration Overview.....	73-1
Part XV Using TopLink Overview		
74	Understanding the Persistence Layer	
	Overview of the Persistence Layer.....	74-1
	Sessions	74-1
	Data Access	74-1
	Cache.....	74-2
	Queries and Expressions.....	74-2
	Transactions.....	74-3
Part XVI TopLink Sessions		
75	Understanding TopLink Sessions	
	Session Types	75-1
	Session Concepts	75-2

Session Architecture	75-2
Object Cache	75-3
Connection Pools	75-3
Query Mechanism.....	75-4
Java Object Builder	75-4
Session Configuration and the sessions.xml File.....	75-4
Session Customization	75-4
Acquiring a Session at Run Time With the Session Manager.....	75-5
Managing Session Events With the Session Event Manager	75-5
Session Event Manager Events	75-6
Session Event Listeners	75-7
Logging.....	75-7
Log Types	75-8
Log Output.....	75-10
Log Level.....	75-10
Logging SQL.....	75-10
Logging Chained Exceptions	75-10
Viewing TopLink Log Messages From the Application Server Control Console	75-10
Profiler	75-11
TopLink Profiler.....	75-11
Oracle Dynamic Monitoring System (DMS)	75-11
Integrity Checker.....	75-12
Exception Handlers.....	75-12
Registering Descriptors	75-12
Sessions and CMP	75-13
Sessions and Sequencing.....	75-13
Server and Client Sessions	75-13
Three-Tier Architecture Overview	75-14
Advantages of the TopLink Three-Tier Architecture	75-15
Shared Resources	75-15
Providing Read Access.....	75-15
Providing Write Access.....	75-16
Security and User Privileges	75-17
Concurrency.....	75-18
Connection Allocation.....	75-18
Unit of Work Sessions	75-19
Isolated Client Sessions	75-19
Isolated Client Sessions and Oracle Virtual Private Database (VPD)	75-20
VPD With Oracle Database Proxy Authentication	75-21
VPD Without Oracle Database Proxy Authentication	75-21
Isolated Client Session Life Cycle.....	75-22
Isolated Client Session Limitations.....	75-23
Historical Client Sessions.....	75-24
Historical Client Session Limitations	75-25
Session Broker and Client Sessions	75-25
Session Broker Architecture	75-26
Committing a Transaction with a Session Broker	75-27

Committing a Session with a JTA Driver: Two-Phase Commits	75-27
Committing a Session Without a JTA Driver: Two-Stage Commits.....	75-27
Session Broker Session Limitations	75-27
Many-to-Many Join Tables and Direct Collection Tables	75-27
Session Broker Alternatives.....	75-28
Database Linking	75-28
Multiple Sessions	75-28
Database Sessions	75-28
Remote Sessions	75-29
Architectural Overview.....	75-30
Application Layer	75-30
Transport Layer.....	75-30
Server Layer.....	75-31
Remote Session Concepts.....	75-31
Securing Remote Session Access	75-31
Queries.....	75-31
Refreshing	75-31
Indirection	75-32
Cursored Streams.....	75-32
Unit of Work	75-32
Sessions and the Cache	75-32
Server and Database Session Cache	75-32
Isolated Session Cache.....	75-33
Historical Session Cache	75-33
Understanding the Session API	75-33

76 Creating Sessions

Session Creation Overview	76-1
Creating a Sessions Configuration	76-1
Using TopLink Workbench	76-2
Configuring a Sessions Configuration	76-2
Using TopLink Workbench	76-2
Creating a Server Session	76-4
Using TopLink Workbench	76-4
Using Java.....	76-5
Creating Session Broker and Client Sessions	76-6
Using TopLink Workbench	76-6
Using Java.....	76-8
Creating Database Sessions	76-8
Using TopLink Workbench	76-8
Using Java.....	76-10
Creating Remote Sessions	76-10
Using Java.....	76-10
Server	76-11
Client.....	76-11

77 Configuring a Session

Configuring Common Session Options	77-1
Configuring a Primary Mapping Project.....	77-2
Using TopLink Workbench	77-3
Using Java.....	77-3
Configuring a Session Login.....	77-4
Configuring Logging	77-4
Using TopLink Workbench	77-5
Using Java.....	77-7
Using Session Logging API	77-7
Configuring a Session to use java.util.logging Package.....	77-8
Configuring Logging in a CMP Application	77-9
Configuring Multiple Mapping Projects.....	77-9
Using TopLink Workbench	77-9
Using Java.....	77-10
Configuring a Performance Profiler	77-10
Using TopLink Workbench	77-11
Using Java.....	77-12
Configuring an Exception Handler.....	77-12
Using TopLink Workbench	77-12
Using Java.....	77-12
Configuring Customizer Class	77-13
Using TopLink Workbench	77-14
Configuring the Server Platform.....	77-14
Using TopLink Workbench	77-15
Using Java.....	77-17
Configuring Session Event Listeners	77-17
Using TopLink Workbench	77-17
Using Java.....	77-18
Configuring the Integrity Checker	77-18
Using Java.....	77-19
Configuring Connection Policy	77-19
Using TopLink Workbench	77-20
Using Java.....	77-21
Configuring Named Queries at the Session Level	77-21
Using Java.....	77-22

78 Acquiring and Using Sessions at Run Time

Session Acquisition Overview	78-1
Understanding the Session Manager	78-2
Multiple Sessions.....	78-2
Acquiring the Session Manager	78-2
Acquiring a Session from the Session Manager	78-3
Loading a Session from sessions.xml Using Defaults.....	78-3
Loading a Session from sessions.xml with an Alternative Class Loader.....	78-4
Loading a Session from an Alternative Session Configuration File	78-4

Loading a Session Without Logging In.....	78-5
Reloading and Refreshing Session Configuration.....	78-5
Refreshing a Session when the Class Loader Changes.....	78-6
Acquiring a Client Session	78-6
Acquiring an Isolated Client Session	78-7
Acquiring a Historical Client Session.....	78-7
Acquiring a Client Session That Uses Exclusive Connections	78-7
Acquiring a Client Session that Uses Connection Properties	78-8
Acquiring a Client Session that Uses a Named Connection Pool.....	78-8
Acquiring a Client Session that Does Not Use Lazy Connection Allocation	78-9
Logging In to a Session	78-9
Using Session API	78-9
Logging Out of a Session	78-10
Storing Sessions in the Session Manager Instance	78-10
Destroying Sessions in the Session Manager Instance	78-10
 79 Configuring Server Sessions	
Server Session Configuration Overview	79-1
Configuring Internal Connection Pools	79-1
Configuring External Connection Pools	79-2
 80 Configuring Exclusive Isolated Client Sessions for Virtual Private Database	
Exclusive Isolated Client Session Configuration Overview.....	80-1
PostAcquireExclusiveConnection Event Handler	80-1
Using Java.....	80-2
PreReleaseExclusiveConnection Event Handler.....	80-2
Using Java.....	80-2
NoRowsModifiedSessionEvent Event Handler.....	80-3
Using Java.....	80-3
ValidationException Handler.....	80-3
 81 Configuring Historical Client Sessions	
Historical Client Session Configuration Overview.....	81-1
Configuring Historical Client Sessions Using an Oracle Platform	81-1
Configuring Historical Client Sessions Using a TopLink HistoryPolicy.....	81-1
 82 Configuring Session Broker and Client Sessions	
Session Broker and Client Session Configuration Overview	82-1
Removing, Renaming, or Adding Sessions	82-1
Using TopLink Workbench	82-2
 83 Configuring Database Sessions	
Database Session Configuration Overview.....	83-1
Configuring External Connection Pools	83-1

Part XVII Data Access

84 Understanding Data Access

Data Access Concepts	84-1
Externally Managed Transactional Data Sources.....	84-1
Data Source Login Types	84-2
DatabaseLogin.....	84-2
EISLogin	84-3
Data Source Platform Types	84-3
Database Platforms.....	84-3
EIS Platforms	84-4
Authentication	84-5
Simple JDBC Authentication.....	84-5
Oracle Database Proxy Authentication.....	84-5
Auditing	84-6
Connections.....	84-6
Connection Pools.....	84-7
Internal Connection Pools.....	84-7
External Connection Pools.....	84-8
Default (Write) and Read Connection Pools.....	84-8
Sequence Connection Pools.....	84-8
Application-Specific Connection Pools.....	84-9
Understanding Data Access API	84-9
Login Inheritance Hierarchy	84-10
Platform Inheritance Hierarchy	84-10

85 Configuring a Data Source Login

Configuring Common Data Source Login Options	85-1
Configuring User Name and Password	85-1
Using TopLink Workbench	85-2
Configuring Password Encryption	85-2
Using Java.....	85-2
Configuring External Connection Pooling	85-2
Using TopLink Workbench	85-3
Configuring Properties	85-4
Using TopLink Workbench	85-4
Using Java.....	85-5
Configuring a Default Null Value at the Login Level	85-5
Using Java.....	85-6

86 Configuring a Database Login

Database Login Configuration Overview	86-1
Configuring a Relational Database Platform at the Session Level	86-1
Using TopLink Workbench	86-1
Configuring Database Login Connection Options	86-2
Using TopLink Workbench	86-2

Configuring Sequencing at the Session Level.....	86-4
Using TopLink Workbench	86-4
Using Java.....	86-5
Using the Platform Default Sequence	86-5
Configuring Multiple Sequences	86-6
Configuring Query Sequencing	86-7
Configuring a Table Qualifier	86-8
Using TopLink Workbench	86-8
Configuring JDBC Options	86-9
Using TopLink Workbench	86-9
Using Java.....	86-11
Configuring Advanced Options	86-11
Using TopLink Workbench	86-11
Configuring Oracle Database Proxy Authentication	86-12
Using Java.....	86-13

87 Configuring an EIS Login

EIS Login Configuration Overview.....	87-1
Configuring an EIS Data Source Platform at the Session Level	87-1
Using TopLink Workbench	87-1
Configuring EIS Connection Specification Options at the Session Level.....	87-2
Using TopLink Workbench	87-2

88 Creating an Internal Connection Pool

Internal Connection Pool Creation Overview	88-1
Using TopLink Workbench	88-1

89 Configuring an Internal Connection Pool

Internal Connection Pool Configuration Overview	89-1
Configuring Connection Pool Sizes	89-1
Using TopLink Workbench	89-2
Configuring Properties.....	89-2
Using TopLink Workbench	89-2
Using Java.....	89-3
Configuring a Nontransactional Read Login.....	89-3
Using TopLink Workbench	89-3
Configuring Connection Pool Connection Options	89-4
Using TopLink Workbench	89-4
Configuring Exclusive Read Connections.....	89-6
Using TopLink Workbench	89-6

Part XVIII Cache

90 Understanding the Cache

Cache Architecture	90-1
Session Cache.....	90-2

Unit of Work Cache	90-2
Cache Concepts	90-2
Cache Type and Object Identity	90-3
Full Identity Map	90-3
Weak Identity Map	90-3
Soft and Hard Cache Weak Identity Maps.....	90-4
No Identity Map.....	90-4
Guidelines for Configuring the Cache and Identity Maps	90-4
Understanding the Internals of Soft and Hard Cache Weak Identity Map	90-5
Querying and the Cache.....	90-6
Handling Stale Data.....	90-6
Configure a Locking Policy	90-6
Configure the Cache on a Per-Class Basis	90-7
Force a Cache Refresh When Required on a Per-Query Basis.....	90-7
Configure Cache Invalidation	90-7
Configure Cache Coordination.....	90-7
Explicit Query Refreshes.....	90-7
Refresh Policy	90-8
EJB Finders and Refresh Policy	90-8
Cache Invalidation	90-8
Cache Coordination	90-9
Cache Isolation	90-9
Cache Locking and Transaction Isolation.....	90-9
Cache Optimization	90-9
Understanding Cache Coordination	90-10
When to use Cache Coordination.....	90-10
Coordinated Cache Architecture	90-11
Session	90-11
Descriptor	90-11
Unit of Work	90-12
Coordinated Cache Types.....	90-12
JMS Coordinated Cache	90-12
RMI Coordinated Cache	90-12
CORBA Coordinated Cache.....	90-13
Custom Coordinated Cache	90-13
Understanding the Cache API	90-13
Object Identity API	90-13
Cache Refresh API	90-14
Cache Invalidation API	90-14
Cache Coordination API	90-15

91 Configuring a Coordinated Cache

Configuring Common Coordinated Cache Options	91-1
Configuring the Synchronous Change Propagation Mode	91-2
Using TopLink Workbench	91-2
Configuring a Service Channel	91-3
Using TopLink Workbench	91-3

Configuring a Multicast Group Address	91-4
Using TopLink Workbench	91-5
Configuring a Multicast Port	91-5
Using TopLink Workbench	91-6
Configuring a Naming Service Type	91-7
Configuring JNDI Naming Service Information	91-7
Using TopLink Workbench	91-8
Configuring RMI Registry Naming Service Information	91-9
Using TopLink Workbench	91-10
Configuring an Announcement Delay	91-11
Using TopLink Workbench	91-12
Configuring Connection Handling	91-13
Using TopLink Workbench	91-13
Configuring Context Properties	91-14
Using TopLink Workbench	91-14
Configuring a Packet Time-to-Live	91-15
Using TopLink Workbench	91-16

92 Configuring a JMS Coordinated Cache

JMS Coordinated Cache Configuration Overview	92-1
Configuring a Topic Name	92-1
Using TopLink Workbench	92-1
Configuring a Topic Connection Factory Name	92-2
Using TopLink Workbench	92-2
Configuring a Topic Host URL	92-3
Using TopLink Workbench	92-3

93 Configuring an RMI Coordinated Cache

RMI Coordinated Cache Configuration Overview	93-1

94 Configuring a CORBA Coordinated Cache

CORBA Coordinated Cache Configuration Overview	94-1

95 Configuring a Custom Coordinated Cache

Custom Coordinated Cache Configuration Overview	95-1
Configuring Transport Class	95-1
Using TopLink Workbench	95-1

Part XIX Queries

96 Understanding TopLink Queries

Query Types	96-1
Query Concepts	96-2
Call.....	96-3
DatabaseQuery	96-3

Data-Level and Object-Level Queries	96-3
Summary Queries	96-3
Descriptor Query Manager	96-3
TopLink Expressions	96-3
Query Keys.....	96-4
Query Languages	96-4
SQL Queries.....	96-4
EJB QL Queries.....	96-5
XML Queries.....	96-5
EIS Interactions.....	96-6
Query by Example	96-6
Building Queries	96-6
Executing Queries	96-7
Handling Query Results	96-8
Collection Query Results.....	96-8
Report Query Results	96-8
Stream and Cursor Query Results	96-8
Session Queries	96-9
Read-Object Session Queries	96-9
Create, Update, and Delete Object Session Queries.....	96-10
Database Queries	96-10
Object-Level Read Query	96-11
ReadObjectQuery	96-11
ReadAllQuery.....	96-11
Partial Object Queries.....	96-11
Join Reading and Object-Level Read Queries.....	96-12
Fetch Groups and Object Level Read Queries	96-12
Data-Level Read Query	96-13
DataReadQuery	96-13
DirectReadQuery	96-13
ValueReadQuery	96-13
Object-Level Modify Query	96-13
WriteObjectQuery.....	96-14
UpdateObjectQuery	96-14
InsertObjectQuery.....	96-14
DeleteObjectQuery.....	96-14
UpdateAllQuery.....	96-14
Object-Level Modify Queries and Privately Owned Parts	96-14
Data-Level Modify Query	96-14
Report Query	96-15
Named Queries	96-15
Call Queries	96-16
SQL Calls	96-16
SQLCall.....	96-17
StoredProcedureCall.....	96-17
StoredFunctionCall	96-17
Oracle Extensions.....	96-17

EJB QL Calls	96-18
Enterprise Information System (EIS) Interactions	96-18
IndexedInteraction	96-19
MappedInteraction	96-19
XMLInteraction	96-19
XQueryInteraction	96-19
QueryStringInteraction	96-19
Redirect Queries	96-19
Historical Queries	96-20
Using an ObjectLevelReadQuery With an AsOfClause	96-21
Using an ObjectLevelReadQuery With Expression Operator asOf	96-21
Using an ObjectLevelReadQuery in a Historical Session.....	96-22
Interface and Inheritance Queries	96-22
Descriptor Query Manager Queries	96-22
Configuring Named Queries	96-22
Configuring Default Query Implementations	96-22
Configuring Additional Join Expressions	96-23
EJB Finders	96-23
Predefined Finders	96-24
Predefined CMP Finders for EJB 2.0	96-24
Predefined CMP Finders for EJB 1.1	96-25
Predefined BMP Finders	96-25
Default Finders	96-26
Call Finders	96-26
DatabaseQuery Finders.....	96-26
Named Query Finders.....	96-27
Primary Key Finders.....	96-27
Expression Finders.....	96-27
EJB QL Finders.....	96-27
SQL Finders.....	96-28
Redirect Finders.....	96-28
The ejbSelect Method	96-28
Queries and the Cache	96-29
Configuring the Cache	96-29
Using In-Memory Queries	96-30
Configuring Cache Usage for In-Memory Queries.....	96-30
Expression Options for In-Memory Queries.....	96-31
Handling Exceptions Resulting From In-Memory Queries.....	96-33
Primary Key Queries and the Cache	96-34
Disabling the Identity Map Cache Update During a Read Query	96-34
Refreshing the Cache	96-35
Object Refresh.....	96-35
Cascading Object Refresh	96-35
Refreshing the Identity Map Cache During a Read Query	96-35
Caching Query Results in the Session Cache	96-36
Caching Query Results	96-36
Internal Query Cache Restrictions.....	96-36

Caching and EJB Finders.....	96-36
Caching Options.....	96-37
Disabling Cache for Returned Finder Results	96-37
Refreshing Finder Results.....	96-38
Understanding the Query API.....	96-38

97 Understanding TopLink Expressions

Understanding the Expression Framework.....	97-1
Expressions Compared to SQL	97-1
Expression Components.....	97-2
Boolean Logic.....	97-3
Database Functions	97-3
Mathematical Functions	97-4
XMLType Functions	97-4
Platform and User-Defined Functions	97-4
Expressions for One-to-One and Aggregate Object Relationships.....	97-5
Expressions for Joining and Complex Relationships	97-5
Understanding Joins.....	97-5
Using TopLink Expression API For Joins.....	97-6
Parameterized Expressions.....	97-7
Expression Method getParameter.....	97-8
Expression Method getField.....	97-9
Query Keys and Expressions.....	97-9
Using Multiple Expressions	97-10
Subselects and Subqueries	97-10
Parallel Expressions	97-11
Data Queries and Expressions	97-11
getField	97-11
getTable	97-12
Creating an Expression.....	97-12
Using TopLink Workbench	97-12
Adding Arguments.....	97-14
Using Java.....	97-15
Creating and Using a User-Defined Function	97-16
Making a User-Defined Function Available to a Specific Platform.....	97-16
Making a User-Defined Function Available to All Platforms	97-17
Using a User-Defined Function	97-17

98 Using Basic Query API

Using Session Queries.....	98-1
Reading Objects With a Session Query	98-1
Reading an Object With a Session Query	98-2
Reading All Objects With a Session Query	98-2
Refreshing an Object with a Session Query	98-2
Creating, Updating, and Deleting Objects With a Session Query	98-3
Writing a Single Object to the Database with a Session Query	98-3

Writing All Objects to the Database With a Session Query	98-3
Adding New Objects to the Database with a Session Query	98-4
Modifying Existing Objects in the Database with a Session Query	98-4
Deleting Objects in the Database with a Session Query	98-4
Using DatabaseQuery Queries	98-4
Reading Objects Using a DatabaseQuery	98-4
Basic DatabaseQuery Read Operations	98-5
Reading Objects Using Partial Object Queries	98-5
Reading Objects Using Report Queries	98-6
Reading Objects Using Query By Example	98-6
Specifying Read Ordering	98-8
Specifying a Collection Class	98-9
Specifying the Maximum Rows Returned	98-9
Configuring Query Timeout at the Query Level	98-10
Using Batch Reading	98-10
Using Join Reading	98-11
Creating, Updating, and Deleting Objects with a DatabaseQuery	98-12
Write Query Overview	98-12
UpdateAll Queries	98-13
Noncascading Write Queries	98-13
Disabling the Identity Map Cache During a Write Query	98-14
Reading Data With a DatabaseQuery	98-14
Using a DataReadQuery	98-15
Using a DirectReadQuery	98-15
Using a ValueReadQuery	98-15
Updating Data With a DatabaseQuery	98-15
Specifying a Custom SQL String in a DatabaseQuery	98-16
Specifying a Custom EJB QL String in a DatabaseQuery	98-16
Using Parameterized SQL and Statement Caching in a DatabaseQuery	98-17
Using Named Queries	98-17
Using SQL Calls	98-18
Using an SQLCall	98-18
Specifying a SQLCall Input Parameter	98-19
Specifying a SQLCall Output Parameter	98-19
Specifying a SQLCall Input / Output Parameter	98-20
Using a StoredProcedureCall	98-20
Specifying an Input Parameter	98-21
Specifying an Output Parameter	98-21
Specifying an Input / Output Parameter	98-22
Using an Output Parameter Event	98-22
Using a StoredFunctionCall	98-22
Using EJB QL Calls	98-23
Using EIS Interactions	98-24
Handling Exceptions	98-24
Handling Collection Query Results	98-25
Handling Report Query Results	98-25

99 Using Advanced Query API

Using Redirect Queries	99-1
Creating a Redirect Query	99-1
Using Historical Queries	99-2
Using Queries With Fetch Groups	99-2
Configuring Default Fetch Group Behavior	99-3
Querying With a Static Fetch Group	99-3
Querying With a Dynamic Fetch Group	99-4
Querying on Interfaces	99-4
Querying on an Inheritance Hierarchy	99-4
Appending Additional Join Expressions	99-4
Using Java	99-5
Using Queries on Variable One-to-One Mappings	99-5
Using Oracle Database Features	99-5
Oracle Hints	99-6
Hierarchical Queries	99-6
startWith Parameter	99-7
connectBy Parameter	99-7
orderSibling Parameter	99-7
Using EJB Finders	99-7
Creating a Finder	99-8
ejb-jar.xml Finder Options	99-9
Using DatabaseQuery Finders	99-10
Using Named Query Finders	99-10
Using Primary Key Finders	99-11
Using EJB QL Finders	99-11
Using SQL Finders	99-12
Using Redirect Finders	99-12
Using the ejbSelect Method	99-15
Handling Cursor and Stream Query Results	99-16
Cursors and Java Iterators	99-16
Traversing Data With Scrollable Cursors	99-16
Java Streams	99-17
Cursored Stream Support	99-17
Optimizing Streams	99-18
Using Cursors and Streams With EJB Finders	99-19
Building the Query	99-19
Executing the Finder From the Client in EJB 1.1	99-19
Executing the Finder From the Client in EJB 2.0	99-20
Using Queries and the Cache	99-21
Caching Results in a ReadQuery	99-21
Configuring Cache Expiration at the Query Level	99-22

Part XX Transactions

100 Understanding TopLink Transactions

Unit of Work Architecture	100-1
Unit of Work Transaction Context.....	100-2
Unit of Work Transaction Demarcation.....	100-2
JTA Controlled Transactions	100-3
OTS Controlled Transactions	100-3
CMP Controlled Transactions	100-3
Unit of Work Transaction Isolation	100-4
Unit of Work Concepts	100-4
Unit of Work Benefits	100-4
Unit of Work Life Cycle	100-5
Unit of Work and Change Policy	100-6
Deferred Change Detection Policy	100-7
Object-Level Change Tracking Policy	100-7
Attribute Change Tracking Policy	100-8
Change Policy Mapping Support	100-8
Clones and the Unit of Work.....	100-9
Nested and Parallel Units of Work.....	100-9
Nested Unit of Work	100-10
Parallel Unit of Work.....	100-10
Commit and Rollback Transactions	100-10
Commit Transactions	100-10
Rollback Transactions.....	100-11
Primary Keys	100-11
Unit of Work Optimization.....	100-11
Understanding the Unit of Work API	100-11
Unit of Work as Session	100-12
Reading and Querying Objects with the Unit of Work.....	100-12
Locking and the Unit of Work	100-12
Example Model Object and Schema	100-12

101 Using Basic Unit of Work API

Acquiring a Unit of Work	101-1
Creating an Object	101-2
Modifying an Object	101-2
Associating a New Target to an Existing Source Object	101-3
Associating Without Reference to the Cache Object.....	101-3
Associating With Reference to the Cache Object.....	101-4
Associating a New Source to an Existing Target Object	101-6
Associating an Existing Source to an Existing Target Object	101-7
Deleting Objects	101-7
Using privateOwnedRelationship	101-8
Explicitly Deleting from the Database	101-9
Understanding the Order in Which Objects Are Deleted	101-9

102 Using Advanced Unit of Work API

Registering and Unregistering Objects	102-1
Creating and Registering an Object in One Step	102-2
Using registerNewObject	102-2
Registering a New Object with registerNewObject	102-2
Associating New Objects With One Another	102-3
Using registerAllObjects	102-4
Using Registration and Existence Checking	102-5
Check Database	102-5
Assume Existence	102-5
Assume Nonexistence	102-5
Working with Aggregates.....	102-6
Unregistering Working Clones	102-6
Declaring Read-Only Classes	102-6
Configuring Read-Only Classes for a Single Unit of Work	102-6
Configuring Default Read-Only Classes	102-7
Read-Only Descriptors	102-7
Writing Changes Before Commit Time	102-7
Using Conforming Queries and Descriptors	102-8
Guidelines for Using Conforming	102-8
Use Only In-Memory Queries that Support Conforming.....	102-8
Consider How Conforming Affects Database Results	102-9
If You Want an Object in the Conformed Results, Register it in the Unit of Work.....	102-9
Using Conforming Queries.....	102-10
Using Conforming Descriptors	102-11
Conforming Query Alternatives	102-11
Using Unit of Work Method writeChanges Instead of Conforming.....	102-11
Using Unit of Work Properties Instead of Conforming	102-12
Merging Changes in Working Copy Clones	102-12
Resuming a Unit of Work After Commit	102-14
Reverting a Unit of Work	102-14
Using a Nested or Parallel Unit of Work	102-14
Parallel Unit of Work.....	102-15
Nested Unit of Work.....	102-15
Using a Unit of Work With Custom SQL	102-15
Controlling the Order of Delete Operations	102-16
Using the Unit of Work setShouldPerformDeletesFirst Method	102-16
Using the Descriptor addConstraintDependencies Method	102-16
Using deleteAllObjects Without addConstraintDependencies	102-16
Using deleteAllObjects with addConstraintDependencies	102-17
Using Optimistic Read Locking With forceUpdateToVersionField	102-17
Forcing a Check of the Optimistic Read Lock.....	102-17
Forcing a Version Field Update	102-18
Disabling forceUpdateToVersionField	102-20
Implementing User and Date Auditing with the Unit of Work	102-20
Integrating the Unit of Work With an External Transaction Service	102-20
Acquiring a Unit of Work With an External Transaction Service.....	102-21

Using a Unit of Work When an External Transaction Exists	102-21
Using a Unit of Work When No External Transaction Exists.....	102-22
Using the Unit of Work to Handle External Transaction Timeouts and Exceptions	102-23
External Transaction Commit Timeouts.....	102-23
External Transaction Commit Exceptions	102-23
Integrating the Unit of Work With CMP	102-24
CMP Transaction Attribute.....	102-24
Local Transactions.....	102-25
Nondeferred Changes	102-25
Database Transaction Isolation Levels	102-25
General Factors Affecting Transaction Isolation Level.....	102-26
External Applications	102-26
TopLink Coordinated Cache	102-26
DatabaseLogin Method setTransactionIsolation	102-27
Reading Through the Write Connection	102-27
Managing Cache Access.....	102-28
CMP and External Transactions	102-29
Read Uncommitted Level	102-29
Read Committed Level.....	102-30
Repeatable Read Levels.....	102-30
Serializable Read Levels	102-30
Troubleshooting a Unit of Work.....	102-30
Avoiding the Use of Post-Commit Clones	102-30
Determining Whether or Not an Object Is the Cache Object.....	102-31
Dumping the Contents of a Unit of Work	102-32
Handling Exceptions	102-33
Exceptions at Commit Time	102-33
Exceptions During Conforming.....	102-33
Validating a Unit of Work.....	102-34
Validating the Unit of Work Before Commit Time.....	102-34
.....	103-1
.....	103-1
.....	103-1
.....	103-1
.....	103-1
.....	103-1
.....	103-1
.....	103-1
.....	103-1
.....	103-1
.....	103-1
.....	103-1
.....	103-1
.....	103-1
.....	103-1
.....	103-2
.....	103-2

.....	103-2
.....	103-2
.....	103-2
.....	103-2
.....	103-2
.....	103-2
.....	103-2
.....	103-2
.....	103-2
.....	103-2

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.
<i>italic</i>	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 VII

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 19, "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.

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 "[Understanding TopLink Sessions](#)" on page 75-1). The TopLink runtime uses this metadata in all persistence and data transformation operations.

This chapter includes information on:

- [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 19-2) for each class and each descriptor contains a mapping (see "[Mappings](#)" on page 19-2) 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 21-13.

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 77-2.

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 8-3
- ["project.xml File"](#) on page 8-2
- ["Understanding Projects"](#) on page 20-1.

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 19-2). Most descriptor information can be defined by TopLink Workbench, then read from the project XML file at run time.

See ["Understanding Descriptors"](#) on page 26-1 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 19-2).

See ["Understanding Mappings"](#) on page 33-1 for more information.

Part VIII

Projects

This part describes the TopLink artifact used to contain mapping and data source-specific information. It contains the following chapters.

- [Chapter 20, "Understanding Projects"](#)

This chapter describes each of the different TopLink project types and important project concepts.

- [Chapter 21, "Creating a Project"](#)

This chapter contains procedures for creating TopLink projects.

- [Chapter 22, "Configuring a Project"](#)

This chapter explains how to configure TopLink project options common to two or more project types.

- [Chapter 23, "Configuring a Relational Project"](#)

This chapter explains how to configure project options specific to a relational project.

- [Chapter 24, "Configuring an EIS Project"](#)

This chapter explains how to configure project options specific to an enterprise information system (EIS) project.

- [Chapter 25, "Configuring an XML Project"](#)

This chapter explains how to configure project options specific to an XML project.

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](#)

TopLink Project Types

[Table 20–1](#) lists the project types available in TopLink, classifies each as basic or advanced, and indicates how to create each.

Table 20–1 *TopLink Project Types*

Project Type	Description	Type	TopLink Workbench	Java
"Relational Projects" on page 20-6	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 20-7	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 20-9	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 21, "Creating a Project"](#)
- [Chapter 22, "Configuring a Project"](#)
- [Chapter 75, "Understanding TopLink Sessions"](#)

Project Concepts

This section describes concepts unique to TopLink projects, including the following 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 20–2](#) summarizes the relationship between project, descriptor, and mappings.

Table 20–2 *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 20-2) 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 20-4.

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 78, "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 20-3).

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 20-4).

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 84-2.

For information on configuring a session login, see ["Configuring a Session Login"](#) on page 77-4.

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 CMP container uses the

information in the deployment login whenever your business logic performs a persistence operation from within a CMP entity bean.

In this case, you do not configure a session login (see ["Non-CMP Session Role: Session Login"](#) on page 20-3). 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 20-4).

For information on creating a deployment login, see ["Configuring Development and Deployment Logins"](#) on page 23-6.

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 20-3) or the `project.xml` file (see ["CMP Deployment Role: Deployment Login"](#) on page 20-3).

For more information on creating a development login, see ["Configuring Development and Deployment Logins"](#) on page 23-6.

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 20-3.

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 84-3.

Projects and Sequencing

An essential part of maintaining object identity (see ["Cache Type and Object Identity"](#) on page 90-3) 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 20-14).
- For EIS projects, you typically use a returning policy (see ["Configuring Returning Policy"](#) on page 28-65) 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 20-5), 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 20-5)

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 84-8.

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 23-3).
- 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 86-4).

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 29-3).

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 20-7) and XML projects (see ["XML Projects"](#) on page 20-9).

For more information, see ["Understanding XML Namespaces"](#) on page 20-22.

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 20-6) 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 20-6), 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 36-4), 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 ["Understanding TopLink Queries"](#) on page 96-1).

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 20-3](#) describes the components of a relational project for a relational database.

Table 20-3 Components of a Relational Project for a Relational Database

Component	Supported Types
Data Source	For more information, see the following: ▪ "DatabaseLogin" on page 84-2 ▪ "Database Platforms" on page 84-3
Descriptors	For more information, see "Relational Descriptors" on page 26-11.
Mappings	For more information, see the following: ▪ Part X, "Mappings" ▪ Part XI, "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 21-1).
5. Create an object-relational descriptor for each Java class (see ["Creating an Object-Relational Descriptor"](#) on page 27-3).
6. Create object-relational mappings from each persistent field of each Java class to the corresponding object-types and object-type tables (see ["Configuring an Object-Relational Mapping"](#) on page 50-1).

[Table 20-4](#) describes the components of a relational project for an object-relational database.

Table 20-4 Components of a Relational Project for an Object-Relational Database

Component	Supported Types
Data Source	For more information, see the following: ■ "DatabaseLogin" on page 84-2 ■ "Database Platforms" on page 84-3
Descriptors	For more information, see "Object-Relational Descriptors" on page 26-11.
Mappings	For more information, see the following: ■ Part X, "Mappings" ■ Part XII, "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 Customer 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 20-6).
- 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 XIII, "EIS Mappings"](#).

You configure a TopLink EIS descriptor to use a particular EIS record format (see ["Configuring Record Format"](#) on page 31-5). 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 20-5.

[Table 20-5](#) describes the components of an EIS project.

Table 20-5 EIS Project Components

Component	Supported Types
Data Source	For more information, see the following: ■ "EISLogin" on page 84-3 ■ "EIS Platforms" on page 84-4
Descriptors	For more information, see "EIS Descriptors" on page 26-11.
Mappings	For more information, see the following: ■ Part X, "Mappings" ■ Part XIII, "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 20-9) 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 20-9).

In an EIS project, your EIS interactions (see ["Enterprise Information System \(EIS\) Interactions"](#) on page 96-18) can make full use of TopLink queries (see ["Understanding TopLink Queries"](#) on page 96-1). 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 98-24.

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 98-24.

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 20-10 and ["JAXB Validation"](#) on page 20-14). 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 20-5.

[Table 20–6](#) describes the components of an XML project.

Table 20–6 XML Project Components

Component	Supported Types
Data Source	None
Descriptors	For more information, see "XML Descriptors" on page 26-12.
Mappings	For more information, see the following: ■ Part X, "Mappings" ■ Part XIV, "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 21-6) both the required JAXB files (see ["Understanding JAXB-Specific Generated Files"](#) on page 20-10) and the TopLink files (["Understanding TopLink-Specific Generated Files"](#) on page 20-11) 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 20-13.

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 21-6.

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 5, "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:

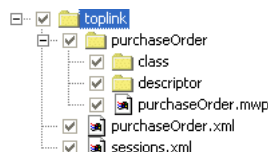
- [TopLink Sessions XML File](#)
- [TopLink Project XML File](#)
- [TopLink Workbench Project](#)
- [Typesafe Enumeration Converter Amendment Method DescriptorAfterLoads Class](#)

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 21-6).

The `toplink` subdirectory is organized as [Figure 20-1](#) illustrates.

Figure 20-1 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 20–1](#) shows a typical `sessions.xml` file where the context path is `examples.ox.model`.

Example 20–1 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 20-13).

TopLink Project XML File The project XML file is named the same as the XSD, but with a file extension of `xml`. In [Figure 20–1](#), 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 20–1](#) 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 20–1](#), 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 33-24).

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 20-12) 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 `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 20-10) and the TopLink runtime.
2. Acquire the session manager (see ["Acquiring the Session Manager"](#) on page 78-2).
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 78-3).

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 20-10) as the context path, as [Example 20-2](#) shows. This example assumes that you configure your application classpath to include your domain object class files.

The `JAXBContext` is thread-safe.

Example 20–2 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 20-13.

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 20–2](#) 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 20-2 Sequencing Elements in a Class Database Table

Table Name → VEHICLE_POOL

Sequencing Column →

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

You configure TopLink sequencing at the project or session level (see "[Configuring Sequencing at the Project Level](#)" on page 23-3 or "[Configuring Sequencing at the Session Level](#)" on page 86-4) 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 20-16).

You configure TopLink sequencing at the descriptor level (see "[Configuring Sequencing at the Descriptor Level](#)" on page 29-3) 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 CMP Entity Beans](#)

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 20-16) or native sequencing (see "[Native Sequencing With an Oracle Database Platform](#)" on page 20-18). 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 20-17). 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 23-3 or "[Configuring Sequencing at the Session Level](#)" on page 86-4.

Using Java, you can configure any sequence type that TopLink supports (see "[Sequencing Types](#)" on page 20-16). 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 20-18). 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 86-5.

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 20–3](#) 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 20–3 TopLink Table Sequence Table

SEQUENCE	
SEQ_NAME	SEQ_COUNT
SEQ_V_POOL	350
SEQ_MACHINERY	800
SEQ_PURCH_ORDER	1550
SEQ_WORK_ORDER	2400

The rows of the `SEQUENCE` table represent 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 29-3), 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 20-20).

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 23-3 or ["Configuring Sequencing at the Session Level"](#) on page 86-4.

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:

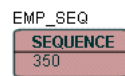
- You want to use an existing sequence table for sequencing.
- You do not want to use the default naming convention for the table and its columns.

Unary Table Sequencing

Although similar to table sequencing (see ["Table Sequencing"](#) on page 20-16), with unary table sequencing, you create a separate sequence table for each sequenced object in the project.

As [Figure 20-4](#) 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 20-4 *TopLink Unary Table Sequence Table*



EMP_SEQ
SEQUENCE 350

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 20-4](#) shows a unary table sequence for the `Employee` class. The `Employee` class descriptor is configured (see ["Configuring Sequencing at the Descriptor Level"](#) on page 29-3) 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 20-20).

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 20-17). 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 86-5).

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 86-7).

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 20-16).

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 29-6.

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 86-5).

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 20-16), 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 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 23-3 or ["Configuring Sequencing at the Session Level"](#) on page 86-4.

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.

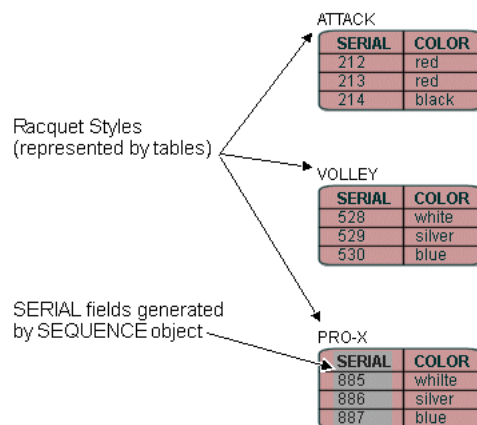
The key difference between this process and the process involved in table sequencing is that TopLink is unaware of the **INCREMENT** construct on the **SEQUENCE** object. TopLink sequencing and the Oracle **SEQUENCE** object operate in isolation. To avoid sequencing errors in the application, set the 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 20–5](#), 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 20–5 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 20-18.

You can configure native 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 23-3 or ["Configuring Sequencing at the Session Level"](#) on page 86-4.

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 20-3](#). 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 23-3 or ["Configuring Sequencing at the Session Level"](#) on page 86-4.

Sequencing With CMP Entity Beans

To implement sequencing for CMP entity beans, use a sequencing strategy that implements preallocation, such as table sequencing or Oracle native sequencing. Preallocation ensures that the 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 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 20-3](#) illustrates this call in a WebSphere integration.

Example 20–3 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 20–4](#) illustrates this call in a WebLogic integration.

Example 20–4 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 20-17) 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 20-7) and XML projects (see ["XML Projects"](#) on page 20-9).

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 4-38.

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 20–5](#) 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 20–5 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 20–6](#) shows an XML document that conforms to this XML schema.

Example 20–6 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 20–7](#) shows the Java code for a `Customer` class `XMLDescriptor` and XML mappings for its attributes to illustrate how this schema configuration affects the XPath expressions you specify for default root element and mappings (for more information, see ["Configuring an XML Descriptor"](#) on page 32-1 and ["Configuring an XML Mapping"](#) on page 66-1).

Example 20–7 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 20–8](#) 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 20–8 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 20–9](#) shows an XML document that conforms to this XML schema.

Example 20–9 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 20–10](#) 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 32-1 and ["Configuring an XML Mapping"](#) on page 66-1).

Example 20–10 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 20–11](#) 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 20–11 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 20–12](#) shows an XML document that conforms to this XML schema.

Example 20–12 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 20–13](#) shows the Java code for a `Customer` class `XMLDescriptor` and XML mappings for its attributes to illustrate how this schema configuration affects the XPath you specify for default root element and mappings (for more information, see ["Configuring an XML Descriptor"](#) on page 32-1 and ["Configuring an XML Mapping"](#) on page 66-1).

Example 20–13 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);

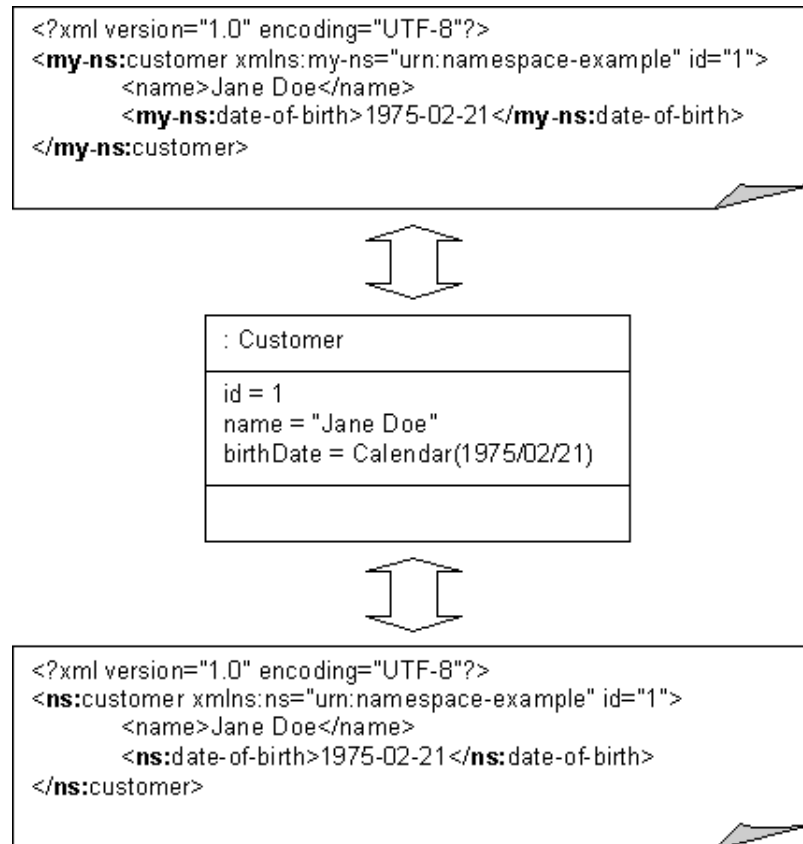
XMLDirectMapping nameMapping = new XMLDirectMapping();
nameMapping.setAttributeName("name");
nameMapping.setXPath("ns:name/text()");
customerDescriptor.addMapping(nameMapping);

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 20–6](#) shows, TopLink will use the namespace prefixes specified in the mapping when creating new documents.

