Oracle® Application Server TopLink

Mapping Workbench User's Guide

10*g* (9.0.4)

Part No. B10316-01

September 2003

ORACLE®

Oracle Application Server TopLink Mapping Workbench User's Guide, 10*g* (9.0.4)

Part No. B10316-01

Primary Author: Rick Sapir

Contributors: Jacques-Antoine Dubé, Ellen Siegal (editor)

Contents

2 Understanding Projects

3 Understanding Databases

4 Understanding Descriptors

5 Understanding Direct Mappings

6 Understanding Relationship Mappings

7 Understanding Object-Relational Mappings

8 Understanding the OracleAS TopLink Sessions Editor

A Object Model Requirements

B Tutorials

C Troubleshooting

Index

Send Us Your Comments

Oracle Application Server TopLink Mapping Workbench User's Guide, 10*g* (9.0.4)

Part No. B10316-01

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

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

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

- Electronic mail: appserverdocs_us@oracle.com
- FAX: (650) 506-7225 Attn: Java Platform Group, Information Development Manager
- Postal service:
  Oracle Corporation
  Java Platform Group, Information Development Manager
  500 Oracle Parkway, Mailstop 4op978
  Redwood Shores, CA 94065
  USA

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

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

Preface

This guide includes the concepts necessary for using the Oracle Application Server TopLink Mapping Workbench, a standalone application that creates and manages your descriptors and mappings for a project. This document includes information on each OracleAS TopLink Mapping Workbench function and option.

This preface contains the following topics:

Intended Audience

This document is intended for OracleAS TopLink users who are familiar with the object-oriented programming and Java development environments.

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

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

Documentation Accessibility

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

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

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

Organization

This document includes the following chapters:

Chapter 1, "Understanding the OracleAS TopLink Mapping Workbench"
This chapter introduces the OracleAS TopLink Mapping Workbench – a tool to graphically configure descriptors and map your OracleAS TopLink project.

Chapter 2, "Understanding Projects"

This chapter contains instructions for creating and maintaining OracleAS TopLink project files, including workbench preferences and team development.

Chapter 3, "Understanding Databases"

This chapter describes how to create database sessions and import/export database tables to and from your OracleAS TopLink project.

Chapter 4, "Understanding Descriptors"

This chapter summarizes OracleAS TopLink descriptors, including standard and advanced properties and mappings.

Chapter 5, "Understanding Direct Mappings"

This chapter summarizes the direct mapping types supported by OracleAS TopLink.

Chapter 6, "Understanding Relationship Mappings"

This chapter summarizes the relational mapping types supported by OracleAS TopLink.

Chapter 7, "Understanding Object-Relational Mappings"

This chapter summarizes the object relational mapping types supported in OracleAS TopLink.

Chapter 8, "Understanding the OracleAS TopLink Sessions Editor"

This chapter contains information on using the OracleAS TopLink Sessions Editor to create and maintain OracleAS TopLink sessions.

Appendix A, "Object Model Requirements"

This section summarizes OracleAS TopLink object model requirements.

Appendix B, "Tutorials"

This section includes a basic and advanced tutorials that provide step-by-step instructions for using the OracleAS TopLink Mapping Workbench.

Appendix C, "Troubleshooting"

This section contains information on the types of errors that may occur and how to correct them.

Related Documentation

For more information, see these Oracle resources:

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

Printed documentation is available for sale in the Oracle Store at

`http://oraclestore.oracle.com`

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

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

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

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

Conventions

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

- Conventions in Text
- Conventions in Code Examples

Conventions in Text

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

Convention	Meaning	Example
Bold	Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.	When you specify this clause, you create an **index-organized table**.
Italics	Italic typeface indicates book titles or emphasis.	*Oracle9i Database Concepts* Ensure that the recovery catalog and target database do *not* reside on the same disk.
`UPPERCASE monospace (fixed-width) font`	Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, usernames, and roles.	You can specify this clause only for a `NUMBER` column. You can back up the database by using the `BACKUP` command. Query the `TABLE_NAME` column in the `USER_TABLES` data dictionary view. Use the `DBMS_STATS.GENERATE_STATS` procedure.
`lowercase monospace (fixed-width) font`	Lowercase monospace typeface indicates executables, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, usernames and roles, program units, and parameter values. **Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	Enter `sqlplus` to open SQL*Plus. The password is specified in the `orapwd` file. Back up the datafiles and control files in the `/disk1/oracle/dbs` directory. The `department_id`, `department_name`, and `location_id` columns are in the `hr.departments` table. Set the `QUERY_REWRITE_ENABLED` initialization parameter to `true`. Connect as `oe` user. The `JRepUtil` class implements these methods.
`lowercase italic monospace (fixed-width) font`	Lowercase italic monospace font represents placeholders or variables.	You can specify the *`parallel_clause`*. Run `U`*`old_release`*`.SQL` where *`old_release`* refers to the release you installed prior to upgrading.

Conventions in Code Examples

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

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

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

Convention	Meaning	Example
[ ]	Brackets enclose one or more optional items. Do not enter the brackets.	`DECIMAL (digits [ , precision ])`
{ }	Braces enclose two or more items, one of which is required. Do not enter the braces.	`{ENABLE | DISABLE}`
|	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	`{ENABLE | DISABLE}` `[COMPRESS | NOCOMPRESS]`
. . .	Horizontal ellipsis points indicate either:	
	■ That we have omitted parts of the code that are not directly related to the example	`CREATE TABLE ... AS subquery;`
	■ That you can repeat a portion of the code	`SELECT col1, col2, ... , coln FROM employees;`
. . .	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	
Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown.	`acctbal NUMBER(11,2);` `acct CONSTANT NUMBER(4) := 3;`
Italics	Italicized text indicates placeholders or variables for which you must supply particular values.	`CONNECT SYSTEM/system_password` `DB_NAME = database_name`

Convention	Meaning	Example

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

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

1

Understanding the OracleAS TopLink Mapping Workbench

The Oracle Application Server TopLink Mapping Workbench is a separate component from Oracle Application Server TopLink — it allows you to graphically configure descriptors and map your project. The OracleAS TopLink Mapping Workbench can verify the descriptor options, access the data source, and create the database schema. With the OracleAS TopLink Mapping Workbench you can define OracleAS TopLink descriptors *without using code*.

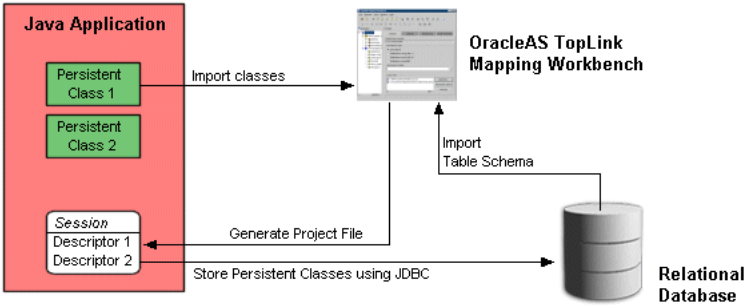
This chapter includes the following subjects:

- Starting the OracleAS TopLink Mapping Workbench
- Working with the OracleAS TopLink Mapping Workbench
- Working with Workbench Preferences
- Working with the OracleAS TopLink Mapping Workbench in a Team Environment

To Use the OracleAS TopLink Mapping Workbench in a Java Application:

1. Define an object model (a set of Java classes) to describe and solve your problem domain.
2. Use the OracleAS TopLink Mapping Workbench to create a project, import your Java classes and relational tables, and specify descriptors to describe how the classes map to your relational model.
3. In your Java application, create an OracleAS TopLink session and register your descriptors. Add logic to your application to use the session to retrieve and store objects from and to the database.

Figure 1–1 Using the OracleAS TopLink Mapping Workbench



Starting the OracleAS TopLink Mapping Workbench

Use this procedure to start the OracleAS TopLink Mapping Workbench.

To Start the OracleAS TopLink Mapping Workbench:

Use one of the following methods to start the OracleAS TopLink Mapping Workbench:

- For Windows environments: From the **Start** menu, choose **Program Files** > **OracleAS TopLink** > **Mapping Workbench**.
- For non-Windows environments: Execute the `<ORACLE_HOME>\toplink\bin\workbench.sh` file.

The splash screen appears, followed by the OracleAS TopLink Mapping Workbench window.

Working with the OracleAS TopLink Mapping Workbench

The OracleAS TopLink Mapping Workbench window includes these parts:

- Menu

  Pull-down menus for each OracleAS TopLink Mapping Workbench function. Some objects also contain context-sensitive pop-up menus. See "Using the Menus" on page 1-4 for more information.

- Toolbars

  Shortcuts to specific functions. See "Using the Toolbars" on page 1-5 for more information.

- Navigator pane

  The project tree for all open projects (see "Using the Navigator Pane" on page 1-9). Click the plus or minus (+/–) next to an object (or double-click the object) to expand/collapse the tree. When you select an object in the **Navigator** pane, its properties appear in the **Editor** pane.

- Editor pane

  Specific property sheets and option tabs for the currently selected object. See "Using the Editor Pane" on page 1-10 for more information.

- Status bar

  Provides instant information regarding the status of projects, descriptors, and mappings.

Figure 1–2 OracleAS TopLink Mapping Workbench

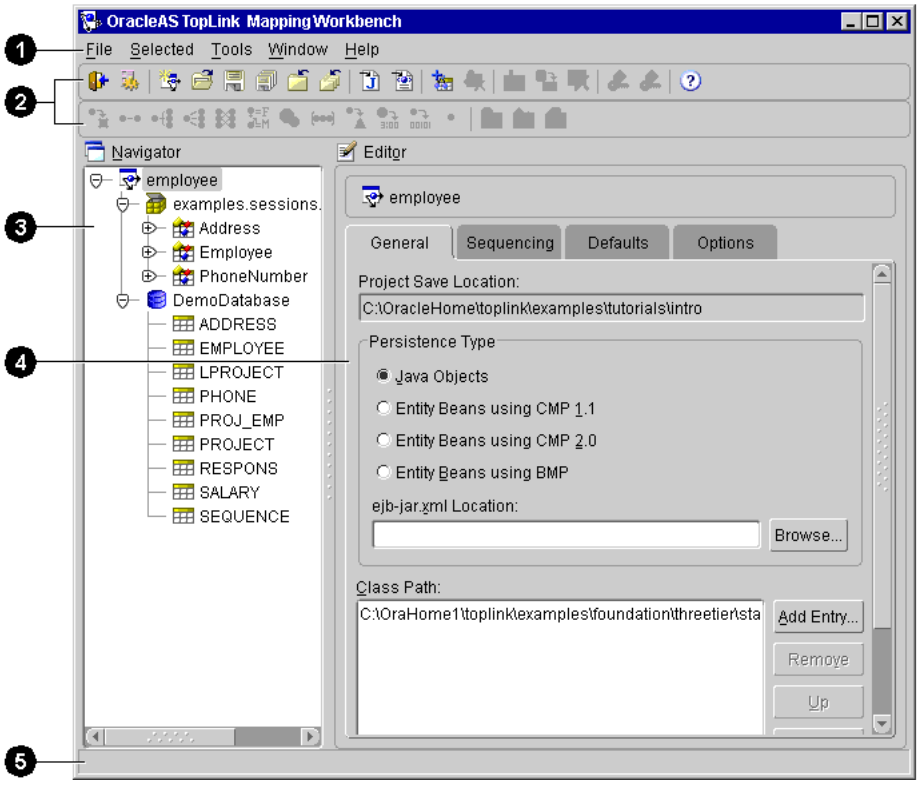


Figure 1–2 calls out the following user-interface components.

1. Menu bar
2. Toolbars
3. Navigator pane
4. Editor pane
5. Status bar

Using the Menus

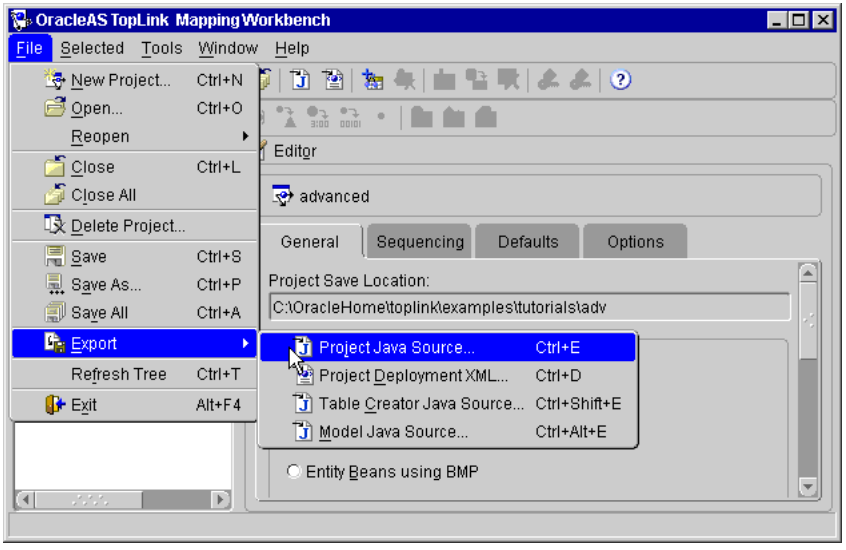
The OracleAS TopLink Mapping Workbench contains two types of menus:

- Menu Bar Menus
- Pop-Up Menus

Menu Bar Menus

The menu bar provides pull-down menus for each OracleAS TopLink Mapping Workbench function. Some menus (such as **Selected**) are context-sensitive — the available options may vary, depending on the currently selected object.

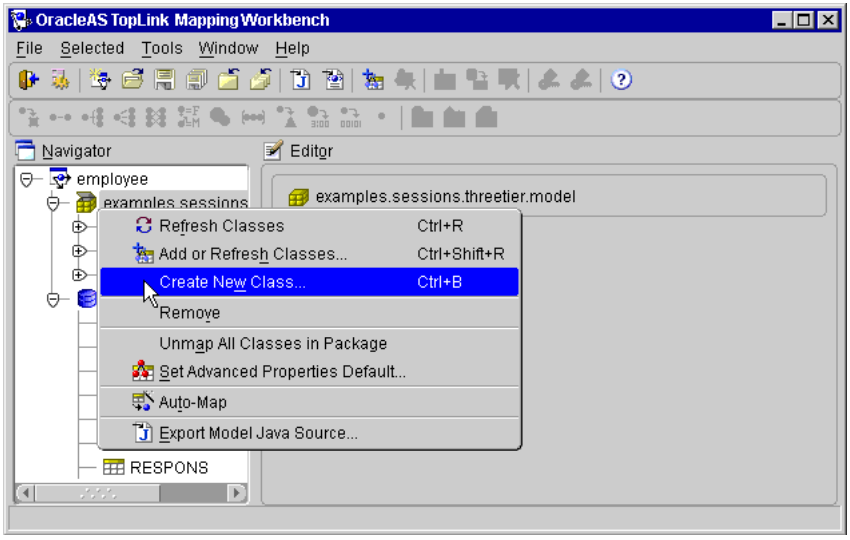
Figure 1–3 Sample Menu Bar Menu



Pop-Up Menus

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

Figure 1–4 Sample Pop-up Menu



Using the Toolbars

The OracleAS TopLink Mapping Workbench contains two types of toolbars:

- Standard Toolbar
- Mapping Toolbar
- OracleAS TopLink Sessions Editor Navigator Toolbar

Use these toolbars to select options and functions.

Standard Toolbar

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

Button	Description
	Exit
	Preferences
	New project
	Open project
	Save
	Save all
	Save as
	Close
	Close all
	Export Java source
	Export deployment XML
	Add or refresh classes
	Remove class
	Add table
	Add or update existing tables from database
	Remove table
	Login to database

Button	Description
	Logout of database
	Online help

Mapping Toolbar

The mapping toolbar provides quick access to create mapping and descriptor types. You can specify a mapping or descriptor type by selecting the object in the **Navigator** pane, then clicking the appropriate button on the mapping toolbar.

You can also right-click the object and choose the appropriate mapping from the pop-up menu.

Button	Description
	Direct-to-Field mapping
	One-to-One mapping
	Variable One-to-One mapping
	One-to-Many mapping
	Many-to-Many mapping
	Object Type mapping
	Aggregate mapping
	Direct Collection mapping
	Transformation mapping
	Type conversion mapping
	Serialized mapping
	Unmap

Button	Description
	Aggregate descriptor
	Class descriptor
	EJB descriptor

OracleAS TopLink Sessions Editor Navigator Toolbar

The OracleAS TopLink Sessions Editor contains a toolbar that appears in the **Navigator** pane and offers quick access to create and modify configurations, sessions, and connection pools.

Button	Description
	Caching
	Login
	Connection Pool
	Rename
	Session
	Session Broker

Using the Navigator Pane

OracleAS TopLink displays each project's descriptors, mappings, and database tables in the **Navigator** pane on the left side of the workbench.

Figure 1–5 Sample Navigator Pane

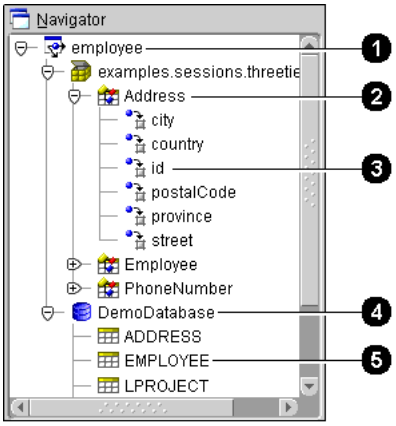


Figure 1–5 identifies the following user-interface components:

1. Project
2. Descriptor
3. Attribute/mapping
4. Database
5. Database table

Click the **+/–** symbol next to the item, or double-click the item name to expand and collapse the item.

When you select an item in the **Navigator** pane, its properties appear in the **Editor** pane (see "Using the Editor Pane" on page 1-10). You can perform specific functions for an item by selecting the item in the **Navigator** pane and:

- Right-clicking on the object and selecting the function from the pop-up menu (see "Pop-Up Menus" on page 1-5).
- Choosing a function from the **Selected** menu (see "Menu Bar Menus" on page 1-4).

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

Figure 1–6 Sample Active/inactive Descriptors

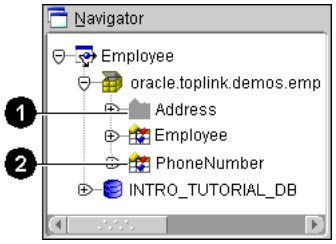


Figure 1–6 shows the following user-interface components.

1. Inactive descriptor
2. Active descriptor

If an element in the project (such as a descriptor mapping), contains an error or deficiency (sometimes called *neediness*), a warning icon ⚠ appears beside the element icon in the **Navigator** pane, and a message displays in the status bar detailing the error. Appendix C, "Troubleshooting" contains complete information on each OracleAS TopLink Mapping Workbench error message.

Using the Editor Pane

The **Editor** pane, on the right side of the OracleAS TopLink Mapping Workbench (see Figure 1–2), displays the property sheet associated with the currently selected item in the **Navigator** pane.

Working with Workbench Preferences

This section includes information on customizing the following aspects of the OracleAS TopLink Mapping Workbench:

- General Preferences
- Warning and Confirmation Preferences
- Class Preferences

- EJB Preferences
- Database Preferences
- Help Preferences

General Preferences

Use the General preferences to customize the "look and feel" (the graphical user interface) of the OracleAS TopLink Mapping Workbench.

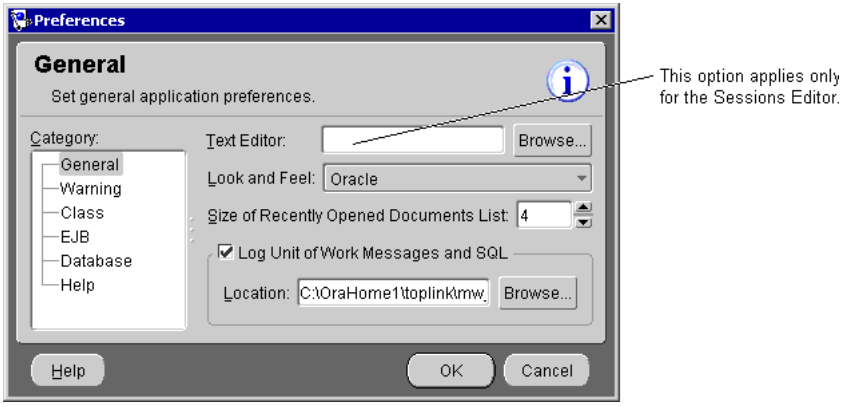
To Change the General Preferences:

1. Click the **Preferences** button on the toolbar. The Preferences dialog appears.

   You can also display the Preferences dialog box by choosing **Tools** > **Preferences** from the menu.

2. Select **General** in the **Category** pane.

Figure 1–7 General Preferences



3. Use this table to enter data in each field and click **OK**.

Field	Description
Text Editor	Click **Browse** and select a text editor (used when viewing the session configuration source). See "Working with the Source" on page 8-17 for more information. **Note**: This field appears for the OracleAS TopLink Sessions Editor only.

Field	Description
Look and Feel	Select the "look and feel" to use for the OracleAS TopLink Mapping Workbench. See Figure 1–8 for a sample of each option.
Size of Recently Opened Documents List	Select the number of projects to maintain in the **File** > **Reopen** option. See "Opening Existing Projects" on page 2-3 for more information.
Log Unit of Work Messages and SQL	Specify if the OracleAS TopLink Mapping Workbench logs runtime messages to help troubleshoot projects. Click **Browse** to select a filename (default `mw_xml.txt`) and **Location**.

You must restart the OracleAS TopLink Mapping Workbench to apply the changes.

Figure 1–8 Look and Feel Samples

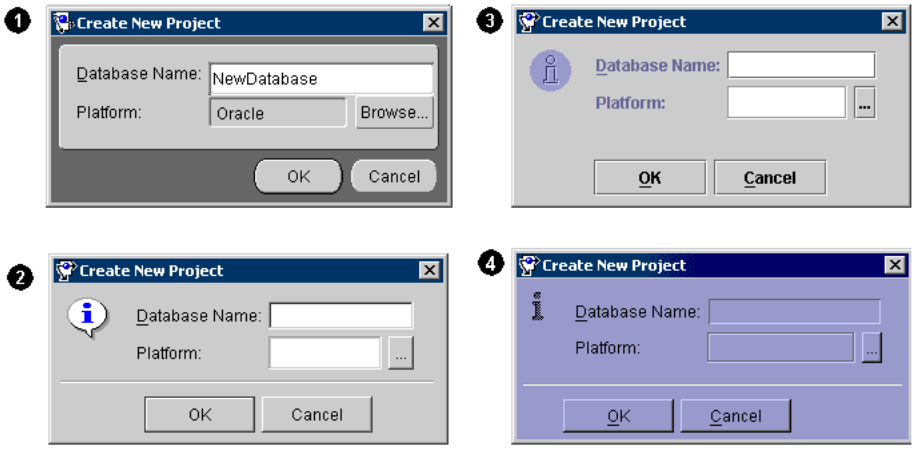


Figure 1–8 shows the following "look and feel" samples:

1. Oracle
2. Windows
3. Metal (Java)
4. CDE/Motif

Warning and Confirmation Preferences

Use the **Warnings** preferences to specify if the OracleAS TopLink Mapping Workbench displays certain warning and confirmation dialog boxes.

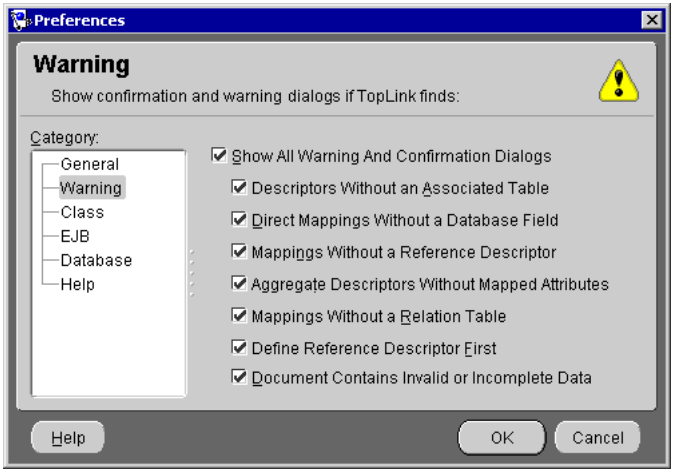
To Specify Warnings Preferences:

1. Click the **Preferences** button on the toolbar. The Preferences dialog box appears.

   You can also display the Preferences dialog box by choosing **Tools** > **Preferences** from the menu.

2. Select **Warning** in the **Category** pane.

Figure 1–9 Warning Preferences



3. Use this table to enter data in each field and click **OK**.

Field	Description
Show All Warning and Confirmation Dialogs	Specify which (if any) of the following confirmation and warning messages are displayed.
Descriptors Without an Associated Table	Specify if the OracleAS TopLink Mapping Workbench displays a warning message if you attempt to create a table reference before defining the descriptor's associated table.

Field	Description
Direct Mappings Without a Database Field	Specify if the OracleAS TopLink Mapping Workbench displays a warning message if you attempt to create a table reference for a direct collection mapping before mapping's direct field and target table.
Mappings Without a Reference Descriptor	Specify if the OracleAS TopLink Mapping Workbench displays a warning message if you attempt to create a mapping before defining a reference descriptor.
Aggregate Descriptors Without Mapped Attributes	Specify if the OracleAS TopLink Mapping Workbench displays a warning message if you attempt to create field references for an aggregate mapping before mapping the aggregate descriptor's attributes.
Mappings Without a Relation Table	Specify if the OracleAS TopLink Mapping Workbench displays a warning message if you attempt to create a table reference for a many-to-many mapping before defining the mapping's relation table.
Define Reference Descriptor First	Specify if the OracleAS TopLink Mapping Workbench displays a warning message if you attempt to create a table reference before defining the mapping's reference descriptor.
Invalid or Incomplete Data	Specify if the OracleAS TopLink Sessions Editor displays a warning message if your session configuration contains invalid or incomplete data.

Class Preferences

Use this procedure to specify how the OracleAS TopLink Mapping Workbench imports and edits classes. See "Working with Classes" on page 2-14 for more information.

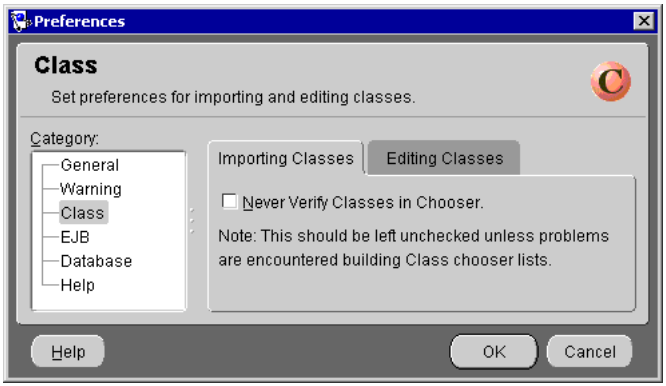
To Specify Class Import Options:

1. Click the **Preferences** button  on the toolbar. The Preferences dialog box appears.

   You can also display the Preferences dialog box by choosing **Tools** > **Preferences** from the menu.

2. Select **Class** in the **Category** pane.
3. Click the **Importing Classes** tab.

Figure 1–10 Class Preferences — Importing Classes

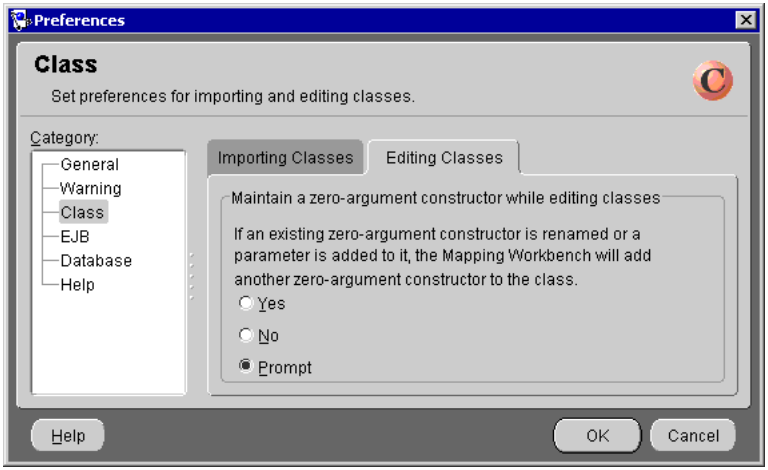


4. Select **Never Verify Classes in Chooser** if you do not want to verify the classes when performing an Add or Refresh.

> **Caution:** By default, the OracleAS TopLink Mapping Workbench always verifies the classes. Select this option only if you encounter errors when displaying classes in the Select Classes dialog box.

5. Click the **Editing Classes** tab.

Figure 1–11 Class Preferences — Editing Classes



6. Specify how the OracleAS TopLink Mapping Workbench maintains classes when renaming or editing a zero-argument constructor.
7. Click **OK**.

EJB Preferences

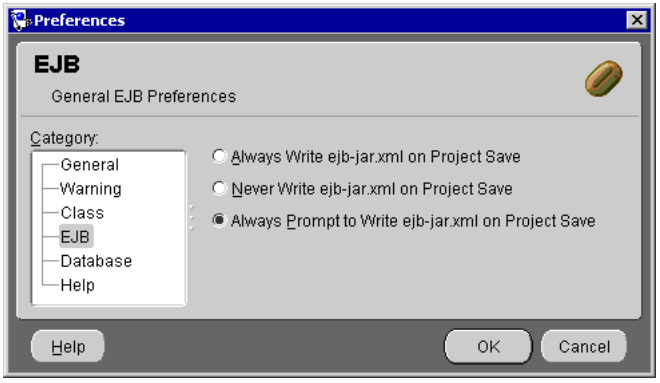
Use this procedure to specify how the OracleAS TopLink Mapping Workbench updates the `ejb-jar.xml` file when saving projects.

To Specify EJB Preferences:

1. Click the **Preferences** button on the toolbar. The Preferences dialog box appears.

   You can also display the Preferences dialog box by choosing **Tools** > **Preferences** from the menu.
2. Select **EJB** in the **Category** pane.

Figure 1–12 EJB Preferences



3. Specify whether the OracleAS TopLink Mapping Workbench prompts before updating the `ejb-jar.xml` file when you save the project.
4. Click **OK**.

Database Preferences

Use the **Database** preferences to specify custom database divers and connection URLs for the OracleAS TopLink Mapping Workbench. These drivers and URLs can then be used when defining database logins.