Figure 20-6 *Namespaces in TopLink*

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 20-1.

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 21-2.

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 21-3 and Oracle TopLink API Reference.

You can use TopLink to create a project:

- If you have both an object and data model: see ["Creating a Project for an Existing Object and Data Model"](#) on page 21-5.
- If you have an object model but no data model yet: see ["Creating a Project From an Existing Object Model"](#) on page 21-5.
- If you have a data model but no object model yet: ["Creating a Project From an Existing Data Model"](#) on page 21-5.
- If you have an XML schema (XSD) document but no object model yet: see ["Creating an XML Project From an XML Schema"](#) on page 21-6.

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 21-2.

- If 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 21-9.
- If you are deploying an EJB 2.0 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 2.0 CMP EAR at Deployment Time"](#) on page 21-10.

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 21-13.

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 ["Integrating TopLink With an Application Server"](#) on page 7-1. To create a new project from JAXB, see ["TopLink Support for Java Architecture for XML Binding \(JAXB\)"](#) on page 20-10.

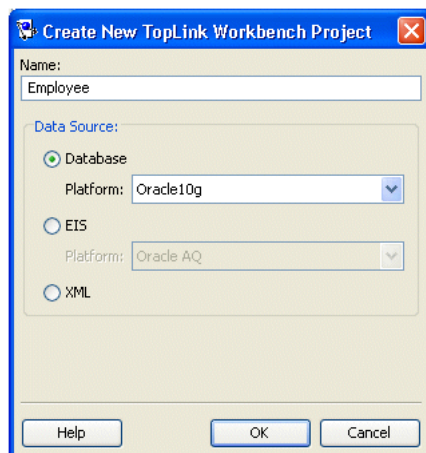
To create a new TopLink Workbench project, use this procedure:



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 21-1 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 <code>.mwp</code> 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 20-6 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 20-7 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 21-6). See "XML Projects" on page 20-9 for more information.

For more project information, continue with the following:

- Configure the project (see [Chapter 22, "Configuring a Project"](#)).
- Add mappings and descriptors (see [Chapter 26, "Understanding Descriptors"](#) and [Chapter 33, "Understanding Mappings"](#)).
- Export the project for use with the TopLink runtime (see ["Exporting Project Information"](#) on page 21-13).

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 21-1](#)).
2. Compile the project class.

Example 21-1 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 21-14.

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 21-2).
2. Configure the project classpath (see ["Configuring Project Classpath"](#) on page 22-3).
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 22, "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 21-2).
2. Configure the project classpath (see ["Configuring Project Classpath"](#) on page 22-3).
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 22, "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 21-2).
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 Entities and Descriptors From Database Tables"](#) on page 4-31
4. Configure project options (see [Chapter 22, "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 21-2.

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 21-6) or from the command line (see ["Using the Command Line"](#) on page 21-8). 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 5, "Using an Integrated Development Environment"](#).

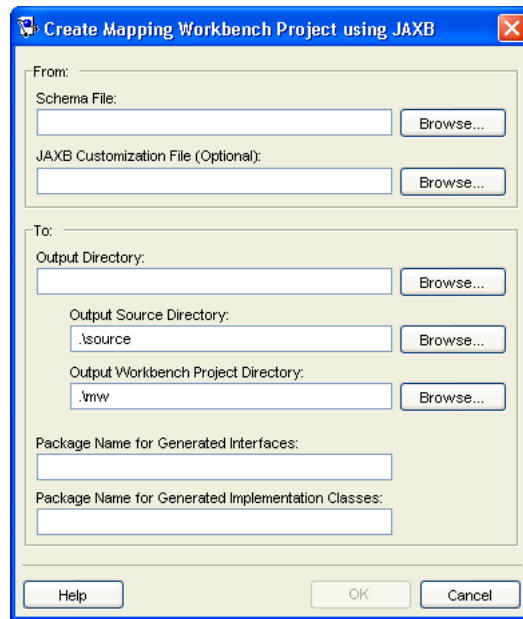
For more information, see the following:

- ["TopLink Support for Java Architecture for XML Binding \(JAXB\)"](#) on page 20-10
- ["Using TopLink JAXB Compiler Generated Files at Run Time"](#) on page 20-13

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 21-2 Create TopLink Workbench Project using JAXB Dialog Box

This illustration shows the **Create TopLink Workbench Project using JAXB** dialog box. The following table describes each field on the dialog box.

2. Complete each field on the Create TopLink Workbench Project using JAXB dialog box, and then click **OK**.

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 is 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.

Field	Description
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 <code>jaxbderived.<schema name></code> is used where <code><schema name></code> is the name of the schema specified by the Schema File field.
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 <code>jaxbderived.<schema name></code> is used where <code><schema name></code> 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 20-10) and TopLink-specific files (see "[Understanding TopLink-Specific Generated Files](#)" on page 20-11).

Optionally, open the generated TopLink Workbench project (see "[TopLink Workbench Project](#)" on page 20-12), 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 5, "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 `tljxb.cmd` or `tljxb.sh` file (located in the `<ORACLE_HOME>/toplink/bin` directory) as follows:

1. Using a text editor, edit the `tljxb.cmd` or `tljxb.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 21-2](#) or [Example 21-3](#) and edit them to set your proxy host (name or IP address) and port:

Example 21-2 Proxy Settings in `tljxb.cmd`

```
@REM set JVM_ARGS=%JVM_ARGS% -DproxySet=true -Dhttp.proxyHost= -Dhttp.proxyPort=
```

Example 21-3 Proxy Settings in `tljxb.sh`

```
# JVM_ARGS="{JVM_ARGS}" -DproxySet=true -Dhttp.proxyHost= -Dhttp.proxyPort=
```

2. Execute the `tljxb.cmd` or `tljxb.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 20-10) and TopLink-specific files (see "[Understanding TopLink-Specific Generated Files](#)" on page 20-11).

[Example 21-4](#) illustrates how to generate an object model from a schema using the Toplink JAXB compiler. [Table 21-1](#) lists the compiler arguments.

Example 21–4 Generating an Object Model from a Schema with tljxb.cmd

```
tljxb.cmd -sourceDir ./app/src -workbenchDir ./app/mw -schema purchaseOrder.xsd
-targetPkg examples.ox.model.if -implClassPkg examples.ox.model.impl
```

Table 21–1 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 <code>source</code> in the specified output directory.	Yes
-workbenchDir	The path to the directory into which the TopLink Workbench project files are written. Default: directory named <code>mw</code> 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 <code>-implClassPkg</code> argument. Default: a package name of <code>jaxbderived.<schema name></code> where <code><schema name></code> is the name of the schema specified by the <code>-schema</code> 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 <code>-targetPkg</code> 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

- Optionally, open the generated TopLink Workbench project (see ["TopLink Workbench Project"](#) on page 20-12) 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 5, "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 EJB 2.0 CMP relational projects only.

For more information, see ["Persistence Manager Migration"](#) on page 7-4.

Creating a Project From an OC4J EJB 2.0 CMP EAR at Deployment Time

For an EJB 2.0 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 the EJB 2.0 CMP relational projects deployed to OC4J configured to use TopLink as the default persistence manager.

For more information, see ["Default Mapping in EJB 2.0 or 3.0 CMP Projects Using OC4J at Run Time"](#) on page 33-4.

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 22, "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.

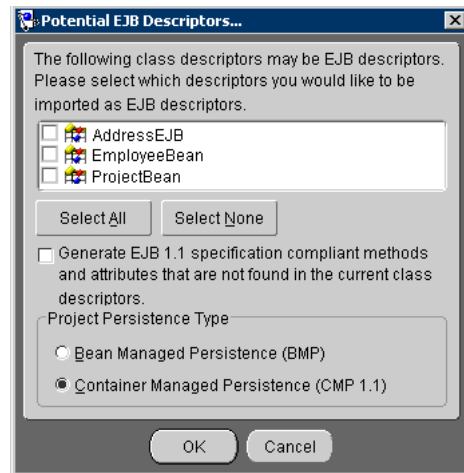


1. Click **Open Project** on the toolbar. The Choose a File dialog box appears. You can also open a project by choosing **File > Open** from the menu.

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.

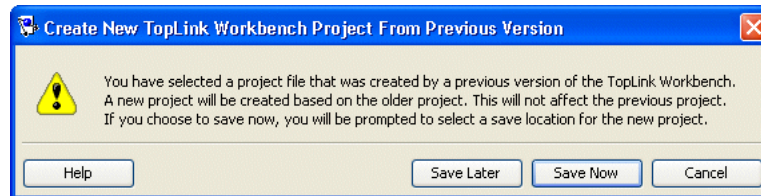
If you open a TopLink Workbench version 3.*n* project that contains EJB information, the Potential EJB Descriptors dialog box appears.

Figure 21–3 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 21–4 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.

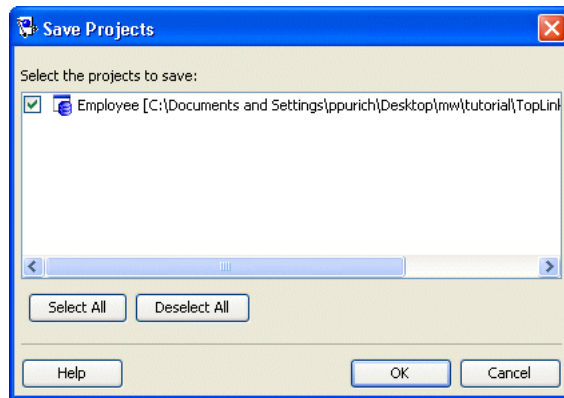
To save your project(s), use this procedure:



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.

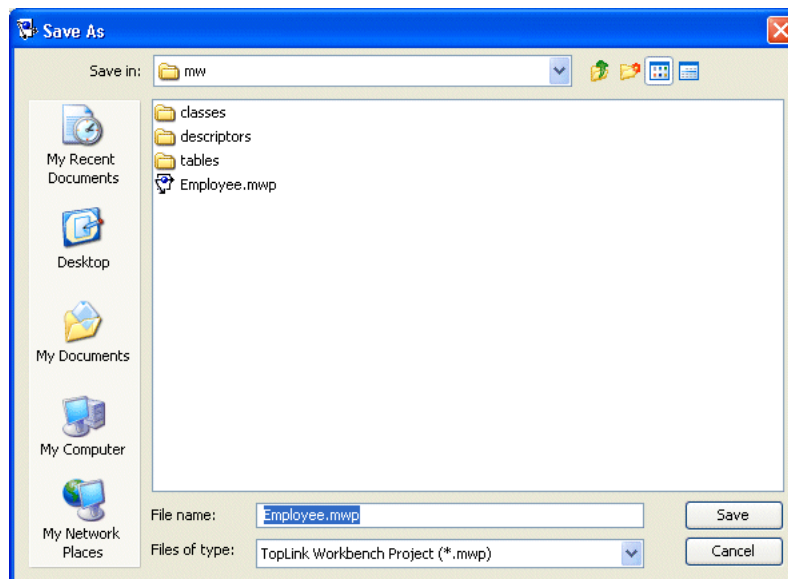
Figure 21–5 Save Projects Dialog Box

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 21–6 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](#)

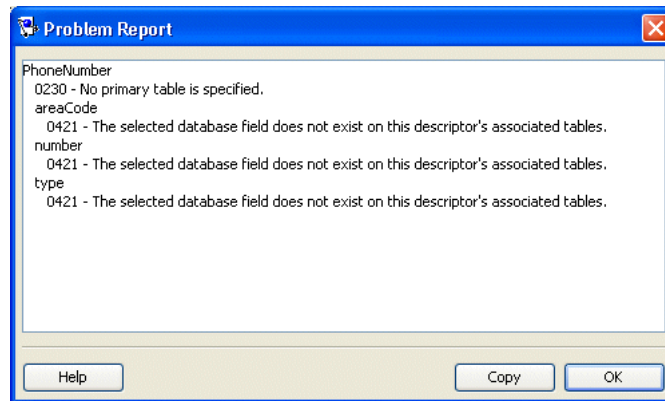


With [TopLink Workbench](#)" on page 4-3), but lets you easily copy and paste the errors into documents or messages. To generate the project status report, use this procedure:

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 21–7 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 85-2.

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 22, "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 10g Release 3 (10.1.3), see ["Configuring Deprecated Direct Mappings"](#) on page 22-12.

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 22-1) 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 27-6 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 22-1) 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 10g Release 3 (10.1.3), see ["Configuring Deprecated Direct Mappings"](#) on page 22-12.

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 22, "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 persistence, 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 21-2](#) describes how fields in the `ejb-jar.xml` file correspond to specific functions in TopLink Workbench.

Table 21-2 *ejb-jar.xml Fields and TopLink Workbench*

ejb-jar.xml Fields	TopLink Workbench
primaryKey	Bean attribute mapped to the primary key in the database table (see "Configuring Primary Keys" on page 28-2).

Table 21-2 (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 28-44).
abstract-schema-name	Descriptor Alias field (see "Configuring Descriptor Alias" on page 28-7).
cmp-field	Direct (non-relationship) attributes on the Descriptor Info tab (see "Configuring Common Descriptor Options" on page 28-1).
cmp-version	Persistence Type field on the General tab (see "Configuring Persistence Type" on page 22-5). The persistence-type is set to container.
query	Queries listed in Queries tab (see "Configuring Named Queries at the Descriptor Level" on page 28-10). 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 XI, "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.

Configuring a Project

This chapter describes how to configure TopLink projects.

[Table 22-1](#) 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 22-1 *Configuring TopLink Projects*

If you are creating...	See also...
Relational Projects	Chapter 23, "Configuring a Relational Project"
EIS Projects	Chapter 24, "Configuring an EIS Project"
XML Projects	Chapter 25, "Configuring an XML Project"

[Table 22-2](#) lists the configurable options shared by two or more TopLink project types.

For more information, see the following:

- ["Project Creation Overview"](#) on page 21-1
- ["Understanding Projects"](#) on page 20-1

Configuring Common Project Options

[Table 22-2](#) 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 22-1](#).

Table 22-2 *Common Project Options*

Option	Type	TopLink Workbench	Java
"Configuring Project Save Location" on page 22-2	Basic	✓	
"Configuring Project Classpath" on page 22-3	Basic	✓	
"Configuring Mapped Field Access at the Project Level" on page 22-4	Basic	✓	
"Configuring Persistence Type" on page 22-5	Basic	✓	
"Configuring Default Descriptor Advanced Properties" on page 22-7	Advanced	✓	
"Configuring Existence Checking at the Project Level" on page 22-8	Advanced	✓	✓
"Configuring Project Deployment XML Options" on page 22-10	Advanced	✓	
"Configuring Model Java Source Code Options" on page 22-11	Advanced	✓	

Table 22–2 (Cont.) Common Project Options

Option	Type	TopLink Workbench	Java
"Configuring Deprecated Direct Mappings" on page 22-12	Advanced	✓	
"Configuring Cache Type and Size at the Project Level" on page 22-13	Advanced	✓	✓
"Configuring Cache Isolation at the Project Level" on page 22-16	Advanced	✓	✓
"Configuring Cache Coordination Change Propagation at the Project Level" on page 22-17	Advanced	✓	✓
"Configuring Cache Expiration at the Project Level" on page 22-19	Advanced	✓	✓
"Configuring Project Comments" on page 22-21	Advanced	✓	

Configuring Project Save Location

You can configure a project save location only when using TopLink Workbench.

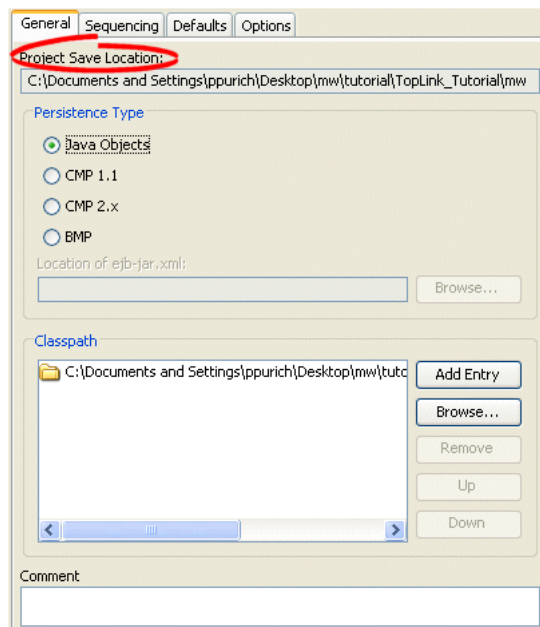
Table 22–3 summarizes which projects support a project save location.

Table 22–3 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 22–1 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 21-11 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 22-4](#) summarizes which projects support project classpath configuration.

Table 22-4 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).

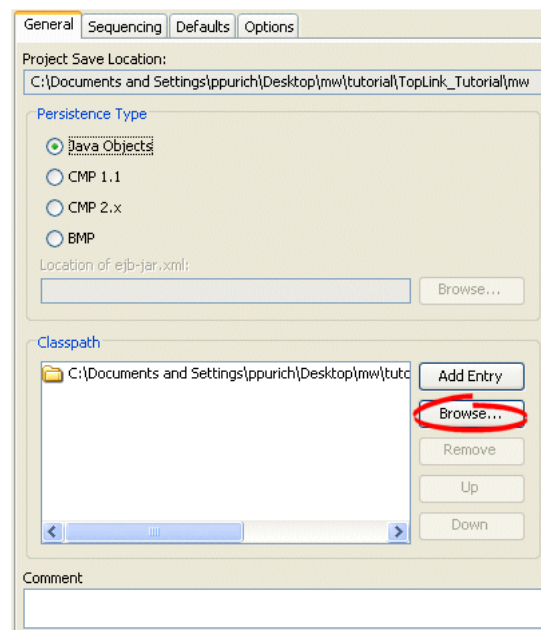
After you configure the project classpath, you can use TopLink Workbench to import classes into your project (see ["Importing and Updating Classes"](#) on page 4-50).

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 22-2 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 22–5](#) summarizes which projects support mapped field access configuration.

Table 22–5 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.

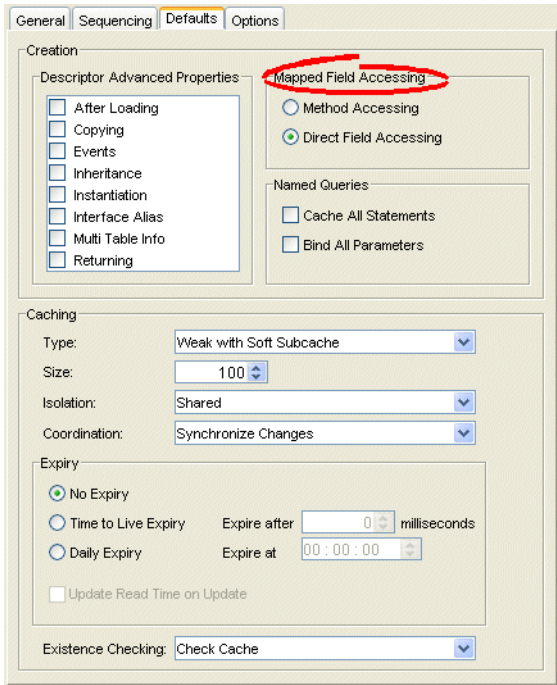
You can also configure mapped field access at the mapping level to override this project-level configuration on a mapping-by-mapping basis. For more information, see ["Configuring Method Accessing"](#) on page 35-14.

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 22–3 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 CMP, or entity beans with bean-managed persistence (BMP).

Table 22–6 summarizes which projects support a persistence type configuration.

Table 22–6 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 19-2 for more information on TopLink descriptors.

For EJB projects, you can specify an `ejb-jar.xml` file from which TopLink will read and write the necessary persistence information. You use the `ejb-jar.xml` file to map the virtual fields of the EJB 2.0 CMP entity beans (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 28-44).

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 ejb-jar.xml** from the menu.

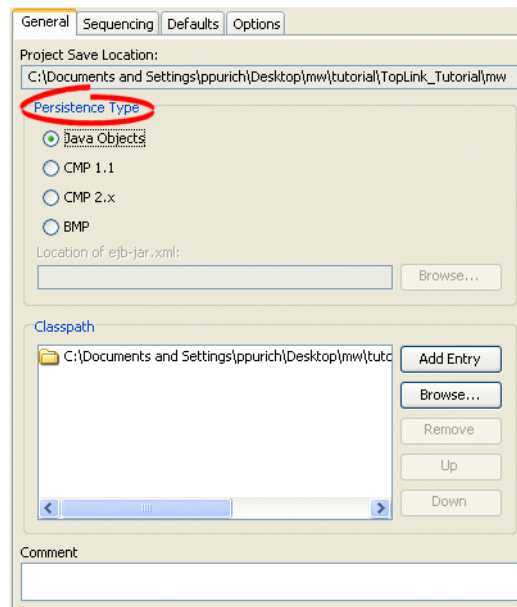
For more information on creating and using deployment files such as the `ejb-jar.xml` file, see the following:

- ["Understanding TopLink Deployment File Creation"](#) on page 8-1
- ["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

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 22–4 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.x , CMP 2.x , or BMP . For EJB projects, specify the location of the <code>ejb-jar.xml</code> file. Note: This field does not apply to XML projects.
Location of ejb-jar.xml	Specify the location of the <code>ejb-jar.xml</code> file for this project. "Working With the ejb-jar.xml File" on page 21-15 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 28, "Configuring a Descriptor"](#)).

[Table 22–7](#) summarizes which projects support default descriptor advanced property configuration.

Table 22–7 Project Support for Default Descriptor Advanced Properties

Descriptor	Using TopLink Workbench	Using Java
Relational Projects	✓	
EIS Projects	✓	

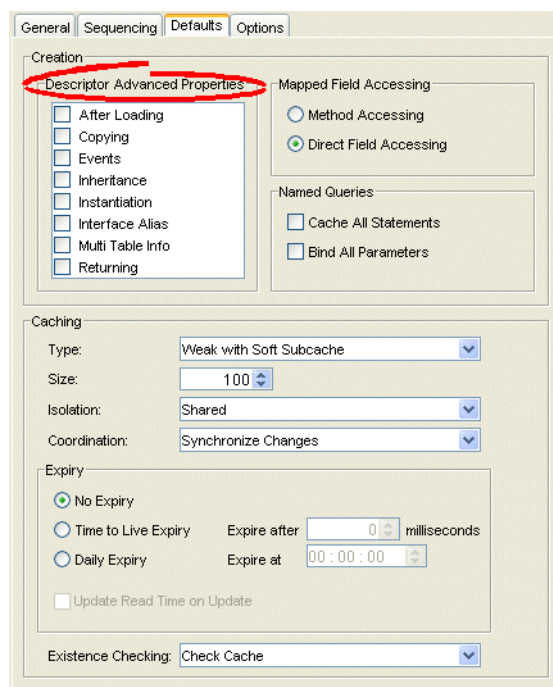
Table 22–7 (Cont.) Project Support for Default Descriptor Advanced Properties

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 22–5 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 28, "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 22–8](#) summarizes which projects support existence checking configuration.

Table 22–8 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 28-42.

For more information see:

- ["Cache Type and Object Identity"](#) on page 90-3
- ["Queries and the Cache"](#) on page 96-29
- ["Using Registration and Existence Checking"](#) on page 102-5.

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 22–6 Defaults Tab, Existence Checking Options

The screenshot shows the 'Defaults' tab in the TopLink Workbench. The 'Existence Checking' section at the bottom is circled in red. It includes a dropdown menu set to 'Check Cache'. Other visible options include 'Creation' (Descriptor Advanced Properties, Mapped Field Accessing, Named Queries), 'Caching' (Type, Size, Isolation, Coordination), and 'Expiry' (No Expiry, Time to Live Expiry, Daily Expiry).

3. Complete the Existence Checking options on the tab.

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 102-5.
Assume Existence	Always assume objects exist: always do an update (never do an insert). For more information, see "Assume Existence" on page 102-5.
Assume Nonexistence	Always assume objects do not exist: always do an insert (never do an update). For more information, see "Assume Nonexistence" on page 102-5.

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 22-2), and will contain folders for each generated package.

[Table 22-9](#) summarizes which projects support deployment XML options.

Table 22-9 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 22–7 Options Tab, Project Deployment XML Options

The screenshot shows the 'Options' tab with the following sections:

- Deployment and Java Source Code Generation**
 - Project Java Source**: Class Name, Root Directory, Browse...
 - Project Deployment XML** (circled in red): File Name (EmployeeProject.xml), Directory (config), Browse...
 - Table Creator Java Source**: Class Name, Root Directory, Browse...
 - Model Java Source**: Root Directory, Browse...
- Table Generation**: Default Primary Key (ID), Primary Key Search Pattern (*ID), ☐ Generate Deprecated Direct Mappings

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 22-2), and will contain folders for each generated package.

[Table 22–10](#) summarizes which projects support model Java source code options.

Table 22–10 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 22–8 Options Tab, Model Java Source options

The screenshot shows the 'Options' tab in the TopLink Workbench. The 'Model Java Source' section is highlighted with a red circle. It contains a 'Root Directory' text field and a 'Browse...' button. Above it are sections for 'Project Java Source' and 'Project Deployment XML'. Below it is the 'Table Generation' section with 'Default Primary Key' and 'Primary Key Search Pattern' fields. At the bottom is a checkbox labeled 'Generate Deprecated Direct Mappings'.

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 27-6.

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 22–11](#) summarizes which projects support deprecated direct mapping options.

Table 22–11 Project Support for Deprecated Direct Mapping Options

Descriptor	Using TopLink Workbench	Using Java
Relational Projects	✓	
EIS Projects		

Table 22-11 (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 22-9 Options Tab, Generate Deprecated Direct Mappings Option

The screenshot shows the 'Options' tab in the TopLink Workbench. The 'Generate Deprecated Direct Mappings' checkbox is checked and highlighted with a red circle. The other options in the tab are:

- Project Java Source:** Class Name: [], Root Directory: [] Browse...
- Project Deployment XML:** File Name: EmployeeProject.xml, Directory: config Browse...
- Table Creator Java Source:** Class Name: [], Root Directory: [] Browse...
- Model Java Source:** Root Directory: [] Browse...
- Table Generation:** Default Primary Key: ID, Primary Key Search Pattern: *ID

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 22-12 summarizes which projects support identity map configuration.

Table 22–12 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 28-35.

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 are affected.

For detailed information on caching and object identity, and the recommended settings to maximize TopLink performance, see ["Cache Type and Object Identity"](#) on page 90-3.

For more information about the cache, see ["Understanding the Cache"](#) on page 90-1.

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 22–10 Defaults Tab, Cache Identity Map Options

The screenshot shows the 'Defaults' tab of a configuration window. The 'Caching' section is highlighted with red circles around the 'Type' and 'Size' fields. The 'Type' dropdown is set to 'Weak with Soft Subcache' and the 'Size' spinner is set to '100'. Other options include 'Isolation' (Shared), 'Coordination' (Synchronize Changes), 'Expiry' (No Expiry), and 'Existence Checking' (Check Cache).

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 <i>n</i> elements in soft space, anything after that in weak space (see "Soft and Hard Cache Weak Identity Maps" on page 90-4) ■ Weak with Hard Subcache (HardCacheWeakIdentityMap)—cache first <i>n</i> elements in soft space, anything after that in hard space (see "Soft and Hard Cache Weak Identity Maps" on page 90-4) ■ Weak (WeakIdentityMap)—cache everything in weak space (see "Weak Identity Map" on page 90-3) ■ Full (FullIdentityMap)—cache everything permanently (see "Full Identity Map" on page 90-3) ■ None (NoIdentityMap)—cache nothing (see "No Identity Map" on page 90-4) For more information, see "Cache Type and Object Identity" on page 90-3. 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 <i>maximum</i> number of objects stored in the identity map. ■ When using Full or Weak, the size indicates the <i>starting</i> size of the identity map.

Configuring Cache Isolation at the Project Level

If you plan to use isolated sessions ("[Cache Isolation](#)" on page 90-9), 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 22-12](#) summarizes which projects support cache isolation configuration.

Table 22-13 *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 28-37.

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 are 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 22–11 Defaults Tab, Cache Isolation Options

The screenshot shows the 'Defaults' tab of a configuration window. The 'Creation' section includes 'Descriptor Advanced Properties' (a list of checkboxes: After Loading, Copying, Events, Inheritance, Instantiation, Interface Alias, Multi Table Info, Returning) and 'Mapped Field Accessing' (radio buttons for Method Accessing and Direct Field Accessing, with Direct Field Accessing selected). The 'Named Queries' section has checkboxes for 'Cache All Statements' and 'Bind All Parameters'. The 'Caching' section includes 'Type' (Weak with Soft Subcache), 'Size' (100), 'Isolation' (highlighted with a red circle), and 'Coordination' (Synchronize Changes). The 'Expiry' section has radio buttons for 'No Expiry' (selected), 'Time to Live Expiry' (with 'Expire after' set to 0 milliseconds), and 'Daily Expiry' (with 'Expire at' set to 00:00:00). There is also an 'Update Read Time on Update' checkbox. The 'Existence Checking' dropdown is set to 'Check Cache'.

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 90-9.
- **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 90-10), you can configure how and under what conditions a coordinated cache propagates changes.

[Table 22–12](#) summarizes which projects support cache coordination change propagation configuration.

Table 22–14 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 28-38.

To complete your coordinated cache configuration, see ["Configuring a Coordinated Cache"](#) on page 91-1.

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 are affected.

Using TopLink Workbench

To specify the coordinated cache change propagation options, use this procedure:

1. Select the project object in the **Navigators**.
2. Select the **Defaults** tab in the **Editor**. The Defaults tab appears.

Figure 22–12 Defaults Tab, Coordination Options

The screenshot shows the 'Defaults' tab in the TopLink Workbench. The 'Coordination' field under the 'Caching' section is highlighted with a red circle. The 'Caching' section includes fields for 'Type' (Weak with Soft Subcache), 'Size' (100), 'Isolation' (Shared), and 'Coordination' (Synchronize Changes). The 'Expiry' section has radio buttons for 'No Expiry' (selected), 'Time to Live Expiry', and 'Daily Expiry'. The 'Existence Checking' field is set to 'Check Cache'.

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 101-7) or garbage-collected when using a weak identity map (see ["Configuring Cache Type and Size at the Project Level"](#) on page 22-13). 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 28-27).

[Table 22-15](#) summarizes which projects support cache invalidation configuration.

Table 22-15 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 28-40) or query level (see ["Configuring Cache Expiration at the Query Level"](#) on page 99-22).

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 28-38.

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 90-8.

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 22–13 Defaults Tab, Cache Expiry Options

The screenshot shows the 'Defaults' tab in the TopLink Workbench. The 'Cache Expiry' section is highlighted with a red circle. It contains the following options:

- Creation:**
 - Descriptor Advanced Properties:** After Loading, Copying, Events, Inheritance, Instantiation, Interface Alias, Multi Table Info, Returning.
 - Mapped Field Accessing:** Method Accessing, Direct Field Accessing (selected).
 - Named Queries:** Cache All Statements, Bind All Parameters.
- Caching:**
 - Type: Weak with Soft Subcache
 - Size: 100
 - Isolation: Shared
 - Coordination: Synchronize Changes
 - Expiry:**
 - No Expiry (selected)
 - Time to Live Expiry: Expire after 0 milliseconds
 - Daily Expiry: Expire at 00:00:00
 - Update Read Time on Update: ☐
 - Existence Checking: Check Cache

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 wish: 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 22–16 summarizes which projects support this option.

Table 22–16 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 **Navigators**.
2. Select the **General** tab in the **Editor**. The General tab appears.

Figure 22–14 General Tab, Comments Options

The screenshot shows the 'General' tab of a configuration window. It includes tabs for 'General', 'Sequencing', 'Defaults', and 'Options'. The 'Project Save Location' is at the top. Below it, the 'Persistence Type' section has radio buttons for 'Java Objects' (selected), 'CMP 1.1', 'CMP 2.x', and 'BMP'. A 'Location of ejb-jar.xml' field with a 'Browse...' button is next. The 'Classpath' section contains a list box and buttons for 'Add Entry', 'Browse...', 'Remove', 'Up', and 'Down'. At the bottom, a 'Comment' field is highlighted with a red circle.

3. Enter descriptive text information in the **Comment** field.

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 21-1
- ["Relational Projects"](#) on page 20-6

Relational Project Configuration Overview

In addition to the configurable options described here, you must also configure the base class options described in [Table 22-2](#) on page 22-1.

[Table 23-1](#) lists the configurable options for relational projects.

Table 23-1 Configurable Options for Relational Projects

Option	Type	TopLink Workbench	Java
"Configuring Project Save Location" on page 22-2	Basic	✓	
"Configuring Persistence Type" on page 22-5	Basic	✓	
"Configuring Project Classpath" on page 22-3	Basic	✓	
"Configuring Project Comments" on page 22-21	Basic	✓	
"Configuring Mapped Field Access at the Project Level" on page 22-4	Basic	✓	✓
"Configuring Default Descriptor Advanced Properties" on page 22-7	Basic	✓	✓
"Configuring Existence Checking at the Project Level" on page 22-8	Basic	✓	✓
"Configuring Project Deployment XML Options" on page 22-10	Basic	✓	✓
"Configuring Model Java Source Code Options" on page 22-11	Basic	✓	✓
"Configuring Deprecated Direct Mappings" on page 22-12	Basic	✓	✓
"Configuring Relational Database Platform at the Project Level" on page 23-2	Basic	✓	✓
"Configuring Sequencing at the Project Level" on page 23-3	Basic	✓	✓
"Configuring Login Information" on page 23-5	Basic	✓	✓
"Configuring Development and Deployment Logins" on page 23-6	Basic	✓	
"Configuring Cache Type and Size at the Project Level" on page 22-13	Advanced	✓	✓
"Configuring Cache Isolation at the Project Level" on page 22-16	Advanced	✓	✓

Table 23–1 (Cont.) Configurable Options for Relational Projects

Option	Type	TopLink Workbench	Java
"Configuring Cache Coordination Change Propagation at the Project Level" on page 22-17	Advanced	✓	✓
"Configuring Cache Expiration at the Project Level" on page 22-19	Advanced	✓	
"Configuring Named Query Parameterized SQL and Statement Caching at the Project Level" on page 23-7	Advanced	✓	
"Configuring Table Generation Options" on page 23-9	Advanced	✓	
"Configuring Table Creator Java Source Options" on page 23-10	Advanced	✓	
"Configuring Project Java Source Code Options" on page 23-11	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 23-7.

Configuring Relational Database Platform at the Project Level

For each relational project, you must specify the database platform (such as Oracle Database 10g). This platform configuration is overridden by the session login, if configured.

For more information, see

- ["Configuring a Relational Database Platform at the Session Level"](#) on page 86-1
- ["Data Source Platform Types"](#) on page 84-3

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 23–1 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 84-3.

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 86-4).

Using TopLink Workbench (see ["Using TopLink Workbench"](#) on page 23-3), you can configure table sequencing (see ["Table Sequencing"](#) on page 20-16) and native sequencing (["Native Sequencing With an Oracle Database Platform"](#) on page 20-18 and ["Native Sequencing With a Non-Oracle Database Platform"](#) on page 20-19) and you can configure a preallocation size that applies to all sequences (see ["Sequencing and Preallocation Size"](#) on page 20-20).

Using Java (see ["Using Java"](#) on page 23-4), you can configure any sequence type that TopLink supports (["Sequencing Types"](#) on page 20-16). 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 20-17). 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 29-3).

For more information about sequencing, see ["Understanding Sequencing in Relational Projects"](#) on page 20-14.

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 23–2 Sequencing Tab

The screenshot shows a configuration window with four tabs: General, Sequencing, Defaults, and Options. The 'Sequencing' tab is active. It contains a 'Preallocation Size' field with a value of 50. Below this are three radio buttons: 'Default Sequence Table', 'Native Sequencing', and 'Custom Sequence Table'. The 'Custom Sequence Table' option is selected. Under this option, there are three dropdown menus: 'Name' (set to SEQUENCE), 'Name Field' (set to SEQUENCE.SEQ_NAME (VARCHAR2)), and 'Counter Field' (set to SEQUENCE.SEQ_COUNT (NUMBER)).

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 20-20). 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 20-16) 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 20-18 or "Native Sequencing With a Non-Oracle Database Platform" on page 20-19) 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 20-16) 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 23–1](#) shows.

Example 23–1 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 separate 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 23-2).

After you create a login (see ["Configuring Login Information"](#) on page 23-5) and specify it as a development login (see ["Configuring Development and Deployment Logins"](#) on page 23-6), you can log in to a database instance (see ["Logging in to the Database"](#) on page 23-7).

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 23-3 Database Property Sheet, Database Login Fields

The screenshot shows the 'Database (Oracle10g)' property sheet. The 'Defined Logins' section is circled in red and contains a list with 'RelationalLogin' selected. To the right of this list are buttons for 'Change...', 'Add...', 'Rename...', and 'Remove'. Below the list are fields for 'Driver Class' (oracle.jdbc.driver.OracleDriver), 'URL' (jdbc:oracle:thin:@TL5VRDB5:1521:TOPLINK), 'User Name' (ppurich), 'Password' (masked with asterisks), a checked 'Save Password' checkbox, 'Development Login' (RelationalLogin), and 'Deployment Login' (RelationalLogin).

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 21-13), TopLink Workbench writes the database password (if applicable) using JCE encryption (when using JDK 1.4). Refer to <i>Oracle TopLink Getting Started Guide</i> 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 85-2.

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 23-5. 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 20-4), you must define a login (see ["Configuring Login Information"](#) on page 23-5) 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 23-7.

CMP Deployment Role

If you are creating a CMP project (see ["CMP Deployment Role: Deployment Login"](#) on page 20-3), you may define a run-time login (see ["Configuring Login Information"](#) on page 23-5) 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 20-3), 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 84-2).

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 23-4 Database Property Sheet, Development and Deployment Login Options

Database (Oracle10g)

Database Platform: Oracle10g [Change...]

Defined Logins: RelationalLogin [Add...][Rename...][Remove]

Driver Class: oracle.jdbc.driver.OracleDriver [v]

URL: jdbc:oracle:thin:@TL\$VRDB5:1521:TOPLINK [v]

User Name: ppurich

Password: *****

☒ Save Password

Development Login: RelationalLogin [v]

Deployment Login: RelationalLogin [v]

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 23-7.
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 23-5) and specify it as a development login (see "[Configuring Development and Deployment Logins](#)" on page 23-6), 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 settings apply only to named queries (see ["Named Queries"](#) on page 96-15)—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 86-9). 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 23–2](#) summarizes which projects support query statement caching and binding configuration.

Table 23–2 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 28-22).

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 23–5 Defaults Tab, Named Queries Options

The screenshot shows the 'Defaults' tab in the TopLink Workbench. The 'Named Queries' section is highlighted with a red circle. It contains two checkboxes: 'Cache All Statements' and 'Bind All Parameters'. The 'Caching' section has a 'Type' dropdown set to 'Weak with Soft Subcache', a 'Size' spinner set to '100', an 'Isolation' dropdown set to 'Shared', and a 'Coordination' dropdown set to 'Synchronize Changes'. The 'Expiry' section has three radio buttons: 'No Expiry' (selected), 'Time to Live Expiry', and 'Daily Expiry'. The 'Time to Live Expiry' section has an 'Expire after' spinner set to '0' milliseconds. The 'Daily Expiry' section has an 'Expire at' spinner set to '00:00:00'. There is also an 'Update Read Time on Update' checkbox. The 'Existence Checking' dropdown is set to 'Check Cache'.

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 23–6 Options Tab, Table Generation Options

The screenshot shows the 'Options' tab in the TopLink Workbench interface. The 'Table Generation' section is highlighted with a red circle. It contains the following fields and options:

- Project Java Source:** Class Name (empty), Root Directory (empty) with a 'Browse...' button.
- Project Deployment XML:** File Name (EmployeeProject.xml), Directory (config) with a 'Browse...' button.
- Table Creator Java Source:** Class Name (empty), Root Directory (empty) with a 'Browse...' button.
- Model Java Source:** Root Directory (empty) with a 'Browse...' button.
- Table Generation:** Default Primary Key (ID), Primary Key Search Pattern (*ID).
- ☐ Generate Deprecated Direct Mappings

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 23–7 Options Tab, Table Creator Java Source Options

The screenshot shows the 'Options' tab in the TopLink Workbench interface. It contains several sections for configuring Java source code generation:

- Project Java Source:** Includes fields for 'Class Name' and 'Root Directory' with a 'Browse...' button.
- Project Deployment XML:** Includes fields for 'File Name' (set to 'EmployeeProject.xml') and 'Directory' (set to 'config') with a 'Browse...' button.
- Table Creator Java Source:** This section is circled in red. It includes fields for 'Class Name' and 'Root Directory' with a 'Browse...' button.
- Model Java Source:** Includes a 'Root Directory' field with a 'Browse...' button.
- Table Generation:** Includes fields for 'Default Primary Key' (set to 'ID') and 'Primary Key Search Pattern' (set to '*ID').
- Generate Depreciated Direct Mappings:** A checkbox that is currently unchecked.

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 21-14.

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 23–8 Options Tab, Project Java Source Options

General Sequencing Defaults **Options**

Deployment and Java Source Code Generation

Project Java Source

Class Name:

Root Directory: Browse...

Project Deployment XML

File Name: .xml

Directory: Browse...

Table Creator Java Source

Class Name:

Root Directory: Browse...

Model Java Source

Root Directory: Browse...

Table Generation

Default Primary Key:

Primary Key Search Pattern:

☐ Generate Depreciated Direct Mappings

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.

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 21-1
- ["EIS Projects"](#) on page 20-7

EIS Project Configuration Overview

lists the configurable options for EIS projects.

Table 24-1 Configurable Options for EIS Projects

Option	Type	TopLink Workbench	Java
"Configuring Project Save Location" on page 22-2	Basic	✓	
"Configuring Persistence Type" on page 22-5	Basic	✓	
"Configuring Project Classpath" on page 22-3	Basic	✓	
"Configuring Project Comments" on page 22-21	Advanced	✓	
"Configuring Mapped Field Access at the Project Level" on page 22-4	Basic	✓	
"Configuring Default Descriptor Advanced Properties" on page 22-7	Advanced	✓	
"Configuring Existence Checking at the Project Level" on page 22-8	Advanced	✓	✓
"Configuring Project Deployment XML Options" on page 22-10	Advanced	✓	
"Configuring Model Java Source Code Options" on page 22-11	Advanced	✓	
"Configuring EIS Data Source Platform at the Project Level" on page 24-2	Basic	✓	✓
"Configuring EIS Connection Specification Options at the Project Level" on page 24-2	Basic	✓	✓
"Configuring Model Java Source Code Options" on page 22-11	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 4-38	Advanced	✓	✓
"Configuring Cache Type and Size at the Project Level" on page 22-13	Advanced	✓	✓

Table 24–1 (Cont.) Configurable Options for EIS Projects

Option	Type	TopLink Workbench	Java
"Configuring Cache Isolation at the Project Level" on page 22-16	Advanced	✓	✓
"Configuring Cache Coordination Change Propagation at the Project Level" on page 22-17	Advanced	✓	✓
"Configuring Cache Expiration at the Project Level" on page 22-19	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.

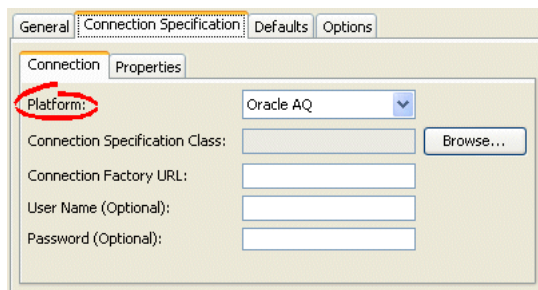
For more information, see the following:

- "Configuring an EIS Data Source Platform at the Session Level" on page 87-1
- "Data Source Platform Types" on page 84-3

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 24–1 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 84-3.

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 87-2.

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 24–2 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 <code>oracle.toplink.eis.aq.AQPlatform</code> , use <code>oracle.toplink.eis.aq.AQEISConnectionSpec</code>). For more information on platform configuration, see "Configuring an EIS Data Source Platform at the Session Level" on page 87-1.
Connection Factory URL	Specify the appropriate connection factory URL (as a J2EE JNDI name) for the selected Connection Specification Class (example: <code>java:comp/env/eis/attuntiy</code>).
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 21-13), TopLink Workbench writes the database password (if applicable) using JCE encryption (when using JDK 1.4). Refer to <i>Oracle TopLink Getting Started Guide</i> 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 85-2.

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 21-1
- ["XML Projects"](#) on page 20-9

XML Project Configuration Overview

[Table 25–1](#) lists the configurable options for XML projects.

Table 25–1 Configurable Options for XML Projects

Option	Type	TopLink Workbench	Java
"Configuring Project Save Location" on page 22-2	Basic	✓	
"Configuring Project Classpath" on page 22-3	Basic	✓	
"Configuring Project Comments" on page 22-21	Advanced	✓	
"Configuring Mapped Field Access at the Project Level" on page 22-4			
"Configuring Project Deployment XML Options" on page 22-10	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 4-38	Advanced	✓	✓
"Configuring Model Java Source Code Options" on page 22-11	Advanced	✓	
"Configuring Default Descriptor Advanced Properties" on page 22-7	Advanced	✓	
"Configuring Mapped Field Access at the Project Level" on page 22-4	Advanced	✓	✓

Part IX

Descriptors

This part describes the TopLink artifact used to describe persistent objects. It contains the following chapters.

- [Chapter 26, "Understanding Descriptors"](#)
This chapter describes each of the different TopLink descriptor types and important descriptor concepts.
- [Chapter 27, "Creating a Descriptor"](#)
This chapter contains procedures for creating TopLink descriptors.
- [Chapter 28, "Configuring a Descriptor"](#)
This chapter explains how to configure TopLink descriptor options common to two or more descriptor types.
- [Chapter 29, "Configuring a Relational Descriptor"](#)
This chapter explains how to configure descriptor options specific to a relational descriptor.
- [Chapter 30, "Configuring an Object-Relational Descriptor"](#)
This chapter explains how to configure descriptor options specific to an object-relational descriptor.
- [Chapter 31, "Configuring an EIS Descriptor"](#)
This chapter explains how to configure descriptor options specific to an EIS descriptor.
- [Chapter 32, "Configuring an XML Descriptor"](#)
This chapter explains how to configure descriptor options specific to an XML descriptor.

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 26–1](#) lists the descriptor types you use to describe the classes in your object model and classifies them as basic or advanced.

Table 26–1 *TopLink Descriptor Types*

Descriptor Type	Description	Type	TopLink Workbench	Java
"Relational Descriptors" on page 26-11	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 26-11	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 26-11	Describes Java objects that you map to an EIS data source by way of a J2C adapter.	Basic	✓	✓
"XML Descriptors" on page 26-12	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 27, "Creating a Descriptor"](#)
- [Chapter 28, "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 26–2](#) summarizes the relationship between project, descriptor, and mappings.

Table 26–2 *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 26-12.

Descriptors and EJB

You can use descriptors to describe the characteristics of CMP and BMP entity beans.

When mapping EJB, 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 EJB type (such as CMP or BMP) and import the `ejb-jar.xml` file for the beans into the TopLink Workbench project.

For CMP 2.0 projects, you use the `ejb-jar.xml` file to define the bean's mapped attributes. A CMP bean descriptor contains a CMP policy used to configure CMP-specific options.

For CMP 3.0 projects, you can use annotations to define the bean's mapped attributes.

This section describes:

- [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 CMP 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 28-44.

Creating a New Entity Bean and `ejbCreate` / `ejbPostCreate` Methods

When you create a new entity bean, by default, the EJB life cycle can be thought of as follows:

1. `ejbCreate` method:

After the insert, the CMP 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 CMP container updates container-managed relationship (CMR) fields. The CMP 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 28-44.

Inheritance

Although the EJB 2.0 specification does not define inheritance, TopLink allows you to configure inheritance for CMP descriptors, with some reservations.

For more information, see ["Inheritance and EJB"](#) on page 26-17.

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:

- ["Configuring Fetch Groups"](#) on page 28-77
- ["Fetch Groups and Object Level Read Queries"](#) on page 96-12

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 28-78.

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 27-2).

This lets you configure an aggregate mapping (see ["Configuring a Relational Aggregate Object Mapping"](#) on page 47-1) 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.

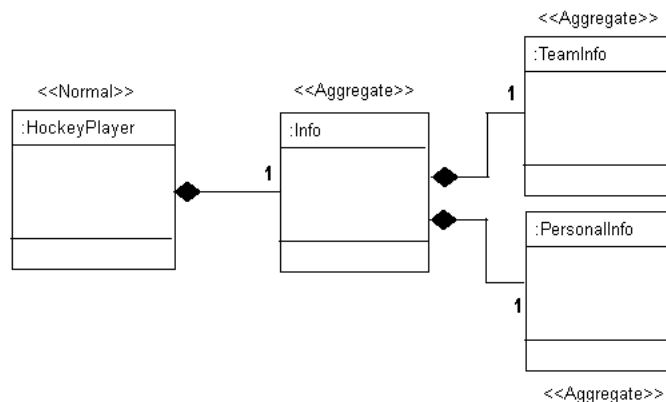
When working with aggregate relational descriptors, consider the following:

- [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 29-11.

Relational Aggregates and Nesting TopLink supports nested aggregates. In [Figure 26-1](#) 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 26-1 *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 CMP application 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 26-1](#) shows the classes from [Figure 26-1](#) 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 26-1 *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 26-3), 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 26-3).

For more information on using relational aggregates and EJB 3.0, see ["Relational Aggregates and Nesting"](#) on page 26-6.

Root and Composite Descriptors in EIS Projects

In an EIS project, you can designate the descriptor as a composite (see ["EIS Composite Descriptors"](#) on page 27-5).

The type of EIS mapping you wish to create will determine whether you configure an EIS descriptor as a composite or root (see ["Composite and Reference EIS Mappings"](#) on page 56-4).

For more information, see ["Configuring an EIS Descriptor as a Root or Composite Type"](#) on page 31-8.

You cannot configure EJB information for an EIS descriptor designated as an composite (see ["Descriptors and EJB"](#) on page 26-3).

You can configure inheritance for an EIS descriptor designated as a composite (see ["Descriptors and Inheritance"](#) on page 26-3), 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 26-3).

Descriptor Event Manager

In relational and EIS projects, TopLink raises various instances of `DescriptorEvent` (see [Table 28–25](#) and [Table 28–27](#)) 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 28-57
- ["Configuring a Descriptor Event Listener as an Event Handler"](#) on page 28-60

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 28-10)

- custom default queries for basic persistence operations (see ["Configuring Default Query Implementations"](#) on page 96-22)
- additional join expressions (see ["Configuring Additional Join Expressions"](#) on page 96-23)

For more information on using the query manager, see ["Descriptor Query Manager Queries"](#) on page 96-22.

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 20-4.

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 23-3). 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 86-4).

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 28-2), with its own sequence (see ["Configuring Sequencing at the Descriptor Level"](#) on page 29-3).

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 26-17 and ["Optimistic Field Locking Policies"](#) on page 26-20).
- 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 26-21).
- 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 26-17
- ["Configuring Locking Policy"](#) on page 28-62

Default Root Element

You must configure EIS root descriptors (["Configuring Default Root Element"](#) on page 31-3) and XML descriptors (["Configuring Default Root Element"](#) on page 32-5) with a default root element so that the TopLink runtime knows the data source data type associated with the class the descriptor describes.

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 26–2](#).

Example 26–2 Customer and Address Classes

Class: `Customer`

Default Root: `customer`

Attributes and Mappings:

<code>name:String</code>	Direct Mapping to	<code>name/text()</code>
<code>billingAddress:Address</code>	Composite Object Mapping to	<code>billing-address</code>
<code>shippingAddress:Address</code>	Composite Object Mapping to	<code>shipping-address</code>

Class: `Address`

Default Root: `address`

Attributes and Mappings:

<code>street:String</code>	Direct Mapping to	<code>street/text()</code>
<code>city:String</code>	Direct Mapping to	<code>city/text()</code>

These classes correspond to the XML schema shown in [Example 26–3](#).

Example 26–3 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 20-6).

Using relational descriptors in a relational project, you can configure relational mappings (see ["Relational Mapping Types"](#) on page 36-1).

For more information, see the following:

- ["Creating a Relational Descriptor"](#) on page 27-1.
- [Chapter 29, "Configuring a Relational Descriptor"](#)

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 49-1).

For more information, see the following:

- ["Creating an Object-Relational Descriptor"](#) on page 27-3
- [Chapter 30, "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 56-1) 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 27-4

- [Chapter 31, "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 65-1), in memory, to XML elements defined by an XSD.

For more information, see the following:

- ["Creating an XML Descriptor"](#) on page 27-5
- [Chapter 32, "Configuring an XML Descriptor"](#)

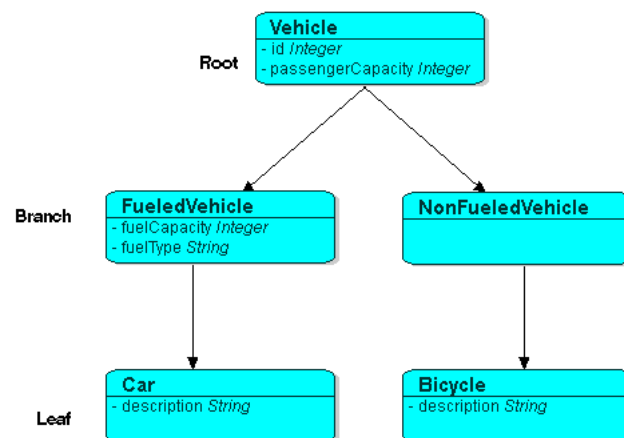
Understanding Descriptors and Inheritance

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 26–2](#) 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 26–2 Example Inheritance Hierarchy

Java 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 26–2](#) 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 26-2](#) 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 26-2](#) 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 28-50.

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 28-49.

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 48, "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 26–3](#) illustrates the class indicator dictionary for the `Vehicle` class's subclasses as shown in [Figure 26–2](#).

Table 26–3 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 26-15).

For example, [Table 26–4](#) 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 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 26-13). To resolve this issue, you could use the class extraction method shown in [Example 26–4](#).

Table 26–4 *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 26–4 *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 26-14), 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 28-53).

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 26-15) or to multiple tables (["Multitable Inheritance"](#) on page 26-16). 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 26–2](#) can share the same table, as in [Figure 26–3](#). 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 26–3](#) shows, this approach uses a class indicator field. For more information, see ["Specifying a Class Indicator"](#) on page 26-13.

Figure 26–3 Inheritance Using a Superclass Table with Optional Fields

VEHICLE table

ID	PASS_CAP	VHCL_TYPE	FUEL_CAP	FUEL_TYPE	CAR_DESCR	BICYCLE_DESCR
1	1	B				Mountain Bike
2	3	V				
3	8	F	20	Diesel		
4	5	C	15	Unleaded	Toyota Camry	

Class Indicator Field:
V = Vehicle
F = Fueled Vehicle
N = Non-Fueled Vehicle
C = Car
B = Bicycle

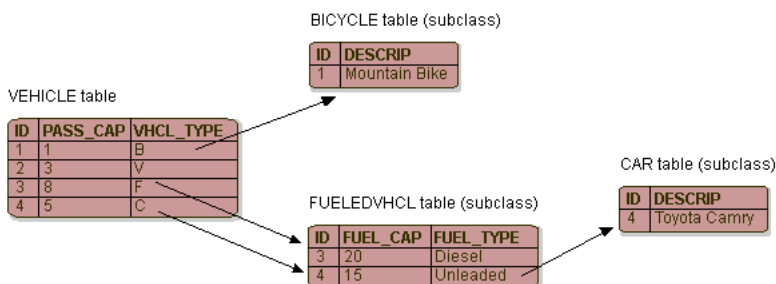
Multitable Inheritance

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 26–2](#), 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 26-13) to determine the class of object to create, then uses the descriptor for that class to read from the subclass tables.

[Figure 26–4](#) 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 26–4 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 28-47.

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 26-5).

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 current EJB specification contains only general information regarding inheritance. You should fully understand this information before implementing EJB inheritance. Be aware of the fact that the next EJB specification 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 28-62.

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 26-21).

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 35-2).

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 26-18.

When using optimistic version locking with the unit of work, consider ["Using Optimistic Read Locking With forceUpdateToVersionField"](#) on page 102-17.

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 102-17) 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 28-65).

After you enable optimistic version locking cascading, when a privately owned child object is modified, `TopLink` will traverse the privately owned foreign reference mappings, updating all the parent objects back to the root.

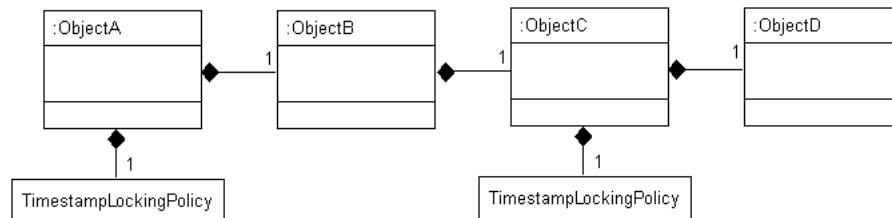
Optimistic version locking cascading is only applied if the child object is registered in a unit of work.

`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 26–5](#)

Figure 26–5 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 102-14) 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 100-8).

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 persistence type of CMP (see "[Configuring Persistence Type](#)" on page 22-5) and with descriptors that have EJB information (see "[Configuring a Descriptor With EJB Information](#)" on page 28-44).

You can also use pessimistic locking (but not a pessimistic locking policy) at the query level (see "[Configuring Named Query Options](#)" on page 28-22).

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 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 28-63).

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.

You 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 26-21).

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 26–5](#) illustrates the descriptor types that derive from class `oracle.toplink.descriptors.ClassDescriptor`.

Example 26–5 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
```

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 27-1](#)
- ["Creating an Object-Relational Descriptor" on page 27-3](#)
- ["Creating an EIS Descriptor" on page 27-4](#)
- ["Creating an XML Descriptor" on page 27-5](#)

After you create a descriptor, you must configure its various options (see [Chapter 28, "Configuring a Descriptor"](#)) and use it to define mappings.

For complete information on the various types of mapping that TopLink supports, see [Chapter 33, "Understanding Mappings"](#) and [Chapter 34, "Creating a Mapping"](#).

For complete information on the various types of descriptor that TopLink supports, see ["Descriptor Types" on page 26-23](#).

Creating a Relational Descriptor

You can create a relational descriptor using TopLink Workbench (see ["Using TopLink Workbench" on page 27-2](#)) or Java code (see ["Using Java" on page 27-2](#)). Oracle recommends that you use TopLink Workbench to create and manage your relational descriptors.

For more information, see ["Relational Descriptors" on page 26-11](#).

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 22-3), 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 27-2). 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 ["Configuring a Relational Aggregate Collection Mapping"](#) on page 44-1 and ["Configuring a Relational Aggregate Object Mapping"](#) on page 47-1), you must designate the target object's descriptor as an aggregate (see ["Configuring a Relational Descriptor as a Class or Aggregate Type"](#) on page 29-11).

Relational Interface Descriptors



If you add an interface to a relational project (see ["Configuring Project Classpath"](#) on page 22-3), 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 28-33). A read query on the interface results in reading one or more of its implementors.

Using Java

[Example 27-1](#) shows how to create a relational descriptor using Java code.

Example 27-1 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 29-2) and field mappings (see ["Understanding Mappings"](#) on page 33-1).

To allow a relational descriptor to participate in an aggregate collection mapping (see ["Aggregate Collection Mapping"](#) on page 36-10), use `ClassDescriptor` method `descriptorIsAggregateCollection`. For a `RelationalDescriptor` configured for use with an aggregate collection mapping, you do not define primary keys (see ["Configuring Primary Keys"](#) on page 28-2) and an associated table (see ["Configuring Associated Tables"](#) on page 29-2), 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 29-10).

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 26-11.

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 27-2](#) shows an `Employee` object that is mapped to an Oracle Database using its object-relational features.

Example 27-2 Employee Class

```
public class Employee {
    Long id;
    String firstName;
    String lastName;

    ...
}
```

[Example 27-3](#) 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 27-3 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 27-4](#) shows how to code an object-relational descriptor in Java to describe the object-relational database type `Employee_t`.

Example 27-4 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 30-1.

For more information on the object-relational mappings that TopLink supports, see [Chapter 49, "Understanding Object-Relational Mappings"](#).

Creating an EIS Descriptor

You can create an EIS descriptor using TopLink Workbench (see ["Using TopLink Workbench"](#) on page 27-4) or Java code (see ["Using Java"](#) on page 27-5). Oracle recommends that you use TopLink Workbench to create and manage your EIS descriptors.

For more information, see ["EIS Descriptors"](#) on page 26-11.

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 31-8). 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 98-24). 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 22-3), 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 27-5).

Using Java

[Example 27-5](#) shows how to create a relational descriptor using Java code.

Example 27-5 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 27-5) or Java code (see ["Using Java"](#) on page 27-5). Oracle recommends that you use TopLink Workbench to create and manage your XML descriptors.

For more information, see ["XML Descriptors"](#) on page 26-12.

Using TopLink Workbench



When you add a class to an XML project (see ["Configuring Project Classpath"](#) on page 22-3), TopLink Workbench creates an XML descriptor for the class.

An XML descriptor is always a composite type.

Using Java

[Example 27-6](#) shows how to create a relational descriptor using Java code.

Example 27-6 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 75-12.
- Review the project status report. For more information, see ["Generating the Project Status Report"](#) on page 21-12.

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 21-14.

Configuring a Descriptor

This chapter describes how to configure TopLink descriptors.

[Table 28-1](#) 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 28-1 *Configuring TopLink Descriptor*

If you are creating...	See...
Relational Descriptors	Chapter 29, "Configuring a Relational Descriptor"
Object-Relational Descriptors	Chapter 30, "Configuring an Object-Relational Descriptor"
EIS Descriptors	Chapter 31, "Configuring an EIS Descriptor"
XML Descriptors	Chapter 32, "Configuring an XML Descriptor"

[Table 28-2](#) lists the configurable options shared by two or more TopLink descriptor types.

For more information, see the following:

- ["Descriptor Creation Overview"](#) on page 27-1
- ["Understanding Descriptors"](#) on page 26-1

Configuring Common Descriptor Options

[Table 28-2](#) 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 28-1](#).

Table 28-2 *Common Descriptor Options*

Option	Type	TopLink Workbench	Java
"Configuring Primary Keys" on page 28-2	Basic	✓	✓
"Configuring Read-Only Descriptors" on page 28-4	Basic	✓	✓
"Configuring Unit of Work Conforming at the Descriptor Level" on page 28-6	Advanced	✓	✓
"Configuring Descriptor Alias" on page 28-7	Advanced	✓	✓
"Configuring Descriptor Comments" on page 28-77	Advanced	✓	
"Configuring Classes" on page 4-41	Basic	✓	

Table 28-2 (Cont.) Common Descriptor Options

Option	Type	TopLink Workbench	Java
"Configuring Named Queries at the Descriptor Level" on page 28-10	Advanced	✓	✓
"Configuring Query Timeout at the Descriptor Level" on page 28-26	Advanced	✓	✓
"Configuring Cache Refreshing" on page 28-27	Advanced	✓	✓
"Configuring Query Keys" on page 28-29	Advanced	✓	✓
"Configuring Interface Query Keys" on page 28-33	Advanced	✓	✓
"Configuring Cache Type and Size at the Descriptor Level" on page 28-35	Advanced	✓	✓
"Configuring Cache Isolation at the Descriptor Level" on page 28-37	Advanced	✓	✓
"Configuring Cache Coordination Change Propagation at the Descriptor Level" on page 28-38	Advanced	✓	✓
"Configuring Cache Expiration at the Descriptor Level" on page 28-40	Advanced	✓	✓
"Configuring Cache Existence Checking at the Descriptor Level" on page 28-42	Advanced	✓	✓
"Configuring a Descriptor With EJB Information" on page 28-44	Advanced	✓	✓
"Configuring Reading Subclasses on Queries" on page 28-47	Advanced	✓	✓
"Configuring Inheritance for a Child (Branch or Leaf) Class Descriptor" on page 28-49	Advanced	✓	✓
"Configuring Inheritance for a Parent (Root) Descriptor" on page 28-50	Advanced	✓	✓
"Configuring Inheritance Expressions for a Parent (Root) Class Descriptor" on page 28-53	Advanced	✓	✓
"Configuring Inherited Attribute Mapping in a Subclass" on page 28-56	Advanced	✓	✓
"Configuring a Domain Object Method as an Event Handler" on page 28-57	Advanced	✓	✓
"Configuring a Descriptor Event Listener as an Event Handler" on page 28-60	Advanced	✓	✓
"Configuring Locking Policy" on page 28-62	Advanced	✓	✓
"Configuring Returning Policy" on page 28-65	Advanced	✓	✓
"Configuring Instantiation Policy" on page 28-68	Advanced	✓	✓
"Configuring Copy Policy" on page 28-69	Advanced	✓	✓
"Configuring Change Policy" on page 28-70	Advanced		✓
"Configuring a History Policy" on page 28-73	Advanced		✓
"Configuring Wrapper Policy" on page 28-75	Advanced		✓
"Configuring Fetch Groups" on page 28-77	Advanced		✓
"Configuring Amendment Methods" on page 28-78	Advanced	✓	
"Configuring a Mapping" on page 35-1	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 28–3](#), you must configure a primary key and you must ensure that your class contains one or more persistent fields suitable for this purpose.

[Table 28–3](#) summarizes which descriptors support primary keys.

Table 28–3 *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 27-2).

² EIS root descriptors only (see ["EIS Root Descriptors"](#) on page 27-5).

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 29-2).

For an EIS root descriptor (see ["Configuring an EIS Descriptor as a Root or Composite Type"](#) on page 31-8), 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 31-2).

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 28–1 *Descriptor Info Tab, Primary Key Options*

The screenshot shows the 'Descriptor Info' tab in the TopLink Workbench. The 'Associated Table' is set to 'ADDRESS'. Under the 'Primary Keys' section, 'ADDRESS.ADDRESS_ID' is listed. Below this, there are checkboxes for 'Use Sequencing', 'Read-Only', and 'Conform Results in Unit of Work'. The 'Descriptor Alias' is set to 'Address'. A 'Comment' field is also present at the bottom.

Use this table to enter data in each 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 29-2). - 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 31-2). For more information on choosing an element or attribute, see "Choosing a Schema Context" on page 31-3. 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 100, "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 28-5.

[Table 28-4](#) summarizes which descriptors support read-only configuration.

Table 28–4 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 27-2).

² EIS root descriptors only (see "EIS Root Descriptors" on page 27-5)

Note: Relational aggregate and EIS composite descriptors get their read-only setting from their owner.

Using Read-Only Entity Beans

TopLink can declare a CMP entity bean 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 28–2 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 28–5 summarizes which descriptors support descriptor level unit of work conforming.

Table 28–5 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 27-2).

² EIS root descriptors only (see "EIS Root Descriptors" on page 27-5)

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 102-8) and make sure that conforming is actually necessary.

For examples, see the following:

- ["Using Conforming Queries and Descriptors"](#) on page 102-8
- ["Conforming Query Alternatives"](#) on page 102-11

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 28-3 *Descriptor Info Tab, Conform Results in Unit of Work Option*

The screenshot shows the 'Descriptor Info' tab in the TopLink Workbench. The 'Associated Table' dropdown is set to 'ADDRESS'. Under the 'Primary Keys' section, 'ADDRESS.ADDRESS_ID' is entered in the text field, with 'Add...' and 'Remove' buttons. The 'Use Sequencing' section is collapsed. The 'Read-Only' checkbox is unchecked. The 'Conform Results in Unit of Work' checkbox is checked and circled in red. The 'Descriptor Alias' text field contains 'Address'. There is also a 'Comment' text area at the bottom.

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 102-8.

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 CMP entity bean. 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 22-5) to use EJB.

[Table 28–6](#) summarizes which descriptors support descriptor alias configuration.

Table 28–6 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 27-5).

For more information, see the following:

- ["ejb-jar.xml File"](#) on page 8-5
- ["EJB Finders"](#) on page 96-23

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 28–4 Descriptor Info Tab, Descriptor Alias Field

The screenshot shows the 'Descriptor Info' tab in the TopLink Workbench. The 'Associated Table' is set to 'ADDRESS'. Under 'Primary Keys', 'ADDRESS.ADDRESS_ID' is listed. There are checkboxes for 'Use Sequencing', 'Read-Only', and 'Conform Results in Unit of Work'. The 'Descriptor Alias' field is highlighted with a red circle and contains the text 'Address'. Below it is a 'Comment' 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 wish: 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 28–7 summarizes which descriptors support descriptor comment configuration.

Table 28–7 *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 28–5 *Descriptor Info Tab, Comment Field*

The screenshot shows the 'Descriptor Info' tab in the TopLink Workbench. The 'Associated Table' is set to 'ADDRESS'. Under 'Primary Keys', 'ADDRESS.ADDRESS_ID' is listed. There are checkboxes for 'Use Sequencing', 'Read-Only', and 'Conform Results in Unit of Work'. The 'Descriptor Alias' is 'Address'. The 'Comment' field is highlighted with a red circle and contains a large empty text area for input.

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 77-21).

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 28-10).

Using Java, you can create named queries for all query types and store them in a descriptor's `DescriptorQueryManager` (see ["Using Java"](#) on page 28-25).

[Table 28-4](#) summarizes which descriptors support named query configuration.

Table 28-8 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 27-2).

² EIS root descriptors only (see ["EIS Root Descriptors"](#) on page 27-5).

For 2.0 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 21-15 for more information.

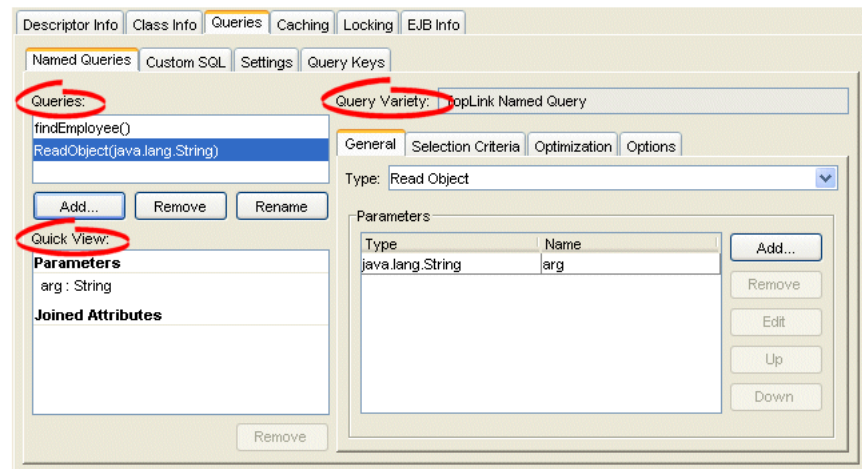
After you create a named query, you can execute it by name and class on the TopLink session (see ["Using Named Queries"](#) on page 98-17).

For more information about named queries, see ["Named Queries"](#) on page 96-15.

Using TopLink Workbench

To create a named query, use this procedure

1. In the **Navigators**, 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 28–6 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 28-12). - 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 28-12).
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:

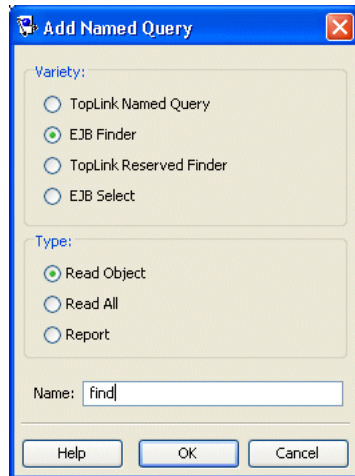
- **General**–See ["Configuring Named Query Type and Parameters"](#) on page 28-13.
- **Selection Criteria**–See ["Configuring Named Query Selection Criteria"](#) on page 28-14.
- **Order**–This tab appears for ReadAllQuery queries only. See ["Configuring Read All Query Order"](#) on page 28-15.
- **Optimization**–See ["Configuring Named Query Optimization"](#) on page 28-16.
- **Attributes**–This tab appears for ReportQuery queries only. See ["Configuring Named Query Attributes"](#) on page 28-17.
- **Group/Order**–This tab appears for ReportQuery queries only. ["Configuring Named Query Group/Order Options"](#) on page 28-19.
- **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 28-20.

- **Options**—See ["Configuring Named Query Options"](#) on page 28-22.

Adding Named Queries

Use this dialog box to create a new named query.

Figure 28–7 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 98-17).
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 96-24.
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 <code>ejbSelect</code> . The TopLink runtime executes this query whenever the <code>ejbSelect</code> method is called.

Field	Description
Type	Select the type of query: ■ ReadObject (ReadObjectQuery) ■ ReadAll (ReadAllQuery) ■ Report¹ (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 96-24. ■ If Variety is set to EJB Select, the name must be ejbSelect.

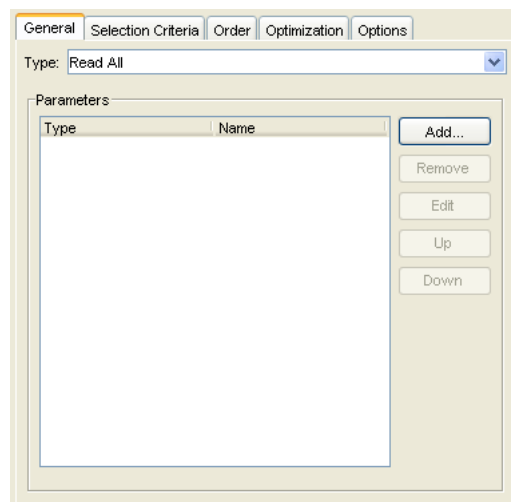
¹ Relational descriptors only.

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 28–8 *Named Queries, General Tab*



Use the following information to complete each field on this tab:

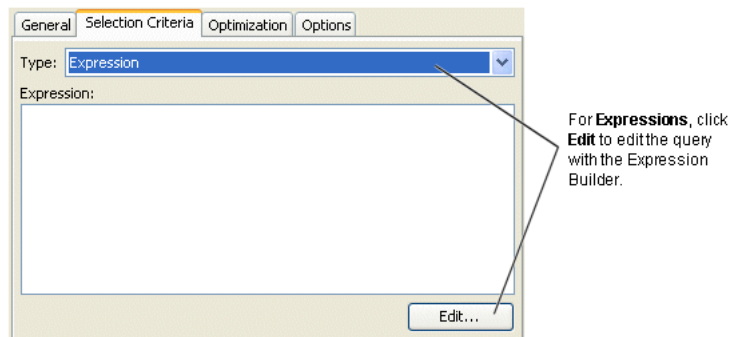
Field	Description
Type	Select the type of query from the list. You can create any of the following query types: ■ ReadAllQuery ■ ReadObjectQuery ■ ReportQuery¹ To create other types of query, you must use Java (see "Using Java" on page 28-25). When you change the type of an existing query, TopLink Workbench 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 28–9 *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.

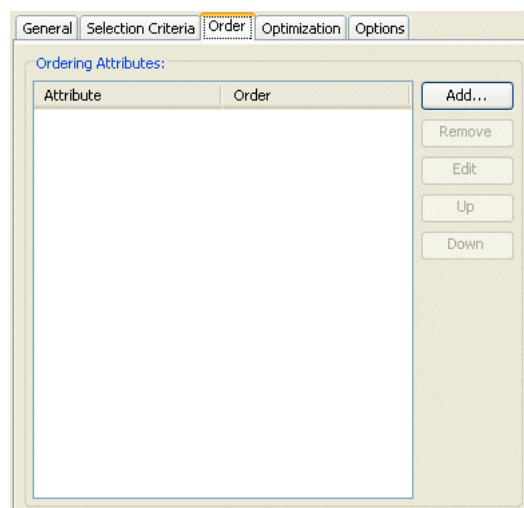
If you define your query using the correct syntax, the generated Java code will contain no errors and be similar to the following:

```
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 28–10 *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 `ReadObjectQuery`) attributes.

For more information on using batch reading, see ["Query Optimization"](#) on page 11-16, ["Reading Case 2: Batch Reading Objects"](#) on page 11-22, and ["Using Batch Reading"](#) on page 98-10.

For more information on joining, see ["Join Reading and Object-Level Read Queries"](#) on page 96-12 and ["Using Join Reading"](#) on page 98-11.

Use this tab to specify batch reading and joining attributes.

Figure 28–11 *Named Queries, Optimization Tab*



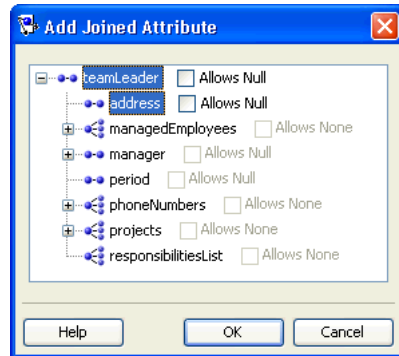
Select one of the following actions for **Batch Read Attributes** (`ReadAllQuery` only):

- 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 28–12 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**.

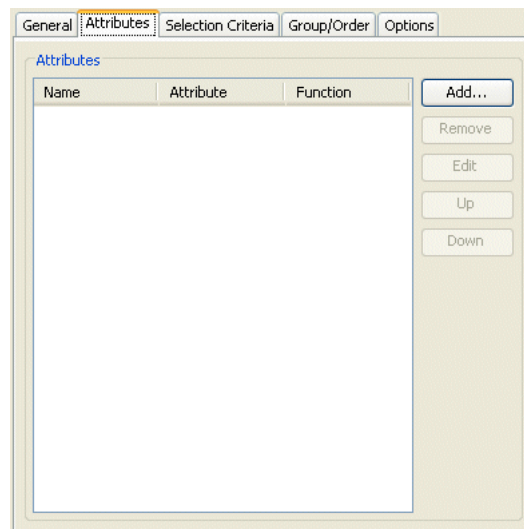
Configuring Named Query Attributes

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 96-15.

Use this tab to configure report query attributes.

Figure 28–13 Named Queries, Attributes Tab



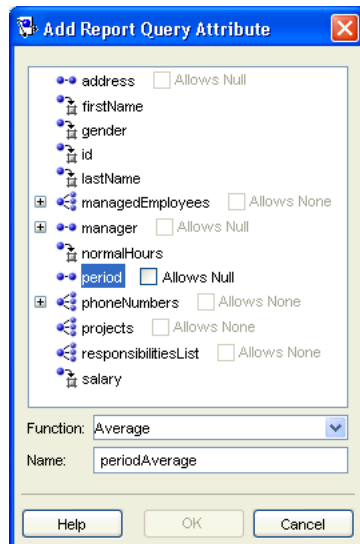
Select one of the following actions for **Attributes** (ReportQuery only):

- To add a new report query attribute, click **Add**. The Add Joined Attribute dialog box appears. Continue with ["Adding Report Query Attributes"](#) on page 28-18.
- 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 28–14 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 <code>getAllowingNull</code>. Check the Allows None option for Collection attributes to use the ExpressionBuilder method <code>anyOfAllowingNone</code>. For more information, see "Using TopLink Expression API For Joins" on page 97-6.

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 <code>getFunction</code> in the API reference.
Name	The name associated with the calculated value. By default, the name is <code><AttributeName><FunctionName></code> .

Enter the necessary information and click **OK**. TopLink Workbench adds the report query attribute to the list of attributes in the Attribute tab.

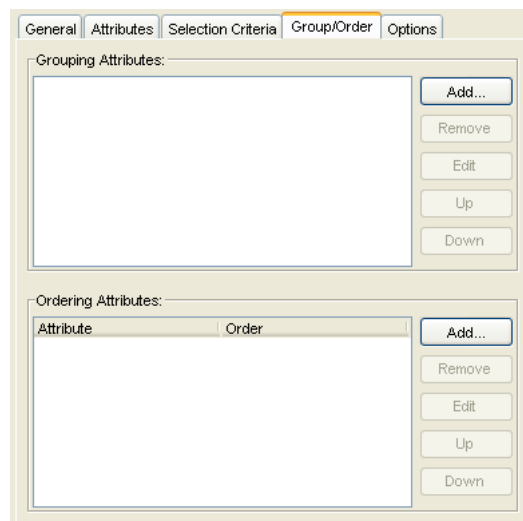
Configuring Named Query Group/Order Options

For ReportQuery queries only, you can configure grouping and ordering attributes.

For more information about report queries, see ["Report Query"](#) on page 96-15.

Use this tab to specify grouping and ordering attributes.

Figure 28–15 *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 28-20.
- 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 28–16 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 28-17). 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 31-6); 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 28–17 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/ <i>first-name</i> .
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/ <i>first-name</i> . 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 <code>exec-find-order</code> , then under an <code>arguments</code> 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 <code>order</code> , if the results were return under a root element <code>results</code> , then under an <code>order</code> element.
Properties	Specify any properties required by your EIS platform. For example, property name <code>operation</code> (from <code>AQPlatform.QUEUE_OPERATION</code>) and property value <code>enqueue</code> (from <code>AQPlatform.ENQUEUE</code>).

Configuring Named Query Options

Use this tab to configure additional options for the query.

Figure 28–18 *Named Queries, Options Tab*

Use the following information to complete each field on the tab:

Field	Description
Refresh Identity Map Results²	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 Statement¹	Caches the prepared statements. This requires full parameter binding as well (see Bind Parameters).
Bind Parameters¹	Binds all of the query's parameters.
Cache Usage²	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 96-30 ■ "Configuring Unit of Work Conforming at the Descriptor Level" on page 28-6

Field	Description
In Memory Query Indirection²	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 96-33 ■ "Indirection" on page 33-5.
Return Choice³	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 "Report Query Results" on page 96-8.
Retrieve Primary Keys³	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.

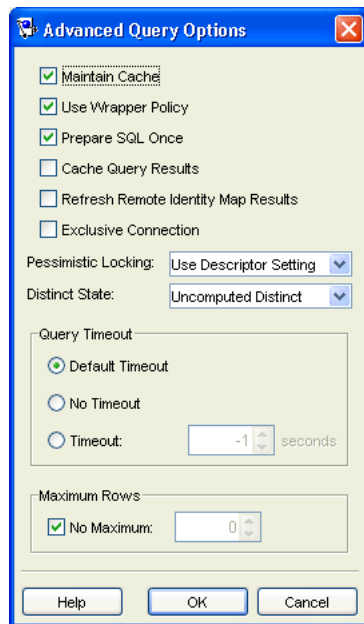
Click **Advanced** to configure additional options. See ["Configuring Named Query Advanced Options"](#) on page 28-24.

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 28–19 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 96-11), whose results cannot be cached. For more information, see "Disabling the Identity Map Cache Update During a Read Query" on page 96-34.
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 28-75.
Prepare SQL Once	Specify the <code>setShouldPrepare()</code> 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 <code>equal</code> where the argument value could be <code>null</code>. This may cause problems on databases that require <code>IS NULL</code>, instead of <code>= NULL</code>. ■ Expressions that use <code>in</code> and use parameter binding. This will cause problems as the <code>in</code> values must be bound individually.

Field	Description
Cache Query Results	Specify the <code>cacheQueryResults</code> 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 99-21.
Refresh Remote Identity Map Results	Specify the <code>refreshRemoteIdentityMapResult</code> 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 77-19).
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 <code>DISTINCT</code> 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 28-78). [Example 28–1](#) illustrates an amendment method that creates a named query and adds it to the `DescriptorQueryManager`.