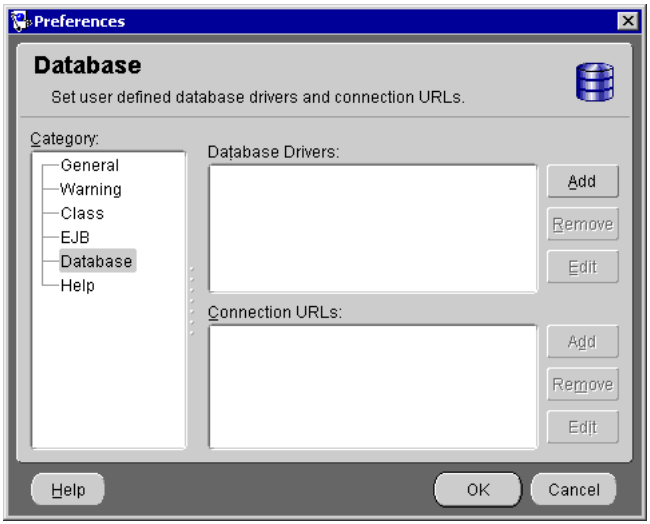
To Specify Database Preferences:

1. Click the **Preferences** button on the toolbar. The Preferences dialog appears.

   You can also display the Preferences dialog box by choosing **Tools** > **Preferences** from the menu.

2. Select **Database** in the **Category** pane.

Figure 1–13 Database Preferences



3. Use this table to enter data in each field and click **OK**.

Field	Description
Database Drivers	Click **Add** to add a custom database driver.
Connection URLs	Select a Database Driver, then click **Add** to add a custom database connection URL.

To edit or remove an existing driver or URL, select the driver or URL and click **Remove** or **Edit**.

Help Preferences

Use the **Help** preferences to select a Web browser to use the online help and Web-based support.

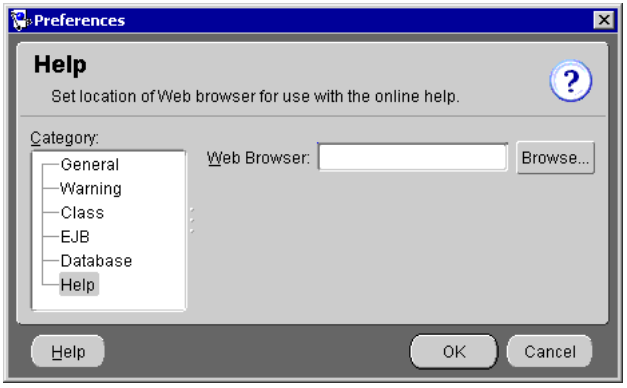
To Specify the Help Preferences:

1. Click the **Preferences** button on the toolbar. The Preferences dialog box appears.

   You can also display the Preferences dialog box by choosing **Tools** > **Preferences** from the menu.

2. Select **Help** in the **Category** pane.

Figure 1–14 Help Preferences



3. Use this table to enter data in each field and click **OK**.

Field	Description
Web Browser	Click **Browse** and select the location of your default Web browser. You must specify a Web browser to use the online help and Web-based support.

Working with the OracleAS TopLink Mapping Workbench in a Team Environment

When using an OracleAS TopLink Mapping Workbench project in a team environment, you must synchronize your changes with other developers. See "Merging Files" on page 1-20 for more information.

You can use the OracleAS TopLink Mapping Workbench with a source control system (see "Using a Source Control Management System" on page 1-19) to facilitate enterprise-level team development. If you have a small development team, you can manage the changes from within XML files (see "Sharing Project Objects" on page 1-23).

Using a Source Control Management System

If you use an enterprise, file-based, source control management system to manage your Java source files, you can use the same system with your OracleAS TopLink Mapping Workbench project files. These project files are maintained by the OracleAS TopLink Mapping Workbench and written in XML file format.

The source control system's *check in* and *out* mechanism defines how to manage the source (the XML source and OracleAS TopLink Mapping Workbench project file) in a multi-user environment.

To Check Out and In OracleAS TopLink Mapping Workbench Project Files:

Although your actual development process will vary, depending on your SCM tool, a typical process involves:

1. Determine (based on your SCM system) which files to retrieve from the source management system.
2. Edit the project using the OracleAS TopLink Mapping Workbench.
3. Save the edited project. If the OracleAS TopLink Mapping Workbench displays the Read Only Files dialog (see Figure 1–15 on page 1-24), make a note of these files, they must be unlocked and possibly merged.
4. Merge the required project files. See "Merging Files" on page 1-20 for details.
5. Check in the modified files, then retrieve from the repository any files that have been added or modified for this OracleAS TopLink Mapping Workbench project.

Merging Files

The most difficult aspect of team development is merging changes from two (or more) members that have simultaneously edited the same file. If one developer checks in his or her changes, a *merge* condition exists. Use a file comparison tool to determine the merged aspects of the project. The files to edit will vary, depending on the type of merge:

- Merging Project Files
- Merging Table, Descriptor, and Class Files

Example 1–1 and Example 1–2 demonstrate a "merge out" merging technique.

Merging Project Files

Project files contain references to the objects in the project. Generally, your project `<projectName>.mwp` contains:

- Database information – `<database>`
  - Database tables – `<tables>`
- Descriptors – `<descriptors>`
- Repository – `<repository>`
  - Classes – `<classpath-entries>`

Changes in these parts of the `.mwp` file are normally caused by adding, deleting, or renaming project elements.

To Merge Project Files:

Generally, you will need to merge a project file if another developer has added or removed a descriptor, table, or class, and checked in the project while you were adding or removing descriptors, tables, or classes from the same project. Use this procedure to merge the project's `.mwp` file:

1. Perform a file comparison between the `<projectName>.mwp` file in the repository and the your local copy. The file comparison shows the addition or removal of an element inside the owner (i.e., `<database>`, `<descriptors>`, or `<repository>`).
2. Insert or delete the XML into your local `<projectName>.mwp` file (inside the *corresponding owner element*). This brings your local code up to date to the current code in the code repository.

3. Retrieve any updated files, as indicated by your source control system. Your local source now matches the repository.

Example 1–1 Merging Projects

Another developer has added and checked in a new **Employee** class descriptor to the `com.demo` package while you were working with the same OracleAS TopLink Mapping Workbench project. To merge your work with the newly changed project:

1. Perform a file comparison on the `<projectName>.mwp` to determine the differences between your local file and the file in the repository. Your SCM system may show the file in *merge* status.

   The file comparison shows the addition of the `<package-descriptor>` tag and a `<name>` element inside that tag:

   ```
   <package-descriptor>
       <name>com.demo.Employee.ClassDescriptor</name>
   </package-descriptor>
   ```

2. Insert this XML into your `<projectName>.mwp` file (inside the `<descriptors>` element) to bring it up to date to the current files the source repository.
3. Retrieve any new or updated files from your source control system. This includes the newly added **Employee** class descriptor.
4. Check in files that you have modified.

Merging Table, Descriptor, and Class Files

Developers who concurrently modify the same existing table, descriptor, or class file will create a merge condition for the following files:

- Table – `<tableName>.xml` (one for each table)
- Descriptor – `<descriptorName.type>.xml` (one for each descriptor)
- Class – `<className>.xml` (one for each class)

The OracleAS TopLink Mapping Workbench changes these files when saving a project if you have changed any of the contents within them (such as adding a mapping to a descriptor, adding an attribute to a class, or a changing a field reference in a table).

To Merge a Table, Descriptor, or Class File:

If another developer has changed an attribute in a table, descriptor, or class, while you were changing a different mapping on that same descriptor, you will need to merge your project.

1. Perform a file comparison on the specific `.xml` file(s) in merge status (i.e., table, descriptor, or class). The file comparison shows the addition or removal of an XML element.
2. Insert or remove the XML into your local `.xml` file to bring it up to date to the current files the source repository.

Example 1–2 Merging Files

Another developer has added and checked in the **firstName** mapping to the **Employee** class descriptor while you were changing a different mapping on that same descriptor. To merge your work with the newly changed project:

1. Perform a file comparison on the `com.demo.Employee.ClassDescriptor.xml` file located in *<projectRoot>*`/Descriptor/` that is in merge status.

   The file comparison shows the addition of the `<mapping>` tag and the elements inside that tag:

```
<mapping>
    <uses-method-accessing>false</uses-method-accessing>
    <inherited>false</inherited>
    <read-only>false</read-only>
    <instance-variable-name>firstName</instance-variable-name>
    <default-field-names>
        <default-field-name>direct field=</default-field-name>
    </default-field-names>
    <field-handle>
        <field-handle>
            <table>EMPLOYEE</table>
            <field-name>F_NAME</field-name>
        </field-handle>
    </field-handle>
<mapping-class>MWDirectToFieldMapping </mapping-class>
</mapping>
```

2. Insert this XML block into your local `com.demo.Employee.ClassDescriptor.xml` file (inside the existing `<mappings>` element) to bring it up to date to the current files in the source repository.
3. Retrieve any new files noted as missing by your source control system. This includes any tables or descriptors that may be referenced by the new mapping.
4. Check in files that you have modified.

Sharing Project Objects

You can also share project objects by copying the table or descriptor file(s) into the appropriate directories in the target project.

After copying the file(s), insert a reference to the table, descriptor, or class in the appropriate place in the `<projectName>.mwp` file. All references contained within the project file must refer to an existing object in the project.

Managing the ejb-jar.xml File

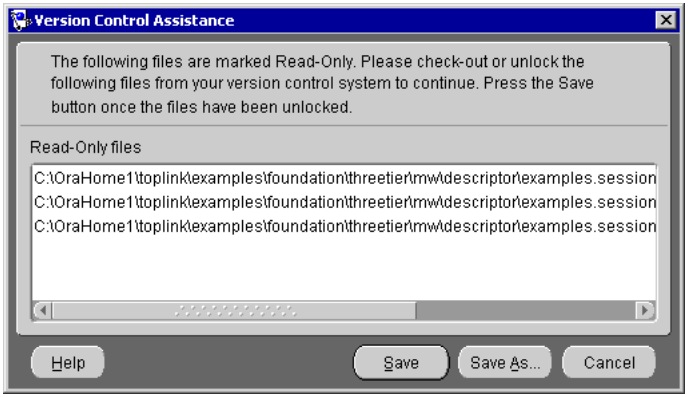
When working in a team environment, manage the `ejb-jar.xml` file similarly to the `.xml` project files. The OracleAS TopLink Mapping Workbench edits and updates the `ejb-jar.xml` file, if necessary, when working with an EJB project.

If you use a version control system, perform the same check in and check out procedures. For merge conditions, use a file comparison tool to determine which elements have been added or removed. Modify the file as necessary and check in the file to version control your work.

Working with Locked Files

When working in a team environment, your source control system may lock files when you retrieve them from the repository. If the OracleAS TopLink Mapping Workbench attempts to save a locked file, the Version Control Assistance dialog box appears, showing the locked files.

Figure 1–15 Version Control Assistance



Select one of the following methods to save your project:

- Use your source control system to unlock the files, then click **Save**.
- Click **Save As** to save the project to a new location.

See "Saving Projects" on page 2-4 for more information.

2

Understanding Projects

The Oracle Application Server TopLink Mapping Workbench project (`.mwp` file) stores the information about how classes map to database tables. These are language-independent XML files, different from the deployment XML files generated by the OracleAS TopLink Mapping Workbench, that are read in by the application using the `XMLProjectReader` class.

You can edit each component of the project, including:

- Project settings, such as the project class path and sequence information
- Database information, such as driver, URL, and login information
- Table schema information for the database
- Packages and classes associated with the project
- Descriptor information for each class

Working with Projects

The OracleAS TopLink Mapping Workbench displays projects and their contents in the **Navigator** pane. When you select a project, its attributes display in the **Editor** pane. The OracleAS TopLink Mapping Workbench can log runtime XML calls (in the `mw_xml.log` file) to help troubleshoot projects (see "General Preferences" on page 1-11 for more information).

Creating new Projects

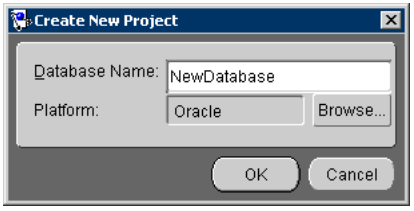
Use this procedure to create a new OracleAS TopLink Mapping Workbench project.

To Create a New Project:

1. Click the **Create New Project** button on the toolbar. The Create New Project dialog box appears.

   You can also create a new project by choosing **File** > **New Project**.

Figure 2–1 Create New Project



2. Enter the database name and platform for the new project and click **OK**. The Save As dialog box appears.

   To select a different database, click **Browse**. See "Working with Databases" on page 3-2 for more information.

3. Select a location in which to save the project and click **Save**.

> **Note:** Always use a new folder to save a project. After creating the `.mwp` project, do not rename the file. See "Saving Projects" on page 2-4 to save your project with a different name.

The OracleAS TopLink Mapping Workbench window appears, showing the project name in the **Navigator** pane. Continue with "Working with Project Properties" on page 2-6.

Opening Existing Projects

Use this procedure to open an existing project.

> **Caution:** To upgrade from a previous version of OracleAS TopLink, you **must** follow specific upgrade procedures and use the Package Rename tool. Refer to the *Oracle Application Server TopLink Release Notes* and *Oracle Application Server TopLink Getting Started Guide* for more information.

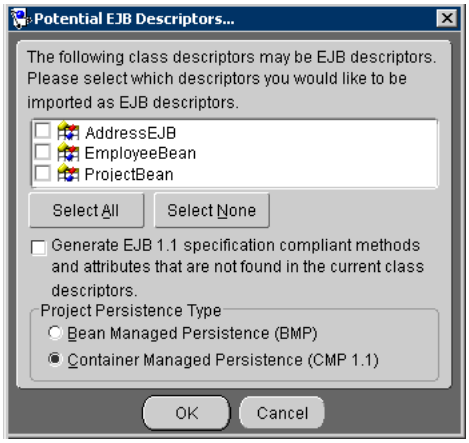
To Open an Existing Project:

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

   > **Note:** The **File** > **Reopen** menu option contains a list of recently opened projects. You can select one of these projects to open.

2. Select the OracleAS TopLink Mapping Workbench project file (`.mwp`) to open and click **Open**. The OracleAS TopLink Mapping Workbench displays the project information. If you open an OracleAS TopLink Mapping Workbench version 3.x project that contains EJBs, the Potential EJB Descriptors dialog box appears.

Figure 2–2 Potential EJB Descriptors



3. Select which of the descriptors should be imported as EJB descriptors, the project persistence type, and click **OK**.

   You can also specify whether the OracleAS TopLink Mapping Workbench generates methods and attributes that comply with the EJB specification if they are not found within the current class descriptor(s).

Saving Projects

The OracleAS TopLink Mapping Workbench does not automatically save your project. Be sure to save your project often to avoid losing data.

To Save Your Project(s):

Click the **Save Selected Project** button or **Save All Projects** button to save your project(s). You can also save a project by choosing **File** > **Save Project** or **File** > **Save All** from the menu.

To Save Your Project with a Different Name or Location:

1. Choose **File** > **Save As**. The Save As dialog box appears.
2. Browse to the directory in which to save the project. In the **File Name** field, type the name of the project and click **Save**.

> **Caution:** Do not rename the `.mwp` file outside of the OracleAS TopLink Mapping Workbench; instead use the **Save As** option.

Refreshing the Navigator Pane

If the OracleAS TopLink Mapping Workbench interface becomes corrupt, use the **Refresh Tree** option to refresh the **Navigator** pane.

To Refresh the Navigator:

Choose **File** > **Refresh** from the menu, or press **Ctrl+T**.