Example 28–1 Creating a Named Query with an Amendment Method

```
public class EmployeeAmendmentMethodClass {
    ....
    // Create named query
    public static void createEmployeeQuery(ClassDescriptor descriptor) {
        ExpressionBuilder emp = new ExpressionBuilder();
        Expression firstNameExpression =
            emp.get("firstName").equal(emp.getParameter("firstName"));
        ReadObjectQuery query = new ReadObjectQuery();
        query.setReferenceClass(Employee.class);
        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 28–4 summarizes which descriptors support query timeout configuration.

Table 28–9 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 27-2).

You can also configure a timeout on a per-query basis. For more information, see the following:

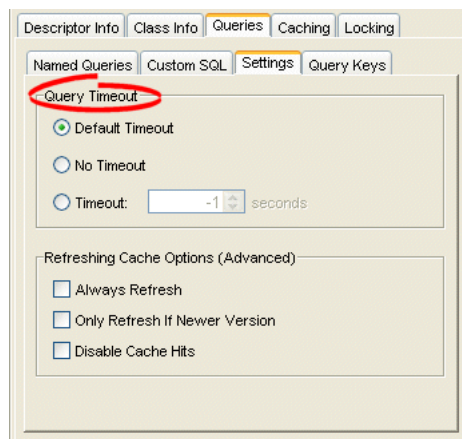
- "Configuring Named Query Advanced Options" on page 28-24
- "Configuring Query Timeout at the Query Level" on page 98-10

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 28–20 Descriptor Queries Settings Tab, Query Timeout Options



Field	Description
Default Timeout	TopLink throws a <code>DatabaseException</code> 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 <code>DatabaseException</code> 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 ["Understanding the Cache"](#) on page 90-1). 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 28-62) 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 28-4](#) summarizes which descriptors support query cache refresh configuration.

Table 28-10 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 27-2).

² EIS root descriptors only (see ["EIS Root Descriptors"](#) on page 27-5).

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 96-35)
- cache expiration (see ["Configuring Cache Expiration at the Descriptor Level"](#) on page 28-40)
- isolated caching (see ["Cache Isolation"](#) on page 90-9)

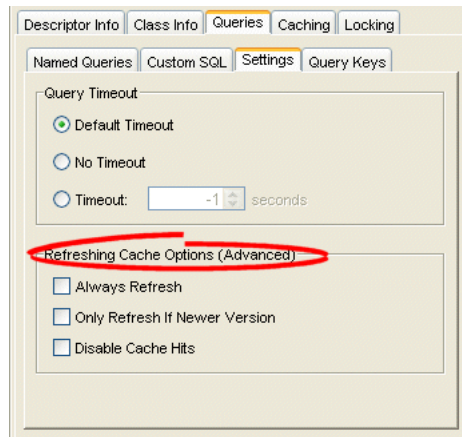
For more information, see ["Cache Optimization"](#) on page 11-13

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 28–21 Descriptor Queries Settings Tab, Cache Refreshing Options



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 96-35) ■ cache expiration (see "Configuring Cache Expiration at the Descriptor Level" on page 28-40) ■ isolated caching (see "Cache Isolation" on page 90-9)
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 28-62 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 96-35), or cache expiration (see "Configuring Cache Expiration at the Descriptor Level" on page 28-40).

Field	Description
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 96-35) or configuring cache expiration (see ["Configuring Cache Expiration at the Descriptor Level"](#) on page 28-40). 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 28-78) as [Example 28–2](#) illustrates.

Example 28–2 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 28–11](#) summarizes which descriptors support query keys.

Table 28–11 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 96-4.

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 27-2) 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.

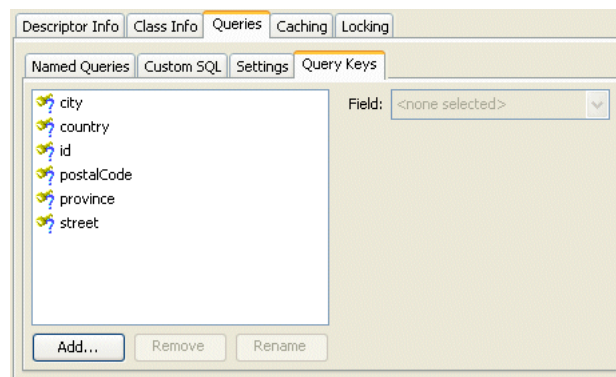
TopLink Workbench provides support for choosing the implementors of an interface that share at least one common automatically generated query key (see ["Configuring Interface Query Keys"](#) on page 28-33).

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 28-22 *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 28-78) 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 28-3, 28-4, and 28-5 illustrate how to define a query key in Java code.

Example 28–3 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 28–4 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 28–5 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 28–6](#) 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 28–6 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 28-29.

Interface descriptors (see ["Relational Interface Descriptors"](#) on page 27-2) 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 28-11](#) summarizes which descriptors support interface query keys.

Table 28-12 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 28-23](#) illustrates the generated keys:

Figure 28-23 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 99-5).

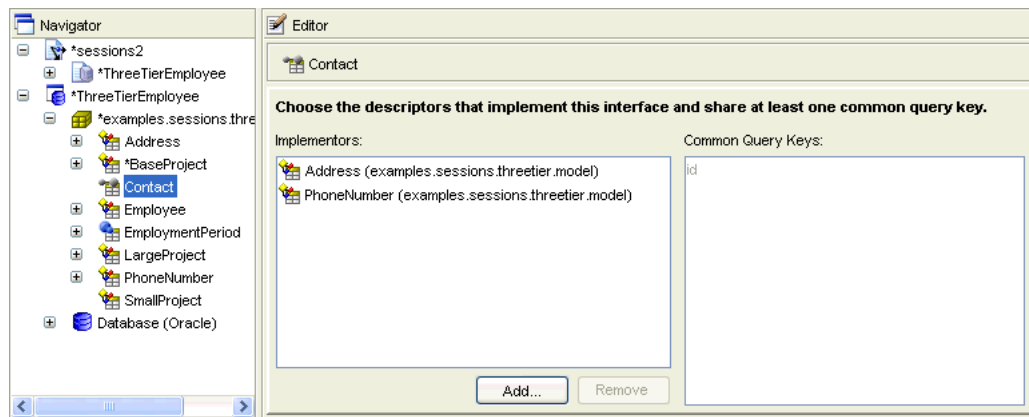
For example, you can now create a variable one-to-one mapping for the `contact` attribute of `Employee`. When you edit the foreign key field information for the mapping, you must match the `Employee` descriptor's tables to query keys from the `Contact` interface descriptor.

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 28–24 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 28–7](#) shows how to define the `Contact` interface and `Email` and `Phone` implementors in Java.

Example 28–7 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 28–13](#) summarizes which descriptors support identity map configuration.

Table 28–13 *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 27-2).

² EIS root descriptors only (see ["EIS Root Descriptors"](#) on page 27-5).

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 22-13).

For detailed information on caching and object identity, and the recommended settings to maximize TopLink performance, see ["Cache Type and Object Identity"](#) on page 90-3.

For more information about the cache, see ["Understanding the Cache"](#) on page 90-1.

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 28–25 Caching Tab, Identity Map Options

Descriptor Info Class Info Queries **Caching** Locking

Type: Default (Weak with Soft Subcache) ▼

Size: 100 ▼ ☒ Default

Isolation: Default (Shared) ▼

Coordination: Default (Synchronize Changes) ▼

Expiry

☒ Project Default

☒ No Expiry

☐ Time to Live Expiry Expire after: 0 milliseconds

☐ Daily Expiry Expire at: 00 : 00 : 00

☐ Update Read Time on Update

Existence Checking: Default (Check Cache) ▼

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 <i>n</i> elements in soft space, anything after that in weak space (see "Soft and Hard Cache Weak Identity Maps" on page 90-4) ■ Weak with Hard Subcache (HardCacheWeakIdentityMap)—cache first <i>n</i> elements in hard space, anything after that in weak space (see "Soft and Hard Cache Weak Identity Maps" on page 90-4) ■ Weak (WeakIdentityMap)—cache everything in weak space (see "Weak Identity Map" on page 90-3) ■ Full (FullIdentityMap)—cache everything permanently (see "Full Identity Map" on page 90-3) ■ None (NoIdentityMap)—cache nothing (see "No Identity Map" on page 90-4) For more information, see "Cache Type and Object Identity" on page 90-3. Changing the project's default identity map does not affect descriptors that already exist in the project.
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 <i>starting size</i> 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 26-4.

Using Java

Use one of the following `ClassDescriptor` methods to configure the descriptor to use the appropriate type of identity map:

- `useFullIdentityMap`
- `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 ("[Cache Isolation](#)" on page 90-9), 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 28-13](#) summarizes which descriptors support cache isolation configuration.

Table 28-14 *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 27-2).

² EIS root descriptors only (see "[EIS Root Descriptors](#)" on page 27-5).

This configuration overrides the default cache isolation configuration defined at the project level (see "[Configuring Cache Isolation at the Project Level](#)" on page 22-16).

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 28-38).

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 28–26 Caching Tab, Isolation Options

The screenshot shows the 'Caching' tab in the Oracle TopLink Workbench. The 'Isolation' dropdown is highlighted with a red circle. The 'Type' is set to 'Default (Weak with Soft Subcache)', 'Size' is '1.00', 'Coordination' is 'Default (Synchronize Changes)', and 'Existence Checking' is 'Default (Check Cache)'. The 'Expiry' section shows 'Project Default' selected, with options for 'No Expiry', 'Time to Live Expiry', and 'Daily Expiry'.

- **Isolated**—if you want all objects confined to an isolated client session cache. For more information, see ["Cache Isolation"](#) on page 90-9.
- **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 28-78) to call `ClassDescriptor` method `setIsIsolated`, passing in a boolean of `true`.

Configuring Cache Coordination Change Propagation at the Descriptor Level

If you plan to use a coordinated cache (see ["Understanding Cache Coordination"](#) on page 90-10), you can configure how, and under what conditions, a coordinated cache propagates changes for a given descriptor.

[Table 28–13](#) summarizes which descriptors support cache isolation configuration.

Table 28–15 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 27-2).

² EIS root descriptors only (see ["EIS Root Descriptors"](#) on page 27-5).

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 22-17).

To complete your coordinated cache configuration, see ["Configuring a Coordinated Cache"](#) on page 91-1.

Note: If you configure a descriptor as isolated (see ["Configuring Cache Isolation at the Descriptor Level"](#) on page 28-37), 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 28–27 Caching Tab, Coordination Options

The screenshot shows the 'Caching' tab in the TopLink Workbench. The 'Coordination' dropdown menu is highlighted with a red circle. The options are: Default (Weak with Soft Subcache), 1.00, Default, Default (Shared), and Default (Synchronize Changes). The 'Expiry' section has radio buttons for Project Default, No Expiry, Time to Live Expiry, and Daily Expiry. The 'Existence Checking' dropdown is set to Default (Check Cache).

3. Complete the **Coordination** options on the **Caching** tab.

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.

Using Java

Use a descriptor amendment method (see ["Configuring Amendment Methods"](#) on page 28-78) 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 101-7) or garbage collected when using a weak identity map (see ["Configuring Cache Isolation at the Project Level"](#) on page 22-16). 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 28-27).

[Table 28-16](#) summarizes which descriptors support a cache invalidation policy.

Table 28-16 *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 27-2).

² EIS root descriptors only (see ["EIS Root Descriptors"](#) on page 27-5).

You can override the project-level cache invalidation configuration (see ["Configuring Cache Expiration at the Project Level"](#) on page 22-19) by defining cache invalidation at the descriptor or query level (see ["Configuring Cache Expiration at the Query Level"](#) on page 99-22).

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 28-38.

For more information, see ["Cache Invalidation"](#) on page 90-8

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 28–28 Caching Tab, Expiration Options

The screenshot shows the 'Caching' tab in the TopLink Workbench. The 'Expiry' section is highlighted with a red circle. The 'Project Default' checkbox is checked. Other options include 'No Expiry', 'Time to Live Expiry' (with 'Expire after' field set to 0 milliseconds), and 'Daily Expiry' (with 'Expire at' field set to 00:00:00). The 'Update Read Time on Update' checkbox is unchecked. The 'Existence Checking' dropdown is set to 'Default (Check Cache)'.

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 22-19 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 28–17](#) summarizes which descriptors support existence checking.

Table 28–17 *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 27-2).

² EIS root descriptors only (see ["EIS Root Descriptors"](#) on page 27-5).

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 22-8).

For more information see:

- ["Cache Type and Object Identity"](#) on page 90-3
- ["Queries and the Cache"](#) on page 96-29
- ["Using Registration and Existence Checking"](#) on page 102-5

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 28–29 Caching Tab, Existence Checking Options

The screenshot shows the 'Caching' tab of a configuration window. It includes fields for 'Type' (Default (Weak with Soft Subcache)), 'Size' (100), 'Isolation' (Default (Shared)), and 'Coordination' (Default (Synchronize Changes)). Under the 'Expiry' section, 'Project Default' is checked, and 'No Expiry' is selected. There are also options for 'Time to Live Expiry' and 'Daily Expiry'. At the bottom, the 'Existence Checking' dropdown is circled in red and set to 'Default (Check Cache)'.

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 102-5.
Assume Existence	Always assume objects exist: always do an update (never do an insert). For more information, see "Assume Existence" on page 102-5.
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 102-5.

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 28–8](#)):

- `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 102-5.
- `assumeExistenceForDoesExist`—always assume objects exist: always do an update (never do an insert). For more information, see ["Assume Existence"](#) on page 102-5.

- `assumeNonExistenceForDoesExist`—always assume objects do not exist: always do an insert (never do an update). For more information, see ["Assume Nonexistence"](#) on page 102-5.

Example 28–8 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 22-5), you can use descriptors to describe the characteristics of CMP and BMP entity beans.

[Table 28–18](#) summarizes which descriptors support EJB information.

Table 28–18 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 27-2).

² EIS root descriptors only (see ["EIS Root Descriptors"](#) on page 27-5).

When mapping EJB, 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 EJB type (such as CMP or BMP) and import the `ejb-jar.xml` file for the beans into the TopLink Workbench project.

For CMP 2.0 projects, the `ejb-jar.xml` file defines the bean's attributes to be mapped. A CMP bean descriptor contains a CMP policy used to configure CMP-specific options.

For more information, see ["Descriptors and EJB"](#) on page 26-3.

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 28-30 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. When using EJB 2.0, this is specified in the <ejb-name> element of the <code>ejb-jar.xml</code> file and is for display only.
Primary Key Class	Enter the primary key. When using EJB 2.0, this is specified in the <prim-key-class> element of the <code>ejb-jar.xml</code> 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. When using EJB 2.0, this is specified in the <local> element of the <code>ejb-jar.xml</code> file and is for display only.
Local Home Interface	Enter the local home interface. When using EJB 2.0, this is specified in the <local-home> element of the <code>ejb-jar.xml</code> file and is for display only.
Remote Interface	Enter the remote interface. When using EJB 2.0, this is specified in the <remote> element of the <code>ejb-jar.xml</code> file and is for display only.
Remote Home Interface	Enter the remote interface. When using EJB 2.0, this is specified in the <home> element of the <code>ejb-jar.xml</code> 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 CMP containers, such as OC4J. For more information, see "Nondeferred Changes" on page 26-3). 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 CMP containers, such as OC4J. For more information, see "Nondeferred Changes" on page 26-3).
Insert New Objects After	Specify to send new object insert changes to the database after bean life cycle method <code>ejbCreate</code> (default) or <code>ejbPostCreate</code>. This is only relevant when not deferring changes (see "Configuring Change Policy" on page 28-70). If non-null foreign key constraints cannot be satisfied when the insert is performed after <code>ejbCreate</code>, you may consider configuring TopLink CMP to do the insert after <code>ejbPostCreate</code>, if supported by your container. For more information, see "Creating a New Entity Bean and <code>ejbCreate</code> / <code>ejbPostCreate</code> Methods" on page 26-4.

Using Java

Using Java code, you can use descriptors to describe the characteristics of CMP (see ["Configuring CMP Information"](#) on page 28-46) and BMP (see ["Configuring BMP Information"](#) on page 28-47) entity beans.

Configuring CMP Information

To configure EJB CMP specific information on a descriptor, define a `CMPPolicy`:

```
descriptor.setCMPPolicy(new CMPPolicy());
```

You can use the following `CMPPolicy` API to configure optional EJB behavior:

Note: Most of these options are provided for compatibility with other EJB 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 non-deferred writes (see `setDeferModificationsUntilCommit`), use this method to configure TopLink to insert a new EJB before (`CMPPolicy.AFTER_EJBCREATE`) or after (`CMPPolicy.AFTER_EJBPOSTCREATE`) the `ejbPostCreate` method.
- `setForceUpdate`—use this method to make TopLink write all EJB that have been accessed to the database regardless if they changed or not. Normally, this is set during migration if the `orion-ejb-jar.xml` file has force update enabled. For more information, see ["Migrating OC4J Orion Persistence to OC4J TopLink Persistence"](#) on page 7-5
- `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

EJB 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 28-75.

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 28-19](#) summarizes which descriptors support inherited attribute mapping configuration.

Table 28-19 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 26-3.

Using TopLink Workbench

To configure reading classes on subqueries, use this procedure:

1. In the **Navigators**, 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 28–31 Inheritance Tab, Read Subclasses on Query Option

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 28-78) 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 28–9](#) shows an amendment method for the `Person` class. In this example, subclasses are not read on queries.

Example 28–9 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 28–38](#) summarizes which descriptors support child inheritance configuration.

Table 28–20 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 26-3.

For more information about configuring inheritance for a parent (root) class descriptor, see ["Configuring Inheritance for a Parent \(Root\) Descriptor"](#) on page 28-50.

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 28–32 Inheritance Tab, Child Descriptor Option

The screenshot shows a configuration window with tabs: Descriptor Info, Class Info, Queries, Caching, Locking, and Inheritance. The Inheritance tab is active. It contains several sections: a 'Read Subclasses on Query' checkbox (unchecked), a 'Read Subclasses View (Optional)' dropdown (set to '<none selected>'), a red circle around the 'Is Child Descriptor' radio button (selected), a 'Parent Descriptor' dropdown (set to 'Project (examples.servletjsp.model)'), an 'Is Root Parent Descriptor' radio button (unchecked), a 'Use Class Extraction Method' dropdown (set to '<none selected>'), a 'Use Class Indicator Field' radio button (unchecked), a 'Field Selection' dropdown (set to '<none selected>'), an 'Indicator Selection' section with 'Use Class Name as Indicator' and 'Use Class Indicator Dictionary' radio buttons (both unchecked), an 'Indicator Type' dropdown (set to '<none selected>'), and a table with columns 'Include', 'Class', and 'Indicator Value' (currently empty) and an 'Edit' button.

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 26-3 for more information.

Using Java

Using Java, you can configure an inheritance child descriptor using InheritancePolicy method setParentClass as Example 28–10 shows.

Example 28–10 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 28–23 summarizes which descriptors support parent inheritance configuration.

Table 28–21 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 28-49.

For more information, see ["Descriptors and Inheritance"](#) on page 26-3.

Using TopLink Workbench

To create a root class for an inheritance, use this procedure.

1. In the **Navigat**or, 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 28–33 Inheritance Tab, Configuring Inheritance for a Root Descriptor

The screenshot shows the 'Order' descriptor configuration window in TopLink Workbench, with the 'Inheritance' tab selected. The 'Is Root Parent Descriptor' radio button is selected. Under 'Field Selection', 'Specify Field' is chosen, and the 'Indicator Selection' section shows 'Use Class Indicator Dictionary' selected. The 'Indicator Type' is set to 'String (java.lang)'. A table at the bottom lists the indicator values for the 'Order' class.

Include	Class	Indicator Value
☐	Order (examples.eis.attunit...)	

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 26-14.
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 26-13.
Field Selection	Choose the field to use as the class indicator field.
Use XML Schema "Type" Attribute¹	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 29-2). 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 27-5) or XML descriptors (see ["XML Descriptors"](#) on page 26-12) only.

Using Java

Create a descriptor amendment method (["Configuring Amendment Methods"](#) on page 28-78) to customize the root class descriptor's `InheritancePolicy` using `InheritancePolicy` methods `setParentClass`, `setClassIndicatorFieldName`, `addClassIndicator`,

`useClassNameAsIndicator` and `setClassExtractionMethodName`, as required.

[Example 28–13](#) 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 26-13).

Example 28–11 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 26-14), you may also need to use `InheritancePolicy` methods `setOnlyInstancesExpression` and `setWithAllSubclassesExpression` (see ["Configuring Inheritance Expressions for a Parent \(Root\) Class Descriptor"](#) on page 28-53).

[Example 28–13](#) 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 26-13).

Example 28–12 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 26-12) with a class extraction method (see ["Using Class Extraction Methods"](#) on page 26-14) you must provide `TopLink` with expressions to correctly filter sibling instances for all classes that share a common table.

Table 28–23 summarizes which descriptors support inheritance expression configuration.

Table 28–22 Descriptor Support for Inheritance Expression Configuration

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors		✓
Object-Relational Descriptors		✓
EIS Descriptors		
XML Descriptors		

Figure 28–34 shows a typical inheritance hierarchy. In this example, instances of both `Person` and `Student` are stored in the same `PERSON` table as Figure 28–35 shows: an instance of `Person` has a `null` value for `STUDENT_NUMBER`. Instances of `Company` are stored in a separate `COMPANY` table.

Figure 28–34 Example Inheritance Hierarchy

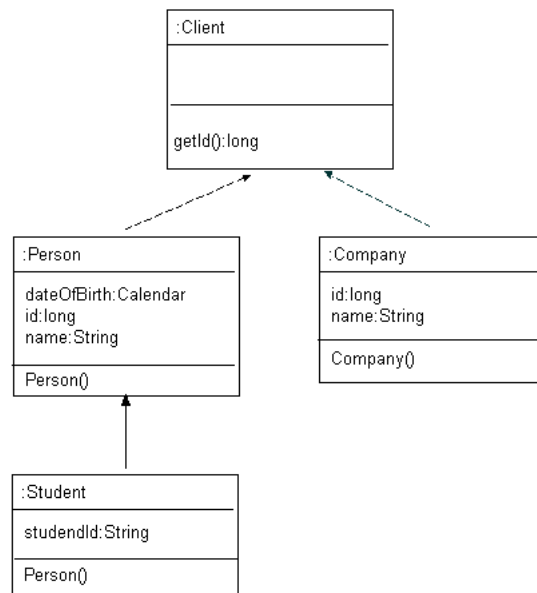


Figure 28–35 PERSON Table

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 26-13), TopLink automatically generates the required expressions.

If you use a class extraction method (see ["Using Class Extraction Methods"](#) on page 26-14), 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 28-47).

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 28-47), 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 26-15.

Using Java

Create a descriptor amendment method (["Configuring Amendment Methods"](#) on page 28-78) to customize the root class descriptor's InheritancePolicy using InheritancePolicy methods setOnlyInstancesExpression and setWithAllSubclassesExpression, as required.

[Example 28-13](#) shows amendment methods for the Person and Student descriptors based on the class hierarchy shown in [Figure 28-34](#) and the database table shown in [Figure 28-35](#).

Example 28-13 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 28–14](#) shows amendment methods for the `Bicycle` and `NonFueledVehicle` descriptors based on the class hierarchy shown in [Figure 26–2](#) 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 28–14 Configuring Only-Instances and With-All-Subclasses Expressions

```
// Bicycle amendment
public static void addToBicycleDescriptor(Descriptor descriptor) {
    ExpressionBuilder builder = new ExpressionBuilder();
    descriptor.getInheritancePolicy().setOnlyInstancesExpression(
        builder.getField("BICYCLE_DESCR").NotNull()
    );
}

// NonFueledVehicle amendment
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 28–23](#) summarizes which descriptors support inherited attribute mapping configuration.

Table 28–23 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 26-3.

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 34, "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 28–25](#). You can register any domain object method that:

- Is public.
- Returns void.
- Takes a single parameter of type `DescriptorEvent`

[Table 28–24](#) summarizes which descriptors support domain object method event handler configuration.

Table 28–24 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 28–25](#) 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 28–25 Descriptor Events

Category	Descriptor Event ID	Description
Delete	<code>PreDeleteEvent</code>	Occurs before an object is deleted from the data source.
	<code>AboutToDeleteEvent</code>	Occurs when an object is deleted from the data source.
	<code>PostDeleteEvent</code>	Occurs after an object is deleted from the data source.

Table 28–25 (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 28-60).

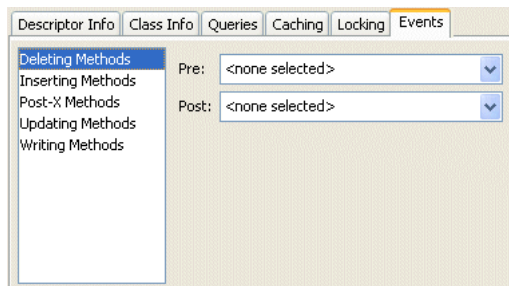
Using TopLink Workbench

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 28–36 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 <i>before</i> the instance is deleted from the data source.
	Post	Select the domain object method that is invoked on an instance of its reference class <i>after</i> 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 <i>before</i> 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 <i>after</i> 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 <i>after</i> the instance is <i>built</i> from the data source.
	Clone	Select the domain object method that is invoked on an instance of its reference class <i>after</i> the instance is <i>cloned</i> into a unit of work.
	Merge	Select the domain object method that is invoked on an instance of its reference class <i>after</i> the instance is <i>merged</i> from a unit of work.
	Refresh	Select the domain object method that is invoked on an instance of its reference class <i>after</i> the instance is <i>refreshed</i> from the data source.
Updating Methods	Pre	Select the domain object method that is invoked on an instance of its reference class <i>before</i> 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 <i>only</i> 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 <i>after</i> 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 <i>before</i> 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 <i>after</i> 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 28–15](#) shows a domain object class with method `handlePostDelete` defined to handle `PostDeleteEvent` descriptor events. [Example 28–16](#) shows 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 28–15 Domain Object Method as a Descriptor Event Handler

```
public class Employee {
    // domain object methods
    ...
    public void handlePostDelete(DescriptorEvent event) {
        // handler implementation
    }
}
```

Example 28–16 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 `DescriptorEventListener` and register it with a `DescriptorEventManager` in a descriptor amendment method. You can also configure a `DescriptorEventListener` 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 28–27](#)). To quickly implement this interface, you can extend abstract class `DescriptorEventAdapter` and override only the methods for the events you are interested in.

[Table 28–26](#) summarizes which descriptors support descriptor event listener configuration.

Table 28–26 Descriptor Support for Descriptor Event Listener Configuration

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors	✓	✓
Object-Relational Descriptors		✓

Table 28-26 (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 `PostBuildEvent` to the event listener's `postBuild` method.

[Table 28-27](#) lists the `DescriptorEventListener` methods associated with each descriptor event. The **Descriptor Event Listener Method** column lists the `DescriptorEventListener` methods associated with each `DescriptorEvent`.

Table 28-27 Descriptor Events

Category	Descriptor Event Listener Method	Description
Delete	<code>preDelete</code>	Occurs before an object is deleted from the data source.
	<code>aboutToDelete</code>	Occurs when an object is deleted from the data source.
	<code>postDelete</code>	Occurs after an object is deleted from the data source.
Insert	<code>preInsert</code>	Occurs before an object is inserted in the data source.
	<code>aboutToInsert</code>	Occurs when an object is inserted in the data source.
	<code>postInsert</code>	Occurs after an object is inserted into the data source.
Post-X	<code>postBuild</code>	Occurs after an object is built from the data source.
	<code>postClone</code>	Occurs after an object has been cloned into a unit of work.
	<code>postMerge</code>	Occurs after an object has been merged from a unit of work.
	<code>postRefresh</code>	Occurs after an object is refreshed from the data source.
Update	<code>preUpdate</code>	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.
	<code>aboutToUpdate</code>	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.
	<code>postUpdate</code>	Occurs after an object is updated in the data source.
Write	<code>preWrite</code>	Occurs before an object is inserted or updated in the data source. This occurs before <code>PreInsertEvent</code> and <code>PreUpdateEvent</code> .
	<code>postWrite</code>	Occurs after an object is inserted or updated in the data source. This occurs after <code>PostInsertEvent</code> and <code>PostUpdateEvent</code> .

Alternatively, you can configure a domain object method as an event handler (see ["Configuring a Domain Object Method as an Event Handler"](#) on page 28-57).

Using Java

[Example 28–17](#) shows a `DescriptorEventListener` that handles `PostBuildEvent` descriptor events. [Example 28–18](#) 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 28–17 *DescriptorEventListener*

```
public class MyDescriptorEventListener extends DescriptorEventAdapter {
    public void postBuild(DescriptorEvent event) {
        // handler implementation
    }
}
```

Example 28–18 *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 28–28](#) summarizes which descriptors support locking policies.

Table 28–28 *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 27-2).

² EIS root descriptors only (see ["EIS Root Descriptors"](#) on page 27-5).

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 26-21).

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 28–37 Locking Tab for a Descriptor

The screenshot shows the 'Locking' tab of a descriptor configuration window. The tabs at the top are 'Descriptor Info', 'Class Info', 'Queries', 'Caching', 'Locking' (selected), and 'EJB Info'. The 'Locking' section has three radio buttons: 'No Locking', 'Optimistic Locking' (selected), and 'Pessimistic Locking'. Under 'Optimistic Locking', there are two sub-sections. The first, 'By Version', has a 'Database Field' dropdown menu currently showing '<none selected>'. Below this are three radio buttons: 'Version Locking' (selected), 'Timestamp Locking', and a checked checkbox for 'Store Version in Cache'. The second sub-section, 'By Fields', has three radio buttons: 'All Fields', 'Changed Fields', and 'Selected Fields'. Below these is a text box and two buttons, 'Add...' and 'Remove'. The 'Pessimistic Locking' section has a checked checkbox for 'Wait for Lock'.

Figure 28–38 Locking Tab for an EIS Root Descriptor

The screenshot shows the 'Locking' tab for an EIS root descriptor. The tabs at the top are 'Descriptor Info', 'Class Info', 'Queries', 'Caching', 'Locking' (selected), and 'EJB Info'. The 'Locking' section has three radio buttons: 'No Locking', 'Optimistic Locking' (selected), and 'Pessimistic Locking'. Under 'Optimistic Locking', there is an 'XPath' text box with a 'Browse...' button next to it. Below this are three radio buttons: 'Version Locking' (selected), 'Timestamp Locking', and a checked checkbox for 'Store Version in Cache'.

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 26-21.
By Fields ¹	Specify to use optimistic locking, based on database fields. These fields appear for relational descriptors only.
All Fields	Select <i>all</i> fields for optimistic locking.
Changed Fields	Select <i>only the changed</i> fields for optimistic locking.
Selected Fields	Click Add to select <i>specific</i> 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 28-44).
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 <code>DatabaseException</code> 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 100-8).

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 26-18.

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 28-44) 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 28-66).

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 28-67).

[Table 28-38](#) summarizes which descriptors support returning policy configuration.

Table 28-29 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 27-5).

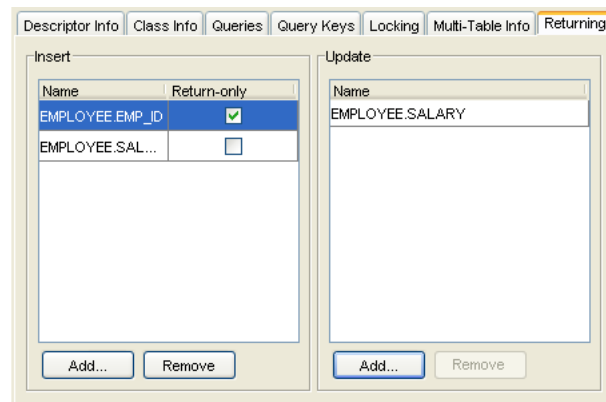
Using TopLink Workbench

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 28–39 Returning 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 36-15).

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 28–30](#) 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 28–30 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
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 28–31](#) summarizes which descriptors support an instantiation policy.

Table 28–31 *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

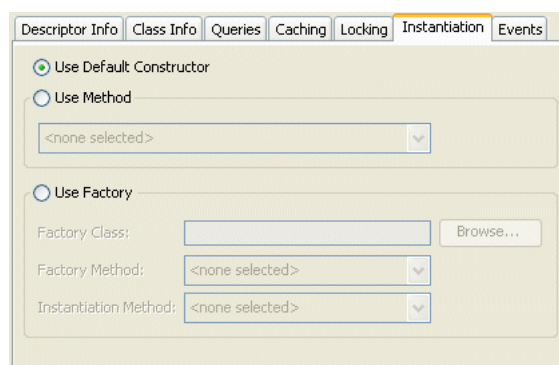
To set the instantiation policy for a descriptor, use this procedure:

1. In the **Navigators**, 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 28–40 *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 28–32 summarizes which descriptors support a copy policy.

Table 28–32 *Configuring Descriptors with a Copy Policy*

Descriptor	Using TopLink Workbench	Using Java
Relational Descriptors	✓	✓
Object-Relational Descriptors		✓
EIS Descriptors	✓	✓
XML Descriptors	✓	✓

TopLink supports two ways of copying objects:

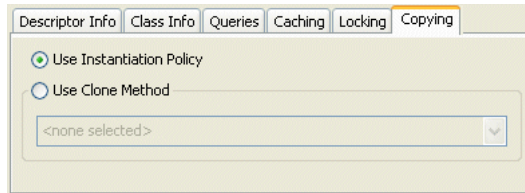
- **Instantiation policy:** By default, TopLink creates a new copy of an object by using the currently configured instantiation policy (see "[Configuring Instantiation Policy](#)" on page 28-68).
- **Method:** TopLink creates a new copy of an object by calling a method on the object that you specify. 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 28–41 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 28-68).
Use Clone Method	Specifies whether or not to call the <code>clone</code> 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 28–33](#) summarizes which descriptors support a change policy.

Table 28–33 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 27-2).

² EIS root descriptors only (see ["EIS Root Descriptors"](#) on page 27-5).

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 100-8).

For EJB 1.*n*, 2.*n*, and 3.0 CMP applications deployed to OC4J, TopLink automatically uses the attribute change tracking policy.

For more information, see ["Unit of Work and Change Policy"](#) on page 100-6.

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 100-7).

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 28-78) that sets the change policy, as [Example 28-19](#) 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 100-7).

For EJB 2.*n* or 3.0 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 CMP 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 28-72).

To configure TopLink to use an `ObjectChangeTrackingPolicy`, use this procedure:

1. Create a descriptor amendment method (see ["Configuring Amendment Methods"](#) on page 28-78) that sets the change policy, as [Example 28-19](#) illustrates.

Example 28–19 Setting the ObjectChangeTrackingPolicy

```
descriptor.setObjectChangePolicy(new ObjectChangeTrackingPolicy());
```

2. For CMP 1.*n* or plain Java objects, code each of your persistent classes to implement the `ChangeTracker` interface as [Example 28–20](#) illustrates.

Example 28–20 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 and 2.*n* CMP applications deployed to OC4J. For more information, see "[Attribute Change Tracking Policy](#)" on page 100-8).

Note: You cannot use the `AttributeChangeTrackingPolicy` if you are using any instance of `FieldsLockingPolicy` (see "[Optimistic Field Locking Policies](#)" on page 26-20).

When you deploy a TopLink-enabled EJB 3.0 or 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 EJB 1.*n* CMP, plain Java objects, or other application servers, use this procedure:

1. Create a descriptor amendment method (see ["Configuring Amendment Methods"](#) on page 28-78) that sets the change policy as [Example 28–21](#) illustrates.

Example 28–21 Setting the `DeferredChangeDetectionPolicy`

```
descriptor.setObjectChangePolicy(new AttributeChangeTrackingPolicy());
```

2. Code each of your persistent classes to implement the `ChangeTracker` interface as [Example 28–22](#) illustrates.

Example 28–22 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 client sessions (see ["Historical Client Sessions"](#) on page 75-24) to execute historical queries (see ["Historical Queries"](#) on page 96-20) 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 Oracle9i 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 Client Sessions Using an Oracle Platform"](#) on page 81-1.

[Table 28–34](#) summarizes which descriptors support history policy configuration.

Table 28–34 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 Client Session Limitations"](#) on page 75-25).

Example Historical Schema

As shown in [Table 28–35](#) and [Table 28–36](#), 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 28–35](#) shows the schema for an `EMPLOYEE` table. The table currently contains one `EMPLOYEE` instance.

Table 28–35 Example Table for `EMPLOYEE`

EMP_ID	F_NAME	L_NAME	SALARY
1	Jane	Doe	55000

[Table 28–36](#) 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 28–36 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 28–36](#), the unique identifier of the current version is given by $(EMP\_ID, START) = (1, 31/08/2004)$.

Using Java

[Example 28–23](#) shows how to describe the schema shown in [Table 28–35](#) and [Table 28–36](#) using the TopLink HistoryPolicy:

Example 28–23 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 28–24](#). 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 28–24 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 (["Configuring Default Query Implementations"](#) on page 96-22).

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, 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 28-37](#) summarizes which descriptors support a wrapper policy.

Table 28-37 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 EJB 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 28-44).

For EJB 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 28-76).

Using Java

Use the `ClassDescriptor` method `setWrapperPolicy` to set the appropriate instance of `WrapperPolicy`.

[Example 28-25](#) shows how to amend a BMP descriptor with the required EJB BMP information.

Example 28–25 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 99-3).

Currently, you can use fetch groups only in CMP 2.0 projects for EJB objects. For non-CMP classes, use partial object querying (see ["Partial Object Queries"](#) on page 96-11).

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 28–38](#) summarizes which descriptors support fetch group configuration.

Table 28–38 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 26-4
- ["Fetch Groups and Object Level Read Queries"](#) on page 96-12
- ["Configuring Default Fetch Group Behavior"](#) on page 99-3

Using Java

To configure a fetch group, use a descriptor amendment method (see ["Configuring Amendment Methods"](#) on page 28-78) as [Example 28-26](#) shows.

Example 28-26 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 99-2.

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 28-39](#) summarizes which descriptors support amendment methods.

Table 28–39 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 26-5.

To customize a session, use a session customizer class (see ["Configuring Customizer Class"](#) on page 77-13).

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 28–42 After Load Tab

Use the following information to enter data in each field on the 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 <code>oracle.toplink.descriptors.ClassDescriptor</code> .

Configuring a Relational Descriptor

This chapter describes how to configure a relational descriptor.

For more information, see the following:

- ["Descriptor Creation Overview"](#) on page 27-1
- ["Relational Descriptors"](#) on page 26-11

Relational Descriptor Configuration Overview

[Table 29-1](#) lists the default configurable options for a relational descriptor.

Table 29-1 Configurable Options for Relational Descriptor

Option	Type	TopLink Workbench	Java
"Configuring Associated Tables" on page 29-2	Basic	✓	✓
"Configuring Primary Keys" on page 28-2	Basic	✓	✓
"Configuring Sequencing at the Descriptor Level" on page 29-3	Basic	✓	✓
"Configuring Read-Only Descriptors" on page 28-4	Advanced	✓	✓
"Configuring Unit of Work Conforming at the Descriptor Level" on page 28-6	Advanced	✓	✓
"Configuring Descriptor Alias" on page 28-7	Advanced	✓	
"Configuring Descriptor Comments" on page 28-9	Advanced	✓	
"Configuring Classes" on page 4-41	Basic	✓	
"Configuring Named Queries at the Descriptor Level" on page 28-10	Advanced	✓	✓
"Configuring Custom SQL Queries for Basic Persistence Operations" on page 29-6	Advanced	✓	✓
"Configuring Query Timeout at the Descriptor Level" on page 28-26	Advanced	✓	✓
"Configuring Cache Refreshing" on page 28-27	Advanced	✓	✓
"Configuring Query Keys" on page 28-29	Advanced	✓	✓
"Configuring Interface Query Keys" on page 28-33	Advanced	✓	✓
"Configuring Interface Alias" on page 29-10	Advanced	✓	✓
"Configuring Cache Type and Size at the Descriptor Level" on page 28-35	Advanced	✓	✓
"Configuring Cache Isolation at the Descriptor Level" on page 28-37	Advanced	✓	✓
"Configuring Cache Coordination Change Propagation at the Descriptor Level" on page 28-38	Advanced	✓	✓

Table 29–1 (Cont.) Configurable Options for Relational Descriptor

Option	Type	TopLink Workbench	Java
"Configuring Cache Expiration at the Descriptor Level" on page 28-40	Advanced	✓	✓
"Configuring Cache Existence Checking at the Descriptor Level" on page 28-42	Advanced	✓	✓
"Configuring a Descriptor With EJB Information" on page 28-44	Advanced	✓	✓
"Configuring a Relational Descriptor as a Class or Aggregate Type" on page 29-11	Advanced	✓	✓
"Configuring Reading Subclasses on Queries" on page 28-47	Advanced	✓	✓
"Configuring Inheritance for a Child (Branch or Leaf) Class Descriptor" on page 28-49	Advanced	✓	✓
"Configuring Inheritance for a Parent (Root) Descriptor" on page 28-50	Advanced	✓	✓
"Configuring Inheritance Expressions for a Parent (Root) Class Descriptor" on page 28-53	Advanced	✓	✓
"Configuring Inherited Attribute Mapping in a Subclass" on page 28-56	Advanced	✓	✓
"Configuring Multitable Information" on page 29-13	Advanced	✓	✓
"Configuring a Domain Object Method as an Event Handler" on page 28-57	Advanced	✓	✓
"Configuring a Descriptor Event Listener as an Event Handler" on page 28-60	Advanced	✓	✓
"Configuring Locking Policy" on page 28-62	Advanced	✓	✓
"Configuring Returning Policy" on page 28-65	Advanced	✓	✓
"Configuring Instantiation Policy" on page 28-68	Advanced	✓	✓
"Configuring Copy Policy" on page 28-69	Advanced	✓	✓
"Configuring Change Policy" on page 28-70	Advanced		✓
"Configuring a History Policy" on page 28-73	Advanced		✓
"Configuring Wrapper Policy" on page 28-75	Advanced		✓
"Configuring Fetch Groups" on page 28-77	Advanced		✓
"Configuring Amendment Methods" on page 28-78	Advanced	✓	
"Configuring a Mapping" on page 35-1	Basic	✓	✓

Configuring Associated Tables

Each relational class descriptor (see "[Relational Class Descriptors](#)" on page 27-2) 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 27-2).

Using TopLink Workbench

To associate a descriptor with a database table, use this procedure:

1. Select a descriptor in the **Navigators**. Its properties appear in the Editor.
2. Click the **Descriptor Info** tab. The Descriptor Info tab appears.

Figure 29-1 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 23-3) or session level ("[Configuring Sequencing at the Session Level](#)" on page 86-4) 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 28-2). 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 20-14.

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 29–2 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 20-16. ■ 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 20-18 ■ 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 20-19.

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 28-2). ■ 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 20-19).

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 29-1](#).

Example 29-1 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 29-2](#) 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 29-1](#). 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 29-2 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 29-3](#) 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 20-20.

Example 29-3 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 29-4](#), 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 29-1](#), the TopLink runtime will create a DefaultSequence named NEW_SEQ for this descriptor. For more information about DefaultSequence, see ["Default Sequencing"](#) on page 20-18.

Example 29-4 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 29-7).

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 29-8). Using a `Call`, you can define more complex SQL strings and invoke custom stored procedures.

For 2.0 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 21-16), or define them on the **Queries** tab and write them to the file (see ["Writing to the ejb-jar.xml File"](#) on page 21-16).

Note: When you customize the update persistence operation for an application that uses optimistic locking (see ["Configuring Locking Policy"](#) on page 28-62), 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.

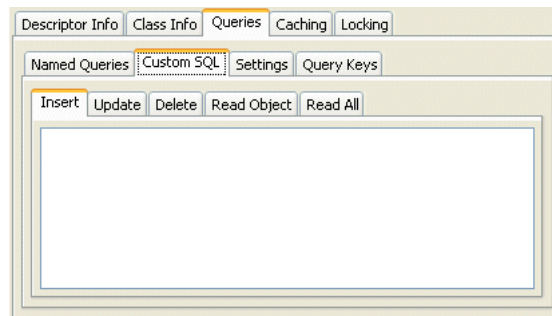
Note: TopLink does not validate the SQL code that you enter. Enter the SQL code appropriate for your database platform (see ["Data Source Platform Types"](#) on page 84-3).

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 29-3 *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 <code>ReadObjectQuery</code>, whose selection criteria is based on the object's primary key. When you define a descriptor's read-object query, your implementation overrides any <code>ReadObjectQuery</code>, whose selection criteria is based on the object's primary key. TopLink generates dynamic SQL for all other <code>Session readObject</code> method signatures. To customize other <code>Session readObject</code> method signatures, define additional named queries and use them in your application instead of the <code>Session</code> methods.
Read All	Defines the read-all SQL that TopLink uses when you call <code>Session readAllObjects(java.lang.Class)</code> passing in the <code>java.lang.Class</code> that this descriptor represents. When you define a descriptor's read-all query, your implementation overrides only the <code>Session</code> method <code>readAll(java.lang.Class)</code>, not the version that takes a <code>Class</code> and <code>Expression</code>. As a result, this query reads every single instance. TopLink generates dynamic SQL for all other <code>Session readAll</code> method signatures. To customize other <code>Session readAll</code> method signatures, define additional named queries and use them in your application instead of the <code>Session</code> 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 29–2](#) lists.

Table 29–2 *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 29–5](#) 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 98-18).

Example 29–5 *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 29-6](#) 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 98-20). This example uses output cursors to return the result set (see ["Handling Cursor and Stream Query Results"](#) on page 99-16).

Example 29-6 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

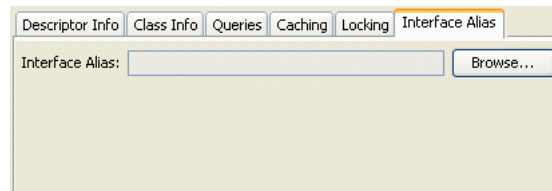
Use the **Interface Alias** tab to specify a descriptor's alias. Each descriptor can have one interface alias. Use the interface in queries and relationship mappings.