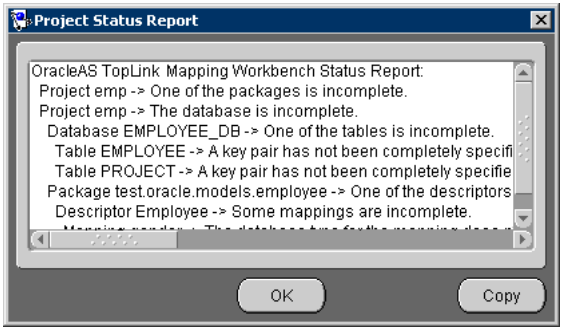
Generating the Project Status Report

Use the Project Status Report to display a list of all warnings and errors in the OracleAS TopLink Mapping Workbench project. This report will quickly identify any incomplete mappings and indicate the project's current status.

To Generate the Status Report:

Choose **Tools** > **Generate Project Status Report** from the menu. The Project Status Report dialog box appears, displaying the status of each OracleAS TopLink Mapping Workbench project.

Figure 2–3 Project Status Report



See "Error Messages" on page C-2 for information on each reported error.

To copy the report to another application (such as a text editor or e-mail message), click **Copy**.

Working with Project Properties

Each project in the **Navigator** pane contains different editable parameters. To edit the project's properties, select the project object in the **Navigator** pane. The following tabs appear in the **Editor** pane.

- **General** project properties (persistence type and class path)
- **Sequencing** table defaults
- **Default** project properties (identity map, existence checking, and field access method)
- **Options** (when generating Java source and creating Java information)

Working with General Project Properties

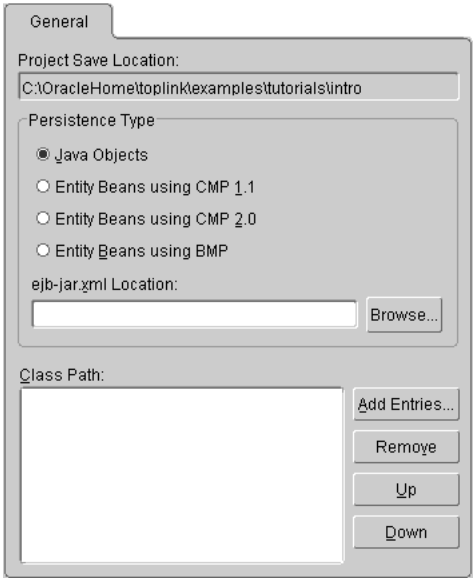
Use the project's **General** tab to specify the default persistence type and class path information. Each OracleAS TopLink project uses a class path — a set of directories, `.jar` files, and `.zip` files — when importing Java classes and defining object types.

To create a descriptor for a persistent class, the OracleAS TopLink Mapping Workbench reads a compiled Java `.class` file to read its attributes and relationships.

To Specify the General Properties:

1. Select the project object in the **Navigator** pane.
2. Select the **General** tab in the **Editor** pane. The **General** tab appears.

Figure 2–4 General Tab



3. Use this table to enter data in each field:

Field	Description
Project Save Location	Location of the project's `.mwp` file and associated folders. This field is for display only. All relative locations are based on the **Project Save Location**.
Persistence Type	Specify the project's persistence type. For EJB projects, specify the location of the `ejb-jar.xml` file.
ejb-jar.xml Location	Location of the `ejb-jar.xml` file for this project. See "Mapping EJB 2.0 Entities" on page 2-8 and "Working with the ejb-jar.xml File" on page 2-18 for more information. **Note**: This field applies for EJB projects only.

Field	Description
Class Path	Location of the classes and packages for this project. See "Working with Classes" on page 2-14 for information. ■ To add a new class path entry, click **Add Entries** and select the directory, `.jar` file, or `.zip` file to add. ■ To remove a class path entry, select the entry and click **Remove**. ■ To change the order of the entries, select the entry and click **Up** or **Down**. To create a relative class path, select an entry and edit the path, as necessary. The path will be relative to the **Project Save Location**.

Mapping EJB 2.0 Entities

You can create an OracleAS TopLink Mapping Workbench project based on information in the `ejb-jar.xml` file. Use this file to map the EJB 2.0 Container Managed Persistence (CMP) entity beans' virtual fields (called *Container Managed Fields*, defined by `<cmp-field>` tag) or relationships (called *Container Managed Relationship*, defined by `<cmr-field>` tag) to database tables. You must specify an `.xml` file or a `.jar` file that contains an `ejb-jar.xml` file.

The OracleAS TopLink Mapping Workbench defines all descriptors for entity classes (as defined in the `ejb-jar.xml` file) as EJB descriptors  . The OracleAS TopLink Mapping Workbench does not create (or remove) descriptors for the interfaces and primary key class for the entity when refreshing from the `ejb-jar.xml`.

> **Note:** The OracleAS TopLink Mapping Workbench creates class descriptors for entity classes not defined in the `ejb-jar.xml` file. You must manually change the descriptor type (see "Specifying Descriptor Types" on page 4-3).

To update your project from the `.xml` file, right-click an EJB descriptor and select **Update Descriptors from ebj-jar.xml**. You can also update the project by choosing **Selected** > **Update Descriptors from ebj-jar.xml** from the menu. See "Working with the ejb-jar.xml File" on page 2-18 for more information.

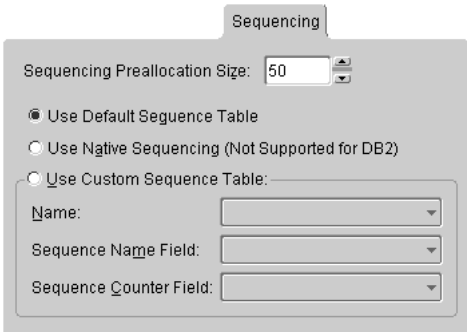
Working with Sequencing Properties

Sequence numbers are artificial keys that uniquely identify the records in a table. Use the project's **Sequencing** tab to specify default sequencing properties for all descriptors in the project. See "Working with Sequencing" on page 4-36 for more information.

To Specify Default Sequencing Properties:

1. Select the project object in the **Navigator** pane.
2. Click the **Sequencing** tab in the **Editor** pane. The **Sequencing** tab displays.

Figure 2–5 Sequencing Tab



3. Use this table to enter data in each field:

Field	Description
Sequencing Preallocation Size	Default preallocation size. Default is **50**.
Sequencing Type	Specify whether the project uses: ■ OracleAS TopLink **Default** sequencing ■ The database's **Native** sequencing ■ **Custom** sequencing table
Custom Sequence Table Information	Use these fields to specify a sequence name, the name field, and counter field. These fields apply only you select **Use Custom Sequencing Table**.

Working with Default Properties

Use the project's **Default** tab to specify the default identity map, existence checking, and field access method.

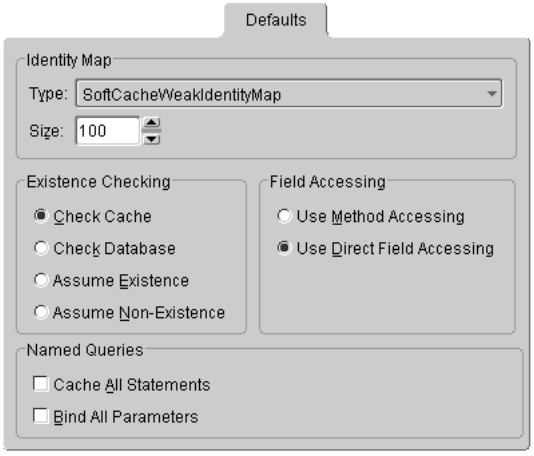
Use the project's **Default** tab to specify the default:

- Identity map and existence checking policy for descriptors (if they do not have a specific identity policy)
- Field accessing applied to newly created descriptors

To Specify Default Project Properties:

1. Select the project object in the **Navigator** pane.
2. Select the **Defaults** tab in the **Editor** pane. The **Defaults** tab displays.

Figure 2–6 Defaults Tab



3. Use this table to enter data in each field:

Field	Description
Identity Map	Use the **Type** drop-down list to select the default identity map and its **Size** for descriptors in this project (see "Working with Identity Maps" on page 4-58).
Existence Checking	Specify the type of existence checking to use.

Field	Description
Field Accessing	Specify whether the descriptors use **Method** or **Direct** field accessing (see "Specifying Direct Access and Method Access" on page 4-70).
Named Queries	Specify if OracleAS TopLink **Caches All Statements** or **Binds All Parameters** for named queries. See "Named Queries" on page 4-15 for more information, including how to override this setting for specific queries.

Renaming Packages

To rename your packages, you must edit each of the project's associated `.xml` files in the following subdirectories:

- **Class**
- **ClassRepository**
- **Descriptor**
- **Package**

You must also edit the package and class names in the `.mwp` file. After changing the package names in all files, open the project in the OracleAS TopLink Mapping Workbench. OracleAS TopLink now uses the new package name.

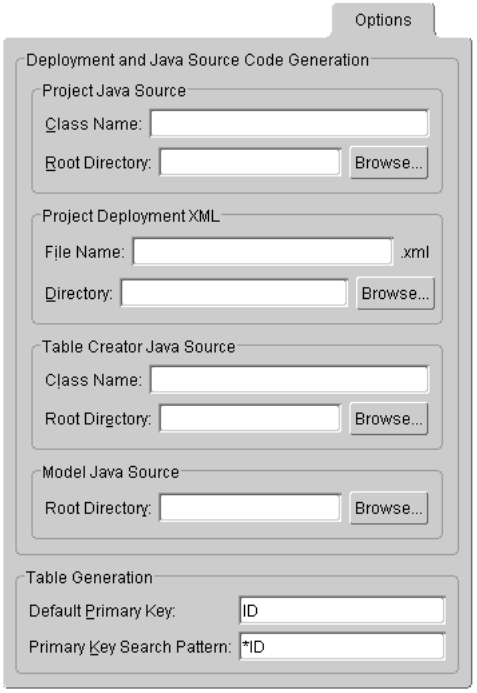
Working with Project Options

Use the project's **Options** tab to specify the default file names, class names, and directories, when exporting or generating Java source code and deployment XML. In addition, you can specify the primary key name and primary key search pattern (database schema) to use when generating tables. The resulting tables and columns will conform to the naming restrictions of the project's target database.

To Specify Project Options:

1. Select the project object in the **Navigator** pane.
2. Click the **Options** tab in the **Editor** pane. The **Options** tab displays.

Figure 2–7 Options Tab



3. Use this table to enter data in each field:

Field	Description
Deployment and Java Source Code Generation	Specify the defaults to use when exporting project information. Directories are relative to the project location and will contain folders for each generated package.
Project Java Source	Java class name and project root directory when generating Java source. See "Exporting Project to Java Source" on page 2-16 for more information.
Project Deployment XML	Filename (`*.xml`) and directory when generating Java source. See "Exporting Deployment XML" on page 2-17 for more information.
Table Creator Java Source	Java class name and project root directory when generating table source. See "Exporting Table Creator Files" on page 2-17 for more information.

Field	Description
Model Java Source	Project root directory when generating Java model source. See "Generating Java Code for Descriptors" on page 4-5 for more information.
Table Generation	
Default Primary Key Name	Default name to use when generating primary keys.
Primary Key Search Pattern	Default search pattern to use when generating primary keys.

Setting Default Advanced Properties

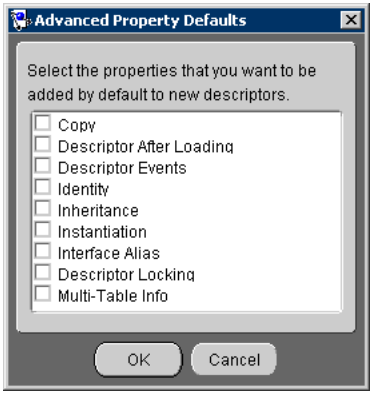
In addition to a descriptor's standard property tabs, you can specify advanced properties for each descriptor (see "Working with Advanced Properties" on page 4-26 for more information). You can also specify which of these advanced properties appear, by default, for new descriptors.

To Specify the Default Advanced Properties for Descriptors:

1. Right-click the project object in the **Navigator** pane and choose **Set Advanced Property Defaults** from the pop-up menu. The Advanced Property Defaults dialog box appears.

   You can also set the default advanced properties by choosing the project object and choosing **Selected** > **Set Advanced Property Defaults** from the menu.

Figure 2–8 Advanced Property Default



2. Select each advanced property to display, by default, when creating and editing descriptors.
3. Click **OK**.

Working with Classes

This section describes how to create descriptors from Java classes and packages and includes information on the following:

- Creating Classes
- Updating Classes

Creating Classes

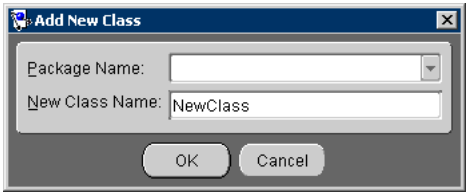
Use this procedure to create new classes and packages from within the OracleAS TopLink Mapping Workbench.

To Create a New Class:

1. Right-click the project in the **Navigator** pane and choose **Create New Class** from the pop-up menu.

   You can also choose **Selected** > **Create New Class** from the menu.

Figure 2–9 Add New Class



2. Use the **Package Name** drop-down list to choose a package or enter a new package name.
3. In the **New Class Name** field enter a class name and click **OK**. The OracleAS TopLink Mapping Workbench adds the new class to your project in the **Navigator** pane.

Note: The **Class Name** must be unique within the package.

Updating Classes

Use this procedure to update or refresh the classes in the OracleAS TopLink Mapping Workbench project.

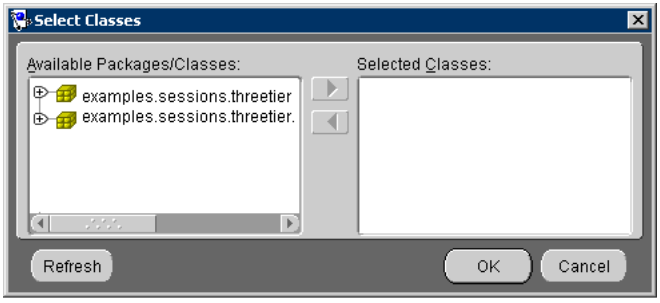
> **Note:** If the class exists on both the system class path and the project class path, the OracleAS TopLink Mapping Workbench will update the class from the system class path. To update or refresh from the project class path, remove the class from the system class path and restart the OracleAS TopLink Mapping Workbench.

To Update Classes:

1. Define the available class(es) and package(s) for the project on the **General** tab. See "Working with General Project Properties" on page 2-6 for information on classes and packages.
2. Click on **Add/Refresh Class** button  . The Select Classes dialog box appears.

   You can also update the classes by choosing **Selected** > **Add/Refresh Classes** from the menu.

Figure 2–10 Select Classes



3. Select the package(s) and/or class(es) to add to the project and click **OK**. The OracleAS TopLink Mapping Workbench adds the new classes to your project in the **Navigator** pane.

> **Note:** The OracleAS TopLink Mapping Workbench creates *class* descriptors for each package/class. See "Specifying Descriptor Types" on page 4-3 for to change the descriptor type.

To Remove a Class from a Project:

Select on the descriptor and click the **Remove Class** button  , or choose **Selected** > **Remove Class** from the menu.

Exporting Project Information

To use your project with the Oracle Application Server TopLink Foundation Library, you must either generate deployment XML or export the project to Java source code. The OracleAS TopLink Mapping Workbench can generate and export the following project information:

- Exporting Project to Java Source
- Exporting Deployment XML
- Exporting Table Creator Files
- Exporting Java Model Source

Note: When exporting Java source and deployment XML, the OracleAS TopLink Mapping Workbench writes the password using JCE encryption (when using JDK 1.4). Refer to the *Oracle Application Server TopLink Getting Started Guide* for information on using password encryption with pre-JDK 1.3.