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 29–4 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 28-78) and use InterfacePolicy method `addParentInterface` as [Example 29–7](#) shows.

Example 29–7 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 22-3), 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 ["Configuring a Relational Aggregate Collection Mapping"](#) on page 44-1 and ["Configuring a Relational Aggregate Object Mapping"](#) on page 47-1), 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 28-49), 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 26-17.

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 28-44).

For more information, see ["Descriptors and Aggregation"](#) on page 26-5.

Using TopLink Workbench

To configure a relational descriptor as class or aggregate, use this procedure.

1. In the **Navigators**, 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 **Navigators** 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 38, "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 26-5.

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 29-13).

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 29-14).

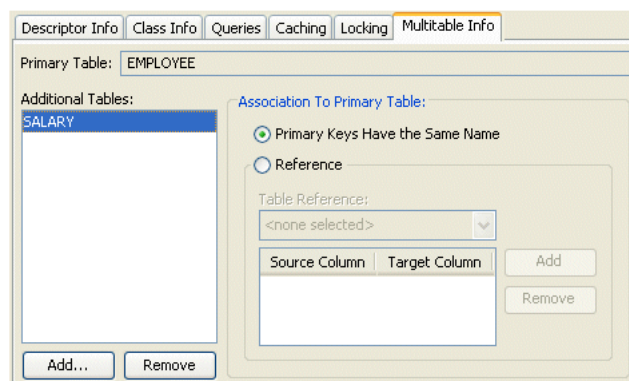
Using TopLink Workbench

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 29-5 Multitable Info Tab



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

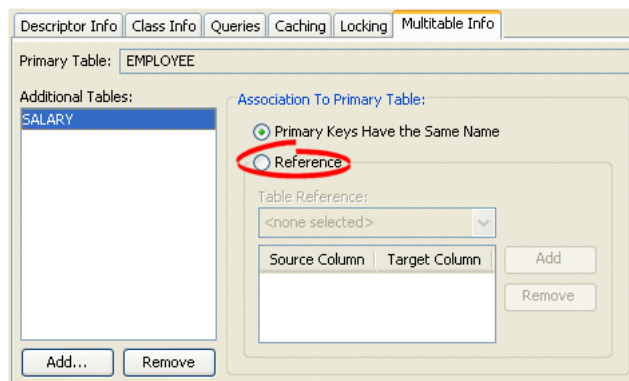
Field	Description
Primary Table	The primary table for this descriptor. This field is for display only.

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 29-14.

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 29-6 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 28-78) and add the join expression using `DescriptorQueryManager` method `setMultipleTableJoinExpression`. For more information, see "[Appending Additional Join Expressions](#)" on page 99-4.

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 27-1
- ["Object-Relational Descriptors"](#) on page 26-11

Object-Relational Descriptor Configuration Overview

Table 30-1 lists the configurable options for an object-relational descriptor.

Table 30-1 Configurable Options for Object-relational Descriptor

Option	Type	TopLink Workbench	Java
"Configuring Field Ordering" on page 30-2	Basic		✓
"Configuring Primary Keys" on page 28-2	Basic		✓
"Configuring Read-Only Descriptors" on page 28-4	Advanced		✓
"Configuring Unit of Work Conforming at the Descriptor Level" on page 28-6	Advanced		✓
"Configuring Query Keys" on page 28-29	Advanced		✓
"Configuring Cache Expiration at the Descriptor Level" on page 28-40	Advanced		✓
"Configuring Amendment Methods" on page 28-78	Advanced		✓
"Configuring Reading Subclasses on Queries" on page 28-47	Advanced		✓
"Configuring Inheritance for a Child (Branch or Leaf) Class Descriptor" on page 28-49	Advanced		✓
"Configuring Inheritance for a Parent (Root) Descriptor" on page 28-50	Advanced		✓
"Configuring Inheritance Expressions for a Parent (Root) Class Descriptor" on page 28-53	Advanced		✓
"Configuring Inherited Attribute Mapping in a Subclass" on page 28-56	Advanced		✓
"Configuring Cache Type and Size at the Descriptor Level" on page 28-35	Advanced		✓
"Configuring a Domain Object Method as an Event Handler" on page 28-57	Advanced		✓
"Configuring a Descriptor Event Listener as an Event Handler" on page 28-60	Advanced		✓
"Configuring Locking Policy" on page 28-62	Advanced		✓

Table 30–1 (Cont.) Configurable Options for Object-relational Descriptor

Option	Type	TopLink Workbench	Java
"Configuring Copy Policy" on page 28-69	Advanced		✓
"Configuring Instantiation Policy" on page 28-68	Advanced		✓
"Configuring Wrapper Policy" on page 28-75	Advanced		✓
"Configuring a History Policy" on page 28-73	Advanced		✓
"Configuring Returning Policy" on page 28-65	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 30–1](#) shows how to specify the order of the object-relational database fields `OBJECT_ID`, `F_NAME`, and `L_NAME` for the `Employee` descriptor.

Example 30–1 Field Ordering

```
descriptor.addFieldOrdering("ID");
descriptor.addFieldOrdering("F_NAME");
descriptor.addFieldOrdering("L_NAME");
```

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 27-1
- ["EIS Descriptors"](#) on page 26-11

EIS Descriptor Configuration Overview

[Table 31-1](#) lists the default configurable options for an EIS descriptor.

Table 31-1 Configurable Options for EIS Descriptor

Option	Type	TopLink Workbench	Java
"Configuring XML Schema Namespace" on page 4-38	Basic	✓	✓
"Configuring XML Schema Reference" on page 4-36	Basic	✓	✓
"Configuring Schema Context for an EIS Descriptor" on page 31-2	Basic	✓	✓
"Configuring Default Root Element" on page 31-3	Basic	✓	
"Configuring Primary Keys" on page 28-2 ¹	Basic	✓	✓
"Configuring Read-Only Descriptors" on page 28-4 ¹	Basic	✓	✓
"Configuring Unit of Work Conforming at the Descriptor Level" on page 28-6 ¹	Advanced	✓	✓
"Configuring Descriptor Alias" on page 28-7	Advanced	✓	
"Configuring Descriptor Comments" on page 28-9	Advanced	✓	
"Configuring Record Format" on page 31-5	Basic		✓
"Configuring Classes" on page 4-41	Basic	✓	
"Configuring Named Queries at the Descriptor Level" on page 28-10 ¹	Advanced	✓	✓
"Configuring Custom EIS Interactions for Basic Persistence Operations" on page 31-6 ¹	Advanced	✓	✓
"Configuring Cache Refreshing" on page 28-27 ¹	Advanced	✓	✓
"Configuring Cache Type and Size at the Descriptor Level" on page 28-35 ¹	Advanced	✓	✓
"Configuring Cache Isolation at the Descriptor Level" on page 28-37	Advanced	✓	✓
"Configuring Cache Coordination Change Propagation at the Descriptor Level" on page 28-38	Advanced	✓	✓

Table 31-1 (Cont.) Configurable Options for EIS Descriptor

Option	Type	TopLink Workbench	Java
"Configuring Cache Expiration at the Descriptor Level" on page 28-40	Advanced	✓	✓
"Configuring Cache Existence Checking at the Descriptor Level" on page 28-42	Advanced	✓	✓
"Configuring a Descriptor With EJB Information" on page 28-44	Advanced	✓	✓
"Configuring an EIS Descriptor as a Root or Composite Type" on page 31-8	Basic	✓	✓
"Configuring Inheritance for a Child (Branch or Leaf) Class Descriptor" on page 28-49	Advanced	✓	✓
"Configuring Inheritance for a Parent (Root) Descriptor" on page 28-50	Advanced	✓	✓
"Configuring Inherited Attribute Mapping in a Subclass" on page 28-56	Advanced	✓	✓
"Configuring a Domain Object Method as an Event Handler" on page 28-57	Advanced	✓	✓
"Configuring a Descriptor Event Listener as an Event Handler" on page 28-60	Advanced	✓	✓
"Configuring Locking Policy" on page 28-62 ¹	Advanced	✓	✓
"Configuring Returning Policy" on page 28-65	Advanced	✓	✓
"Configuring Instantiation Policy" on page 28-68	Advanced	✓	✓
"Configuring Copy Policy" on page 28-69	Advanced	✓	✓
"Configuring Change Policy" on page 28-70	Advanced		✓
"Configuring Wrapper Policy" on page 28-75	Advanced		✓
"Configuring Amendment Methods" on page 28-78	Advanced	✓	✓
"Configuring a Mapping" on page 35-1	Basic	✓	✓

¹ EIS root descriptors only (see "Configuring an EIS Descriptor as a Root or Composite Type" on page 31-8).

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 31-8) 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 31-1 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 31-3.

Choosing a Schema Context

Use the Choose Schema Context dialog box to select a specific schema element (such as when mapping an element).

Figure 31-2 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 27-5) 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 26-9.

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 31-3 *Descriptor Info Tab, Default Root Element Option*

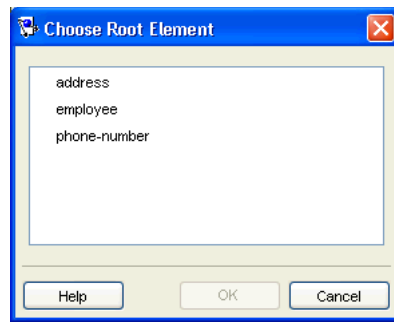
The screenshot shows the 'Descriptor Info' tab in the TopLink Workbench interface. The 'Default Root Element' field is highlighted with a red circle and contains the text 'ADDRESS'. The 'Schema Context' field contains the text 'schema::order//element::ORDER/element::ADDRESS'. Below these fields are buttons for 'Browse...'. The 'Primary Keys' section shows a list with '@ADDRESSEE' and buttons for 'Add...' and 'Remove'. There are also checkboxes for 'Read-Only' and 'Conform Results in Unit of Work'. At the bottom, there is a 'Descriptor Alias' field and a 'Comment' text area.

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 31-4 for more information.

Choosing a Root Element

Use the Choose Root Element dialog box to select a specific root element.

Figure 31–4 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 31–2](#) shows.

Table 31–2 EIS Record Formats

EISDescriptor Method	EIS Record Type
<code>useMappedRecordFormat</code>	All EIS mappings owned by this descriptor map to EIS mapped records.
<code>useIndexedRecordFormat</code>	All EIS mappings owned by this descriptor map to EIS indexed records.
<code>useXMLRecordFormat</code>	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 20-5.

For more information, see ["EIS Record Type"](#) on page 56-2.

Using Java

To configure the EIS record format for an EIS descriptor, use one of the `EISDescriptor` methods listed in [Table 31-2](#) as shown in [Example 31-1](#).

Example 31-1 *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 31-8).

For 2.0 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 21-16 and "[Reading From the ejb-jar.xml File](#)" on page 21-16.

Using TopLink Workbench, you can create `XMLInteraction` objects, in which there is a single query per interaction (see "[Using TopLink Workbench](#)" on page 31-6).

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 31-8).

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 57-3).

Using TopLink Workbench

To configure custom EIS interactions for basic persistence operations, use the following procedure:

1. In the **Navigators**, 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 31–5 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/<i>first-name</i>.
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/<i>first-name</i>. 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 <i>exec-find-order</i>, then under an <i>arguments</i> 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 <code>order</code> , if the results were return under a root element <code>results</code> , then under an <code>order</code> element.
Properties	Any properties required by your EIS platform. For example, property name <code>operation</code> (from <code>AQPlatform.QUEUE_OPERATION</code>) and property value <code>enqueue</code> (from <code>AQPlatform.ENQUEUE</code>).

Using Java

Using Java, you can create any type of EIS interaction that TopLink supports (see ["Using EIS Interactions"](#) on page 98-24).

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 31–2](#) shows.

Example 31–2 *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 27-5) or composite (see ["EIS Composite Descriptors"](#) on page 27-5).

When you designate an EIS descriptor as a root, you tell the TopLink runtime that the EIS descriptor's reference class is a parent 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 and you can configure the descriptor with EIS interactions (see ["Using EIS Interactions"](#) on page 98-24). 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 ["Configuring an EIS Composite Object Mapping"](#) on page 60-1 and ["Configuring an EIS Composite Collection Mapping"](#) on page 61-1). 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 28-49), 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 26-17.

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 28-44).

For more information, see the following:

- ["Descriptors and Aggregation"](#) on page 26-5
- ["Composite and Reference EIS Mappings"](#) on page 56-4

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 28-78) 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`.

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 27-1
- ["XML Descriptors"](#) on page 26-12

XML Descriptor Configuration Overview

[Table 32–1](#) lists the default configurable options for an XML descriptor.

Table 32–1 Configuration Options for XML Descriptors

Option	Type	TopLink Workbench	Java
"Configuring XML Schema Namespace" on page 4-38	Basic	✓	✓
"Configuring XML Schema Reference" on page 4-36	Basic	✓	✓
"Configuring Schema Context for an XML Descriptor" on page 32-1	Basic	✓	✓
"Configuring for Complex Type of anyType" on page 32-3	Advanced	✓	
"Configuring Default Root Element" on page 32-5	Basic	✓	✓
"Configuring Document Preservation" on page 32-6	Basic	✓	✓
"Configuring Descriptor Comments" on page 28-9	Advanced	✓	
"Configuring Classes" on page 4-41	Basic	✓	
"Configuring Inheritance for a Child (Branch or Leaf) Class Descriptor" on page 28-49	Advanced	✓	✓
"Configuring Inheritance for a Parent (Root) Descriptor" on page 28-50	Advanced	✓	✓
"Configuring Inherited Attribute Mapping in a Subclass" on page 28-56	Advanced	✓	✓
"Configuring Instantiation Policy" on page 28-68	Advanced	✓	✓
"Configuring Copy Policy" on page 28-69	Advanced	✓	✓
"Configuring Amendment Methods" on page 28-78	Advanced	✓	✓
"Configuring a Mapping" on page 35-1	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.

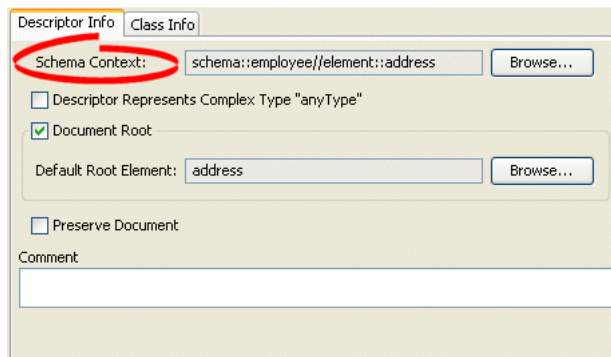
The TopLink runtime uses the schema context to validate XML fragments.

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 32–1 *Descriptor Info Tab, Schema Context Option*

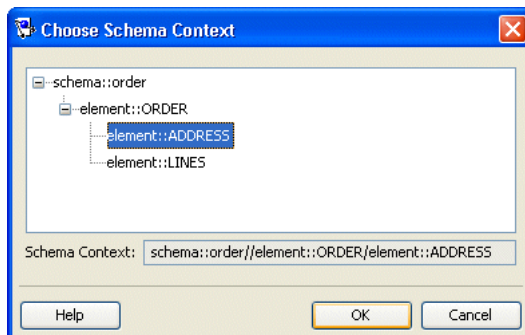


Click **Browse** to select the schema element to associate with this descriptor. For more information, see ["Choosing a Schema Context"](#) on page 32-2.

Choosing a Schema Context

Use the Choose Schema Context dialog box to select a specific schema element (such as when mapping an element).

Figure 32–2 *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 28-78) that uses XMLSchemaReference method `setSchemaContext`, as [Example 32-1](#) shows.

Example 32-1 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.

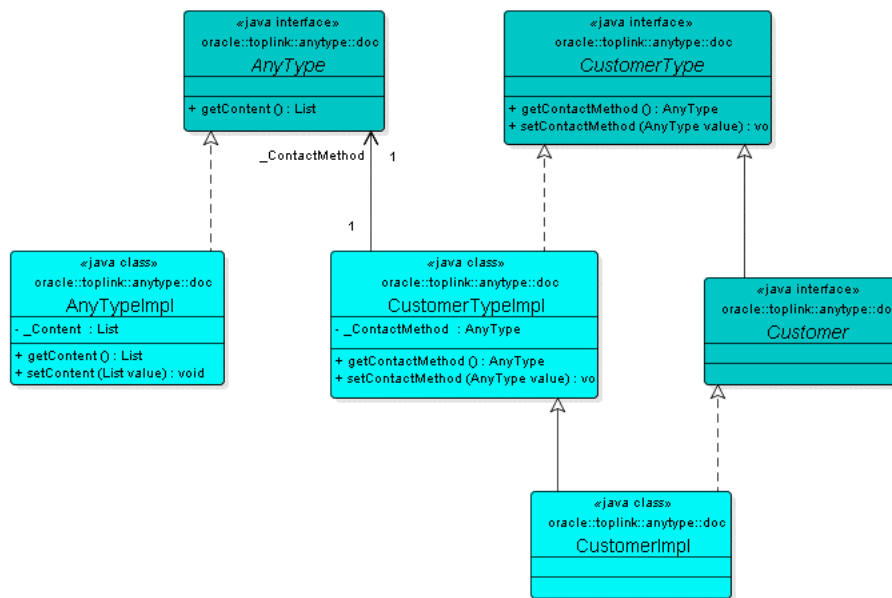
In general, TopLink Workbench assumes that every XML descriptor must have a schema context (see ["Configuring Schema Context for an XML Descriptor"](#) on page 32-1). However, if a class in your project does not relate to an element in your schema, then it does not have a schema context.

For example, consider the schema that [Example 32-2](#) shows.

Example 32-2 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 32-3](#). 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 32-3 Class Representing xsd:anyType

Note: See also "Configuring Maps to Wildcard" on page 66-3.

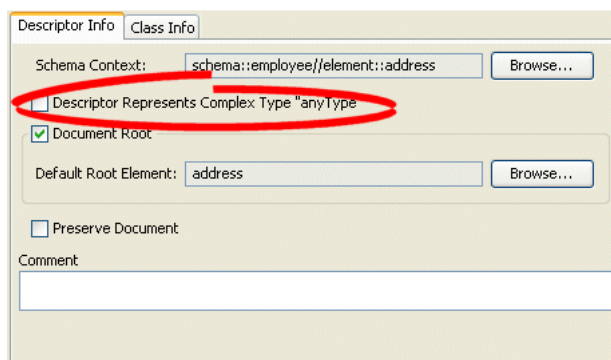
For more information, see "xs:any and xs:anyType Support" on page 65-4.

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 32-4 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.

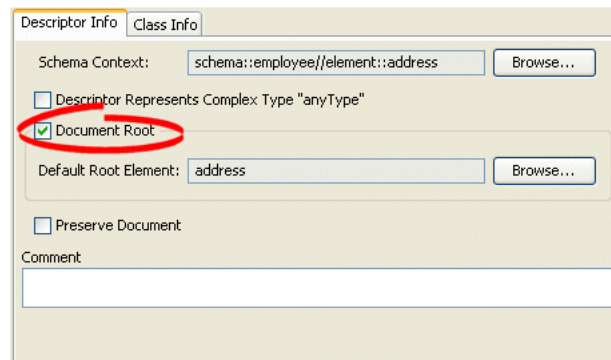
For more information, see ["Default Root Element"](#) on page 26-9.

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 32-5 *Descriptor Info Tab, Default Root Option*



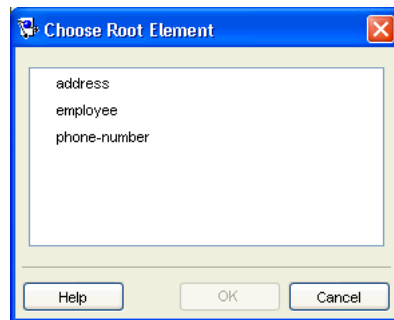
Select the **Default Root Element** option to specify this descriptor as the root element.

Click **Browse** to select the schema element to identify as the root element for this descriptor. See ["Choosing a Root Element"](#) on page 32-5 for more information.

Choosing a Root Element

Use the Choose Root Element dialog box to select a specific root element.

Figure 32-6 *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).

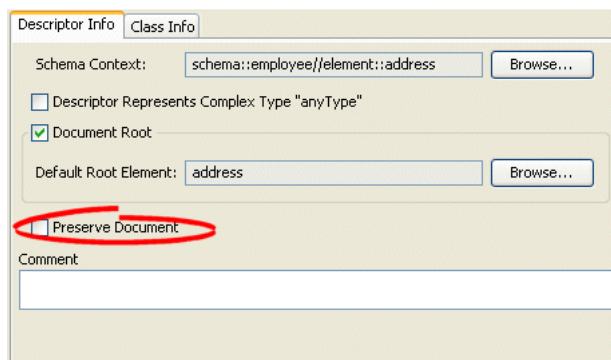
This permits round-tripping from XML to objects and back to XML without losing any data.

Using TopLink Workbench

To preserve the entire XML source document, use this procedure:

1. Select a descriptor in the **Navigators**. Its properties appear in the Editor.
2. Click the **Descriptor Info** tab. The Descriptor Info tab appears.

Figure 32–7 *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 28-78) that configures the descriptor using XMLDescriptor method `setShouldPreserveDocument`.

Part X

Mappings

This part describes general mapping information. It contains the following chapters:

- [Chapter 33, "Understanding Mappings"](#)

This chapter describes each of the different TopLink mapping types and important mapping concepts.

- [Chapter 34, "Creating a Mapping"](#)

This chapter contains procedures for creating TopLink mappings.

- [Chapter 35, "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 XI, "Relational Mappings"](#)

- [Part XII, "Object-Relational Mappings"](#)

- [Part XIII, "EIS Mappings"](#)

- [Part XIV, "XML Mappings"](#)

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 33–1](#) describes the mapping types that TopLink supports.

Table 33–1 *TopLink Mapping Types*

Type	Description	Type	TopLink Workbench	Java
"Relational Mappings" on page 33-25	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 33-26	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 33-27	Mappings that transform object data members to the EIS record format defined by the object's descriptor.	Advanced	✓	✓
"XML Mappings" on page 33-27	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 34-1
- ["Configuring a Mapping"](#) on page 35-1

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 EJB 2.0 or 3.0 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 33-3.

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 20-1.

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 26-1.

TopLink provides mappings to handle a wide variety of data types and data representations. For more information, see ["Mapping Types"](#) on page 33-1.

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 33-25.

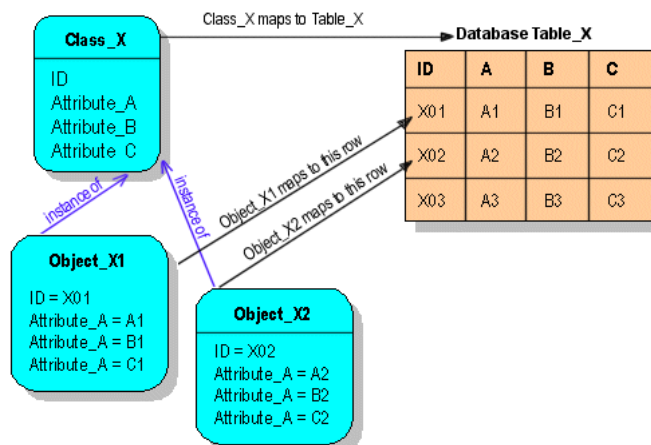
Example Mapping

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 33-1](#) 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 33-1 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 33-1](#), to complex mappings.

For an additional example of a relational mapping, see [Figure 36–1, "Direct-to-Field Mapping"](#) on page 36-4.

For an example of a nonrelational mapping, see [Figure 65–34, "XML Transformation Mappings"](#) on page 65-31.

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 34-1).

Alternatively, you can take advantage of the following:

- [Automapping With TopLink Workbench at Development Time](#)
- [Default Mapping in EJB 2.0 or 3.0 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 34-2).

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 EJB 2.0 or 3.0 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 36-2).
- 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 EJB 2.0 and 3.0 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 EJB 2.0 or 3.0 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 or if equivalent EJB 3.0 annotations are specified, 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

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 65, "Understanding XML Mappings"](#).

Indirection

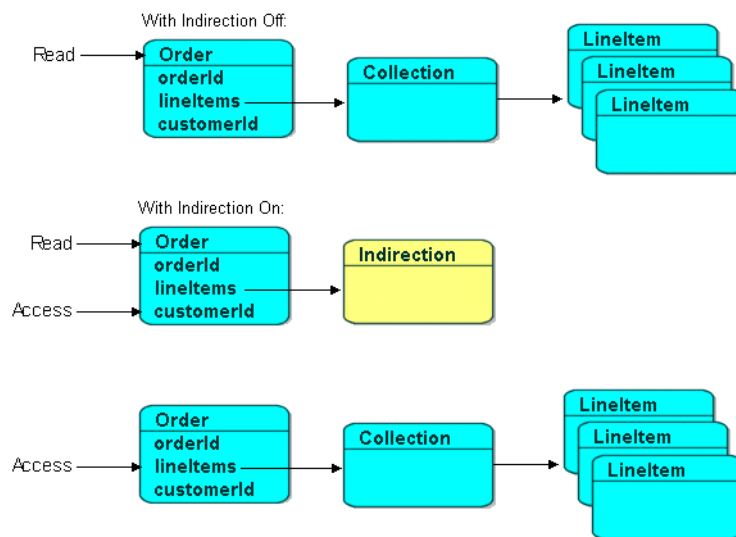
By default, when TopLink retrieves a persistent object, it retrieves all of the dependent objects to which it refers. When you configure indirection (also known as lazy reading, lazy loading, and just-in-time reading) for an attribute mapped with a relationship mapping, TopLink uses an indirection object 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 let 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 33-8).

Figure 33–2 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 33–2 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 33-9).

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 33-9).

For information on configuring indirection, see ["Configuring Indirection"](#) on page 35-3.

Value Holder Indirection

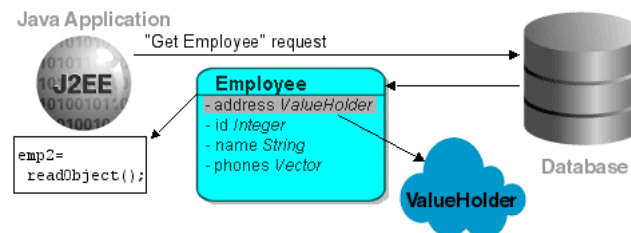
Persistent classes that use indirection must replace relationship attributes with value holder attributes. A value holder is an instance of a class that implements the

`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.

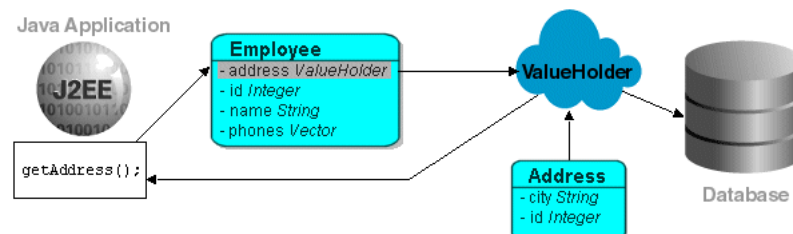
Figure 33-3 shows the `Employee` object being read from the database. The `Address` object is not read and will not be created unless it is accessed.

Figure 33-3 Address Object Not Read



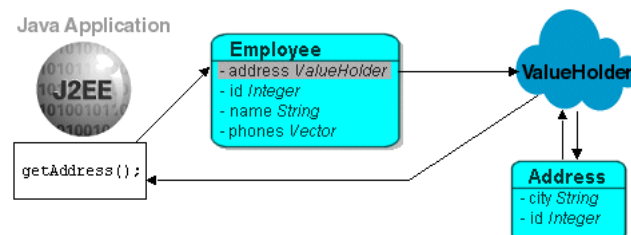
The first time the address is accessed, as in Figure 33-4, the `ValueHolder` reads and returns the `Address` object.

Figure 33-4 Initial Request



Subsequent requests for the address do not access the database, as shown in Figure 33-5.

Figure 33-5 Subsequent Requests



If you are using method access ("[Configuring Method Accessing](#)" on page 35-14), 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 35-7.

Transparent Indirect Container Indirection

Transparent indirect container (see ["Configuring Container Policy"](#) on page 35-26) 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.Hashtable`
- `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 35-4](#)) 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 35-14), 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 33-8).

To configure proxy indirection, you can use TopLink Workbench (see ["Using TopLink Workbench"](#) on page 35-15) or Java in an amendment method (see ["Configuring Proxy Indirection"](#) on page 35-9).

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 100](#), "[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 33-8) for relationships to an EJB because EJB do not directly implement their remote or local interfaces.

When using indirection with an EJB that your application serializes, you must consider the effect of any untriggered indirection objects at deserialization time (see "[Indirection, Serialization, and Detachment](#)" on page 33-9).

EJB 3.0 CMP and OC4J When using EJB 3.0 CMP 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 33-6) for all container-managed relationships (CMRs).

EJB 2.*n* CMP When using EJB 2.*n* 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 33-6) for all CMRs.

EJB 1.*n* CMP and BMP When using EJB 1.*n* CMP or BMP, the usage of indirection is up to the application.

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 102-12.

EJB 3.0 CMP and OC4J When using EJB 3.0 CMP with OC4J, indirection does not affect EJB 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 35-8).

EJB 1.*n* and 2.*n* CMP When using EJB 1.*n* or 2.*n* 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 EJB passivation and activation within the same JVM.

However, when you serialize an indirection-enabled EJB 1.*n* or 2.*n* EJB 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 indirection object.

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 22-4) and at the method level (["Configuring Method Accessing"](#) on page 35-14).

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](#)

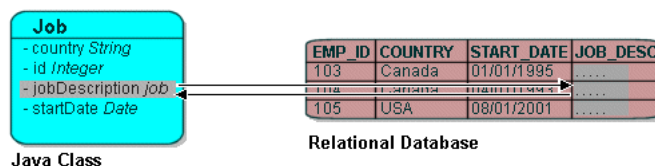
Note: You can use the mapping converters and transformers regardless of whether your data source is relational or nonrelational.

Serialized Object Converter

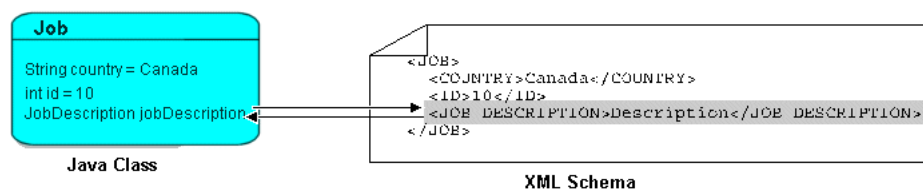
The serialized object converter is an extension of direct and direct collection mappings that lets you map complex objects into binary fields through Java object serialization. Serialized objects are normally stored in RAW or Binary Large Object (BLOB) fields in the database, or HEX or BASE64 elements in an XML document.

[Figure 33–6](#) 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 33–6 *Serialized Object Converter (relational)*



[Figure 33–8](#) 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 33–7 Serialized Object Converter (nonrelational)

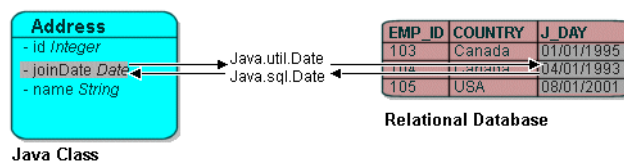
The serialized object converter relies on the Java serializer. Before you map a domain object with the serialized object converter, ensure that the domain object implements the `java.io.Serializable` interface (or inherits that implementation) and marks all nonserializable fields transient.

For more information, see ["Configuring a Serialized Object Converter"](#) on page 35-18.

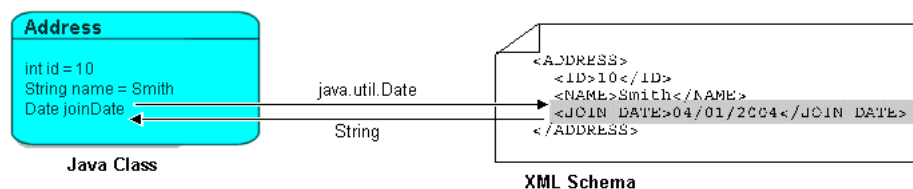
Type Conversion Converter

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 33–8](#) 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 33–8 Type Conversion Mapping (relational)

[Figure 33–9](#) illustrates a type conversion mapping (nonrelational). `java.util.Date` object is mapped to a `String` in a XML schema.

Figure 33–9 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 33-14).

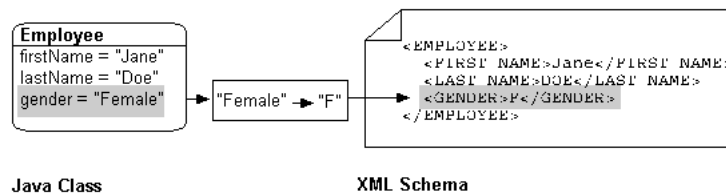
For more information, see ["Configuring a Type Conversion Converter"](#) on page 35-20.

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 33-10](#) 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 33-10 Object Type XML Converter



For more information, see ["Configuring an Object Type Converter"](#) on page 35-22.

Simple Type Translator

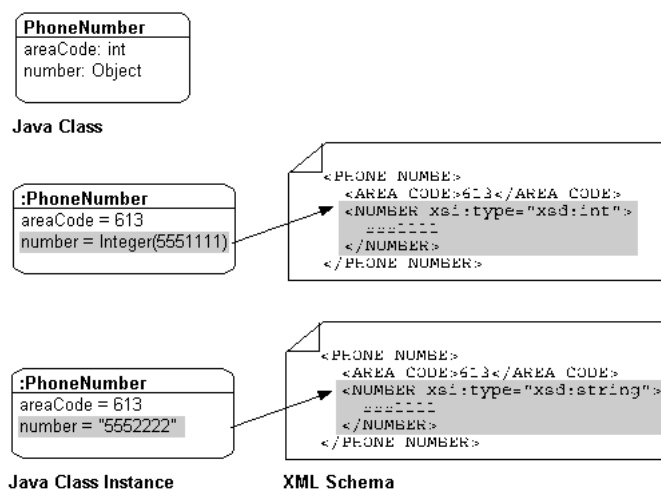
The simple type translator is an extension of direct and direct collection mappings that lets you automatically translate an XML element value to an appropriate Java type based on the element's `<type>` attribute as defined in your XML schema.

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 33–11 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 33–11 Simple Type Translator



By default, TopLink uses built-in read and write conversion pairs (see ["Default Read Conversions"](#) on page 33-13 and ["Default Write Conversions"](#) on page 33-14).

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 35-23.

Default Read Conversions Table 33–2 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 33–2 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
long	Long
short	Short

Table 33–2 (Cont.) Simple Type Translator Read Conversions

Schema Type	Java Type
string	String
time	Calendar
unsignedByte	Short
unsignedInt	Long
unsignedShort	Integer

Default Write Conversions Table 33–3 lists the built-in conversion pairs for writing XML. When a Java class attribute is of a type in Table 33–3 and the simple type translator is enabled, the corresponding schema type is specified on the element written.

Table 33–3 Simple Type Translator Write Conversions

Java Type	Schema Type
Byte[]	hexBinary
BigInteger	integer
Boolean	boolean
Byte	byte
Calendar	dateTime
GregorianCalendar	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 35-29): performs the object attribute transformation at read (unmarshal) time
- field transformer (see ["Configuring Field Transformer Associations"](#) on page 35-31): 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 vice versa.

For more information, see the following:

- ["Transformation Mapping"](#) on page 36-15
- ["EIS Transformation Mapping"](#) on page 56-17
- ["XML Transformation Mapping"](#) on page 65-31

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:

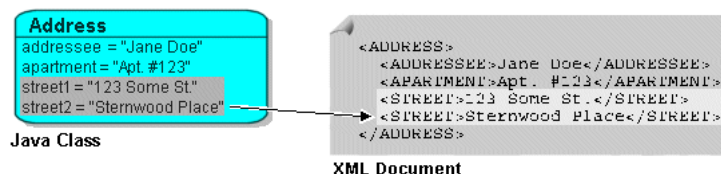
- [XPath by Position](#)
- [XPath by Path and Name](#)
- [XPath by Name](#)
- [Self XPath](#)

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 33-12](#) 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 33-12](#) 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.

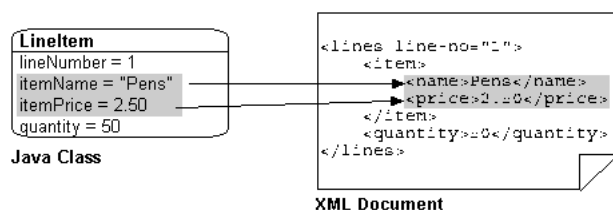
Figure 33-12 Mapping to an XML Document by Position



XPath by Path and Name

In an XML document, attributes and elements are uniquely identified by a combination of name and path. [Figure 33-13](#) 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`.

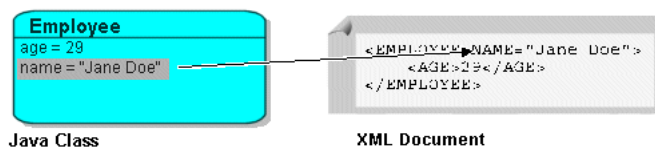
Other XML technologies force you to provide an object relationship for every level of nesting, resulting in the inclusion of many XML elements and classes simply to organize the data to satisfy this restriction. This produces an unnecessarily large object model that does not properly reflect the domain space.

Figure 33-13 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 33-14 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.

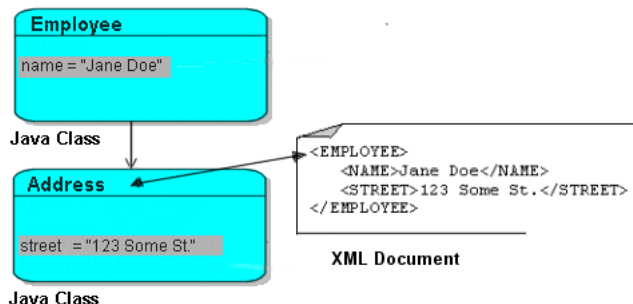
Figure 33-14 Mapping to a Simple XML Document by Name

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 33-15) or XPath by path and name (see ["XPath by Path and Name"](#) on page 33-15) 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 33-15 illustrates mapping to an XML document using the self XPath.

Figure 33-15 Mapping to a XML Document Using Self XPath

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 33-20).

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 33–4](#) shows.

Table 33–4 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 <code>xsd:union</code> Type	✓	
Mapping an <code>xsd:list</code> 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 65-10. The same applies to an `EISDirectMapping` with XML records (see ["EIS Direct Mapping"](#) on page 56-5).

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 `EISCompositedDirectCollectionMapping` (with XML records) or an `XMLCompositedDirectCollectionMapping` 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 `XMLDirectMapping` 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 `XMLCompositedDirectCollectionMapping`, see:

- ["Mapping to a Single Text Node With an XML Composite Direct Collection Mapping"](#) on page 65-17
- ["Mapping to a Single Attribute With an XML Composite Direct Collection Mapping"](#) on page 65-18
- ["Specifying the Content Type of a Collection With an XML Composite Direct Collection Mapping"](#) on page 65-20

The same applies to an `EISCompositedDirectCollectionMapping` (with XML records).

For more information about mapping to an `xsd:list` type using an `XMLDirectMapping`, see ["Mapping to a List Field"](#) ["Mapping to a List Field With an XML Direct Mapping"](#) on page 65-10. The same applies to an `EISDirectMapping` with XML records (see ["EIS Direct Mapping"](#) on page 56-5).

Mapping a List of Unions

Use an `EISCompositedDirectCollectionMapping` (with XML records) or an `XMLCompositedDirectCollectionMapping` 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 65-18. The same applies to an `EISCompositedDirectCollectionMapping` with XML records (see ["EIS Composite Direct Collection Mapping"](#) on page 56-6).

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 `XMLDirectMapping` 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 65-12. The same applies to an `EISDirectMapping` with XML records (see ["EIS Direct Mapping"](#) on page 56-5).
- ["Mapping to a Union of Lists With an XML Composite Direct Collection Mapping"](#) on page 65-19. The same applies to an `EISCompositeDirectCollectionMapping` with XML records (see ["EIS Composite Direct Collection Mapping"](#) on page 56-6).

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>
```

```
        </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 65-12. The same applies to an `EISDirectMapping` with XML records (see ["EIS Direct Mapping"](#) on page 56-5).

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 35-10) 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 group](#)

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 33-23).

all, choice, or sequence Structure

You can use the `jaxb:class` customization with an `all`, `choice`, or `sequence` structure. [Example 33-1](#) shows a `jaxb:class` customization of an `all` structure.

Example 33-1 `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 65-21. The same applies to an `EISCompositeObjectMapping` with XML records (see ["EIS Composite Object Mapping"](#) on page 56-7).

group Structure

You can use the `jaxb:class` customization with a group structure as [Example 33–2](#) shows.

Example 33–2 *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 65-21. The same applies to an `EISCompositeObjectMapping` with XML records (see ["EIS Composite Object Mapping"](#) on page 56-7).

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 33–3](#) shows a `jaxb:class` customization of a sequence structure containing a group structure.

Example 33–3 *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 65-21. The same applies to an `EISCompositeObjectMapping` with XML records (see ["EIS Composite Object Mapping"](#) on page 56-7).

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 33-4](#) shows a `jaxb:class` customization of a `group` structure containing a `sequence` structure.

Example 33-4 *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 65-21. The same applies to an `EISCompositeObjectMapping` with XML records (see ["EIS Composite Object Mapping"](#) on page 56-7).

group Structure Containing a group

You can use the `jaxb:class` customization with a group structure that contains another group structure as [Example 33–5](#) shows.

Example 33–5 `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 65-21. The same applies to an `EISCompositeObjectMapping` with XML records (see ["EIS Composite Object Mapping"](#) on page 56-7).

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 33–6](#). 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 33–6 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 33–7](#)) and define the mappings in the order specified by the XML schema.

Example 33–7 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 33–8](#)).

Example 33–8 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 35-25).

If you create a project and object model using the TopLink JAXB compiler (see ["Creating an XML Project From an XML Schema"](#) on page 21-6), 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 20-12).

Understanding the Mapping API

All the mapping classes are derived from the `DatabaseMapping` class.

Table 33-5 Platform and Mapping Package Compatibility

Platform	Mapping Package
DatabasePlatform	<code>oracle.toplink.mappings</code>
For relational projects	<code>oracle.toplink.xdb</code> <code>oracle.toplink.objectrelational</code>
EISPlatform	<code>oracle.toplink.eis</code>
For EIS projects	<code>oracle.toplink.mappings.TransformationMapping</code>
XMLPlatform	<code>oracle.toplink.ox</code>
For XML projects	<code>oracle.toplink.mappings.TransformationMapping</code>

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 20-6.

For more information about relational mappings, see ["Understanding Relational Mappings"](#) on page 36-1 and ["Configuring a Relational Mapping"](#) on page 37-1.

Note: Do not confuse relational mappings with object-relational mappings (see ["Object-Relational Mappings"](#) on page 33-26). 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 38, "Configuring a Relational Direct-to-Field Mapping"](#)
- [Chapter 39, "Configuring a Relational Direct-to-XMLType Mapping"](#)
- [Chapter 40, "Configuring a Relational One-to-One Mapping"](#)
- [Chapter 41, "Configuring a Relational Variable One-to-One Mapping"](#)
- [Chapter 42, "Configuring a Relational One-to-Many Mapping"](#)
- [Chapter 43, "Configuring a Relational Many-to-Many Mapping"](#)
- [Chapter 44, "Configuring a Relational Aggregate Collection Mapping"](#)
- [Chapter 45, "Configuring a Relational Direct Collection Mapping"](#)
- [Chapter 46, "Configuring a Relational Direct Map Mapping"](#)
- [Chapter 47, "Configuring a Relational Aggregate Object Mapping"](#)
- [Chapter 48, "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 20-6.