Use the OracleAS TopLink Sessions Editor to specify custom password encryption (see "Setting Login Properties" on page 8-10).

Exporting Project to Java Source

Use this procedure to convert the project to Java code. Generally, this generated code executes faster and deploys easier than XML files.

To Export the Project to Java Source Code:

Right-click the project in the **Navigator** pane and choose **Export** > **Project to Java Source** from the pop-up menu.

You can also click the **Export to Java Source** button  , or choose **File** > **Export** > **Export to Java Source** or **Selected** > **Export** > **Export to Java Source** from the menu.

If you have not defined deployment and source code generation defaults (see "Working with Project Options" on page 2-11) the OracleAS TopLink Mapping Workbench prompts for a project class name and source directory.

Note: If your OracleAS TopLink Mapping Workbench project uses UTF-8 character set, you must use a compatible JDK when compiling the exported Java source.

Exporting Deployment XML

Use this procedure to generate XML files from your project that can be read by the OracleAS TopLink Foundation Library. Using this option reduces development time by eliminating the need to regenerate and recompile Java code each time the project changes.

To Export Deployment XML:

Right-click the project in the **Navigator** pane and choose **Export** > **Deployment XML** from the pop-up menu.

You can also click the **Export to Deployment XML** button  or choose **File** > **Export** > **Generate Deployment XML** or **Selected** > **Export** > **Generate Deployment XML** from the menu.

If you have not defined deployment and source code generation defaults (see "Working with Project Options" on page 2-11) the OracleAS TopLink Mapping Workbench prompts for a filename and directory.

Exporting Table Creator Files

Use this procedure to create Java source code to generate database tables.

To Export Table Creator Files:

Right-click the project in the **Navigator** pane and choose **Export** > **Export Table Creator Java Source** from the pop-up menu.

You can also choose **File** > **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 "Working with Project Options" on page 2-11) the OracleAS TopLink Mapping Workbench prompts for a class name and root directory.

Exporting Java Model Source

Use this procedure to generate the project model's Java source.

To Export Java Model Source:

Right-click the project, package, or specific descriptor in the **Navigator** pane and choose **Export** > **Export Java Model Source** from the pop-up menu. The OracleAS TopLink Mapping Workbench creates a `.java` file for each selected descriptor.

You can also choose **File** > **Export** > **Export Java Model Source** or **Selected** > **Export** > **Java Model Source** from the menu.

If you have not defined deployment and source code generation defaults (see "Working with Project Options" on page 2-11) the OracleAS TopLink Mapping Workbench prompts for a root directory.

Note: If your OracleAS TopLink Mapping Workbench project uses UTF-8 character set, you must use a compatible JDK when compiling the exported Java source.

Working with the ejb-jar.xml File

For OracleAS TopLink Mapping Workbench projects that use EJB 2.0 CMP persistence, use the `ejb-jar.xml` file to store persistence information for the application server. With the OracleAS TopLink Mapping Workbench, you can import information from an existing `ejb-jar.xml` file into your project, or you can create/update the `ejb-jar.xml` from your project.

Each OracleAS TopLink Mapping Workbench project uses a single `ejb-jar.xml` file. For each entity from the file, you should have an EJB descriptor in the project. All entities 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 the OracleAS TopLink Mapping Workbench, you can update your project to reflect the current file.

Table 2–1 describes how fields in the `ejb-jar.xml` file correspond to specific functions in the OracleAS TopLink Mapping Workbench:

Table 2–1 ejb-jar.xml Fields and the OracleAS TopLink Mapping Workbench

ejb-jar.xml	OracleAS TopLink Mapping Workbench
`primkey`	Bean attribute mapped to the primary key in the database table (see "Setting Descriptor Information" on page 4-5)
`ejb-name, prim-key-class, local, local-home, remote, home, and ejb-class`	EJB descriptor information on the **EJB Info** tab (see "Displaying EJB descriptor Information" on page 4-25)
`abstract-schema-name`	**Descriptor Alias** field on the **Name Queries** tab (see "Named Queries" on page 4-15)
`cmp-field`	Nonrelational attributes on the **Descriptor Info** tab (see "Setting Descriptor Information" on page 4-5)
`cmp-version`	**Persistence Type** field on the **General** tab (see "Working with General Project Properties" on page 2-6) The `persistence-type` is set to `container`.
`query`	Queries listed in **Queries** tab (see "Specifying Queries and Named Finders" on page 4-12) **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 "Working with Relationship Mappings" on page 6-2)

Writing to the ejb-jar.xml File

Use this procedure to update the `ejb-jar.xml` file, based on the current OracleAS TopLink Mapping Workbench information. Use the EJB preferences to specify whether the OracleAS TopLink Mapping Workbench automatically updates the `ejb-jar.xml` file when you save the project.

Note: You can also write the information to a `.jar` file. The OracleAS TopLink Mapping Workbench automatically places the `ejb-jar.xml` file in the proper location (`META-INF/ejb-jar.xml`).

To Write the ejb-jar.xml File:

Choose **Selected** > **Write Project to ejb-jar.xml** from the menu. You can also right-click the project in the **Navigator** pane and choose **Write Project to ejb-jar.xml** from the pop-up menu.

- If the project does not currently contain an `ejb-jar.xml`, 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 the OracleAS TopLink Mapping Workbench (for example, you changed the file outside the OracleAS TopLink Mapping Workbench), then the system prompts you to read the file before writing the changes.

Reading from the ejb-jar.xml File

Use this procedure to read the `ejb-jar.xml` information and update your OracleAS TopLink Mapping Workbench project.

Tip: To automatically create EJB descriptors in the OracleAS TopLink Mapping Workbench for all entities, read the `ejb-jar.xml` file *before* adding any classes in the OracleAS TopLink Mapping Workbench.

To Read the ejb-jar.xml File:

Choose **Selected** > **Update Project from ejb-jar.xml** from the menu. You can also right-click the project in the **Navigator** pane and choose **Update Project from ejb-jar.xml** from the pop-up menu.

3

Understanding Databases

When you create a descriptor for a class, the Oracle Application Server TopLink Mapping Workbench retrieves the table information from the database. This section includes information on:

- Working with Databases
- Working with Database Tables in the Navigator Pane
- Working with Database Tables in the Editor Pane
- Generating Data from Database Tables

Working with Databases

Each OracleAS TopLink Mapping Workbench project contains a database. You can create multiple logins for each database. This section describes how to specify database properties and how to log into the database.

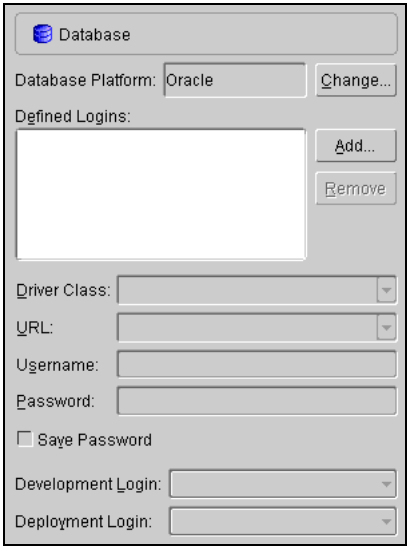
Database Properties

Use the **Database** property sheet to specify information about the database and login(s).

To Specify the Database Properties:

1. Choose the database object in the **Navigator** pane. The Database property sheet appears in the **Editor** pane.

Figure 3–1 Database Property Sheet



2. Use this table to enter data in each field:

Field	Description
Database Platform	Database platform for the project. Click **Change** to select a new database.

Field	Description
Defined Logins	Login used to access the database. Click **Add** to add a new login, or **Remove** to delete an existing login.
Login Fields:	To edit these fields, first select a **Defined Login**.
Driver Class **URL**	The OracleAS TopLink Mapping Workbench connects to databases through JDBC. Contact your database administrator for information on installing and configuring your driver.
Username	Name required to log into the database.
Password	Password required to log into the database. **Note:** When exporting deployment XML and Java source (see "Exporting Project Information" on page 2-16) the OracleAS TopLink Mapping Workbench writes out this password. Use the OracleAS TopLink Sessions Editor (see Chapter 8, "Understanding the OracleAS TopLink Sessions Editor") to specify password encryption options.
Save Password	Specify whether the OracleAS TopLink Mapping Workbench saves the **Password** for this **Defined Login**.
Development Login **Deployment Login**	The OracleAS TopLink Mapping Workbench supports multiple logins. Select a **Defined Login** to use for development and/or deployment.

3. After entering the information, continue with "Logging into the Database" on page 3-3.

Logging into the Database

You must log into the database before importing or exporting table information.

To Log into the Database:

Select the database object in the **Navigator** pane and click the **Login** button  on the toolbar. The OracleAS TopLink Mapping Workbench logs into the database. The database object in the **Navigator** pane changes to  .

You can also right-click on the database object and choose **Log In** from the pop-up menu or choose **Selected** > **Log In** from the menu.

Note: If you have not defined a login, the system displays a warning message. See "Database Properties" on page 3-2 for more information on creating a database login.

Working with Database Tables in the Navigator Pane

When you expand the database object in the **Navigator** pane, the OracleAS TopLink Mapping Workbench displays the database tables associated with the project. You can associate tables by importing them from the database or by creating them within the OracleAS TopLink Mapping Workbench.

Figure 3–2 Sample Database Tables

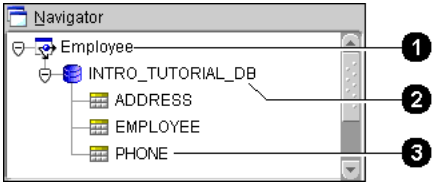


Figure 3–2 calls out the following database icons.

1. Project
2. Database
3. Database table

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

- **Fields** – Add or modify the table's fields, and specify the field's properties (see Figure 3–5).
- **References** – Specify references between tables (see Figure 3–8).

Creating New Tables

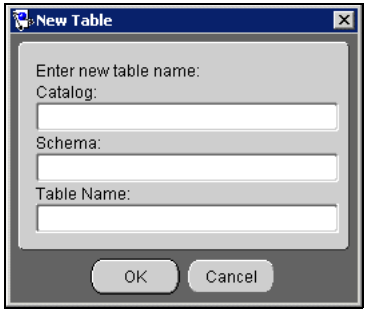
Use this procedure to create a new database table within the OracleAS TopLink Mapping Workbench.

To Create a New Table:

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

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

Figure 3–3 New Table



2. Use this table to enter data in each field.

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

3. Enter the necessary information and click **OK**. The OracleAS TopLink Mapping Workbench adds the database table to the project.

> **Note:** Refer to "Generating Tables on the Database" on page 3-17 to add the table information to the database.

Continue with "Working with Database Tables in the Editor Pane" on page 3-8 to use these tables in your project.

Importing Tables from Database

The OracleAS TopLink Mapping Workbench can automatically read the schema for a database and import the table data into the project. This section describes the driver requirements and how to import, remove, and rename tables.

JDBC Driver Requirements

To retrieve table information from the database, the database driver must support the following JDBC methods:

- `getTables( )`
- `getTableTypes( )`
- `getImportedKeys( )`
- `getCatalogs( )`
- `getPrimaryKeys( )`

To Import Tables from the Database:

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

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

Figure 3–4 Import tables from Database

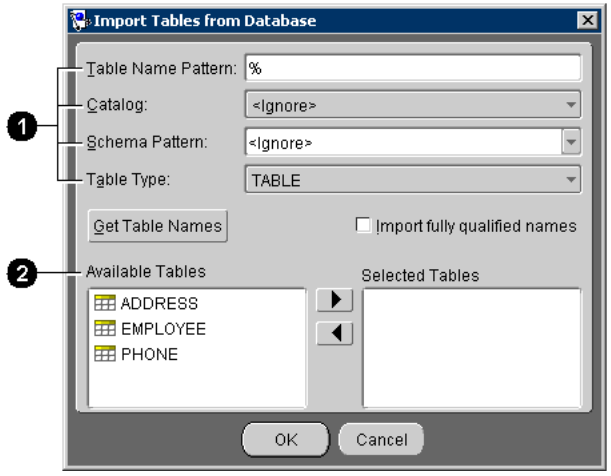


Figure 3–4 calls out the following user-interface components:

1. Filters

2. Database tables that match the filters

2. Use this table to enter data in each filter field on the dialog box:

Field	Description
Table Name Pattern	Name of database table(s) to import. Use **%** (percent character) as a wildcard. Tables that match the **Table Name Pattern** can be imported.
Catalog	Catalog of database table(s) to import.
Schema Pattern	Schema of database table(s) to import.
Table Type	Type of database table(s) to import.
Import Fully Qualified Names	Specify whether the tables' names are fully qualified against the schema and catalog.

3. Enter the filter information and click **Get Table Names**. OracleAS TopLink examines the database and displays the tables that match the filters in the **Available Tables** field.
4. Select the table(s) in the **Available Tables** area to import and click  . OracleAS TopLink adds the table to the **Selected Tables** field.
5. Select all the tables to import, then click **OK**. OracleAS TopLink imports the tables from the database into the OracleAS TopLink Mapping Workbench project.
6. Examine each table's properties to verify that the imported tables contain the correct information. See "Working with Database Tables in the Editor Pane" on page 3-8.

Removing Tables

Use this procedure to remove a database table from the project.

To Remove a Table:

1. Select on a database table in the **Navigator** pane and click the **Remove Table** button  on the toolbar. The OracleAS TopLink Mapping Workbench prompts for confirmation.

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

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

> **Note:** The table remains in the database.

Renaming Tables

Use this procedure to rename a database table in the OracleAS TopLink Mapping Workbench project.

To Rename the Table:

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

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

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

> **Note:** The original table name remains in the database.

Working with Database Tables in the Editor Pane

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

- **Fields** – Add or modify the table's fields, and specify the field's properties (see Figure 3–5).
- **References** – Specify references between tables (see Figure 3–8).

Working with Field Properties

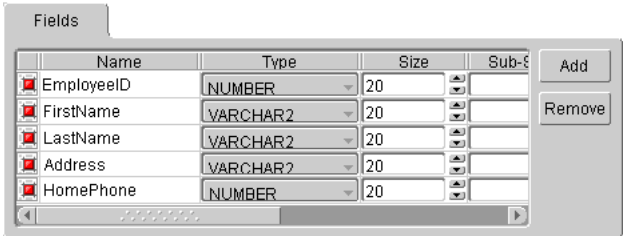
Use the database table's **Field** tab to specify properties for the database table's fields.

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

To Specify Table Field Properties:

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

Figure 3–5 Fields Properties



3. Use this table to enter information in each field. Use the scroll bar to display all fields on the tab.

Field	Description
Name	Name of the field.
Type	Use the drop-down list to select the field's type. **Note:** The valid values will vary, depending on the database.
Size	Size of the field.
Sub-Size	Sub-size of the field.
Allows Null	Specify if this field can be null.
Primary Key	Specify whether this field is a primary key for the table.
Identity	Indicates a Sybase, SQL Server or Informix identity field.
Unique	Specify whether the value must be unique within the table.

> **Note:** Use the scroll bar to display the additional fields.

4. Enter the necessary information for the existing fields, or click **Add Field** to add a new field.

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

Setting a Primary Key for Database Tables

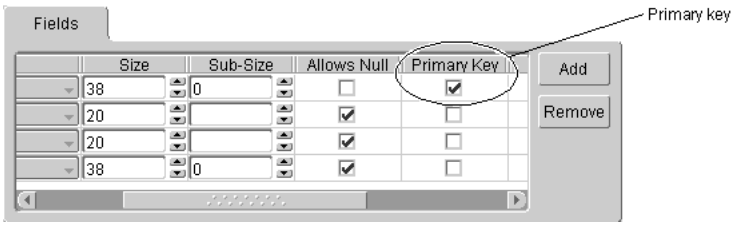
Use this procedure to set primary key(s) for a database table.

> **Note:** The OracleAS TopLink Mapping Workbench can automatically import primary key information if supported by the JDBC driver.

To Set a Primary Key:

1. Select a database table in the **Navigator** pane. Its property sheet appear in the **Editor** pane.
2. Click the **Fields** tab.

Figure 3–6 Setting Primary Key for a Database Table



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

Working with Reference Properties

References are table properties that contain the foreign key — they may or may not correspond to an actual constraint that exists on the database. The OracleAS TopLink Mapping Workbench uses these references when you define relationship mappings and multiple table associations.

When importing tables from the database (see "Importing Tables from Database" on page 3-5), the OracleAS TopLink Mapping Workbench can automatically create references (if the driver supports this), or you can define references from the workbench.

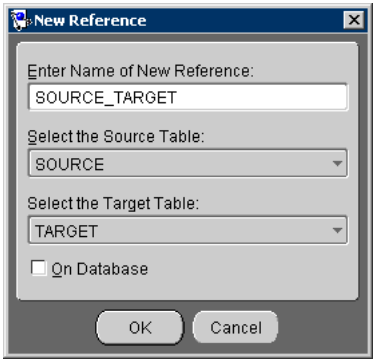
Creating Table References

Use this procedure to create a new table reference.

To Create a New Table Reference:

1. Select a database table in the **Navigator** pane. The table's properties display in the **Editor** pane.
2. Click the **Table Reference** tab (see Figure 3–8).
3. In the **Table References** area, click the **Add** button. The New Reference dialog box appears.

Figure 3–7 New Reference



4. Use this table to enter information in each field.

Field	Description
Name of New Reference	Name of the reference table. If you leave this field blank, the OracleAS TopLink Mapping Workbench automatically creates a name based on the format: `SOURCETABLE_TARGETTABLE`.
Select the Source Table	Name of the database table. This field is for display only.
Select the Target Table	Use the drop-down list to specify the target table for this reference.
On Database	Specify if you want to create the reference on the database when you create the table. Not all database drivers support this option.

Continue with Creating Field References.

Creating Field References

Use this procedure to create a new field reference.

To Specify Table Reference Properties:

1. Select a database table in the **Navigator** pane. The table's properties display in the **Editor** pane.
2. Click on **Table Reference** tab.

Figure 3–8 References Properties

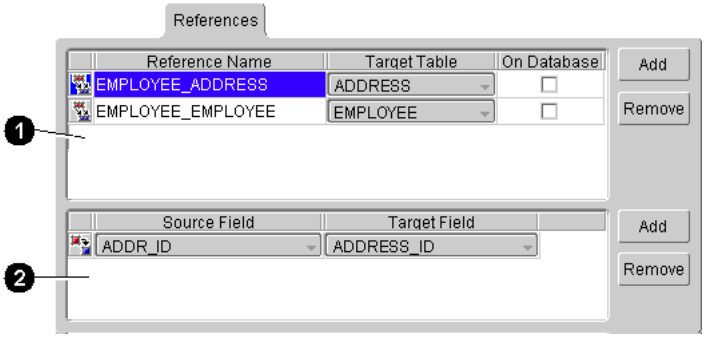


Figure 3–8 calls out the following user-interface components:

1. Table References area
2. Key Pairs area

3. In the **Table References** area, select a Table Reference (see "Creating Table References" on page 3-11).
4. In the **Key Pairs** area, click on **Add** button. The **Source** and **Target** fields appear on the tab.
5. Use the **Source Field** and **Target Field** drop-down lists to choose the key pair for this reference.

Generating Data from Database Tables

The OracleAS TopLink Mapping Workbench can automatically generate the following information from the database tables.

- Generating SQL Creation Scripts

- Generating Descriptors and Classes from Database Tables
- Generating EJB Entities from Database Tables

You can also generate database tables from descriptors in your project.

Generating SQL Creation Scripts

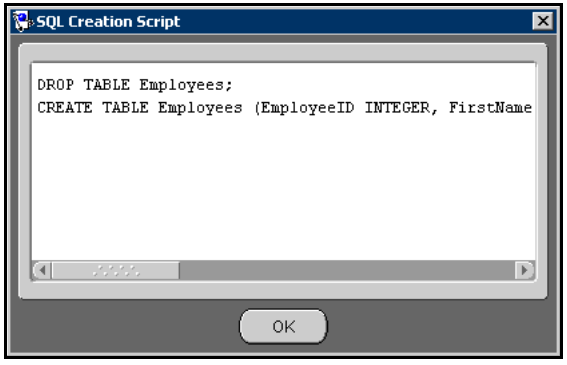
Use this procedure to automatically generate SQL scripts to create the tables in a project.

To Generate SQL Scripts from Database Tables:

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

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

Figure 3–9 SQL Creation Script



3. Copy the script and paste it into a file. You may need to edit the file to include additional SQL information that the OracleAS TopLink Mapping Workbench could not generate.

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

Generating Descriptors and Classes from Database Tables

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

For each table, the OracleAS TopLink Mapping Workbench:

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

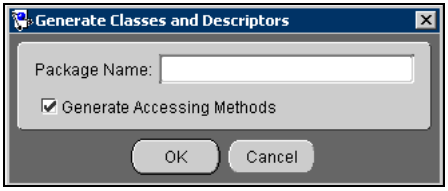
> **Note:** Class and attribute names are generated based on the table and column names. You can edit the class properties to change their names.

To Generate Descriptors and Classes from Database Tables:

1. Select the database table(s) in the **Navigator** pane.
2. Right-click the table(s) and choose **Generate Descriptors and Classes from** > **Selected Table** or **All Tables** from the pop-up menu. The Save Project dialog box appears.

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

Figure 3–10 Generate Classes and Descriptors

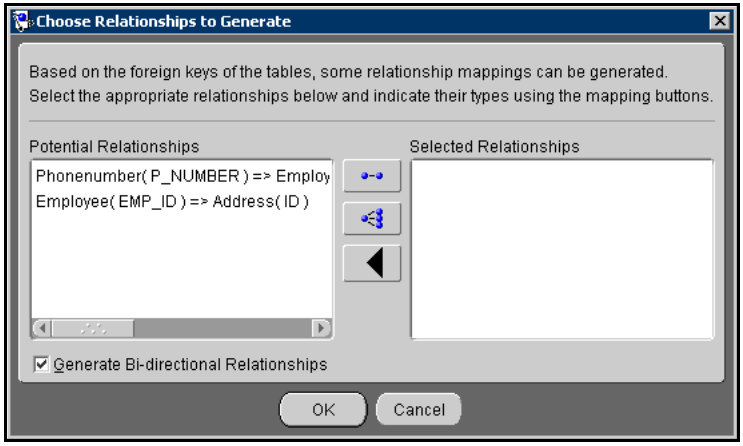


4. Use this table to enter data in each field:

Field	Description
Package Name	Name of package to generate
Generate Accessing Methods	Specify if the OracleAS TopLink Mapping Workbench generates accessing methods for each class and descriptor

5. Enter the information and click **OK**. If the table contains foreign key fields that may represent relationship mappings, then the Choose Relationships to Generate dialog box appears.

Figure 3–11 Choose Relationships to Generate



6. Select a **Potential Relationship** and click the **1:1 Mapping** or **1:M Mapping** button. See Chapter 6, "Understanding Relationship Mappings" for more information on mappings.

   You can also specify whether the relationships are bidirectional.

7. Click **Create** to automatically create the relationships (or click **Skip** to generate the descriptors *without* creating these relationships.).

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

Generating EJB Entities from Database Tables

Use this procedure to automatically generate EJB classes and descriptors for each database table. Generating EJB entities allows you to create:

- One EJB descriptor that implements the `<javax.ejb.EntityBean>` interface and four EJB 1.1 classes for each table
- Bean relation attributes (CMP or BMP)
- Java source fore each class
- EJB-compliant method stubs

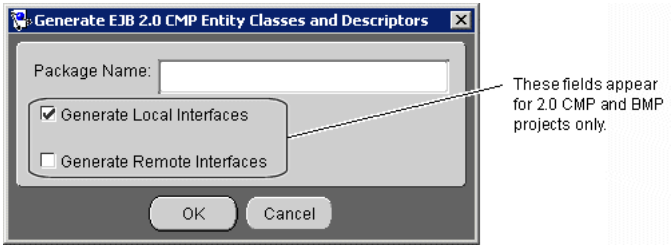
> **Note:** This option is available only for projects with CMP or BMP persistence.

To Generate EJB Entities:

1. Select the database table(s) in the **Navigator** pane.
2. Right-click the table(s) and choose **Generate EJB Entities and Descriptors from** > **Selected Table** or **All Tables** from the pop-up menu. The Save Project dialog box appears.

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

Figure 3–12 Generate EJB Entity Classes and Descriptors



4. Enter a package name, select any persistence type options, and click on **OK**.
5. If the table contains foreign key fields that may represent relationship mappings, then the Choose Relationships To Generate dialog box appears (see

Figure 3–11). Select a potential relationship and click the **1:1 Mapping** or **1:M Mapping** button.

You can also specify whether the relationships are bidirectional.

6. Repeat step 5 for all appropriate sets of tables.
7. Click **Create** to generate the relationship mappings (or click **Skip** to generate the EJB descriptors *without* creating these relationships.).

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

Generating Tables on the Database

Use this procedure to create a table in the database, based on the information in the OracleAS TopLink Mapping Workbench.

To Create a Table on the Database:

1. Select the database table(s) in the **Navigator** pane.
2. Right-click the table(s) and choose **Create on Database** > **Selected Table** or **All Tables** from the pop-up menu. The Save Project dialog box appears.

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

> **Note:** You must log into the database before creating tables. See "Logging into the Database" on page 3-3 for more information.

The OracleAS TopLink Mapping Workbench creates the tables on the database.

The newly created descriptor(s) appear in the **Navigator** pane of the OracleAS TopLink Mapping Workbench. Use the **EJB Info** tab (see Figure 4–16) to modify the EJB information.

4

Understanding Descriptors

Oracle Application Server TopLink uses *descriptors* to store the information that describes how an instance of a particular class can be represented in a relational database. Most descriptor information can be defined by the OracleAS TopLink Mapping Workbench and read from a project file to be registered with an OracleAS TopLink Mapping Workbench session.

For complete information on the OracleAS TopLink API, refer to the *Oracle Application Server TopLink API Reference*.

Note: In this document, *descriptors* refers to OracleAS TopLink descriptors; *deployment descriptors* refers to EJB deployment descriptors.

Working with Descriptors

A descriptor stores all the information describing how an instance of a particular class can be represented in a relational database. The OracleAS TopLink Mapping Workbench reads a project `.mwp` file to load all a project's information (including descriptor information).

You may need to amend a descriptor (for example, to specify a property not supported by the OracleAS TopLink Mapping Workbench) after reading a project file (see "Amending Descriptors After Loading" on page 4-27). However, do not modify any descriptors after registering them with the session.

OracleAS TopLink descriptors contain the following information:

- The persistent Java class it describes and the corresponding database table(s) for storing instances of that class
- A collection of mappings, which describe how the attributes and relationships for that class are stored in the database
- The primary key information of the table(s)
- 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 identity maps, optimistic locking, the Event Manager, and the Query Manager
- Caching refresh options

Understanding Persistent Classes

Any class that registers a descriptor with an OracleAS TopLink Mapping Workbench database session is called a *persistent class*. OracleAS TopLink *does not* require that persistent classes provide public accessor methods for any private or protected attributes stored in the database.

See Appendix A, "Object Model Requirements" for more information on persistent classes object model requirements.

Specifying Descriptor Types

OracleAS TopLink descriptors can be a class descriptor, an aggregate descriptor, or an EJB descriptor. After creating a descriptor, use this procedure to change the descriptor type.

> **Note:** An EJB descriptor cannot be an aggregate.

To specify a descriptor's type:

1. Select the descriptor in the **Navigator** pane.
2. Click the appropriate descriptor icon (**Class**  , **Aggregate**  , or **EJB** ) on the mapping toolbar.

   You can also select the descriptor and choose **Selected** > **Descriptor Type** > *specific descriptor* type from the menu or by right-clicking on the descriptor in the **Navigator** pane and selecting **Descriptor Type** > *specific descriptor type* from the pop-up menu.

   > **Note:** EJB 2.0 descriptors are created only by reading the `ejb-jar.xml` file. See "Mapping EJB 2.0 Entities" on page 2-8 for more information.

When changing a descriptor's type, the OracleAS TopLink Mapping Workbench adds or removes property tabs, as needed.

- Converting a class or EJB descriptor to an aggregate descriptor removes the **Descriptor Info** and **Queries** tabs. Some advanced properties are not available for aggregate descriptors and will be removed from the **Editor** pane.
- Converting an aggregate descriptor to a class descriptor adds the **Descriptor Info** and **Queries** tabs.

Mapping Descriptors

Descriptors define mappings between classes and tables. To display the attributes in a specific class, expand the descriptor item in the **Navigator** pane (see Figure 1–5).

Use the mapping toolbar (see "Mapping Toolbar" on page 1-7) to choose a mapping type for each attribute.

To map a descriptor:

1. Select a descriptor in the **Navigator** pane. Its properties appear in the **Editor** pane.
2. On the **Descriptor Info** tab, associate the descriptor with a database table (see "Setting Descriptor Information" on page 4-5).
3. In the **Navigator** pane, expand the descriptor to display its attributes.
4. Select an attribute and click the appropriate mapping button on the Mapping toolbar (see "Mapping Toolbar" on page 1-7).

Continue with "Working with Mappings" on page 4-67 to modify the mapping.

Automapping Descriptors

The OracleAS TopLink Mapping Workbench can automatically map class attributes to a similarly named database field. The **Automap** function only creates mappings for unmapped attributes — it does not change previously defined mappings.

You can automap classes for an entire project or for specific tables.

Note: You **must** associate a descriptor with a database table before using the **Automap** function. See "Setting Descriptor Information" on page 4-5 for more information.

To automap attributes:

To automap *all* descriptors in a project, right-click the project icon in the **Navigator** pane and choose **Automap** from the pop-up menu or choose **Selected** > **Automap** from the menu.

or

To automap *a specific* descriptor or attribute choose the descriptor/attribute(s). Right-click and select **Automap** from the pop-up menu or choose **Selected** > **Automap** from the menu.

Generating Java Code for Descriptors

Use this procedure to generate the Java class code for the selected descriptor or package.

To generate Java code:

Right-click the descriptor or package and choose **Export Java Model Source** from the pop-up menu. The Choose a Directory dialog box appears.

You can also choose **Selected** > **Export Java Model Source** from the menu.

If you have not defined deployment and source code generation defaults (see "Working with Project Options" on page 2-11) the OracleAS TopLink Mapping Workbench prompts for a filename and directory.

OracleAS TopLink creates the `<DescriptorName>.java` file in the specified directory.

Working with Descriptor Properties

Each descriptor in the OracleAS TopLink Mapping Workbench contains the following default tabs and specific properties.