For more information about object-relational mappings, see ["Understanding Object-Relational Mappings"](#) on page 49-1 and ["Configuring an Object-Relational Mapping"](#) on page 50-1.

Note: Do not confuse object-relational mappings with relational mappings (see ["Relational Mappings"](#) on page 33-25). 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 51, "Configuring an Object-Relational Structure Mapping"](#)
- [Chapter 52, "Configuring an Object-Relational Reference Mapping"](#)
- [Chapter 53, "Configuring an Object-Relational Array Mapping"](#)
- [Chapter 54, "Configuring an Object-Relational Object Array Mapping"](#)
- [Chapter 55, "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 20-9.

For more information about XML mappings, see ["Understanding XML Mappings"](#) on page 65-1 and ["Configuring an XML Mapping"](#) on page 66-1.

TopLink provides the following XML mappings:

- [Chapter 67, "Configuring an XML Direct Mapping"](#)
- [Chapter 68, "Configuring an XML Composite Direct Collection Mapping"](#)
- [Chapter 69, "Configuring an XML Composite Object Mapping"](#)
- [Chapter 70, "Configuring an XML Composite Collection Mapping"](#)
- [Chapter 71, "Configuring an XML Any Object Mapping"](#)
- [Chapter 72, "Configuring an XML Any Collection Mapping"](#)
- [Chapter 73, "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 33-25.

Use EIS mappings in EIS projects. For more information, see ["EIS Projects"](#) on page 20-7.

For more information about EIS mappings, see ["Understanding EIS Mappings"](#) on page 56-1 and ["Configuring an EIS Mapping"](#) on page 57-1.

TopLink provides the following EIS mappings:

- [Chapter 58, "Configuring an EIS Direct Mapping"](#)
- [Chapter 59, "Configuring an EIS Composite Direct Collection Mapping"](#)
- [Chapter 60, "Configuring an EIS Composite Object Mapping"](#)
- [Chapter 61, "Configuring an EIS Composite Collection Mapping"](#)
- [Chapter 62, "Configuring an EIS One-to-One Mapping"](#)
- [Chapter 63, "Configuring an EIS One-to-Many Mapping"](#)
- [Chapter 64, "Configuring an EIS Transformation Mapping"](#)

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 33-1.

During development, you can create mappings individually (see ["Creating Mappings Manually During Development"](#) on page 34-1) or you can allow TopLink Workbench to automatically map all attributes (see ["Creating Mappings Automatically During Development"](#) on page 34-2).

For EJB 2.0 CMP projects using OC4J, you can also configure TopLink to create mappings automatically at deployment time (see ["Creating Mappings Automatically During Deployment"](#) on page 34-2).

After you create a mapping, you must configure its various options (see [Chapter 35](#), ["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 **Navigators**. Its properties appear in the Editor.

2. On the **Descriptor Info** tab, associate the descriptor with the data source (see [Chapter 28, "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 35, "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 29-2) 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 2.0 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 2.0 CMP EAR at Deployment Time"](#) on page 21-10
- ["Default Mapping in EJB 2.0 or 3.0 CMP Projects Using OC4J at Run Time"](#) on page 33-4

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 34-1](#) shows an example of a direct-to-field mapping to a BLOB value.

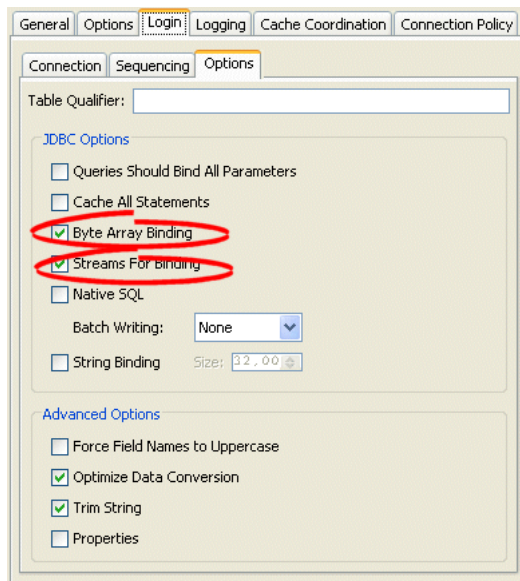
Example 34-1 Mapping a BLOB with a Direct-To-Field Mapping

```
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 `shouldBindAllParameters` is false) and enable the streams for binding to ensure that stream binding is used for byte arrays. [Example 34-2](#) shows configuring the parameter bindings in TopLink Workbench. [Example 34-3](#) shows configuring the parameter bindings in TopLink Workbench.

Example 34–2 Configuring Parameter Bindings in TopLink Workbench

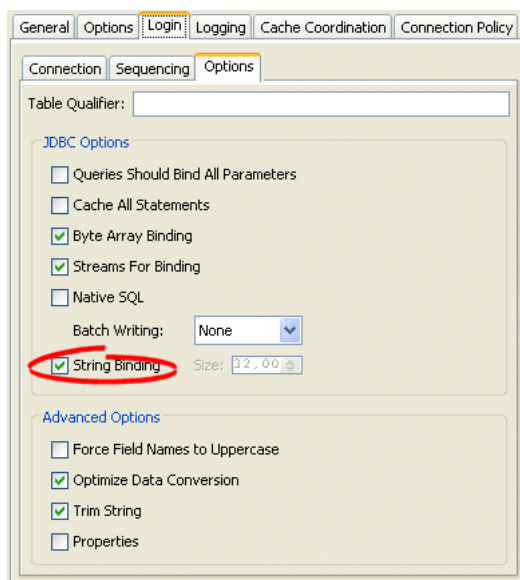


Example 34–3 Configuring Parameter Bindings in Java Code

```
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 34–4](#) shows configuring string binding in TopLink Workbench. [Example 34–5](#) shows configuring string binding in TopLink Workbench.

Example 34–4 Configuring String Binding in TopLink Workbench



Example 34–5 Configuring String Binding in Java Code

```
login.useStringBinding();
```

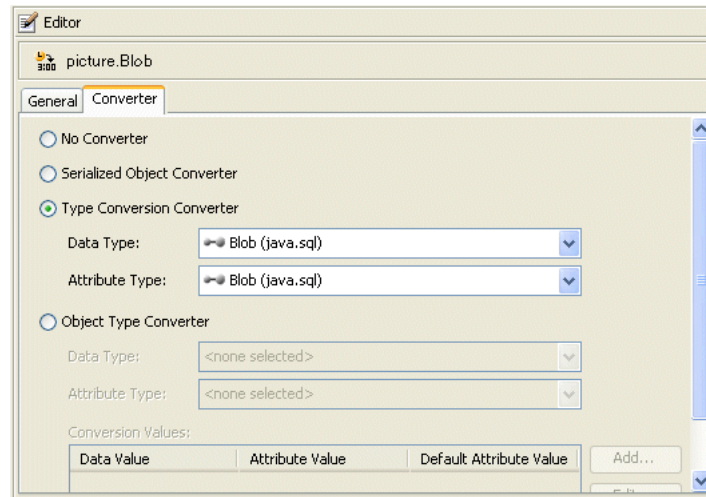

Using the Oracle JDBC Thin Driver

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 35-20) 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 34-6](#) shows a type conversion mapping to a BLOB value in TopLink Workbench. [Example 34-6](#) shows the Java code for the same mapping.

Example 34-6 Mapping a BLOB in TopLink Workbench



Example 34-7 Mapping a BLOB in Java Code

```
TypeConversionMapping pictureMapping = new TypeConversionMapping();
pictureMapping.set.AttributeName("picture");
pictureMapping.set.FieldName("IMAGE.PICTURE");
pictureMapping.set.FieldClassification(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:

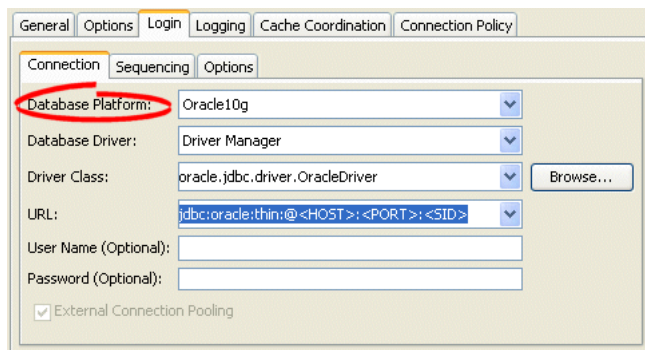
```
login.usePlatform(new Oracle8Platform());
```
 - For Oracle9i Database Server, use the `oracle.toplink.oraclespecific.Oracle9Platform` class:

```
login.usePlatform(new Oracle9Platform());
```
 - Oracle Database 10g, use the `oracle.toplink.oraclespecific.Oracle10Platform` class:

```
login.usePlatform(new Oracle10Platform());
```

In TopLink Workbench, select the appropriate platform in the Database editor.

Figure 34–1 *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.

Configuring a Mapping

This chapter describes how to configure TopLink mappings.

[Table 35–1](#) 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 35–1 *Configuring TopLink Mappings*

If you are creating...	See...
Relational Mappings	Chapter 37, "Configuring a Relational Mapping"
Object-Relational Mappings	Chapter 50, "Configuring an Object-Relational Mapping"
EIS Mappings	Chapter 57, "Configuring an EIS Mapping"
XML Mappings	Chapter 66, "Configuring an XML Mapping"

[Table 35–2](#) lists the configurable options shared by two or more TopLink mapping types.

For more information, see the following:

- ["Mapping Creation Overview"](#) on page 34-1
- ["Understanding Mappings"](#) on page 33-1

Configuring Common Mapping Options

[Table 35–2](#) 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 35–1](#)

Table 35–2 *Common Mapping Options*

Option	Type	TopLink Workbench	Java
"Configuring Read-Only Mappings" on page 35-2	Basic	✓	✓
"Configuring Indirection" on page 35-3	Basic	✓	✓
"Configuring XPath" on page 35-10	Basic	✓	✓
"Configuring a Default Null Value at the Mapping Level" on page 35-12	Basic	✓	✓
"Configuring Method Accessing" on page 35-14	Advanced	✓	✓
"Configuring Private or Independent Relationships" on page 35-16	Advanced	✓	✓
"Configuring Mapping Comments" on page 35-18	Advanced	✓	

Table 35–2 (Cont.) Common Mapping Options

Option	Type	TopLink Workbench	Java
"Configuring a Serialized Object Converter" on page 35-18	Advanced	✓	✓
"Configuring a Type Conversion Converter" on page 35-20	Advanced	✓	✓
"Configuring an Object Type Converter" on page 35-22	Advanced	✓	✓
"Configuring a Simple Type Translator" on page 35-23	Advanced	✓	✓
"Configuring a JAXB Typesafe Enumeration Converter" on page 35-25	Advanced		✓
"Configuring Container Policy" on page 35-26	Advanced	✓	✓
"Configuring Attribute Transformer" on page 35-29	Advanced	✓	✓
"Configuring Field Transformer Associations" on page 35-31	Advanced	✓	✓
"Configuring Mutable Mappings" on page 35-33	Advanced	✓	✓
"Configuring Bidirectional Relationship" on page 35-34	Advanced	✓	✓
"Configuring the Use of a Single Node" on page 35-36	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 28-4).

[Table 35–3](#) summarizes which mappings support this option.

Table 35–3 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 35–1 General Tab, Read-Only Option

The screenshot shows the 'General' tab in the TopLink Workbench. The 'Database Field' dropdown is set to 'ADDRESS.CITY (VARCHAR2)'. Below it, the 'Method Accessing' section has two dropdowns for 'Get Method' and 'Set Method', both set to '<none selected>'. The 'Default Null Value' section has a 'Type' dropdown set to '<none selected>' and an empty 'Value' text box. The 'Read-Only' checkbox is checked and circled in red. At the bottom is a 'Comment' text area.

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 35–1](#) shows how to use these methods with a class that has a read-only attribute named `phones`.

Example 35–1 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 35–4 summarizes which mappings support this option.

Table 35–4 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 33-6) for one-to-one mappings and transparent indirect container indirection (see ["Transparent Indirect Container Indirection"](#) on page 33-8) 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 33-9).

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 33-9).

For more information, see ["Indirection"](#) on page 33-5.

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 35-2 General Tab, Indirection Options

The screenshot shows the 'General' tab of a configuration window. At the top, there are two tabs: 'General' and 'Table Reference'. Below them is a 'Reference Descriptor' dropdown menu showing 'Address (examples.servletjsp.model)'. Under 'Method Accessing', there are 'Get Method' and 'Set Method' dropdowns, both set to '<none selected>'. A group of checkboxes includes 'Read-Only', 'Private Owned', 'Batch Reading', 'Use Joining', and 'Use Indirection' (which is checked and circled in red). Below these is a radio button group with 'ValueHolder' selected and 'Proxy' unselected. At the bottom, there is a 'Maintains Bidirectional Relationship' checkbox and a 'Relationship Partner' dropdown set to '<none selected>'. A 'Comment' text area is at the very bottom.

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 33-6 for more information.
Proxy	Specify that the mapping uses Proxy indirection. See "Proxy Indirection" on page 33-8 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 35-8) or define the `ValueHolder` in the constructor as follows:

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

[Example 35-2](#) illustrates the `Employee` class using `ValueHolder` indirection. The class definition conceals the use of `ValueHolder` within the existing getter and setter methods.

Example 35-2 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 35-3](#) shows how to configure a one-to-one mapping to the `address` attribute.

Example 35-3 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 35-14), 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 35-2](#) 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 35-4 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 35-3](#) shows how to configure a one-to-one mapping to the address attribute.

Example 35-5 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 CMP on OC4J (see "[EJB 3.0 CMP and OC4J](#)" on page 33-9), if your application serializes any indirection-enabled EJB (see "[Indirection, Serialization, and Detachment](#)" on page 33-9), 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>\j lib`) 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 35-2](#) 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 35-6 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 35-3](#) shows how to configure a one-to-one mapping to the `addresses` attribute.

Example 35-7 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 35-8](#) illustrates an `Employee` to `Address` one-to-one relationship.

Example 35-8 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 35-8](#), 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 35–5 summarizes which mappings support this option.

Table 35–5 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 31-5).

- 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 javax.class Customization"](#) on page 33-20).

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 31-2 or ["Configuring Schema Context for an XML Descriptor"](#) on page 32-1).

For more information, see ["Mappings and XPath"](#) on page 33-15.

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 35-3 General Tab, XPath Options

Figure 35-4 XPath Options for Composite Object Mappings

Click **Browse** and select the XPath to map to this attribute (see ["Choosing the XPath"](#)).

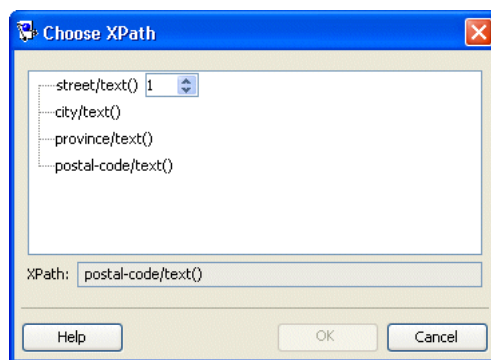
For an EIS composite object mapping using XML records or an XML composite object mapping, you can choose one of the following:

- **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 33-16) 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 33-20).

Choosing the XPath

From the Choose XPath dialog box, select the XPath and click **OK**. TopLink Workbench builds the complete XPath name.

Figure 35–5 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 35–6](#) summarizes which mappings support this option.

Table 35–6 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 35–6 (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 85-5).

Using TopLink Workbench

To configure a default null value for a mapping, use this procedure.

1. Select the mapped attribute in the **Navigators**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.

Figure 35–6 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 <code>null</code> . 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 35-7](#) summarizes which mappings support this option.

Table 35-7 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 22-4.

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 35–7 General Tab, Method Accessing Options

The screenshot shows the 'General' tab of a configuration window. At the top, there are two tabs: 'General' and 'Converter'. Below them, the 'Database Field' is set to 'ADDRESS.CITY (VARCHAR2)'. The 'Method Accessing' checkbox is checked and highlighted with a red circle. Below this, there are two dropdown menus: 'Get Method' and 'Set Method', both set to '<none selected>'. Further down, there is a 'Default Null Value' section with a 'Type' dropdown set to '<none selected>' and a 'Value' text field. Below that is a 'Read-Only' checkbox. At the bottom is a 'Comment' text area.

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 <code>get</code> method.
Set Method	Select a specific <code>set</code> 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 22-4 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 35–9](#) 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 35–9 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 35–8 summarizes which mappings support this option.

Table 35–8 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 35–8 (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 **Navigators**. Its properties appear in the Editor.
2. Click the **General** tab. The General tab appears.

Figure 35–8 General tab, Private Owned option

The screenshot shows the 'General' tab of the TopLink Workbench. The 'Reference Descriptor' is set to 'Address (examples.servletjsp.model)'. The 'Private Owned' checkbox is selected and circled in red. Other options include 'Method Accessing', 'Read-Only', 'Batch Reading', 'Use Joining', 'Use Indirection' (checked), 'ValueHolder' (selected), 'Proxy', 'Maintains Bidirectional Relationship', and 'Relationship Partner'.

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 35–9](#) shows how to use these methods with a class that has a privately owned attribute, `phones`, in a mapping.

Example 35–10 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 wish: 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 35–9](#) summarizes which mappings support this option.

Table 35–9 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 35–9 General Tab, Comment

The screenshot shows the 'General' tab in the TopLink Workbench. The 'Database Field' is set to 'ADDRESS.CITY (VARCHAR2)'. Below this, there are several sections with checkboxes and dropdown menus. The 'Method Accessing' section has checkboxes for 'Method Accessing', 'Get Method', and 'Set Method', all of which are unchecked. The 'Default Null Value' section has checkboxes for 'Default Null Value', 'Type', and 'Value', all of which are unchecked. The 'Read-Only' checkbox is unchecked. The 'Comment' text area is highlighted with a red circle.

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 33-10.

[Table 35–10](#) summarizes which mappings support this option.

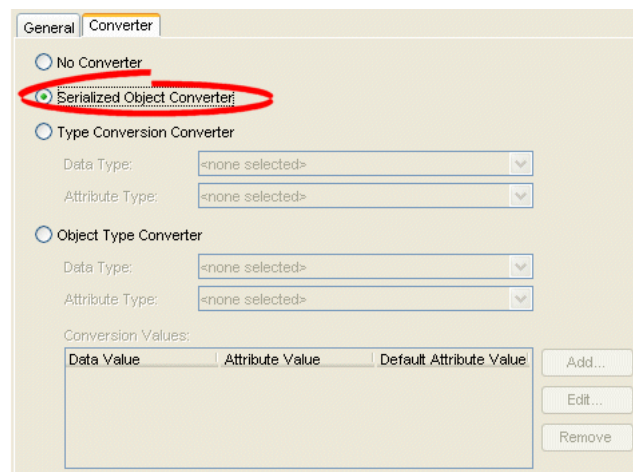
Table 35–10 Mapping Support for Serialized Object Converter

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 35–10 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 35–11](#) shows.

Example 35–11 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 33-11.

[Table 35–11](#) summarizes which mappings support this option.

Table 35–11 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 **Navigators**. Its properties appear in the Editor.
2. Click the **Converter** tab. The Converter tab appears.
3. Select the **Type Conversion Converter** option.

Figure 35–11 Converter Tab, Type Conversion Converter Option

The screenshot shows a configuration window with two tabs: 'General' and 'Converter'. The 'Converter' tab is active. It contains three radio button options: 'No Converter', 'Serialized Object Converter', and 'Type Conversion Converter'. The 'Type Conversion Converter' option is selected and circled in red. Below this, there are two dropdown menus: 'Data Type' and 'Attribute Type', both set to 'String (java.lang)'. Below these is an unselected 'Object Type Converter' section with its own 'Data Type' and 'Attribute Type' dropdowns, both set to '<none selected>'. At the bottom, there is a section titled 'Conversion Values' containing a table with three columns: 'Data Value', 'Attribute Value', and 'Default Attribute Value'. To the right of the table are three buttons: 'Add...', 'Edit...', and 'Remove'.

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 35–12](#) shows.

Example 35–12 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 33-12.

[Table 35-12](#) summarizes which mappings support this option.

Table 35-12 Mapping Support for Object Type Converter

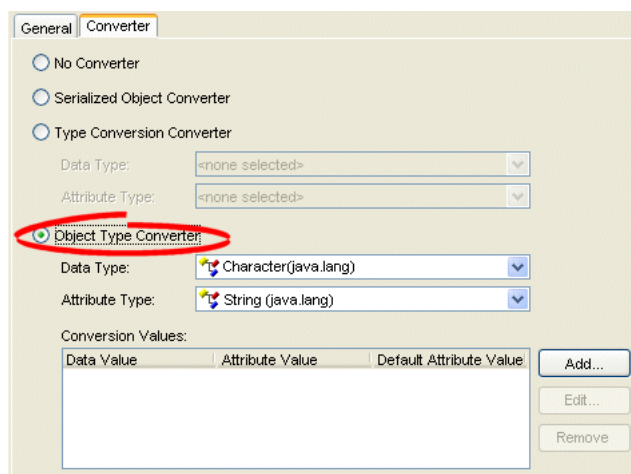
Mapping	Using TopLink Workbench	Using Java
Relational Mappings	✓	✓
Object-Relational Mappings		
Object-Relational Array Mapping		✓
EIS Mappings		
EIS Direct Mapping	✓	✓
EIS Composite Direct Collection Mapping	✓	✓
XML Mappings		
XML Direct Mapping	✓	✓
XML Composite Direct Collection Mapping	✓	✓

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 35-12 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 35–13](#) shows.

Example 35–13 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 33-12.

[Table 35-13](#) summarizes which mappings support this option.

Table 35-13 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 35-13 General Tab, Use XML Schema "type" Attribute Option

The screenshot shows the 'General' tab of a configuration window. It has two sub-tabs: 'General' (selected) and 'Converter'. Under the 'General' sub-tab, there is a section for 'XML Field' with an 'XPath:' text box and a 'Browse...' button. Below this is a checkbox labeled 'Use XML Schema "type" attribute', which is circled in red. Further down, there are sections for 'Method Accessing' (with 'Get Method' and 'Set Method' dropdowns), 'Default Null Value' (with 'Type' and 'Value' fields), and a 'Read-Only' checkbox with a 'Comment' text box.

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 35–14](#) 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 35–14 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 33-24.

[Table 35–14](#) summarizes which mappings support this option.

Table 35–14 Mapping Support for JAXB Typesafe Enumeration Converter

Mapping	Using TopLink Workbench	Using Java
EIS Mappings¹		
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 31-5).

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 35-26).

If you create a project and object model using the TopLink JAXB compiler (see ["Creating an XML Project From an XML Schema"](#) on page 21-6), 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 20-12).

Using Java

To configure a mapping with a `JAXBTypesafeEnumConverter` in Java, use a descriptor amendment method (see ["Configuring Amendment Methods"](#) on page 28-78). [Example 35-15](#) 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 35-15 *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 35–15 summarizes which mappings support this option.

Table 35–15 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 35–14 General Tab, Advanced Container Options

The screenshot shows the 'General' tab of a configuration window. The 'Reference Descriptor' is set to '<none selected>'. Under 'Method Accessing', 'Get Method' and 'Set Method' are also set to '<none selected>'. There are checkboxes for 'Read-Only', 'Private Owned', 'Batch Reading', and 'Use Indirection' (which is checked). Below these are radio buttons for 'ValueHolder' (selected) and 'Transparent'. A 'Maintains Bidirectional Relationship' checkbox is also present, with a 'Relationship Partner' dropdown set to '<none selected>'. The 'Advanced Container Options' section is highlighted with an orange circle and contains the 'Container Type' section with radio buttons for 'List', 'Set', 'Collection', and 'Map'. The 'List' option is selected. Below this is an 'Override Default Class' field containing 'java.util.Vector' and a 'Browse...' button. The 'Key Method' dropdown is set to '<none selected>'. At the bottom is a 'Comment' text area.

Field ¹	Description
Container Type	Specify the type of <code>Collection</code> class to use: ■ List—use a <code>java.util.List</code> ■ Set—use a <code>java.util.Set</code> ■ Collection—use a <code>java.util.Collection</code> ■ Map—use a <code>java.util.Map</code>
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 <code>java.util.Collection</code> 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 <code>Hashtable</code> or <code>Map</code> . This method must return an object that is a valid key in the <code>Hashtable</code> or <code>Map</code> .

¹ Not all mappings support all options. For more information, see [Table 35–15](#).

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 35–16](#) shows how to configure an `ObjectArrayMapping` to use a `java.util.ArrayList` container class.

Example 35–16 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 35-31).

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 35–16](#) summarizes which mappings support this option.

Table 35–16 Mapping Support for Attribute Transformer

Mapping	Using TopLink Workbench	Using Java
Relational Mappings		
Transformation Mapping	✓	✓

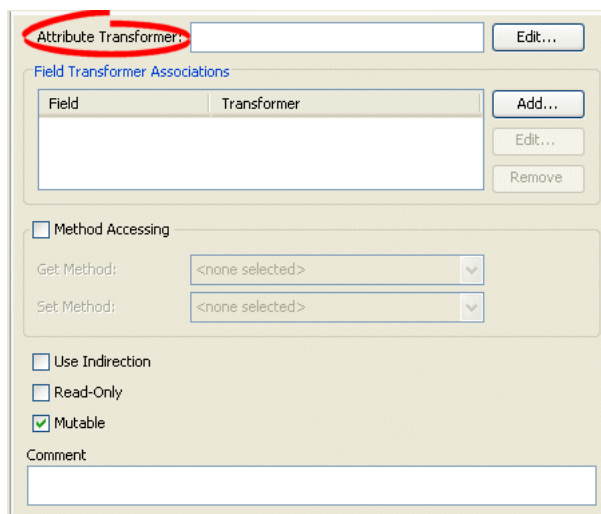
Table 35–16 (Cont.) Mapping Support for Attribute Transformer

Mapping	Using TopLink Workbench	Using Java
EIS Mappings		
EIS Transformation Mapping	✓	✓
XML Mappings		
XML Transformation Mapping	✓	✓

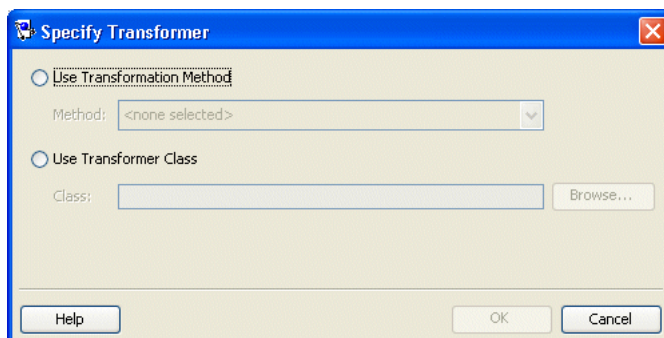
Using TopLink Workbench

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 35–15 Transformation Mapping, Attribute Transformer Field

2. Click **Edit**. The Specify Transformer dialog box appears.

Figure 35–16 Specify Transformer Dialog Box

Click **Edit**. The Specify Transformer dialog box appears. 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.Transformers.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 35-29) 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 35-17](#) summarizes which mappings support this option.

Table 35-17 Mapping Support for Field Transformer

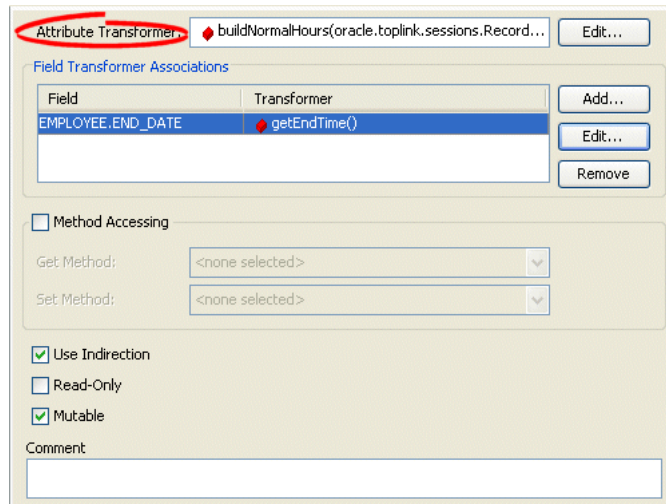
Mapping	Using TopLink Workbench	Using Java
Relational Mappings		
Transformation Mapping	✓	✓
EIS Mappings		
EIS Transformation Mapping	✓	✓
XML Mappings		
XML Transformation Mapping	✓	✓

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 35–17 Transformation Mapping, Field Transformer Associations



2. Click **Add** to add the necessary Field Transformer Associations for the mapping.

To add a new association, click **Add**. Continue with ["Specifying Field-to-Transformer Associations"](#) on page 35-32.

To change an existing association, click **Edit**. Continue with ["Specifying Field-to-Transformer Associations"](#) on page 35-32.

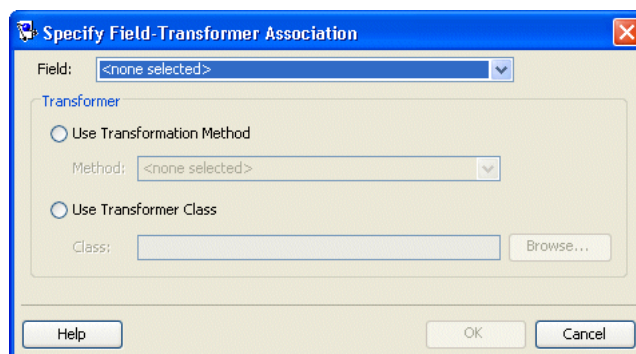
To delete an existing association, select the field transformation association and click **Delete**.

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 35–18 Specify Field-Transformer Association Dialog Box



2. 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.Transformers.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.Transformers.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 28-69).

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 35-18](#) summarizes which mappings support this option.

Table 35-18 Mapping Support for Mutable Mappings

Mapping	Using TopLink Workbench	Using Java
Relational Mappings		
Transformation Mapping	✓	✓
EIS Mappings		
EIS Transformation Mapping	✓	✓

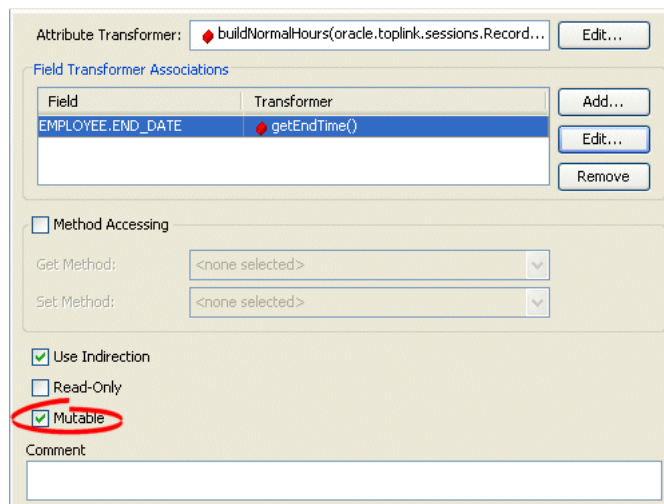
Table 35–18 (Cont.) Mapping Support for Mutable Mappings

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 35–19 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 35–18 summarizes which mappings support this option.

Table 35–19 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 35–20 General tab, Maintains Bidirectional Relationship option

The screenshot shows the 'General' tab of the TopLink Workbench. The 'Reference Descriptor' is set to 'Address (examples.servletjsp.model)'. The 'Method Accessing' section has 'Get Method' and 'Set Method' both set to '<none selected>'. The 'Read-Only' checkbox is unchecked. The 'Private Owned' checkbox is checked. The 'Batch Reading' checkbox is unchecked. The 'Use Joining' checkbox is unchecked. The 'Use Indirection' checkbox is checked. The 'ValueHolder' radio button is selected, and the 'Proxy' radio button is unselected. The 'Maintains Bidirectional Relationship' checkbox is checked and highlighted with a red circle. The 'Relationship Partner' is set to '<none selected>'. There is a 'Comment' text area at the bottom.

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 35–6](#) 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 35–6](#) summarizes which mappings support this option.

Table 35–20 Mapping Support for Use Single Node

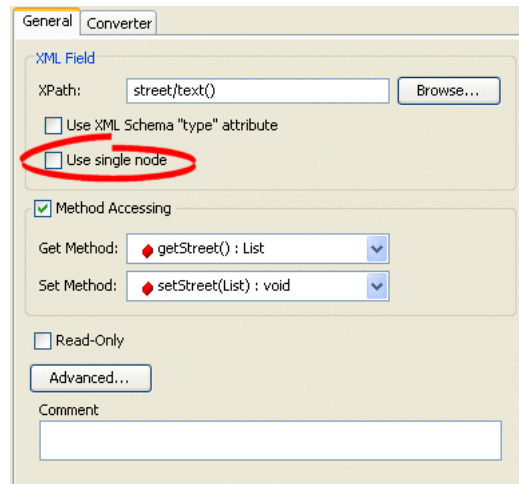
Mapping	Using TopLink Workbench	Using Java
EIS Mappings ¹		
EIS Direct Mapping		✓
EIS Composite Direct Collection Mapping	✓	✓
XML Mappings		
XML Direct Mapping		✓
XML Composite Direct Collection Mapping	✓	✓

¹ When used with XML records only (see "Configuring Record Format" on page 31-5).

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 35–21 General Tab, Use Single Node Option

To configure the mapping to unmarshal (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 35–17](#) shows how to use this method with an `XMLDirectMapping`:

Example 35–17 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);
```

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 includes 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.

A

access

- data access, 84-1
- direct, 35-32
- modifiers, 4-42
- optimizing data access, 11-14
- remote sessions, 75-31

access method

- direct, 35-14
- generating, 4-30
- mappings, 35-15
- method, 35-14
- specifying, 35-32

access modifiers, classes, 4-42

acquiring

- client sessions, 78-6
- sessions, at runtime, 75-5
- unit of work, 101-1

activating descriptors, 4-10

Add Named Query dialog, 28-12, 28-18, 28-20

Add New Class dialog, 4-40, 4-53

Add New Table button, 4-22

Add or Refresh Class button, 4-50

addConstraingDependencies(), 102-16

address

- multicast group, 91-4
- multicast port, 91-5

Add/Update Existing Tables from Database

- button, 4-23

advanced properties for descriptors, 22-7

After Load tab, 28-79

aggregate collection relational mappings

- and EJB, 36-11
- configuring, 44-1
- understanding, 36-10

aggregate descriptors

- about, 26-5
- aggregate object mapping, 26-5
- EIS projects, 26-8
- EJB 3.0, 26-6
- inheritance, 26-17
- relational projects, 26-5
- XML projects, 26-8

aggregate object relational mapping

- aggregate descriptors, and, 26-5

aggregate object relational mappings

- configuring, 47-1
- understanding, 36-12

aggregation, isolated client sessions, 75-24

AllFieldsLockingPolicy, 26-20

allows none, 28-18, 97-14

allows null, 28-18, 97-14

amending descriptors, 2-20, 26-5, 28-78

- see also* after load

announcement delay, 91-11

Ant, integrating with Oracle TopLink

- Workbench, 4-53

any collection XML mappings

- configuring, 72-1
- understanding, 65-29

any object XML mappings

- configuring, 71-1
- understanding, 65-27

application development

- deploying, 10-1
- mapping, 2-20
- querying, 2-13, 74-2
- troubleshooting, 15-1

application layer, remote sessions, 75-30

application servers

- EJB support, customizing, 7-25
- integrating with Oracle TopLink, 7-1
- logging, 75-9
- 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, 1-5, 2-33
- cache, 90-1
- choosing, 2-3
- CMP, 1-5, 2-27
- EIS, 2-4, 56-5
- EJB entity beans, 1-5, 2-27, 2-33
- EJB session bean facade, 1-5, 2-24
- locking, 2-10
- optimistic locking, 2-10

- Oracle TopLink, 1-1
- pessimistic locking, 2-10
- selecting, 2-5
- session brokers, 75-26
- sessions, 75-2
- three-tier, 1-4, 2-21
- two-tier, 1-6, 2-23
- unit of work, 100-1
- web services, 1-5, 2-36
- arguments, binding in query, 29-6
- array
 - dimensionality, 4-45, 4-49
 - object-relational mappings, 49-2
- AsOfClause, 99-2
- asynchronous change propagation, 91-2
- AttributeChangeTrackingPolicy
 - about, 100-8
 - OC4J CMP integration, 100-8
 - OC4J EJB 1.x CMP integration, 100-8
 - OC4J EJB 2.x CMP integration, 100-8
 - OC4J EJB 3.0 CMP integration, 100-8
 - other application servers, 100-8
- attributes
 - adding to descriptors, 4-44
 - array dimensionality, 4-45, 4-49
 - changes, tracking, 28-72
 - final, 4-45, 4-48
 - in TopLink Workbench, Navigator window, 4-9
 - lazy loading. *see* fetch groups
 - static, 4-45, 4-48
 - transforming, 35-29, 35-31, 36-15
 - transient, 4-45, 4-48
 - unmapping, 34-6
 - volatile, 4-45, 4-48
- Attributes tab, 4-44, 4-48
- Attunity Connect platform, 84-3
- auditing
 - authentication, 84-6
 - unit of work, 102-20
- authentication
 - about, 84-5
 - auditing, 84-6
 - proxy authentication, 84-5
 - simple JDBC authentication, 84-5
 - three-tier architecture, 84-5
 - two-tier architecture, 84-5
- Automap, 34-2
- automapping descriptors
 - about, 34-2
 - see also* mappings
- automatic table generation
 - about, 33-4
 - configuring, 8-12

B

- Base64 encoded strings, 35-25
- batch options
 - mappings, 37-7
 - writing, 11-15

- batch reading
 - in query objects, 98-10
 - read optimization, and, 11-19
- batch writing
 - about, 11-15, 11-26, 86-11
 - dynamic, 11-15
 - dynamic, `setMaxBatchWritingSize()`, 11-15
 - MySQL4 platform, 86-9
 - non-parameterized, 11-15
 - parameterized, 11-15
 - `setMaxBatchWritingSize()`, 11-15
- BEA WebLogic
 - deploying to, 10-2
 - deployment exceptions, 15-3
 - modifying persistence descriptor, 8-13
 - setting classpath, 7-15
 - setting shared library, 7-15
 - transport layer, 75-31
 - using a security manager, 7-20
- @Bean fetch=lazy, 33-9
- beans
 - session beans, 2-25
 - stateful beans, 2-25
 - stateless beans, 2-25
- bidirectional relationships
 - about, 36-2
 - generating, 4-31
 - in one-to-one mappings, 35-34
 - target keys, 36-5
 - with indirection, 35-35
- `bindAllParameters()` method, 23-7
- bindings
 - arguments, 29-6
 - input paramters, 98-19
 - JAXB, 20-10
 - LOB mappings, 34-3
 - see* parameter binding
- BLOB
 - mapping to, 34-2
- BMP
 - and EJB 1.1, 2-34
 - and EJB 2.0, 2-34
 - and TopLink, 2-33
 - deployment files, 8-9
 - descriptors, 26-3, 28-44
 - packaging for deployment, 9-5
- boolean logic in expressions, 97-3
- branch classes, 26-12, 26-13
- buttons. *see* toolbars
- bye array binding, LOB mappings, 34-3
- Byte array Base64, 35-25

C

- cache
 - about, 2-12, 2-18, 90-1
 - architecture, 90-1
 - configuring, 90-6, 99-21
 - coordination, 90-9, 91-1, 92-1, 93-1
 - descriptor level, 28-35

- disabling during read query, 96-34
- distributed, 90-9
- expiration, 22-19, 28-40, 99-22
- expression limitations, 96-31
- identity maps, using, 75-29
- in-memory queries, 96-30, 96-31
- internal query object cache, 96-36, 99-21, 99-22
- invalidation, 22-19, 28-40, 90-7, 90-8
- isolated client sessions, 75-24
- isolation, 90-6, 90-9
- object cache, 96-36
- object cascading refresh, 96-35
- object refresh, 96-35
- optimizing, 11-13
- project level, 22-13
- queries, 90-6, 96-29
- query cache, 96-36
- `readObject` method, and, 98-2
- refreshing, 28-27, 90-8, 96-35
- restrictions, 96-36
- service channel, 91-3
- sessions, 75-3, 75-32, 90-2
- stale data, 90-6
- storing query results, 96-36, 99-21
- type and size, 22-13, 28-35
- unit of work cache, 90-2
- cache coordination
 - about, 90-9
 - application server clustering, and, 7-4
 - avoiding stale data, 90-7
 - CMP projects, 8-11
 - EJB Entity Beans with BMP architecture, and, 2-36
 - explicit query refreshes, 90-7
 - JMS, 92-1
 - `orion-ejb-jar.xml`, 8-11
 - packet time-to-live, 91-15
 - permissions, 7-23
 - RMI, 93-1
- cache invalidation, avoiding stale data, 90-7
- cache synchronization. *see* cache coordination
- `cacheAllStatements()`, 23-7
- `cacheQueryResults()`, 28-25
- cache-synchronization property, 8-11
- Caching tab, 28-35, 28-38, 28-39, 28-42
- call
 - call queries, 96-16
 - EIS, 98-24
 - EJBQLCall, 98-23
 - SQLCall, 98-18
 - StoredFunctionCall, 98-22
 - StoredProcedureCall, 98-20
- Call object, queries, 96-3
- cascading
 - object refresh, 96-35
 - optimistic version locking, 26-18
 - write queries, compared to non-cascading, 96-14, 98-13
- catalog, database, 4-22
- `catchExcpetions()`, 77-19
- change policy
 - about, 28-70
 - attribute change tracking, configuring, 28-72
 - deferred change detection, configuring, 28-71
 - empty transaction, 100-6
 - object change tracking, configuring, 28-71
 - unit of work, 100-6
- change tracking
 - attribute, configuring, 28-72
 - deferred, configuring, 28-71
 - object, configuring, 28-71
- changed items, displaying in TopLink Workbench Navigator window, 4-9
- `ChangedFieldsLockingPolicy`, 26-20
- changing package names, 4-52
- `checkDatabase()`, 77-18
- checking in/out projects, 5-3
- `checkInstantiationPolicy()`, 77-19
- Choose a Schema Context dialog box, 31-3, 32-2
- Choose Query Key dialog box, 97-14
- Choose Relationships to Generate dialog box, 4-30, 4-31
- Choose Root Element dialog box, 31-4, 32-5
- class extraction method
 - about, 26-14
 - inheritance, 26-15
- class indicator
 - about, 26-13
 - class extraction method, 26-14
 - class indicator field, 26-13, 41-1
- class loader
 - host application, 2-5
 - loading session, 78-4
- class modifiers, 4-42
- Class tab, 4-41, 4-42, 4-43
- classes
 - access modifiers, 4-42
 - adding and refreshing, 4-50
 - branch, 26-13
 - creating, 4-40, 4-52
 - `CursoredStream`, optimizing, 99-18
 - Database Exception, 98-24
 - `DatabaseMapping`, 33-25
 - default null values, 35-36
 - `DeleteObjectQuery`, 98-13
 - `ExpressionBuilder`, 97-15
 - generating from database, 4-30
 - `InsertObjectQuery`, 98-13
 - `InsertObjectQuery`, 100-12
 - interfaces, 4-43
 - leaf, 26-13
 - merging files, 5-5
 - methods, adding, 4-47
 - naming, 4-41
 - non-descriptor classes, 4-51
 - object model, 2-16
 - Performance Profiler, 11-3
 - persistent requirements, 2-11
 - preferences, 4-16
 - refreshing, 4-50