- **Descriptor Info**
- **Class Info**
- **Query Keys**
- **Queries**
- **EJB Info** (for EJB descriptors only)

Use the **Set Advanced Properties** function (see "Working with Advanced Properties" on page 4-26 and "To Specify the Default Advanced Properties for Descriptors:" on page 2-13) to specify additional properties for each descriptor.

Setting Descriptor Information

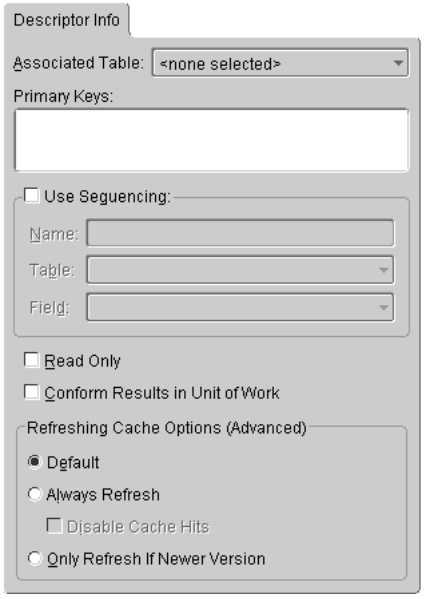
Use the **Descriptor Info** tab to map a descriptor to a specific table in the database, define primary key(s), specify sequencing information, and set caching options.

To map a descriptor to a table:

1. Select a descriptor in the **Navigator** pane. Its properties appear in the **Editor** pane.

2. Click the **Descriptor Info** tab.

Figure 4–1 Descriptor Info Tab



3. Use this table to enter data in each field:

Field	Description
Associated Table	Use the drop-down list to select a database table for the descriptor.
Primary Keys	Specify the primary key(s) for the table.
Use Sequencing	Specify if this descriptor uses sequencing. If selected, specify the **Name**, **Table**, and **Field** for sequencing. See "Working with Sequencing" on page 4-36 for more information.
Read Only	Specify if this descriptor is read-only.
Conform Results in Unit of Work	Specify to use the `conformResultsInUnitOfWork()` method for any read object or read all query. Refer to the *Oracle Application Server TopLink Application Developer's Guide* for more information.

Field	Description
Refreshing Cache	
Default	Use the project's default caching options. OracleAS TopLink will not refresh the cache unless a read query is configured to refresh the cache.
Always Refresh	Refreshes the objects in the cache on all queries. **Note:** Using this property may impact performance.
Disable Cache Hits	Disables the cache hits on primary key read object queries.
Only Refresh if Newer	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 "Working with Optimistic Locking" on page 4-55 for more information.

Note: Use the caching options to specify how descriptors refresh the objects in the cache during queries. This ensures that queries against the session will refresh the objects from the row data.

Setting Class Information

After generating classes and descriptors, use the **Class Info** tab to:

- Rename classes, attributes, and methods
- Add, delete, or edit the generated attributes and methods
- Generate Java source to create new classes

To specify class info:

1. Select a descriptor in the **Navigator** pane. Its properties appear in the **Editor** pane.
2. Click the **Class Info** tab in the **Editor** pane.
3. Select the appropriate tab:
   - Class Tab
   - Attributes Tab
   - Methods Tab

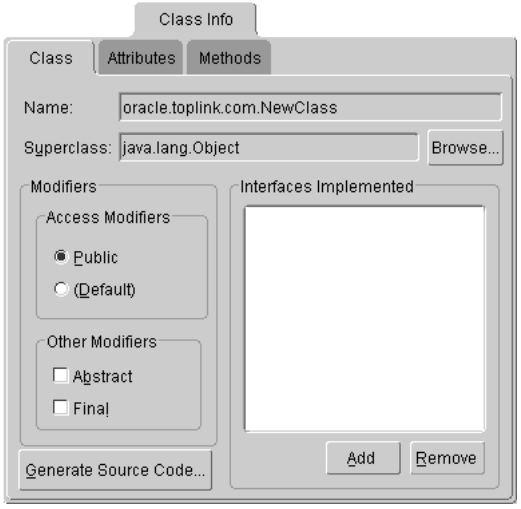
Class Tab

To add a new interface to implement, click **Add.**

To delete an interface, select the interface and click **Remove**.

To generate source code for the descriptor, click **Generate Source Code**.

Figure 4–2 Class Tab



Use this table to enter data in each field:

Field	Description
Name	Name of the class. This field is for display only.
Superclass	Click **Browse** and select a class and package.
Modifiers	
Access Modifiers	Specify if the class is accessible publicly or only within its own package. Non-public classes are not accessible to the OracleAS TopLink Mapping Workbench.
Other Modifiers	Specify if the class is **Final** and/or **Abstract**. Final classes are not included in the superclass selection lists for other classes to extend.

Field	Description
Interfaces Implemented	To add an interface, click **Add** and select the interface and package. To remove an interface, select the interface and click **Remove**. **Note:** You must save your project after removing an interface

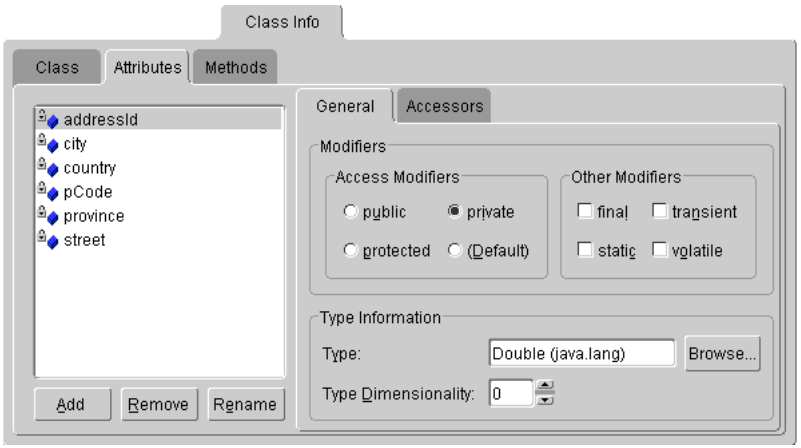
Attributes Tab

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

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

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

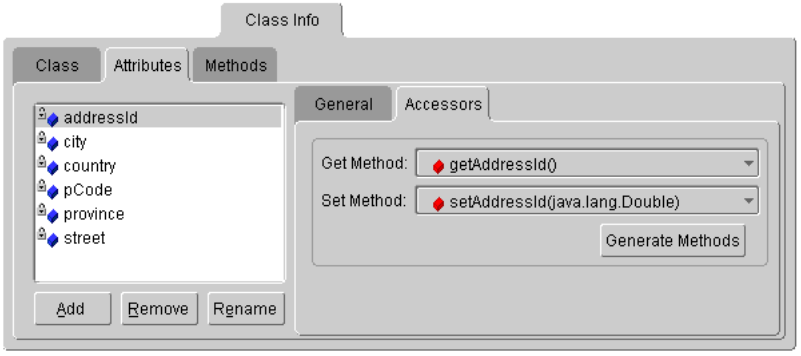
Figure 4–3 Attributes – General Tab



Select an attribute, then use this table to enter data in each field on the **General** tab.

Field	Description
Modifiers	
Access Modifiers	Specify how the attribute is accessible: ▪ **Public** ▪ **Protected** – Public only within its own package ▪ **Private** – Public only for subclasses ▪ **Default** – Public only within its own package
Other Modifiers	Specify whether the attribute is **Final**, **Static**, **Transient**, or **Volatile**. **Note:** Selecting some modifiers may disable others.
Type Information	
Type	Click **Browse** and select a class and package for the attribute.
Type Dimensionality	Specify the length of an array. This field applies only if **Type** is an array.

Figure 4–4 Attributes – Accessors Tab



Select an attribute, then use this table to enter data in each field on the **Accessors** tab.

Field	Description
Get Method	Choose the `get( )` method for the attribute.
Set Method	Choose the `set( )` method for the attribute.

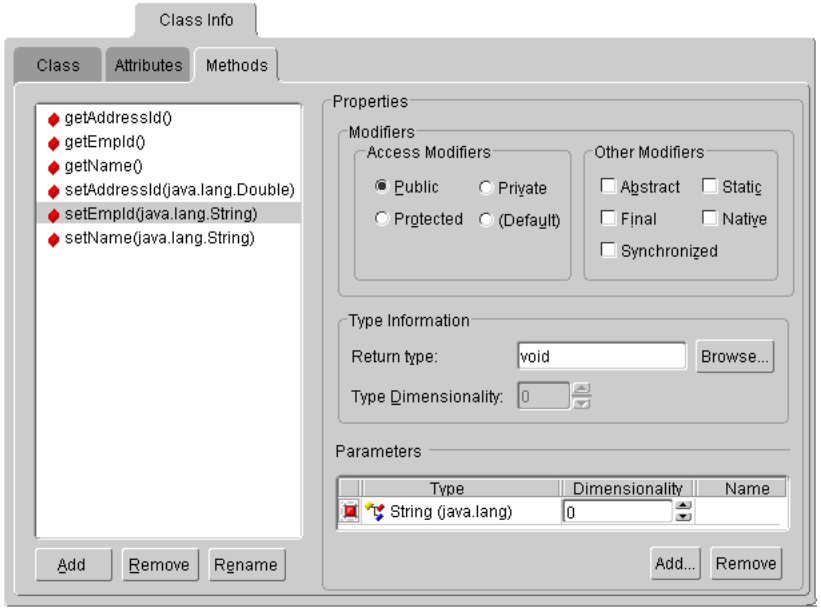
To generate a `get` or `set` method for an attribute, click **Generate Get/Set Methods**.

Methods Tab

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

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

Figure 4–5 Methods Tab



Select a method, then use this table to enter data in each field:

Field	Description
Modifiers	
Access Modifiers	Specify how the attribute is accessible: ▪ **Public**  ▪ **Protected**  – Public only within its own package ▪ **Private**  – Public only for subclasses ▪ **Default**  – Public only within its own package
Other Modifiers	Specify whether the attribute is **Final**, **Static**, **Transient**, or **Volatile**. **Note:** Selecting some modifiers may disable others.
Return Type	Click **Browse** and select a class and package.
Type Dimensionality	Specify the length of the array (**Return Type**).
Parameters	Click **Add** to include parameter(s) for the method. **Note**: The parameters are loaded in the order listed.

Specifying Queries and Named Finders

Use the **Queries** tab to specify EJB QL and SQL queries and finders to use for database access. The **Queries** tab contains two additional tabs: **Named Queries** and **Custom SQL**.

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 the OracleAS TopLink Mapping Workbench, or define them on the **Queries** tab and write them to the file. See "Writing to the ejb-jar.xml File" on page 2-19 for more information.

To create queries:

1. In the **Navigator** pane, select a descriptor.
2. Click the **Queries** tab in the **Editor** pane.

3. Select the appropriate tab:
   - Custom SQL Queries
   - Named Queries

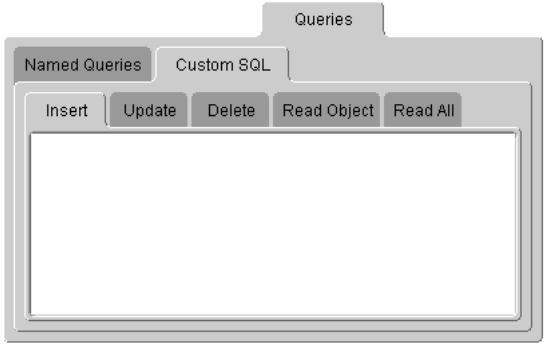
Custom SQL Queries

Use this procedure to create custom SQL queries in the OracleAS TopLink Mapping Workbench. For 2.0 CMP projects, the SQL *is not* written to the `ejb-jar.xml` file.

To create custom SQL queries:

1. In the **Navigator** pane, select a descriptor.
2. Click the **Queries** tab in the **Editor** pane.
3. Click the **Custom SQL** tab.

Figure 4–6 Queries Custom SQL Tab



4. Click the appropriate SQL function tab and type your own query object or SQL string to control these actions for a descriptor.

Tab	Description
Insert	Generates the `insertObject()` method of the `Session` class.

Tab	Description
Update	Generates the `updateObject()` method of the `Session` class. When customizing a descriptor's update **string** 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. Also, the update string must increment the version field if the row is written. In addition, the update string must maintain the row count of the database.
Delete	Generates the `deleteObject()` method of the `Session` class.
Read	Generates the `readObject()` method of the `Session` class Customizing a descriptor's **read-object** query works only for the version of the `readObject()` that takes a primary key expression as a parameter. If other expressions are used, OracleAS TopLink generates dynamic SQL. You can define additional named queries for other read-object queries.
Read All	Generates the `readAllObjects()` method of the `Session` class. Customizing a descriptor's **read-all** query works only for the version of the `readAll()` that takes the class as a parameter—not the version that takes the class and an expression. As a result, the query reads every single instance. You can define additional named queries for other read-all queries.

Note: The OracleAS TopLink Mapping Workbench does not validate the SQL code that you enter. The code is defined by the specific database type.

Example 4–1 Custom Queries

To control the five custom query tabs, you can include your own query object or SQL string for a particular descriptor. The SQL string for each database is defined by the type of database.

For example, the stored procedure to read an object may use the following string:

```
Read_Employee(EMP_ID=>4653)
```

The query manager requires the following string to read the object:

```
Read_Employee(EMP_ID=>#EMP_ID)
```

In this query, the hash character, **#**, binds the argument `EMP_ID` within the SQL string.

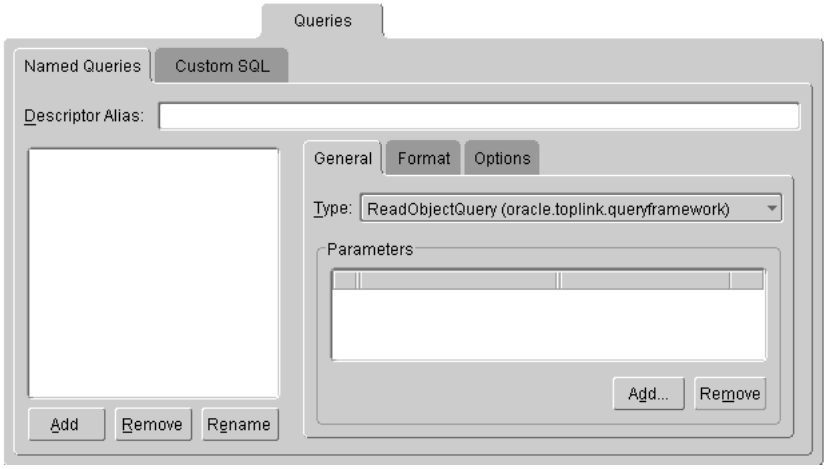
Named Queries

Use named queries to specify SQL, EJB QL, or OracleAS TopLink Expression queries to access the database. EJB QL is a declarative language that presents queries from an object-model perspective. Refer to the EJB specification and the *Oracle Application Server TopLink Application Developer's Guide* for detailed information.