- removing, 4-51
- root, 26-12
- troubleshooting, 14-22
- tutorial, 16-2
- unit of work, 100-12
- UpdateObjectQuery, 98-13, 98-14
- ValueHolderInterface, 2-11, 33-7, 36-9
- VariableOneToOneMapping, 36-6
- see also* specific class name
- classpath
 - adding, 22-3
 - 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, 22-3
 - setting for BEA WebLogic, 7-15
 - troubleshooting, 14-22
 - xdb.jar file, 4-3
- class-table relationships, 16-6
- client sessions
 - about, 75-1, 75-13, 75-25
 - acquiring, 78-6
 - configuration, 82-1
 - example, 75-15
 - shared resources, 75-15
- client-controlled transactions, 102-24
- client-server architecture. *See* two-tier
- CLOB
 - mapping to, 34-2
- clones
 - copying methods, 28-69
 - merging changes, 102-12
 - post-commit, avoiding, 102-30
 - unit of work, 100-2, 100-9
- Cloudscape platform, 84-3
- 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, 26-3, 28-44
 - deploying, 8-9, 10-2
 - descriptors, inheritance, 26-4
 - external transactions, 100-3
 - isolated client sessions, 75-24
 - local transaction, 102-25
 - non-deferred write, 102-25
 - 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, 102-24
 - unit of work, 100-3, 102-24
 - unknown primary key class support, 7-25
- code generation, optimizing, 11-8
- collapsing items in Navigator window, 4-9
- collection class
 - specifying, 98-9
 - specifying in query objects, 98-9
- collections
 - persistent requirements for mappings, 2-11
 - query results, 96-8
- comments
 - descriptors, 28-8, 28-9
 - mappings, 35-18
 - projects, 22-21
- commit
 - and Java Transaction API, 100-10
 - failure, resuming after, 102-14
 - resuming unit of work after, 102-14
- Communication Exceptions, 13-64
- composite collection EIS mappings
 - configuring, 61-1
 - example, 56-8
 - understanding, 56-7
- composite collection XML mappings
 - configuration, 70-1
 - configuring, 70-1
 - understanding, 65-25
- composite descriptors
 - about, 26-5
 - composite object mapping, 26-8
 - EIS projects, 26-8
 - XML projects, 26-8
- composite direct collection EIS mappings
 - configuring, 59-1
 - understanding, 56-6
- composite direct collection XML mappings
 - configuring, 68-1
 - understanding, 65-14
- composite EIS descriptors, 27-5
- composite object EIS mappings
 - composite descriptors, and, 26-8
 - configuring, 60-1
 - understanding, 56-7
- composite object mappings
 - composite descriptors, and, 26-8
- composite object XML mappings
 - configuration, 69-1
 - configuring, 69-1
 - understanding, 65-21
- composite primary key, 36-9
- concrete class. *see* container policy
- concurrency
 - about, 2-17
 - exceptions, 13-25
 - server session, 75-18
- Concurrency Exceptions, 13-25
- configurations
 - about, 76-2

- creating, 76-2
- development environment, 5-1
- new, 76-2
- Oracle JDeveloper, 5-1
- session, 76-2, 76-4, 76-6, 76-9
- conforming
 - about, 102-8
 - alternatives, UnitOfWork method
 - writeChanges, 102-11
 - alternatives, UnitOfWork properties, 102-12
 - descriptors, and, 102-11
 - in-memory queries, 102-9
 - queries, alternatives to, 102-11
- ConnectBy, 99-7
- connection policy
 - configuring, 77-19
 - exclusive connections, 77-19
 - lazy connection acquisition, 77-20
- connection pool
 - about, 84-7
 - connection count, 89-1
 - ConnectionPolicy, 75-20
 - external, 84-8
 - internal, 84-7, 84-8, 84-9
 - lazy connection allocation, 75-18, 78-9
 - named, 84-9
 - parameter binding, 11-16
 - prepared statement caching, 11-16
 - read, 84-8
 - sequence, 20-5, 84-8
 - server session, 75-18
 - sessions, and, 75-3
 - size, 89-1
 - write, 84-8
- Connection Specifications tab, 24-3
- Connection tab, 24-3
- connections
 - about, 84-6
 - connection pool, 84-7
 - exclusive write connection, 77-19
 - lazy acquisition, 77-20
 - reading through the write connection, 102-27
- connector.jar, 4-2, 20-7
- container configuration file, 8-5
- container policy
 - about, 35-26
 - custom Collection class, 4-3
 - sorting, in memory, 35-26
- container-controlled transactions, 102-24
- context
 - JAXB path, 20-14
 - menus, 4-5
 - schema, 31-2, 32-2
- Context.SECURITY_CREDENTIALS, 91-9
- Context.SECURITY_PRINCIPAL, 91-9
- Conversion Exception, 13-26
- Converter tab
 - object type mappings, 35-22
- converters
 - custom, 36-3

- object type, 35-22
- coordinated announcement delay, 91-11
- coordinated cache, 8-11
 - configuring, 91-1
 - naming service, 91-7
 - service channel, 91-3
- copy policy
 - about, 28-69
 - method, 28-69
 - setting, 28-70
- copying project objects, 5-6
- Copying tab, 28-70
- CORBA
 - Oracle TopLink transport layer support, 75-31
 - Transaction Service *see* OTS
- Create New Project button, 21-2
- Create New Project dialog box, 21-2
- Create New Session dialog, 76-4, 76-6, 76-8
- Create Project from JAXB dialog, 21-6
- Create Project from OC4J dialog, 7-9
- creating
 - configurations, 76-2
 - expressions, 97-12
 - sessions, 76-4, 76-6, 76-8, 88-1
- Crimson XML parser, 7-3
- cursored streams
 - example, 99-18
 - optimizing, 99-18
 - remote sessions, 75-32
 - usage example, 75-32
- cursors
 - as query results, 96-8
 - traversing scrollable, 99-16
- Custom Calls tab, 31-6
- custom SQL
 - Custom SQL tab, 29-7
 - unit of work, 102-15
- 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, 84-1
 - authentication, 84-5
 - connection pool, 84-7
 - connections, 84-6
 - optimizing, 11-14, 11-15
 - platforms, 84-3
- data level queries
 - example, 97-12
 - in expressions, 97-11
- data source platform
 - about, 84-3
 - Attunity Connect database, 84-3

- Cloudscape database, 84-3
- databases, 84-3
- DB2 database, 84-3
- EIS, 84-4
- HSQL database, 84-3
- Informix database, 84-3
- J2C adapter, 84-4
- JDBC drivers, 84-3
- Microsoft Access database, 84-3
- MySQL4 database, 84-3
- Oracle database, 84-3
- Oracle8 database, 84-3
- Oracle9 database, 84-3
- PointBase database, 84-3
- SQLAnywhere database, 84-3
- SQLServer database, 84-3
- SybasePlatform database, 84-3
- data sources
 - configuring, 85-1
 - nontransactional, 84-1
 - transactional, 84-1
 - troubleshooting, 14-23
- Database Exceptions, 13-27, 98-24
- database fields, configuring, 37-2
- database functions, in expressions, 97-3
- database login
 - parameter binding, 86-9
 - prepared statement caching, 86-9
- Database Preferences, 4-17
- database queries
 - about, 96-10, 98-4
 - fetch groups, 96-12
 - join reading, 96-12
 - object level modify query, 96-13, 96-14
 - object level read query, 96-11, 96-13
 - partial object query, 96-11
 - read all query, 96-11
 - read object query, 96-11
 - report query, 96-15
- database schema
 - tables, 4-22
 - tutorial, 16-4
- database sessions
 - about, 75-2, 75-28
 - cache, 75-32
 - configuration, 83-1
 - creating, 76-8
- 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, 21-15
 - JDBC driver classpath, 4-23
 - properties, 4-25
 - references, 4-27, 4-28
 - removing, 4-24
 - renaming, 4-24
 - schema, 4-22
 - SQL generation, 4-29
 - TopLink Workbench, Navigator window, 4-9
- DatabaseException class, 98-24
- DatabaseLogin, 84-2
- DatabaseMapping class, 33-25
- DatabaseQuery, 96-3
- DatabaseRow, 36-15
- databases
 - catalog, 4-22
 - common problems, 14-23
 - connect to, 4-21
 - creating reference tables on, 4-27
 - custom drivers, 4-18
 - disconnect from, 4-21
 - drivers, 4-18
 - exceptions, 98-24
 - fields, configuring, 37-2
 - for project, 21-3
 - Java type conversion, 6-3
 - linking, 75-28
 - log out of, 4-22
 - logging into, 4-22, 23-7
 - logins, 86-1
 - mapping. *See* mappings
 - platform, 21-3, 23-2, 24-2, 86-1, 87-1
 - preferences, 4-17
 - schema, 4-22
 - schema manager, 6-1
 - tables, 4-21
 - TopLink Workbench, Navigator window, 4-9
 - troubleshooting, 14-23
 - type conversion, schema manager, 6-3
 - using with Oracle TopLink Workbench, 4-21
- DatabaseSession class
 - logging SQL and messages, 75-10
- DB2
 - platform, 84-3
 - schema manager type conversion, 6-3
- DBase platform, 84-3
- default mapping
 - about, 33-4
 - automatic table generation, and, 33-4
 - configuring, 8-12
 - default table generator, 6-6
- Default Mapping Exception, 13-75
- default table generator
 - default mapping, 6-6
 - table creator, creating, 6-4
- defaults
 - login level null values, 85-5
 - mapping level null values, 35-12
 - null values, 35-12, 35-36, 85-5
 - optimization, 11-8
 - root, 32-5
 - see also* preferences
- DefaultSequence, 86-6
- deferred change detection

- configuring, 28-71
- DeferredChangeDetectionPolicy, 100-7
- Delete All Interaction tab, 63-3
- deleteObject(), 29-8
- DeleteObjectQuery, 98-13
- deletes
 - controlling order, 102-16
 - delete operation, 98-4
 - queries, EIS mappings, 63-3
- demarcation of unit of work transactions, 100-2
- dependent objects
 - non-deferred write, 26-4
- deploy tool
 - about, 10-5
 - troubleshooting, 10-6
 - using with WebSphere Studio Application Developer, 10-5
- deploying
 - about, 2-14, 10-1
 - application server requirements, 7-1
 - BEA WebLogic, 10-2
 - CMP applications, 10-2, 10-4
 - database login, 23-7
 - entity beans overview, 2-14
 - 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, 8-13
 - non-CMP applications, 10-4
 - packaging, 9-1
 - Session Bean applications, 10-1
 - troubleshooting, 15-1
- deployment descriptors, 26-3
- Deployment Exceptions, 13-67
- deployment exceptions
 - BEA WebLogic deployment, 15-3
 - Deployment Exceptions (14001-14033), 13-67
 - IBM WebSphere deployment, 15-12
- deployment files
 - BMP applications, 8-9
 - CMP applications, 8-9
 - creating, 8-1
 - descriptors, 26-3
 - EJB 3.0, 8-2, 9-1
 - JARs, troubleshooting, 15-1
 - Java applications, 8-8
 - JSP and Servlet applications, 8-8
 - Session Bean applications, 8-8
 - XML, generating, 8-3
- deployment XML, exporting, 21-14
- DeploymentXMLGenerator, 8-3
- Descriptor Event Listener, 28-60
- Descriptor Event Manager
 - about, 26-8
 - Descriptor Event Listener, 28-60
 - domain object methods, 28-57
 - event types, 28-57

- handlers, 26-8, 28-57, 28-60
- handlers, Descriptor Event Listener, 28-60
- understanding, 26-8
- descriptor events
 - about, 26-8
 - Descriptor Event Listener, 28-60
 - domain object methods, 28-57
 - handlers, 26-8, 28-57, 28-60
 - types of, 28-57
 - understanding, 26-8
- Descriptor Exceptions, 13-2, 13-29
- Descriptor Info tab, 28-3, 28-5, 28-7, 29-2, 29-4, 31-2, 31-4, 32-2, 32-4, 32-5, 32-6
- DescriptorEventListener, 28-60
- descriptors
 - about, 19-2, 26-1
 - advanced properties, default, 22-7
 - aggregate, 26-5, 27-2
 - aggregate, EJB 3.0 and, 26-6
 - aggregate, relational projects and, 26-5
 - amending, 2-20, 26-5, 28-78
 - API, 26-22
 - architecture, 26-2
 - attributes, adding, 4-44
 - automapping, 34-2
 - automatically mapping, 34-2
 - BMP, 26-3, 28-44
 - cache refreshing, 28-27
 - change policy, 28-70
 - child inheritance, 28-49
 - class, 27-2
 - CMP, 26-3, 28-44
 - CMPPolicy, 26-3, 28-44
 - comments, 28-8, 28-9
 - composite, 26-5, 26-8, 27-5
 - composite EIS, 27-5
 - configuring, 28-1
 - conforming, 102-8
 - creating, 27-1, 27-2, 27-5
 - custom EIS interactions for basic persistence, 31-6
 - custom SQL queries for basic persistence, 29-6
 - deactivating, 4-10
 - default mappings, 33-4
 - default root, 31-4
 - deployment information, 26-3
 - Descriptor Event Listener, 28-60
 - domain object methods, 28-57
 - EIS, 26-11, 27-5, 31-1
 - EIS projects, 26-8
 - EJB, 26-3, 28-44
 - EJB information, 26-3, 28-44
 - errors, 4-10, 4-11, 14-4
 - event handlers, 26-8, 28-57, 28-60
 - events, 26-8, 28-57
 - existence checking, 11-13, 28-42
 - fetch groups, 28-77
 - files, merging, 5-5
 - generating from database, 4-30
 - hierarchy, inheritance, 26-23
 - history policy, 28-73

- identity maps, 22-14, 28-35, 28-37, 28-39
- in Java, 26-22
- inactive, 4-10
- inheritance, 26-3, 26-12, 28-49, 28-50
- instantiation, 11-13
- interface, 27-2, 28-31, 28-33
- mapping, 28-3, 29-2, 33-4, 34-1, 34-2
- merging, 5-5
- named queries, 28-10
- nondesoriptor classes, 4-51
- object-relational, 26-11, 30-1
- optimizing, 11-13
- parent inheritance, 28-50
- projects, 19-2, 26-1
- query key interfaces, 28-31
- query timeout, 28-24, 28-26
- read only, 28-4
- registering with sessions, 75-12, 77-2, 77-3, 77-10
- relational, 26-11, 27-2, 29-1
- removing, 4-51
- returning policy, 28-65
- root EIS, 27-5
- root element, 32-5
- schema context, 31-2, 32-2, 32-3
- sequencing, 26-9, 29-3
- TopLink Workbench, Navigator window, 4-9
- types of, 26-1
- validating, 27-6
- XML, 26-12, 32-1
- XML projects, 26-8
- detachement indirection, 33-9
- developing applications with Oracle TopLink, 2-1
- development environments
 - about, 3-2
 - configuring, 5-1
 - database logins, 23-7
- development process
 - about, 2-1
 - additional support, 2-3
 - stages of, 2-1
 - with Oracle TopLink, 2-1
- development tools
 - about, 3-1
 - profiler, 11-2
 - schema manager, 6-1
- dimensionality, array, 4-45, 4-49
- direct access
 - about, 22-4, 35-14
 - specifying, 35-32
- direct collection relational mappings
 - configuring, 45-1
 - example, 36-11
 - understanding, 36-11
- direct collections
 - session broker limitations, 75-27
- direct EIS mappings
 - configuring, 58-1
 - understanding, 56-5
- direct field
 - in direct collection mappings, 45-2
- direct key fields, 46-2
- direct map relational mappings
 - configuring, 46-1
 - direct keys, 46-2
 - direct value, 46-1
 - understanding, 36-12
- direct mappings
 - generating deprecated, 22-12
 - with EJB, 36-4
- direct value fields, 46-1
- direct XML mappings
 - configuring, 67-1
 - understanding, 65-5
- directionality in mappings, 36-2
- direct-to-field mappings
 - ObjectTypemapping deprecated, 36-3
 - SerializedObjectMapping deprecated, 36-3
 - type conversions, 37-3
 - TypeConversionMapping deprecated, 36-3
- direct-to-field relational mappings
 - configuring, 38-1
 - options, 38-1
 - timestamp support, 37-3
 - understanding, 36-4
- direct-to-XMLType relational mappings
 - configuring, 39-1
 - understanding, 36-4
- Discovery Exception, 13-77
- DMS profiler
 - about, 11-4, 75-11
 - accessing with JMX, 11-7
 - and JMX
 - nouns, 11-4, 75-11
 - selecting, 77-11, 77-14, 77-17, 77-20
 - sensors, 11-4, 75-11
- 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, 99-4

E

- Editor window, about, 4-4, 4-10
- EIS
 - about, 20-8
 - architecture, 2-4
 - call, 98-24
 - custom interactions for basic persistence, per descriptor, 31-6

- indexed records, configuring, 31-5
- interactions, 31-6, 96-6, 98-24
- mapped records, configuring, 31-5
- mappings, 20-8, 56-2
- projects, 21-3
- queries, 96-6
- record format, configuring, 31-5
- XML records, configuring, 31-5
- EIS descriptors
 - composite, 27-5
 - configuring, 31-1
 - default root, 31-4
 - locking policy, 28-62
 - root descriptor, 27-5
 - schema context, 31-2
 - setDataTypeName, 31-5
 - understanding, 26-11
- EIS mappings
 - about, 56-1, 56-2
 - architecture, 56-5
 - composite collection, 56-7, 61-1
 - composite direct collection, 56-6, 59-1
 - composite object, 56-7, 60-1
 - configuring, 57-1
 - direct, 56-5, 58-1
 - jaxb:class support, 56-3
 - list support, 56-3
 - one-to-many, 56-12, 63-1
 - one-to-many, key on source, 56-13
 - one-to-many, key on target, 56-15
 - one-to-one, 56-8, 62-1
 - one-to-one, key on source, 56-9
 - one-to-one, key on target, 56-10
 - transformation, 56-17, 64-1
 - types of, 56-1
 - union support, 56-3
 - xsd:list, 56-3
 - xsd:union, 56-3
- EIS projects
 - configuring, 24-1
 - connector.jar, 20-7
 - indexed records, 20-9
 - mapped records, 20-9
 - sequencing, 20-5
 - understanding, 20-7
 - XML records, 20-9
- EIS queries, 96-6
- EIS record types, supported, 56-2
- EISLogin, 84-2
- EJB
 - descriptors, 21-10
 - isolated client sessions, 75-24
 - 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, 2-34
 - and CMP, 2-27
- EJB 1.x
 - indirection, 33-9
 - serialization, 33-9
- EJB 2.x
 - and BMP, 2-34
 - and CMP, 2-27
 - default mapping, 33-4
 - indirection, 33-9
 - serialization, 33-9
- EJB 3.0
 - <J2EE-Container>-jar.xml file, 8-6
 - and CMP, 2-27
 - attribute change tracking policy, OC4J CMP integration, 7-5, 100-8
 - Bean annotation, fetch=lazy, 33-9
 - default mapping, 33-4
 - deployment files, 8-2, 9-1
 - deployment files, OC4J CMP integration, 7-5
 - detachment, 33-9
 - Embedded annotation, 26-6
 - packaging for deployment, 9-1
 - packaging, OC4J CMP integration, 7-5
 - projects.xml file, 8-3, 8-5
 - serialization, 33-9
 - sessions.xml file, 8-5
 - toplink-ejb.xml file, 8-8
 - value holder indirection, 33-9
- EJB descriptors, opening projects with, 21-10
- EJB entities
 - BMP indirection, 33-9
 - CMP hot deployment, 10-4
 - deployment overview, 2-14
 - EJB 1.x indirection, 33-9
 - EJB 2.x indirection, 33-9
 - EJB 3.0 indirection, 33-9
 - generating, 4-31
 - hot deployment, 10-4
 - indirection, 33-9
 - inheritance, 26-4, 26-17
 - inserting after ejbCreate, 26-4
 - inserting after ejbPostCreate, 26-4
 - mapping, 22-6
 - non-CMP hot deployment, 10-4
 - non-deferred write, understanding, 26-3, 102-25
 - sequencing, 20-21
- EJB entity beans
 - and EJB 1.1, 2-27, 2-34
 - and EJB 2.x, 2-27, 2-34
 - and EJB 3.0, 2-27
 - with BMP architecture, 1-5, 2-33
 - with CMP architecture, 1-5, 2-27
- EJB Exception Factory Exceptions, 13-58
- EJB finders
 - about, 96-23
 - Call finders, 96-26
 - creating, 99-8
 - DatabaseQuery finders, about, 96-26
 - default finders, about, 96-26
 - default finders, creating, 99-8
 - EJB QL finders, about, 96-27
 - ejb-jar.xml options, 99-9
 - ejbSelect method, 96-28

- ejbSelect, creating, 99-15
 - ejbSelect, using, 99-15
 - expression finders, about, 96-27
 - named query finders, about, 96-27
 - predefined, about, 96-24
 - primary key finders, about, 96-27
 - redirect finders, about, 96-28
 - redirect finders, using, 99-12
 - single-object finder return type checking, 7-25
 - SQL finders, about, 96-28
- EJB Info tab, 28-44
- EJB JAR XML Exception, 13-84
- EJB Preferences, 4-16
- EJB QL
 - exceptions, 13-55
 - queries, 29-6, 96-5
 - query language, 96-5
- EJB session bean facade architecture
 - about, 1-5, 2-24
 - understanding, 2-24
- EJB Session Beans, 75-31
- ejbc
 - about, 10-2
 - troubleshooting, 10-3
- ejbCreate, 26-4
- ejb-jar.xml file
 - about, 8-5, 21-15
 - configuring, 8-5
 - corresponding to Oracle TopLink Workbench functions, 21-15
 - EJB finder options, 99-9
 - location, 22-7
 - managing, 5-6
 - preferences, 4-16
 - synchronization under EJB 2.0, 8-5
 - updating from, 21-16
 - writing, 21-16
- ejbPostCreate, 26-4
- @Embeddable, 26-6
- @Embedded, 26-6
- empty unit of work transactions, 100-6
- encrypting login passwords, 21-13
- enhanced validation exceptions, 8-4
- Enterprise Information Systems. *see* EIS
- entity beans
 - deployment, 2-14
 - descriptor information, 26-3
 - direct mappings, 36-4
 - indirection, BMP, 33-9
 - indirection, EJB 1.x, 33-9
 - indirection, EJB 2.x, 33-9
 - indirection, EJB 3.0, 33-9
 - sequencing with, 20-21
- environment
 - configuring, 4-2
 - JAVA_HOME, 4-2
 - JDBC_CLASSPATH, 4-2
 - proxy, 4-13, 4-14, 21-16
- error codes
 - 1-176, 13-2
 - 1-99, 14-1
 - 100-199, 14-2
 - 200-399, 14-4
 - 400-599, 14-11
 - 500-699, 14-15
 - 700-799, 14-20
 - 800-899, 14-21
 - 10001-10047, 13-58
 - 12000-12004, 13-64
 - 13000-13020, 13-64
 - 14001-14027, 13-67
 - 15001-15024, 13-70
 - 17001-17006, 13-71
 - 18001-18002, 13-74
 - 19001-19003, 13-75
 - 22001-22004, 13-77
 - 22101-22105, 13-77
 - 3001-3007, 13-26
 - 4002-4018, 13-27
 - 5001-5008, 13-29
 - 6001-6098, 13-30
 - 7001-7104, 13-42
 - 72000-72023, 13-84
 - 8001-8010, 13-55
 - 9000-9009, 13-56
- 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, 75-5
- events
 - about, 26-8
 - client session, 75-6
 - database access, 75-6
 - Descriptor Event Listener, 28-60
 - domain object methods, 28-57
 - handlers, 26-8, 28-57, 28-60
 - listeners, sessions, 75-7
 - session, 75-5
 - session manager, 75-6
 - sever session, 75-6
 - types of, 28-57
 - unit of work, 75-6
- examples
 - composite collection EIS mapping, 56-8
 - context menu, 4-5
 - cursor streams, 99-17
 - direct collection mappings, 36-11
 - direct-to-field mappings, 36-4
 - exception handler, 77-13
 - indirection, 33-6
 - inheritance, 26-12
 - Oracle TopLink Workbench, 4-3
 - performance optimization, 11-22, 11-24
 - proxy indirection in code, 35-9
 - READALL finders, 99-10
 - report query, 98-6
 - scrollable cursors, 99-17

- serialized mapping, 33-10
- stored procedure call, 98-21, 98-22
- transformation mapping, 36-15
- transformation XML mapping, 56-18, 65-31
- Unit of Work, 100-6, 101-7
- write, write all, 98-3
- exception handler
 - about, 75-12
 - example, 77-13
 - selecting, 77-12
- exceptions
 - chained, 75-10
 - communication exceptions, 13-64
 - conversion exceptions, 13-26
 - database exceptions, 13-27, 98-24
 - Default mapping exception, 13-75
 - deployment exceptions, 13-67
 - descriptor exceptions, 13-2
 - discovery exceptions, 13-77
 - EJB exceptions factory, 13-58
 - EJB JAR XML exceptions, 13-84
 - EJB QL exceptions, 13-55
 - enhanced validation, 8-4
 - java.security.AccessControlException, 15-12, 15-13
 - JMS processing exceptions, 13-74
 - Migration utility exception, 13-82
 - optimistic locking, 13-29
 - query exceptions, 13-30
 - remote command manager exceptions, 13-77
 - SDK data store exceptions, 13-71
 - SDK descriptor exceptions, 13-75
 - selecting exception handler, 77-12
 - session loader exceptions, 13-56
 - synchronization exceptions, 13-70
 - Transaction exception, 13-79
 - validation exceptions, 13-42
 - XML conversion exception, 13-80
 - XML data store exceptions, 13-64
- exclusive connections
 - about, 102-28
 - internal read connection pool, 89-6
 - isolated sessions, 75-20, 77-19
 - named queries, 28-24
- existence checking, 102-5
 - descriptors, 28-42
 - projects, 22-8
- expanding items in Navigator window, 4-9
- expiration of objects in the cache, 22-19, 28-40
- explicit query refreshes, cache coordination, 90-7
- exporting
 - deployment XML, 21-14
 - Java model, 21-14
 - Java source, 21-14
 - preferences, 4-12
 - projects, 21-13
- Expression Builder, 97-12, 97-15
- Expression Builder dialog box, 97-12
- Expression class, 97-1
- ExpressionMath class, 97-1

- expressions
 - about, 96-3, 97-1
 - allows none, 28-18, 97-14
 - allows null, 28-18, 97-14
 - building, 97-12
 - comparing with SQL, 97-1
 - components, 97-2
 - creating, 97-12
 - data level queries, 97-11
 - database functions, 97-3
 - in relationships, 97-5
 - in-memory queries, limitations, 96-31
 - mathematical functions, 97-4
 - multiple, 97-10
 - one-to-one mappings, 97-5
 - outer joins, 98-11
 - parallel expressions, 97-11
 - parameterized, 97-7
 - platform functions, 97-16
 - query keys, 97-9
 - subqueries and subselects, 97-10
 - user-defined functions, 97-4, 97-16
 - using Boolean logic, 97-3
 - XML Type functions, 97-4
 - see also* queries
- external
 - applications, 102-26
 - connection pools, 84-8
 - controller, transaction, 100-1
 - JDBC pools, 2-35
 - transactions, 100-1
- external transaction controller
 - configuration, sessions, 77-14
 - session, 100-2

F

- factory name, JMS connection, 92-2
- failure, resuming unit of work after commit, 102-14
- features of Oracle TopLink, 1-4
- fetch groups
 - about, 26-4, 28-77
 - configuring, 28-77, 99-3
 - default, 28-77, 99-3
 - disabling, 99-3
 - dynamic, 99-4
 - object level read query, 96-12
 - read optimization, and, 28-77
 - size, 11-17
 - static, 99-3
- field references, 37-8
- Field uses XML Schema "type" attribute
 - option, 33-12
- fields in database tables, 4-25
- Fields tab, 47-2
- field-to-object attribute transformation, 35-29, 35-31
- files
 - JAXB-specific, 20-10
 - TopLink-specific, 20-11
 - see also* specific file name

- final attributes, 4-45, 4-48
- findAll, using, 96-26
- finders
 - caching options, 96-37
 - disabling cache, 96-37
 - managing large result sets, 99-19
 - refreshing results, 96-38
 - see also* EJB finders
- flashback queries
 - about, 96-18
 - historical client sessions, 81-1
- forceUpdateToVersionField(), 102-17
- foreign keys
 - about, 2-17, 37-8
 - configuring in EIS mappings, 62-1
 - EIS mappings, 63-1
 - multiple tables, 29-14
 - one-to-many mappings, 36-7
 - one-to-one mappings, 35-34, 36-5
 - parameterized expressions, 97-7
 - references, 14-12, 14-21
 - target, 36-6, 37-9
 - troubleshooting, 14-12, 14-21
- full identity map, 90-3

G

- garbage collection, managing, 101-8
- 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, 22-12
 - see also* exporting
- getCatalogs(), 4-22
- getField(), 97-11
- getImportedKeys(), 4-22
- getParameter, 97-8
- getPrimaryKeys(), 4-22
- getTable(), 97-12
- getTables(), 4-22
- getTableTypes(), 4-22
- getValue(), 33-7
- getValue() method, 33-7

H

- hard cache weak identity map
 - about, 90-4
 - when to use, 90-5
- help
 - about, 4-12
 - displaying, 4-12
- Help button., 4-12
- Help Preferences, 4-14, 4-16
- hierarchical queries
 - about, 96-18

- described, 99-6
- hints, Oracle Hints in queries, 96-18
- historical client sessions
 - about, 75-1, 75-24
 - cache, 75-33
 - limitations of, 75-25
- historical queries, 99-2
 - about, 96-20
 - see also* AsOfClause
- history policy, configuring, 28-73
- holders, value, 33-6
- host URL, JMS topic, 92-3
- 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, 84-3

I

- IBM Informix Database native sequencing, 20-19
- IBM WebSphere
 - deploy tool, 10-5
 - deploying to, 10-3
 - deployment exceptions, 15-12
 - setting classpath, 7-21
- identity
 - about, 2-16, 90-3
 - cache, and, 90-3
 - using cache to preserve, 90-3
 - see also* identity map
- identity map cache
 - disabling during a write query, 98-14
 - refresh in read query, 96-35
- identity maps
 - about, 28-35, 28-37, 28-39, 75-29
 - cascading refresh during read query, 96-35
 - descriptors, 22-14, 28-35
 - example, 96-35
 - full, 90-3
 - guidelines for choosing type, 90-4
 - hard cache weak identity map, 90-4, 90-5
 - isolated client sessions, 75-24
 - no identity map, 90-4
 - refreshing during read query, 96-35
 - soft cache weak identity map, 90-4, 90-5
 - soft cache weak identity map and read optimization, 11-19
 - specifying, 28-35, 28-37, 28-39
 - weak, 90-3
 - weak identity map and read optimization, 11-19
- Identity tab. *see* Caching tab
- impedance mismatch, solving, 1-2
- Implementors tab, 28-33
- Import Tables from Database dialog, 4-23
- importing
 - classes, 4-16

- preferences, 4-12
- inactive descriptors
 - about, 4-10
 - mapping to, 37-6, 57-3
- independent relationships, 35-16
- indexed records, 56-2
- indirection
 - about, 2-18, 33-5
 - bidirectional relationships, 35-35
 - choosing the correct type, 35-4
 - configuring, 35-3
 - EJB, 33-9
 - EJB 1.x CMP, 33-9
 - EJB 2.x CMP, 33-9
 - EJB 3.0 CMP, 33-9
 - example, 33-6
 - many-to-many mappings, 36-9
 - nontransparent, 2-11
 - one-to-many mappings, 35-34
 - proxy indirection, 33-8
 - remote sessions, 75-32
 - serialization, 33-9
 - transparent, 2-11, 33-8
 - value holder, 33-6
 - ValueHolderInterface, 2-11
 - see also* proxy indirection, transparent indirection
- Informix platform, 84-3
- inheritance
 - about, 2-17, 26-3, 26-12
 - aggregate classes, 26-17
 - aggregate collection mappings, 36-10
 - branch classes, 26-12, 26-13
 - child descriptors, 28-49
 - class extraction, 26-14, 26-15
 - class indicator, 26-13, 26-14
 - descriptors, 26-3, 26-12, 28-49, 28-50
 - finding subclasses, 26-13
 - instantiating subclasses, 26-13
 - isolated client sessions, 75-24
 - leaf classes, 26-12, 26-13, 99-4
 - primary keys, 26-15
 - queries, 96-22
 - querying on hierarchy, 99-4
 - relational parent, 28-50
 - root class, 26-12, 28-49, 28-51
 - root class subclasses, finding in inheritance, 26-13
 - supporting with multiple tables, 26-16
 - supporting with one table, 26-15
 - transformed to relational model, 11-11
 - using with EJB, 26-4, 26-17
- inheritance hierarchies
 - descriptors, 26-23
 - querying on, 99-4
- Inheritance tab, 28-49, 28-51
- inherited subclasses, mapping, 28-56
- in-memory query
 - about, 90-6
 - check cache using exact primary key, 96-30
 - check cache using primary key, 96-30
 - check database if not in cache, 96-30
 - conform results in unit of work, 96-31
 - conforming, 102-9
 - expression limitations, 96-31
 - supported, 96-30
 - using, 96-30
- inner join, 97-5
- insert operation, 98-3, 98-4
- insertObject(), 29-7
- instantiation policy
 - about, 28-68
 - setting, 28-68
- Instantiation tab, 28-68
- integrity checker, 27-6
 - about, 75-12
 - configuring, 77-18
- interactions
 - about, 98-24
 - creating, 31-6
- interface alias
 - about, 29-10
 - creating, 29-11
- Interface Alias tab, 29-11
- interfaces
 - classes, implementing, 4-43
 - customizing, 4-13
 - descriptors, 27-2, 28-31, 28-33
 - queries, 96-22
 - query keys, 28-31
 - querying on, 99-4
- internal connection pool
 - about, 84-7
 - named, 84-9
 - read, 84-8
 - sequence, 84-8
 - write, 84-8
- internal query object cache
 - about, 96-36
 - configuring, 99-21, 99-22
 - expiration, 99-22
 - restrictions, 96-36
- internal transactions, 100-1
- invalidation of objects in the cache, 22-19, 28-40, 90-8
- IP address for multicast group, 91-4
- isolated client sessions
 - about, 75-1, 75-19, 80-1
 - configuration, 80-1
 - life cycle, 75-22
 - limitations of, 75-23
 - session event handlers, 75-21
 - with Oracle Virtual Private Database (VPD), 75-20
- isolated session
 - cache, 75-33
 - ConnectionPolicy, 75-20
 - exclusive connections, 75-20
 - supported databases, 75-20
- isolation
 - cache, 90-6
 - transaction levels, 100-2
 - unit of work transactions, 100-4

Iterator interface, 99-16

J

J2C adapters

- about, 84-4
- configuring for Oracle TopLink Workbench, 4-2
- EISLogin, 84-2
- selecting, 84-4
- with EIS, 21-3

<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, 21-15
- exporting to, 21-14
- integration with any datasource, 1-2
- iterators, 99-16
- object model, 2-11

Java applications

- deploying, 10-1
- deployment files, 8-8
- packaging for deployment, 9-1

Java Cryptography Extension, 21-13

Java Management Extensions. *see* JMX

Java Naming and Directory Interface. *See* JNDI

Java Object Builder, 75-4

Java streams

- described, 99-17
- optimizing, 99-18
- support for, 99-17

Java Transaction API

- and unit of work commit, 100-10
- and unit of work rollback, 100-11
- see also* JTA

Java Transaction Service *see* JTA

JAVA_HOME, 4-2

java.security.AccessControlException, 15-12, 15-13

java.util.Collection interface, 35-26

java.util.Map interface, 35-26

java.util.Set interface, 35-26

javax.ejb.EntityBean interface, 4-31

JAXB

- creating projects from, 21-6
- files, 20-10
- generating project from the command line, 21-8
- jaxb:class, and EIS mappings, 56-3
- jaxb:class, and XML mappings, 65-4
- proxy configuration, 21-8
- tljxb.cmd, 21-8
- tljxb.sh, 21-8
- typesafe enumeration converter, 35-25
- understanding, 20-10
- validation, 20-14
- XML projects, 20-10

JAXB typesafe enumeration converter

- configuring, 35-25
- understanding, 33-24

JAXBContext, 20-14

JCE. *see* Java Cryptography Extension

JConnect, 11-14

JDBC

- adaptor for EIS, 56-5
- database gateway for EIS, 56-5
- driver classpath, 4-23
- JConnect, 11-14
- Sybase JConnect, 11-14

JDBC drivers

- about, 84-3
- configuring for Oracle TopLink Workbench, 4-2
- fetch size, 11-14, 11-17
- general properties, 11-14
- mapping LOBs, 34-3
- selecting, 84-3

JDBC pools

- external with EJB Entity Beans with BMP architecture, 2-35

JDBC_CLASSPATH, 4-2

JDeveloper. *See* Oracle JDeveloper

JMS

- connection factory name, 92-2
- coordinated cache, 92-1
- Processing Exceptions, 13-72, 13-74
- topic host URL, 92-3
- topic name, 92-1

JMX

- about
- and DMS profiler
- DMS profiler, 11-7

JNDI naming service, 91-7

joining

- about, 96-12
- expressions, and, 97-5
- mappings, and, 40-1
- one-to-many, about, 98-11
- one-to-one mappings, 40-1, 98-11
- optimizing reads, 11-19
- queries, and, 98-11
- QueryManager expressions, 99-4
- read queries, 11-19, 98-11

JSP and Servlet applications

- deploying, 10-1
- deployment files, 8-8
- packaging for deployment, 9-2

JTA

- about, 100-3
- and unit of work, 100-3
- isolated client sessions, 75-24
- unit of work, 100-1

JTA/JTS

- using with EJB Entity Beans with BMP architecture, 2-36

just-in-time reading. *see* indirection

K

Key Converter tab, 46-3

key pairs

database table reference, 4-28

troubleshooting, 14-12, 14-21

keys

about, 2-16

foreign, 2-17, 35-34, 36-5

foreign, target, 36-6

inheritance, 26-15

multiple tables, 29-13

primary, 26-15, 29-13, 35-2, 36-6, 36-9, 41-3

read-only settings, 35-2

reference key field, 36-11, 36-12

variable class relationships, 36-6, 41-3

L

large result sets, managing in finders, 99-19

lazy attribute loading and read optimization, 28-77

lazy connection

acquisition, 77-20

allocation, 75-18, 78-9

lazy loading. *see* indirection

lazy reading. *see* indirection

leaf classes, 26-12, 26-13, 99-4

life cycle of unit of work, 100-5

LOB

mapping to, 34-2

local

documentation, 4-14

transactions, 102-25

locked files, 5-6

locking policy

AllFieldsLockingPolicy, 26-20

ChangedFieldsLockingPolicy, 26-20

configuring, 28-62

field locking, 26-20

optimistic, 26-9, 26-18, 26-20

optimistic version locking, 26-17

OptimisticLockException, 26-18, 26-21

pessimistic locking policy, 26-9, 26-21

SelectedFieldsLockingPolicy, 26-21

stale data, and, 90-6

three-tier architectures, optimistic locking
and, 26-21

three-tier architectures, pessimistic locking
and, 26-22

TimestampLockingPolicy, 26-18

understanding, 26-9, 26-17

version locking, 26-18

VersionLockingPolicy, 26-18

Locking tab, 28-63

log into database, 4-22

Log Out of Database, 4-22

log out of database, 4-22

logging

application server, 75-9

chained exceptions, 75-10

java.util.logging, 75-9

log level, 75-10

Oracle Enterprise Manager 10g, 75-10

output, 75-10

permissions, 7-24

sessions, 75-7, 77-4, 77-5

TopLink native logging, 75-8

types, 75-8

Logging tab, 77-5

login

CMP deployment, 20-3

database, 23-5, 23-7, 86-1

deployment, 20-3

development, 20-4

platforms, and, 20-4

projects, and, 20-3, 74-1, 84-2

role in project, 20-3, 84-2

session, 20-3, 77-4

session role, non-CMP, 20-3

logMessages method, 75-7

look and feel, specifying, 4-13

M

Manage Non-Descriptor Classes dialog, 4-51

management, source control, 5-3

manager, session events, 75-5

many-to-many mappings

relation table, 36-9

relation tables, 43-1

session broker limitations, 75-27

many-to-many relational mappings

configuring, 43-1

EJB, 36-9

understanding, 36-8

mapped records, 56-3

mapping extensions

custom data types, 12-1

JAXB typesafe enumeration converter, 33-24,
35-25

object type converter, 33-12, 35-22

serialized object converter, 33-10, 35-18

simple type translator, 33-12, 35-23

transformation mappings, 33-14

type conversion converter, 33-11, 35-20

mappings

about, 2-15, 2-16, 19-1, 19-2, 33-1, 34-1

access types, 35-15

aggregate collection mappings and EJB, 36-11

anyType mapping, 66-3

as part of the application development
process, 2-20

automatic, 34-2

batch options, 37-7

class hierarchy, 33-25

comments, 35-18

configuring, 34-1, 35-1

database field, 37-4

default mapping, 33-4

deprecated, generating, 22-12

direct access, 11-13, 35-14

- directionality, 36-2
- EIS mappings, 20-8, 33-27, 56-1, 57-1
- EJB 2.0 entities, 22-6
- errors, 14-11
- example, 33-3
- extensions, about, 33-10
- hierarchy, 33-25
- inactive descriptors, 37-6, 57-3
- indirection, 11-13, 33-5, 35-3
- isolated client sessions, 75-23
- manually configuring, 34-1
- many-to-many, 36-9
- many-to-many, with EJB, 36-9
- method access, 35-14, 35-32
- null values, 35-12, 35-36
- object-relational, 49-1, 50-1
- one-to-many object, with EJB, 36-8
- one-to-one with EJB, 36-6
- optimizing, 11-13
- OX mappings, 33-27
- projects, and, 19-2
- read only, 35-2, 35-3
- relation tables, 43-1, 43-2
- relational, 36-1, 37-1
- removing, 34-6
- to tables, 28-3, 29-2
- TopLink Workbench, Navigator window, 4-9
- types of, 33-1
- XML mappings, 65-1, 66-1
- mathematical functions, in TopLink
 - expressions, 97-4
- menu bar, 4-5
- menus
 - about, 4-4, 4-5
 - context menus, 4-5
 - menu bar, 4-5
- merging
 - changes in clones, 102-12
 - Oracle TopLink Workbench project files, 5-4
 - project files, 5-4
- messages, error, 13-1, 14-1
- metadata
 - about, 2-12, 2-19
 - advantages, 2-19
 - creating, 2-20, 2-21
 - mapping and configuration, 19-1
 - project metadata, 2-20
 - session metadata, 2-21
- Metalink, 2-3
- method access
 - about, 22-4, 35-14
 - setting, 35-32
- methods
 - adding, 4-47
 - getValue(), 33-7
 - setValue(), 33-7
 - wrapper policy, 28-76
 - see also* specific method name
- Microsoft Access
 - platform, 84-3

- schema manager type conversion, 6-3
- Microsoft SQL Database native sequencing, 20-19
- migrating
 - error messages, 7-13
 - OC4J persistence to TopLink, 7-5
 - Oracle TopLink Workbench projects, 21-10
 - troubleshooting, 7-13
- Migration Utility Exception, 13-82
- model source, exporting, 21-14
- modifiers, class, 4-42
- multicast group address, coordinated cache, 91-4
- multicast port, coordinated cache, 91-5
- multiple sessions, 75-28, 78-2
- multiple tables
 - about, 29-13
 - specifying for descriptors, 29-13
- multiplicity in relationships, resolving, 7-14
- multi-processing, 11-29
- Multitable Info tab, 28-48, 29-13
- mutable mappings, 35-33
- .mwp file, 4-1, 21-2
- MySQL4
 - batch writing, 86-9
 - platform, 84-3
 - primary key restrictions, 20-15
 - schema manager type conversion, 6-3

N

- named connection pools, 84-9
- named queries
 - about, 28-10, 77-21, 96-15
 - configuring, 28-10, 77-21
 - descriptor level, 28-10
 - exclusive connections, 28-24
 - options, advanced, 28-24
 - parameter binding, 28-22
 - prepared statement caching, 28-22
 - redirect query, 96-16
 - session level, 77-21
 - using, 98-17
 - when not to use, 96-16
 - when to use, 96-16
- namespaces
 - about, 20-5, 20-22
 - configuring, 4-38
- naming service
 - coordinated cache, 91-7
 - JNDI, 91-7
 - RMI, 91-9
- native sequencing
 - IBM Informix Database, 20-19
 - Microsoft SQL Database, 20-19
 - Microsoft SQL Server, 6-6
 - non-Oracle database, 20-19
 - Oracle Database, 20-18, 20-20
 - Oracle Database SEQUENCE object, 20-19
 - Sybase Database, 6-6, 20-19
- 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, 33-12
- NCLOB, 33-12
- neediness warnings. *See* troubleshooting
- nested table object-relational mappings
 - configuring, 55-1
 - understanding, 49-3
- nested unit of work, 100-9, 102-15
- new projects, 21-2
- New Reference dialog box, 4-27
- New Session button, 76-4, 76-6, 76-8
- New Sessions Configuration, 76-2
- New Table dialog box, 4-22
- newInstance method, 102-2
- no identity map, 90-4
- non-cascading write queries
 - compared to cascading, 96-14, 98-13
 - creating using dontCascadeParts ()
 - method, 96-14, 98-13
- non-deferred write
 - configuring, 102-11
 - dependent objects, 26-4
 - understanding, 26-3, 102-7, 102-25
- nonintrusive persistence, 2-18
- nonpersistent projects, 20-2
- nonrelational projects, 20-2
- nontransactional data sources, 84-1
- nontransparent indirection, 2-11
- nouns
 - DMS profiler, 11-4, 75-11
- null values
 - default, 35-12, 35-36, 85-5
 - in expressions, 97-13
 - login level, 85-5
 - mapping level, 35-12
- NVARCHAR2, 33-12

O

- object array object-relational mappings
 - configuring, 54-1
 - understanding, 49-2
- object cache, 96-36
- object cache, sessions, 75-3
- object change tracking
 - configuring, 28-71
- object identity, 75-29
 - about, 2-16, 90-3
 - cache, and, 90-3
 - using cache to preserve, 90-3
 - see also* identity map
- object indirection
 - read optimization, as, 11-19

- object level modify query
 - about, 96-13, 96-14
- object level read query
 - about, 96-11, 96-13
 - fetch groups, 96-12
 - join reading, 96-12
 - partial object query, 96-11
 - read all query, 96-11
 - read object query, 96-11
- object model
 - about, 2-16
 - generating with t ljaxb.cmd, 21-9
 - optimization, 11-8
 - Oracle TopLink requirements, 2-11
 - tutorial, 16-2
- object type converter
 - about, 12-1, 33-12
 - configuring, 35-22
- object type mappings
 - configuring, 35-22
- ObjectLevelChangeTrackingPolicy, 100-7
- object-relational descriptors
 - configuring, 30-1
 - locking policy, 28-62
 - understanding, 26-11
- object-relational mappings
 - about, 49-1
 - array, understanding, 49-2
 - configuring, 50-1
 - nested table, 49-3, 55-1
 - object array, 49-2, 54-1
 - overview, 2-15
 - reference, 49-2, 52-1, 53-1
 - structure, 49-2, 51-1
- object-relational projects
 - about, 20-6
 - sequencing, 20-5
- objects
 - cascading refresh in cache, 96-35
 - creating and registering, 102-2
 - query, 98-6
 - refreshing in cache, 96-35
 - registering and unregistering, 102-1
- ObjectTypingMapping
 - see* ObjectTypeConverter
- OC4J. *See* Oracle Containers for J2EE
- one-to-many EIS mappings
 - configuring, 63-1
 - key on source, 56-13
 - key on target, 56-15
 - understanding, 56-12
- one-to-many relational mappings
 - configuring, 42-1
 - understanding, 36-7
- one-to-one EIS mappings
 - configuring, 62-1
 - key on source, 56-9
 - key on target, 56-10
 - understanding, 56-8
- one-to-one relational mappings

- configuring, 40-1
 - expressions, 97-5
 - joining, 40-1
 - understanding, 36-5
 - with EJB, 36-6
- online help, 4-14
- Open Project button, 21-10
- opening projects, 21-10
- operators
 - boolean logic, 97-3
- optimistic locking
 - about, 26-9
 - application architecture, 2-10
 - cascading locking policy, 26-18, 28-65
 - database exception, 98-24
 - exceptions, 13-29
 - field locking policy, about, 26-20
 - rollbacks, 26-20
 - version locking policy, 26-17, 26-18, 28-65
 - with `forceUpdateToVersionField()`
 - method, 102-17
- optimistic locking policy
 - field locking, about, 26-20
 - version locking, 26-17, 26-18, 28-65
- `OptimisticLockException`, 26-18, 26-21
- 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, 75-11
 - existence checking, 11-13
 - fetch groups, 11-17
 - fetch size, JDBC, 11-17
 - general, 11-8
 - inheritance, 26-16
 - instantiation, 11-13
 - JDBC driver, 11-14, 11-17
 - 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, 75-11
 - 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, 75-31
- Oracle Containers for J2EE
 - creating projects from, 7-9
 - migrating to TopLink, 7-5
- Oracle Database
 - date and timestamp mappings, 37-3
 - native sequencing, 20-20
 - Oracle Database
 - SEQUENCE object, 20-19
 - platform, 84-3
 - schema manager type conversion, 6-3
- Oracle extensions
 - hierarchical queries, 99-6
 - Oracle Hints, 99-6
- Oracle Hints, using with TopLink queries, 99-6
- Oracle JDeveloper
 - configuring with Oracle TopLink, 5-1
 - TopLink sessions, 5-3
- 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, 33-1
 - 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, 21-1
 - development process, 4-1
 - `DRIVER_CLASSPATH`, 4-2
 - environment, 4-2
 - error messages, 14-1, 14-2
 - `JDBC_CLASSPATH`, 4-2
 - parts of, 4-3
 - preferences, 4-12
 - project, 4-1, 21-2
 - proxy, 4-13, 4-14, 21-16
 - sample, 4-3
 - table creator, creating, 6-4
 - upgrading projects, 21-10
- Oracle Virtual Private Database (VPD)
 - isolated client sessions, 75-20
 - proxy authentication, 75-21, 84-6, 86-13
- `oracle.sql.TimeStamp`, 37-3

- order
 - query keys, 37-8
 - relational mappings, 37-8
- OrderSibling, 99-7
- orion-ejb-jar.xml file
 - about, 8-9
 - entity-deployment attribute pm-name, 8-9
 - modifying for Oracle TopLink, 8-9
 - persistence-manager attribute
 - class-name, 8-10
 - persistence-manager attribute
 - descriptor, 8-10
 - persistence-manager attribute name, 8-9
 - persistence-manager subentry
 - forpm-properties
 - cache-synchronizations, 8-11
 - customization-class, 8-11
 - db-platform-class, 8-11
 - default-mapping, 8-11
 - project-class, 8-10
 - remote-relationships, 8-11
 - session-name, 8-10
- OTN (Oracle Technology Network), 1-4, 2-3
- OTS (Object Transaction Service)
 - about, 100-3
 - unit of work, 100-3
- outer joins
 - in expressions, 98-11
 - inheritance, 28-47
- output parameter event in stored procedures, 98-22
- OX mappings
 - about, 33-27
 - extensions, simple type translator, 33-13, 33-14
 - read conversions, 33-13
 - write conversions, 33-14

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, 91-15
- parallel
 - expressions, 97-11
 - unit of work, 100-9
- parameter binding
 - about, 11-15
 - byte arrays, 11-15
 - configuring, 23-7, 28-22, 86-9, 98-17
 - database login level, 86-9
 - descriptor level, 28-22
 - external connection pools, 11-16
 - internal connection pools, 11-16
 - J2EE, 11-16
 - named queries, 28-22
 - optimizing, 11-15
 - project level, 23-7
 - queries, 98-17
 - streams, 11-15
 - strings, 11-15
 - trouble shooting, 11-15
- parameterized batch writing
 - about, 11-15
 - setMaxBatchWritingSize(), 11-15
- parameterized expressions
 - about, 97-7
 - example, 97-8
- parameterized SQL
 - enabling on queries, 98-17
 - Oracle TopLink optimization features, 11-26
 - See also* parameter binding
- parser conflicts, XML, 7-3
- partial object reading optimization, 11-19
- passwords, encryption, 21-13
- 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, 2-18
 - BEA WebLogic deployment, 8-13
 - by reachability, 101-6
 - components of, 2-12
 - descriptor, 8-13
 - implementation options, 2-11
 - manager, 7-4
 - OC4J, 7-5
 - projects, 20-2
 - types, 22-5
 - using a persistence layer, 2-13
- persistence manager
 - default, 7-4
 - migration, 7-4
 - restrictions, 7-4
- persistent classes
 - requirements, 2-11
 - types, 22-7
- pessimistic locking
 - about, 26-9, 26-21
 - application architecture, 2-10
 - policy, 26-21
- phantom reads, preventing, 102-30
- platforms
 - Attunity Connect database, 84-3
 - Cloudscape database, 84-3

- data source, 84-3
- database, 21-3, 23-2, 24-2, 84-3, 86-1, 87-1
- DB2 database, 84-3
- EIS, 84-4
- functions in expressions, 97-16
- HSQL database, 84-3
- Informix database, 84-3
- J2C adapter, 84-4
- JDBC drivers, 84-3
- Microsoft Access database, 84-3
- MySQL4 database, 84-3
- Oracle database, 84-3
- Oracle8 database, 84-3
- Oracle9 database, 84-3
- parser, XML, 7-2
- PointBase database, 84-3
- projects, and, 20-4
- server, 77-15
- session configuration, 77-15
- SQLAnywhere database, 84-3
- SQLServer database, 84-3
- SybasePlatform database, 84-3
- XML parser, 7-2
- see also* target platforms
- PointBase platform, 84-3
- pop-up menus. *see* context menus
- ports
 - multicast group, 91-5
 - permissions, 7-23, 7-24
- post-commit clones, avoiding, 102-30
- Potential EJB Descriptors dialog box, 21-10
- pre-allocating sequence numbers, 20-20, 23-4, 86-5, 87-2
- 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, 23-7, 28-22, 86-9, 98-17
 - database login level, 86-9
 - descriptor level, 28-22
 - external connection pools, 11-16
 - internal connection pools, 11-16
 - J2EE, 11-16
 - named queries, 28-22
 - optimizing, 11-15
 - project level, 23-7
 - queries, 98-17
 - query level, 98-17
- preserving XML documents, 32-6
- primary key
 - about, 2-16
 - composite, 36-9
 - inheritance, 26-15
 - multiple tables, 29-13
 - primkey in ejb-jar.xml file, 21-15
 - read-only settings, 35-2
 - restrictions, 20-15
 - setting, 4-26, 28-3, 29-2
 - unit of work, 100-11
 - unknown, 28-45
 - variable class relationships, 36-6, 41-3
- private relationships, 35-16
- Problems window
 - about, 4-4, 4-11
 - sample, 4-11
 - see also* error messages
- profiler
 - about, 77-10
 - development tool, 11-2
 - DMS, 11-4, 75-11
 - Oracle TopLink, 11-2, 11-3, 75-11
 - selecting, 77-11, 77-14, 77-17, 77-20
- Project - Multiple Projects tab, 77-10
- Project Status Report dialog box, 21-13
- projects
 - about, 19-1, 20-1, 20-2, 21-2
 - architecture, 20-2
 - cache type and size, 22-13, 28-35
 - comments, 22-21
 - configuring, 22-1
 - copying objects, 5-6
 - creating, 7-9, 21-1, 21-2, 21-3, 21-6
 - deployment login, and, 20-3
 - deployment overview, 2-14
 - descriptors, 19-2
 - development login, and, 20-4
 - direct access to mapped fields, 22-4
 - EIS, 20-7, 20-9, 24-1
 - errors, 14-2
 - existence checking, 22-8
 - exporting, 21-13, 21-14
 - for sessions, 77-9
 - indexed records, 20-9
 - Java, 21-3
 - JAXB, 21-6
 - locked, 5-6
 - login, 20-3, 74-1, 84-2
 - login, and, 20-3
 - mapped field access, default, 22-4
 - mapped records, 20-9
 - mapping projects, creating, 21-2
 - mappings, 19-2, 21-2
 - merging, 5-4
 - metadata, 2-20
 - method access to mapped fields, 22-4
 - model, exporting, 21-14
 - .mwp file, 4-1

- nondesoriptor classes, 4-51
- non-persistent, 20-2
- nonrelational, 20-2
- object-relational, types supported, 20-6
- OC4J, creating from, 7-9
- open, 21-10
- Oracle TopLink Workbench, 21-2
- packages, renaming, 4-52
- persistence type, 20-2, 22-7
- platforms, and, 20-4
- prior TopLink versions, 21-10
- recently opened, 21-10
- relational, 20-2, 20-6, 23-1
- renaming, 21-12
- reopening, 21-10
- saving, 21-11
- sequencing, 20-4, 20-5, 23-3, 86-4
- session login, and, 20-3
- sharing, 5-6
- status report, 21-12
- team development, 5-3
- TopLink Workbench, Navigator window, 4-9
- types of, 20-1, 21-2, 26-1
- updating from `ejb-jar.xml`, 21-16
- upgrading from 2.x or 3.x, 21-10
- writing `ejb-jar.xml` file, 21-16
- XML, 20-9, 25-1
- XML records, 20-9
- `projects.xml` file
 - about, 8-2
 - EJB 3.0, 8-3, 8-5
 - schema, 8-2
 - XSD file, 8-2
- propagation mode, cache, 91-2
- proxies. *see* wrapper policy
- proxy authentication
 - about, 84-5
 - applications, 84-5
 - Oracle Virtual Private Database, 75-21, 84-6, 86-13
 - session events, 75-6
 - use cases, 84-5
- proxy indirection
 - about, 33-8
 - example, 35-9
 - restrictions, 33-8
- proxy settings, preferences, 4-13
- public source code, 12-2

Q

- qualified names, database tables, 4-23, 4-24
- queries
 - about, 96-1
 - application development process, 2-13, 74-2
 - building, 96-6
 - cache, 96-29
 - `Call` queries, 96-3, 96-16
 - cascading, 96-14, 98-13
 - concepts, 96-2

- conforming, 102-8
- database queries, 96-10, 98-4
- `DatabaseQuery`, 96-3
- descriptor query manager, 96-3
- EIS interactions, 96-6
- EJB finders, 96-23
- EJB QL query language, 96-5
- `ejb-jar.xml` file, 28-10, 29-6, 31-6
- executing, 96-7
- expressions, 96-3
- fetch groups, 96-12
- flashback queries, 81-1, 96-18
- hierarchical queries, Oracle extensions, 96-18
- hints, Oracle extensions, 96-18
- historical, 96-20, 99-2
- interface and inheritance queries, 96-22
- joining, 96-12
- languages, about, 96-4
- named queries, 96-15
- object level modify query, 96-13, 96-14
- object level read query, 96-11, 96-13
- on inheritance hierarchies, 99-4
- on interfaces, 99-4
- optimizing, 11-16
- Oracle database features, 99-5
- Oracle extensions, 96-17, 96-18
- parameter binding, 98-17
- partial object query, 96-11
- performance, 11-16
- prepared statement caching, 98-17
- query keys, 96-4
- read all query, 96-11
- read object query, 96-11
- redirect queries, 96-19
- remote sessions, 75-31
- report, 11-21, 96-8
- report query, 96-15
- results, 96-8
- returning policy, 96-17
- session queries, 96-9, 98-1
- SQL query language, 96-4
- stored functions, Oracle extensions, 96-18
- subqueries, 97-10
- summary queries, 96-3
- timeout, 28-24, 28-26, 98-10
- types, 96-1
- `UpdateAll`, 96-14
- XML query language, 96-5
- Queries tab, 28-26, 28-28
- query by example, 98-6
- query cache, 96-36
- Query Exception, 13-30
- query keys
 - about, 28-29, 28-33, 96-4
 - adding, 28-31, 28-34
 - and expressions, 97-9
 - creating, 28-31, 28-34
 - direct mappings, 28-30
 - generating, 28-30
 - in expressions, 97-9

- interface descriptors, 28-31, 28-33
- Java implementation, 28-31
- modifying, 28-30
- order, 37-8
- relationship mappings, 28-30
- specifying, 28-31, 28-34
- unmapped attributes, 28-30, 28-31
- Query Keys tab, 28-31
- query object query. *See* DatabaseQuery
- query objects
 - batch reading, 98-10
 - cache expiration, 99-22
 - caching results, 96-36, 99-21
 - examples, 98-5
 - ordering for ReadAll queries, 98-8
 - report query, 11-21
 - specifying collection class, 98-9
- query results
 - about, 96-8
 - caching, 96-36
 - collections, 96-8
 - cursors, 96-8
 - reports, 96-8
 - streams, 96-8
- query timeout example, 98-10
- QueryManager
 - about, 26-8
 - joining expressions, 99-4
- QuerySequence, 86-8

R

- read access
 - providing in sessions, 75-15
- read all operation, 98-2
- read conversions
 - simple type translator, 33-13
- read only
 - descriptors, 28-4
 - files, 5-6
 - mappings, 35-2, 35-3
- read operation, 98-2
- read optimization
 - about, 11-19
 - batch reading, 11-19
 - fetch groups, 28-77
 - joining, 11-19
 - lazy attribute loading, 28-77
 - object indirection, 11-19
 - partial object reading, 11-19
 - 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, 96-35
 - identity map cache refresh, 96-35
 - refreshing identity maps, 96-35
- ReadAll finders
 - executing, 99-10
- ReadAll queries
 - ordering in query objects, 98-8
- readAllObjects()
 - about, 29-8
 - example, 98-2
- reading
 - ejb-jar.xml, 21-16
 - just-in-time reading, 33-8
 - whole XML documents, 39-1
- reading through the write connection, 102-27
- read-locking, 26-9
- readObject()
 - example, 98-2
- recently opened projects, 21-10
- redirect queries
 - about, 96-19, 99-1
 - creating, 99-1
 - finders, 99-13
 - using, 99-12
- reference key field, 36-11, 36-12
- reference object-relational mappings
 - configuring, 52-1, 53-1
 - understanding, 49-2
- ReferenceMapping class, 49-2
- references
 - database tables, 4-27, 4-28
 - foreign keys, 14-12, 14-21
- Refresh from Database button, 4-24
- refresh policy
 - cache, 90-8
 - EJB finders, 90-8
- refreshing
 - cache, 28-27
 - classes, 4-50
 - refreshObject(), 98-2
 - remote sessions, 75-31
 - sessions, 78-5
- registering objects, 102-1
- registerNewObject method, 102-2
- reimporting schemas, 4-36
- relation tables
 - about, 36-9
 - many-to-many mappings, 43-1
 - mappings, 43-2
- relational descriptors
 - associated table, 29-2
 - configuring, 29-1
 - locking policy, 28-62
 - understanding, 26-11
- relational mappings
 - about, 36-1
 - aggregate collection, 36-10, 44-1
 - aggregate object, 36-12, 47-1
 - configuring, 37-1
 - converters, 36-3
 - direct collection, 36-11, 45-1
 - direct map, 36-12, 46-1
 - direct-to-field, 36-4, 38-1
 - direct-to-XMLType, 36-4, 39-1
 - many-to-many, 36-8, 43-1

- one-to-many, 36-7, 42-1
- one-to-one, 36-5, 40-1
- options, 37-1
- order, 37-8
- transformation, 36-3, 36-15, 48-1
- variable one-to-one, 36-6, 41-1
- relational projects
 - about, 20-2
 - configuring, 23-1
 - object-relational databases, 20-6
 - relational databases, 20-6
 - understanding, 20-6
- relationships
 - about, 2-17
 - bidirectional, 4-31, 35-34, 36-5
 - expressions, 97-5
 - in `ejb-jar.xml` file, 21-16
 - unexpected multiplicity, 7-14
- relative locations
 - about, 22-2
 - classpath, 22-3
- Remote Command Manager Exception, 13-77
- remote connection using RMI, example, 76-10
- remote sessions
 - about, 75-2, 75-29
 - application layer, 75-30
 - creating, 76-10
 - limitations of, 75-31
 - securing access, 75-31
 - server layer, 75-31
 - transport layer, 75-30
 - unit of work, 75-32
- Remove Class button, 4-51
- Remove Table button, 4-24
- removing sessions from brokers, 82-1
- Rename dialog, 4-24
- renaming
 - packages, 4-52
 - projects, 21-10, 21-12
- reopening projects, 21-10
- report query
 - about, 96-15
 - query objects, 11-21
 - read optimization, and, 11-20
 - using, 11-21, 96-8
- reports
 - project status, 21-12
 - query results, 96-8
 - see also* status reports
- resuming unit of work
 - after commit, 102-14
 - after commit failure, 102-14
- returning policy
 - configuring, 28-65
 - SQLCall, 96-17
- Returning tab, 28-66
- RMI
 - coordinated cache, 93-1
 - naming service, 91-9
 - remote session support, 75-31

- rollback
 - and Java Transaction API, 100-11
 - overview, 100-11
 - with optimistic locking, 26-20
- root class
 - about, 26-12
 - inheritance, 26-12
- root EIS descriptor, 27-5
- root element, descriptor, 32-5
- runtime
 - acquiring sessions, 75-5
 - components, 3-2
 - services, configuring sessions, 77-14
 - troubleshooting, 15-14

S

- Save All button, 21-11
- Save button, 21-11
- saving projects, 22-2
- schema context
 - descriptors without, 32-3
 - EIS descriptors, 31-2
 - XML descriptors, 32-2
- Schema Document Info tab, 4-36, 4-38
- schema manager
 - about, 6-1
 - automatic table creation, 6-6
 - creating a table creator, 6-4
 - DB2, 6-3
 - default table generator, 6-4
 - Java table creator, 6-4
 - MS Access, 6-3
 - MySQL, 6-3
 - Oracle, 6-3
 - Oracle TopLink Workbench table creator, 6-4
 - sequencing, 6-3, 6-6
 - Sybase, 6-3
 - table creator, 6-2, 6-4, 6-6
 - table definition, 6-2
 - type conversion, 6-3
 - usage, 6-1
- Schema Structure tab, 4-34
- schemas
 - about, 2-16
 - context for EIS descriptors, 31-2
 - context for XML descriptor, 32-2
 - data storage, 2-16
 - database, 4-22
 - default root for EIS descriptors, 31-4
 - details, 4-34
 - document information, 4-36, 4-38
 - errors, 14-20
 - importing, 4-35
 - optimizing, 11-8
 - properties, 4-36
 - reimporting, 4-36
 - schema manager, 6-1
 - structure, 4-34
 - XML schemas, 4-34

- SCM. *see* source control management
- scripts
 - SQL, generating, 4-29
 - see also* SQL
- scrollable cursor
 - traversing, 99-16
 - using for ReadAllQuery, 99-16
- SDK
 - Data Store Exception, 13-71
 - Descriptor Exception, 13-75
- 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, 21-13
 - 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, 26-21
- sensors, DMS profiler, 11-4, 75-11
- SEQ_COUNT column in sequence table, 20-20
- sequence connection pools, 84-8
- sequence numbers, write optimization
 - features, 11-26
- SEQUENCE objects in Oracle native
 - sequencing, 20-19
- sequencing
 - about, 20-14
 - BEA WebLogic, 20-21
 - BEA WebLogic single column sequence
 - table, 20-17, 23-3, 86-4
 - configuring, 20-15
 - connection pools, 84-8
 - default, 86-5, 86-6
 - DefaultSequence class, 86-6
 - descriptors, 26-9, 29-3
 - entity beans, 20-21
 - IBM WebSphere, 20-21
 - isolated client sessions, 75-24
 - Java configuration, 20-15
 - Microsoft SQL Server, 6-6
 - native, 20-18, 20-19, 20-20
 - non-Oracle database native, 20-19
 - Oracle Database native, 20-18, 20-20
 - Oracle TopLink Workbench configuration, 20-15
 - overriding default, 86-6
 - platform default, 86-5
 - preallocation, 11-28, 20-20
 - QuerySequence class, 86-8
 - schema manager, 6-6
 - SEQ_COUNT, 20-20
 - sequence type, configuring, 23-3, 86-4
 - sessions, and, 75-13
 - stored procedures, 20-18, 86-7
 - Sybase Database, 6-6
 - table, 20-16
 - table, default column and table names, 20-17
 - unary table, 20-17
 - with stored procedures, 86-7
- serialization
 - descriptor exceptions, 102-13
 - EJB 1.x, 33-9
 - EJB 2.x, 33-9
 - EJB 3.0, 33-9
 - indirection, 33-9
 - merging into session cache with unit of
 - work, 102-12
- serialized object converter
 - about, 12-2, 33-10
 - configuring, 35-18
- serialized object mappings, 35-19
- SerializedObjectMapping. *see*
 - SerializedObjectConverter
- server layer, 75-31
- server platform
 - external transaction controller, 77-14
 - runtime services, 77-14
 - session configuration, 77-14
 - session event listener, 77-17
- Server Platform tab, 77-15
- server sessions
 - about, 75-1, 75-13, 75-15
 - cache, 75-32
 - connection options, 75-18
- service channel, coordinated cache, 91-3
- session accessor
 - about, 2-30
 - managing dependent objects with, 2-30
- session beans
 - about, 2-25
 - deploying, 8-8, 9-3, 10-1
 - model, 2-23, 2-25
 - remote session support for, 75-31
- session brokers
 - about, 75-2, 75-25
 - adding sessions to, 82-1
 - alternatives, 75-28
 - architecture, 75-26
 - configuration, 82-1
 - limitations of, 75-27
 - renaming, 82-1
 - two-phase commits, 75-27
 - two-stage commits, 75-27
- session configuration file
 - about, 76-1
 - loading alternative, 78-4

- preferences, 4-18, 4-19
 - see also* sessions.xml file
- session customizer, 75-4
- Session Event Manager, 75-5
- session events
 - example, 75-5, 75-6
 - isolated sessions, and, 75-5
 - manager, 75-5
 - proxy authentication, and, 75-6
- Session Loader Exceptions, 13-56
- Session Manager
 - about, 78-1, 78-2
 - acquiring, 78-2, 78-3
 - defaults, 78-3
 - destroying sessions, 78-10
 - J2EE defaults, 78-3
 - sessions, acquiring, 75-5
 - storing sessions, 78-10
- sessions
 - about, 2-12, 74-1, 75-1, 75-2, 76-1
 - acquiring at runtime, 75-5, 78-1
 - adding to session brokers, 82-1
 - additional mapping projects, 77-9
 - API, 75-33
 - application server logging, 75-9
 - architecture, 75-2
 - cache, 75-32, 90-2
 - client, 75-13
 - configuring, 77-1, 77-17
 - connection policy, 77-19
 - creating, 2-21, 76-1, 76-4, 76-6, 76-8, 88-1
 - customization, 75-4
 - database sessions, 75-28, 76-8
 - destroying in Session Manager, 78-10
 - errors, 14-21
 - event listeners, 75-7, 77-17
 - events, 75-6
 - external transaction controller, 77-14, 100-2
 - historical client sessions, 75-24
 - in sessions.xml file, 75-4
 - isolated client sessions, 75-19, 80-1
 - loading with alternative class loader, 78-4
 - logging, 75-7, 75-9, 75-10, 77-4, 77-5
 - logging into, 78-9
 - logging out of, 78-10
 - logins, 77-4
 - management of, 78-1
 - metadata, about, 2-21
 - multiple sessions, 75-28, 78-2
 - named queries, 77-21
 - object cache, 75-3
 - optimizing, 11-13
 - preferences, 4-18, 4-19
 - queries, 96-7, 96-9, 98-1
 - refreshing, 78-5
 - registering descriptors, 75-12, 77-2, 77-3, 77-10
 - remote sessions, 75-29, 76-10
 - sequencing, about, 75-13
 - server, 75-13
 - server platform, 77-14, 77-15
 - SQL and messages, 75-10
 - storing in Session Manager, 78-10
 - three-tier architecture, 75-14
 - transformation mappings, 36-15
 - types, 75-1
 - unit of work, 75-1, 75-19
- sessions.xml file
 - about, 8-3
 - acquiring, 75-5
 - CMP applications, 8-4, 75-4
 - configuring with JDeveloper, 5-3
 - creating, 76-1
 - default location, 75-5
 - EJB 3.0, 8-5
 - loading alternative configuration file, 78-4
 - non-CMP applications, 8-4
 - schema, 8-4
 - sessions, 75-4
 - XSD file, 8-4
- setAdditionalJoinExpression(), 99-5
- setenv.cmd file, 4-2
- setenv.sh file, 4-2
- setMaxBatchWritingSize(), 11-15
- setMultipleTableJoinExpression(), 99-5
- setShouldPerformDeletesFirst(), 102-16
- Settings tab, 28-26, 28-28
- setValue() method, 33-7
- shared library
 - setting for BEA WebLogic, 7-15
- shared library for BEA WebLogic, 7-15
- simple type translators
 - about, 12-2, 33-12
 - configuring, 35-23
 - in Java, 35-25
 - read conversions, 33-13
 - write conversions, 33-14
- soft cache weak identity map
 - about, 90-4
 - when to use, 90-5
- sorting, in memory, 35-26
- source code, public, 12-2
- source control management
 - projects, 5-3
 - with Oracle TopLink Workbench, 5-3
 - see also* team development
- source table, reference, 4-28
- source.jar, 12-2
- splash screen, 4-13
- SQL
 - call, 98-18
 - comparing with expressions, 97-1
 - custom queries for basic persistence, per descriptor, 29-6
 - EJBQLCall, 98-23
 - generating from database tables, 4-29
 - parameter binding, 11-15
 - parameterized, 11-26, 98-17
 - prepared statement caching, 11-15
 - queries, 96-4
 - scripts with binding arguments, 29-6

- SQLCall, 98-18
 - StoredFunctionCall, 98-22
 - StoredProcedureCall, 98-20
 - unit of work, 102-15
- SQL Creation Script dialog box, 4-29
- SQL DISTINCT, 11-24
- SQL Exception, 13-27
- SQLAnywhere platform, 84-3
- SQLCall
 - about, 96-17
 - binding input parameters, 98-19
 - input parameters, 98-19
 - input-output parameters, specifying, 98-20
 - output parameters, 98-19
 - Returning Policy, 96-17
 - using, 98-18
- SQLServer platform, 84-3
- stages of development with Oracle TopLink, 2-1
- stale data
 - cache, 90-6, 90-7
 - coordination, cache, 90-7
 - invalidating the cache, 90-7
 - locking policy, and, 90-6
 - per-class cache configuration, 90-7
 - per-query cache refresh, 90-7
- StartWith, 99-7
- stateful
 - beans, 2-25
 - comparing with stateless, 2-25
- stateless
 - comparing with stateful, 2-25
- static attributes, 4-45, 4-48
- static fetch groups, querying with, 99-3
- status report, generating, 21-12
- stored functions
 - about, 96-18
 - using, 98-22
- stored procedures
 - output parameter event, 98-22
 - queries, 98-20
 - sequencing, and, 20-18, 86-7
- streams
 - as query results, 96-8
 - cursor, 75-32, 99-18
- string bindings, CLOB mappings, 34-4
- structure object-relational mappings
 - configuring, 51-1
 - understanding, 49-2
- StructureMapping class, 49-2
- subqueries
 - multiple expressions, 97-10
 - subselects in expressions, 97-10
- subselects, in expressions, 97-10
- summary queries, 96-3
- superclass, 4-41
- Sybase
 - database schema manager type conversion, 6-3
 - native sequencing, 20-19
 - platform, 84-3
- Synchronization Exceptions, 13-70

- synchronous change propagation, 91-2
- 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, 6-2
 - creating, 6-4
 - using, 6-6
- Table Creator dialog, 6-4
- table generation. *see* automatic table generation
- table sequence
 - about, 20-16
 - default column and table names, 20-17
- table-class relationships, 16-6
- tables
 - adding database, 4-22
 - associating with relational descriptors, 29-2
 - database, 4-21
 - defining schema, 6-2
 - errors, 14-15
 - generating, automatic, 33-4
 - import filter, 4-23
 - mapping to descriptors, 28-3, 29-2
 - merging files, 5-5
 - multiple, 29-13
 - primary key, 4-26
 - references, 37-8
 - relation tables for mappings, 43-2
 - TableDefinition class, 6-2
 - see also* database tables
- target foreign keys
 - about, 36-6, 37-9
 - configuring, 37-9
- target platforms
 - about, 2-5
 - choosing, 2-3
- target tables
 - in direct collection mappings, 45-1
 - reference, 4-28
- team development, 5-3
- technical support, 2-3
- three-tier architecture
 - about, 1-4, 2-21
 - authentication, 84-5
 - migrating to scalable architecture, 75-29
 - overview, 2-21
 - sessions, 75-14
- timestamp support
 - about, 37-3
 - direct to field mappings, 37-3
 - Oracle Database, 37-3
 - TIMESTAMP timezone, 37-4
- TimestampLockingPolicy, 26-18
- timezone, with TIMESTAMP, 37-4

- tljxb.cmd file, 21-9
- toolbars, 4-4, 4-6
- topic name, 92-1
- TopLink. *see* Oracle TopLink
- TopLink expressions. *see* expressions
- TopLink profiler
 - about, 11-2
 - selecting, 77-11, 77-14, 77-17, 77-20
- TopLink Workbench. *see* Oracle TopLink Workbench
- toplink-ejb-jar.xml file, 8-6, 19-2
- toplink-ejb.xml file, 8-8
- Transaction Exception, 13-79
- transactional data sources, 84-1
- transactions
 - client-controlled, 102-24
 - CMP, 100-3, 102-24, 102-25
 - container-controlled, 102-24
 - demarcation, 100-2
 - external transaction controller, 77-14, 100-2
 - external, integrating, 100-2, 102-20
 - isolated client sessions, 75-24
 - isolation, 100-2, 100-4
 - JTA, 100-3
 - JTS, 100-3
 - local, 102-25
 - local, CMP, 102-25
 - OTS, 100-3
 - overview, 2-13, 74-3, 100-1
 - see also* unit of work
- transformation EIS mappings
 - configuring, 64-1
 - understanding, 56-17
- transformation mappings
 - about, 33-14
 - attribute transformation, 35-29, 35-31
 - mutable, 35-33
- transformation relational mappings
 - configuring, 48-1
 - understanding, 36-15
- transformation XML mappings
 - configuring, 73-1
 - understanding, 65-31
- transient attributes, 4-45, 4-48
- transparent indirection
 - about, 33-8
 - persistent class requirements, 2-11
- transport layer, 75-30
- troubleshooting
 - BEA WebLogic deployment, 15-3
 - data sources, 14-23
 - 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, 102-30, 102-34
- tutorial
 - about, 16-1
 - database schema, 16-4
 - planning, 16-1
 - two-phase commits, 75-27
 - two-stage commits, 75-27
 - two-tier architecture
 - about, 1-6, 2-23
 - authentication, 84-5
 - understanding, 2-23
 - type conversion
 - automatic, 37-2
 - NCHAR, 33-12
 - NCLOB, 33-12
 - NVARCHAR2, 33-12
 - oracle.sql.TimeStamp, 37-3
 - schema manager, 6-3
 - String to TIMESTAMP, 33-12
 - TIMESTAMP to String, 33-12
 - type conversion converter
 - about, 12-2, 33-11
 - configuring, 35-20
 - provided by direct-to-field mappings, 37-3
 - TypeConversionMapping
 - see* TypeConversionConverter
 - types of mappings, 33-1
 - typesafe enumeration, in EIS mappings, 56-3

U

- unary table sequence
 - about, 20-17
 - BEA WebLogic single column sequence
 - table, 20-17, 23-3, 86-4
- undeployment, 10-4
- unexpected relationship multiplicity, 7-14
- unidirectional relationships, 36-2
- unit of work
 - about, 100-1, 100-2, 100-4
 - acquiring, 101-1
 - API, 100-11
 - architecture, 100-1
 - auditing, 102-20
 - benefits of, 100-4
 - cache, 90-2
 - change policy, 100-6
 - clones, 100-9
 - CMP integration, 100-3
 - commit and Java Transaction API, 100-10
 - commit, writing changes before, 102-7, 102-11, 102-23
 - conform results of in-memory query, 96-31
 - creating objects, 101-2
 - deleting objects, 101-7
 - example, 100-6
 - external transaction controller, 77-14, 100-2
 - external transactions, 100-2, 102-20
 - integrating with CMP, 102-24
 - isolation, 100-4
 - JTA integration, 100-3
 - JTS integration, 100-3
 - life cycle, 100-5
 - modifying objects, 101-2

- mutable mappings, 35-33
- nested, 100-9, 102-15
- `newInstance` method, 102-2
- optimization, 11-30, 100-11
- OTS integration, 100-3
- parallel, 100-9, 102-15
- pre-commit validation, 102-34
- primary keys, 100-11
- proxy indirection, 33-8
- queries, 100-12
- read optimization, 11-19
- read-only classes, 102-6, 102-7
- `registerNewObject` method, 102-2
- remote sessions, 75-32
- resuming, 102-14
- reverting, 102-14
- rollback, 100-11
- sessions, 75-1, 75-19
- transaction demarcation, 100-2
- transactions, 100-4
 - unit of work
 - demarcation, 100-2
- troubleshooting, 102-30
- validating objects, 102-34
- with custom SQL, 102-15
- write optimization, 11-26
- writing changes before commit, about, 102-7
- writing changes before commit, and external
 - transaction exceptions, 102-23
- writing changes before commit, and external
 - transaction timeouts, 102-23
- writing changes before commit, as alternative to
 - conforming, 102-11
- unmapping, 34-6
- unregistering objects, 102-1
- unsaved items, displaying in TopLink Workbench
 - Navigator window, 4-9
- update
 - operation, 98-3, 98-4
 - projects from prior versions, 21-10
- `UpdateAll` query, 96-14
- `updateObject()`, 29-8
- Use XML Schema "type" attribute,
 - configuring, 35-23
- `useBatchWriting()`, 86-11
- user-defined functions, in expressions, 97-16

V

- validating
 - descriptors, 27-6
 - JAXB, 20-14
 - projects, 21-12
 - unit of work, 102-34
- Validation Exceptions, 13-42
- Value Converter tab, 46-4
- value holders
 - about, 33-6
 - `ValueHolder` class, 33-6
- `ValueHolderInterface` class, 2-11, 33-7, 36-9

- variable one-to-one relational mappings
 - class indicator, 41-1
 - configuring, 41-1
 - primary key, unique, 41-3
 - understanding, 36-6
- `VariableOneToOneMapping` class, 36-6
- Varray in Oracle database. *see* array mappings
- version control, 5-6
- Version Control Assistance dialog box, 5-6
- `VersionLockingPolicy`, 26-18
- volatile attributes, 4-45, 4-48
- VPD. *see* Oracle Virtual Private Database

W

- warning icon, 4-10, 4-11
- weak identity map, 90-3
- web browser, specifying, 4-14
- web services architecture, 1-5, 2-36
- WebLogic. *see* BEA WebLogic
- `weblogic-ejb-jar.xml` file
 - described, 8-5
 - modifying for Oracle TopLink, 8-13
 - unsupported tags, 8-14
- WebSphere. *see* IBM WebSphere
- welcome screen, 4-14
- wildcard, 66-3
- wrapper policy
 - about, 28-75
 - implementing in Java, 28-76
- write
 - conversions, simple type translator, 33-14
 - write all operation, 98-3
- write query
 - disabling identity map cache, 98-14
 - non-cascading, 98-13
 - objects, 98-12
 - overview, 98-12
- write-locking, 26-9
- writing
 - batch, 11-26, 86-11
 - `ejb-jar.xml` file, 21-16
 - optimization, 11-26
 - sessions write access, 75-16

X

- `xdb.jar` file, 4-3
- XML
 - descriptor, schema context, 32-2
 - generating deployment, 8-3
 - mappings, concepts, 65-2
 - preserving data, 32-6
 - projects, 21-3
 - query language, 96-5
 - reading whole documents, 39-1
 - records, 56-3
 - schemas, 4-34
- XML Conversion Exception, 13-80
- XML Data Store Exceptions, 13-64

- XML descriptors
 - configuring, 32-1
 - schema context, 32-2
 - understanding, 26-12
- XML mappings
 - about, 65-1
 - any collection, 65-29, 72-1
 - any object, 65-27, 71-1
 - any type support, 65-4
 - composite collection, 65-25, 70-1
 - composite direct collection, 65-14, 68-1
 - composite object, 65-21, 69-1
 - concepts, 65-2
 - configuring, 66-1
 - default conversion pairs, customizing, 65-11
 - direct, 65-5, 67-1
 - extensions, 65-4
 - jaxb:class support, 65-4
 - list support, 65-3
 - reference descriptor, configuring, 66-2
 - transformation, 65-31, 73-1
 - types of, 65-1
 - union support, 65-3
 - xsd:list, 65-3
 - xsd:union, 65-3
- 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 projects
 - configuring, 25-1
 - JAXB support, 20-10
 - sequencing, 20-5
 - understanding, 20-9
- XML queries, 96-5
- XML schema
 - jaxb:class, and EIS mappings, 56-3
 - jaxb:class, and XML mappings, 65-4
 - jaxb:class, understanding, 33-20
 - type, 35-23
 - xs:any, understanding, 65-4
 - xs:anyType, understanding, 65-4
 - xsd:list, understanding, 33-17
 - xsd:union, understanding, 33-17
 - see also* schemas
- XML Type functions, 97-4
- XPath
 - by name, 33-16
 - by position, 33-15
 - mapping Java attributes, 35-10
 - support in OX mappings, 33-15
 - support in XML mappings, 56-3, 65-3
- XSD file

- projects.xml file, 8-2
- sessions.xml file, 8-4

Z

- zero-argument constructors
 - editing, 4-16