To create a named query:

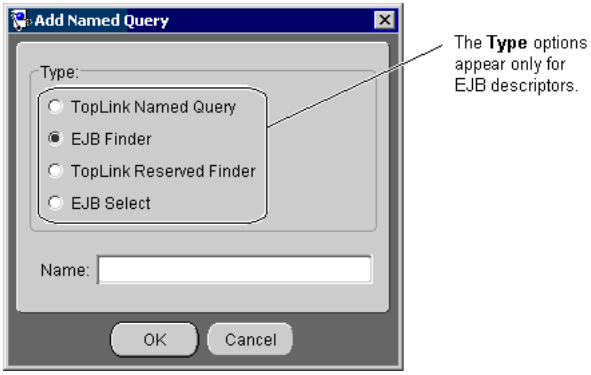
1. In the **Navigator** pane, select a descriptor.
2. Click the **Queries** tab in the **Editor** pane.
3. Click the **Named Queries** tab in the **Queries** tab. The Named Queries tab contains the following additional tabs:
   - **Named Queries General** tab
   - **Named Queries Format** tab
   - **Named Queries Options** tab

Figure 4–7 Named Queries Tab



4. Click **Add** to create a new named query. The Add Named Query dialog box appears.

Figure 4–8 Add named Query

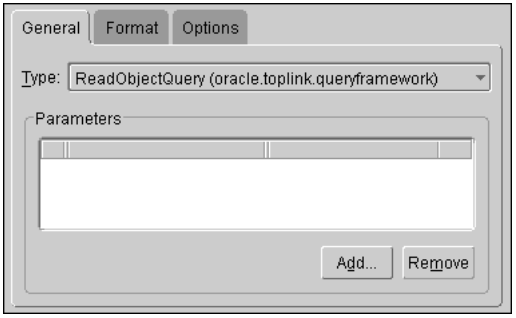


5. Select the query type (for EJB descriptors), enter the query name, and press **Enter**. The OracleAS TopLink Mapping Workbench adds the query to the Named Query tab.

> **Note:** For OracleAS TopLink Reserved Finders, use the drop-down box to select a reserved name. OracleAS TopLink will generate the query at runtime.

6. Click on the **General** tab to specify the query type and parameters.

Figure 4–9 Named Queries General Tab

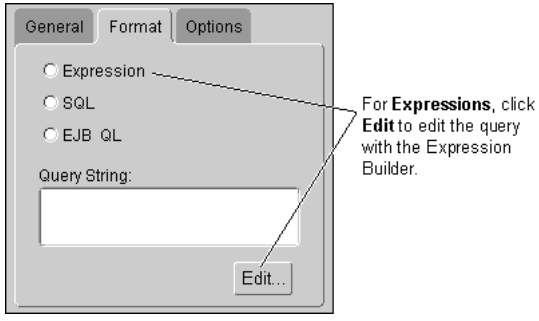


7. Use this table to enter data in each field on the **General** tab.

Field	Description
Descriptor Alias	Alias for the descriptor class. This field applies for EJB finders only.
Name	Name of the query. The prefix of the query name specifies the query type: ■ `find` – EJB 2.0 ■ `ejbSelect` – EJB Select You cannot change the name of OracleAS TopLink Reserved Finders.
Type	Use the drop-down list to specify whether this is a **ReadObject** or **ReadAll query.**
Parameters	Click **Add** to add a parameter of a specific type to this query. **Note**: You cannot add parameters for EJB descriptors.

8. Click on the **Format** tab to specify the named query and its format.

Figure 4–10 Named Queries Format Tab



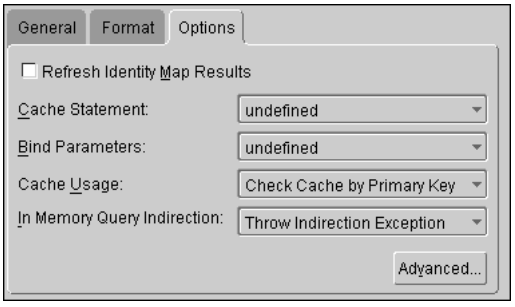
9. Use this table to enter data in each field on the **Format** tab.

Field	Description
Expression	Specify this named query uses an OracleAS TopLink expression.
SQL	Specify this named query is a **SQL** query.
EJB QL	Specify this named query is an **EJB QL** query.

Field	Description
Query String	Entry the query. The OracleAS TopLink Mapping Workbench does not validate the query string. **Note:** For **Expressions**, double-click the query string or click **Edit** to create or edit the query string. See "Building Expressions" on page 4-20 for more information.

10. Click on the **Options** tab to specify additional options for the named query.

Figure 4–11 Named Queries Options Tab



11. Use this table to enter data in each field on the **Options**:

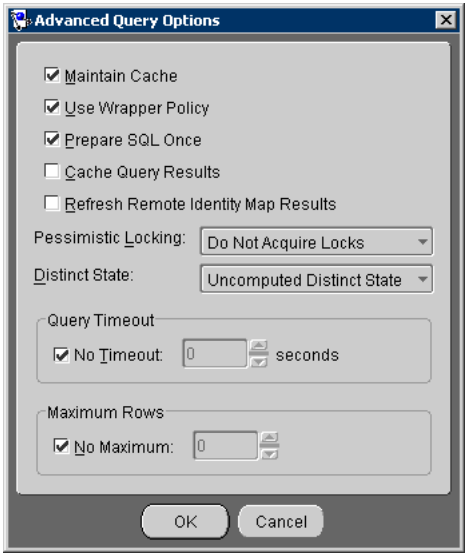
Field	Description
Refresh Identity Map Results	Specify the `refreshIdentityMapResults()` method to refresh the attributes of the object resulting from the query.
Cache Statement	Specify the `cacheStatement()` method for the query. ▪ If **undefined**, the query uses the project's default settings (see "Working with Default Properties" on page 2-10). ▪ If **true**, then **Bind Parameters** must be **true** as well.
Bind Parameters	Specify the `bindAllParameters()` method for the query. ▪ If **undefined**, the query uses the project's default settings (see "Working with Default Properties" on page 2-10).
Cache Usage	Select if/how the query checks the cache before accessing the database.

Field	Description
In Memory Query Indirection	Specify how a query reacts when accessing un-instantiated indirection. Use this option to specify the `InMemoryQueryIndirectionPolicy` policy.

Note: These options are not available for `findOneByQuery` and `findManyByQuery`.

12. Click **Advanced** to specify additional named query options.

Figure 4–12 Advanced Query Options



13. Use this table to enter data in each field and click **OK**:

Field	Description
Maintain Cache	Specify `maintainCache()` for the named query.
Use Wrapper Policy	Specify the `setWrapperPolicy()` for the named query.

Field	Description
Prepare SQL Once	Specify the `setShouldPrepare()` for the named query. By default, OracleAS TopLink optimizes queries to generate their SQL only once. You may need to disable this option for certain types of queries that require dynamic SQL based on their arguments, such as: ■ Expressions that use **equal** where the argument value could be **null**. This may cause problems on databases that require **IS NULL**, instead of **= NULL**. ■ Expressions that use **in** and use parameter binding. This will cause problems as the **in** values must be bound individually.
Cache Query Results	Specify the cacheQueryResults() for the query. OracleAS TopLink can maintains an internal cache of the objects previously returned by a read query.
Refresh Remote Identity Map Results	Specify the `refreshRemoteIdentityMapResult()` method for the query. OracleAS TopLink can refresh the attributes of the object(s) resulting from the query. With cascading, OracleAS TopLink will also refresh the private parts of the object(s).
Pessimistic Locking	Specify the pessimistic locking policy for the query.
Distinct State	Specify if OracleAS TopLink prints the DISTINCT clause, if a distinct has been set.
Query Timeout	Specify if the query will time out (or abort) after a specified number of seconds.
Maximum Rows	Specify if the query will limit the results to a specified number of rows. Use this to option for queries that could return an excessive number of objects.

Refer to the *Oracle Application Server TopLink Application Developer's Guide* for additional information on named queries.

Building Expressions

Use the Expression Builder to create OracleAS TopLink expressions for Named Queries.

To build an expression:

1. From the **Named Queries Format** tab (see Figure 4–10), click **Edit** (or double-click a query string). The Expression Builder dialog box appears.

Figure 4–13 Expression Builder

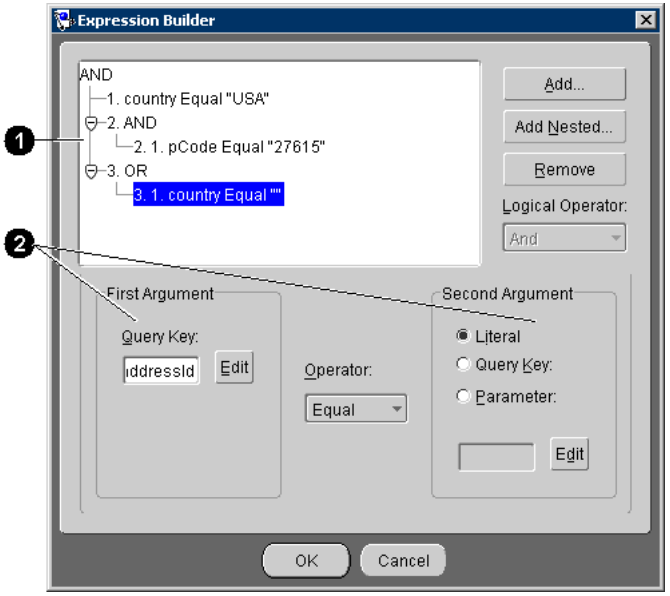


Figure 4–13 calls out the following user-interface components:

1. Expression tree
2. Arguments

2. Click **Add** or **Add Nested** to create a new expression. OracleAS TopLink assigns a sequence number to each node and nested node.

   Click **Remove** to remove an existing expression.

3. Select the node and use the **Logical Operator** drop-down list to specify the operator for the node (**AND**, **OR**, **Not AND**, or **Not OR**).

4. Choose the expression and use this table to enter data in each field:

Field	Description
First Argument	Click **Edit** and select the query key for the first argument. See "Adding Arguments" for more information.

Field	Description
Operator	Specify how OracleAS TopLink should evaluate the expression. Valid operators include: **Equal**, **Not Equal**, **Equal Ignore Case**, **Greater Than**, **Greater Than Equal**, **Less Than**, **Less Than Equal**, **Like**, **Not Like**, **Like Ignore Case**, **Is Null**, and **Not Null**.
Second Argument	Specify the second argument: ■ **Literal** — Click **Edit** and choose a literal type and value. ■ **Query Key** — Click **Edit** and select the query key. ■ **Parameter** — Choose a previously created parameter argument. See "Adding Arguments" for more information.

5. Click **OK**. The OracleAS TopLink Mapping Workbench adds the expression to **Named Queries** tab.

Adding Arguments

Each expression contains elements (*arguments*) to evaluate. Expressions using the **Is Null** or **Not Null** operators require only a single argument.

Use this procedure to add new arguments.

1. Select an existing expression or click **Add** (or **Add Nested**) to add a new expression to the named query.
2. For the **First Argument**, click **Edit**. The Choose Query Key dialog box appears.

Figure 4–14 Choose Query Key

3. Select the attribute, specify if the query allows a null value, and click **OK**.
4. Use the **Operator** drop-down list to specify how OracleAS TopLink should evaluate the expression.
5. For the **Second Argument**, select **Literal**, **Query Key**, or **Parameter**, and click **Edit**.
   - For **Literal** arguments, the Edit the Literal Type and Value dialog box appears. Choose the literal type (such as **String** or **Integer**) and value.
   - For **Query Key** arguments, the Choose Query Key dialog box appears (see Figure 4–14).
   - For **Parameter** arguments, use the drop-down list to select the specific parameter, as created on the Named Queries General tab (see Figure 4–9 on page 4-16).

Repeat this procedure for each expression or sub-expression.

Example 4–2 Sample Expression

This expression:

```
AND
  1.manager(Allows Null).lastName EQUAL "Jones"
  2.OR
    2.1.projects.name LIKE "BETA"
```

```
  2.2.projects.id EQUAL "4"
3.OR
  3.1.AND
    3.1.1.address.country EQUAL "Canada"
    3.1.2.salary GREATER THAN "25000"
  3.2.AND
    3.1.1.address.country EQUAL "United States"
    3.1.2.salary GREATER THAN "37500"
```

will find employees who:

- Have a manager with the last name Jones or have no manager, and
- Work on projects with the name Beta or project ID 4, and
- Live in Canada and have a salary of more than 25,000 or
  Live in the United States and have a salary of more than 37,500

Query Keys

The OracleAS TopLink Mapping Workbench uses query keys as an alias for a field name. With an alias, OracleAS TopLink expressions can use the Java names instead of DBMS-specific field names. See "Automatically-generating Query Keys" on page 4-60 for additional information.

Use the **Query Keys** tab to create user-defined queries or to work with automatically generated query keys.

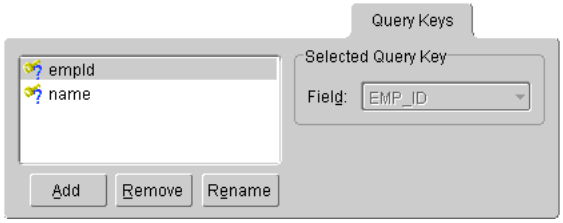
Specifying Query Keys

Use the **Query keys** tab to specify a query key for a descriptor.

To specify query keys:

1. Select a descriptor in the **Navigator** pane. Its properties appear in the **Editor** pane.
2. Click the **Query Keys** tab in the **Editor** pane.

Figure 4–15 Query Keys Tab



3. 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**.

4. Use the **Field** drop-down list to select the field in the table used by the query.

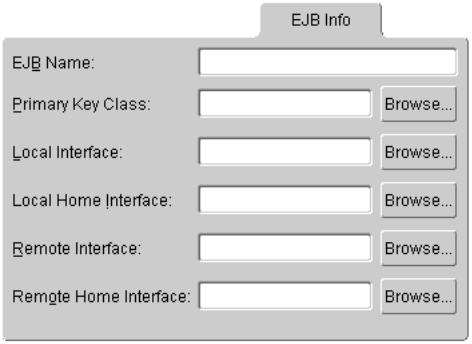
Displaying EJB descriptor Information

Use the **EJB Info** tab to display the EJB descriptor's information (from the `ejb-jar.xml` file). This tab is available only for EJB descriptors.

To display EJB descriptor information:

1. In the **Navigator** pane, select an EJB descriptor.
2. Click the **EJB Info** tab in the **Editor** pane.

Figure 4–16 EJB Info Tab



3. Use this table to identify each field:

Field	Description
EJB Name	Base name. When using EJB 2.0, this is specified in the **<ejb-name>** element of the `ejb-jar.xml` file.
Primary Key Class	Primary key. When using EJB 2.0, this is specified in the **<prim-key-class>** element of the `ejb-jar.xml` file.
Local Interface	Local interface. When using EJB 2.0, this is specified in the **<local>** element of the `ejb-jar.xml` file.
Local Home Interface	Local home interface. When using EJB 2.0, this is specified in the **<local-home>** element of the `ejb-jar.xml` file.
Remote Interface	Remote interface. When using EJB 2.0, this is specified in the **<remote>** element of the `ejb-jar.xml` file.
Remote Home Interface	Remote interface. When using EJB 2.0, this is specified in the **<home>** element of the `ejb-jar.xml` file.

Note: When using EJB 2.0 persistence, these fields are for display only.

Working with Advanced Properties

You can also specify the following advanced properties for each descriptor:

- Amending Descriptors After Loading
- Specifying Events
- Specifying Inheritance
- Specifying Optimistic Locking
- Specifying Multi-table Info
- Setting the Copy Policy
- Specifying Identity Mapping
- Setting Instantiation Policy
- Specifying an Interface Alias

To display advanced properties:

Right-click a descriptor in the **Navigator** pane and choose **Set Advanced Properties** > *specific property* from the pop-up menu or choose **Selected** > **Set Advanced Properties** > *specific property* from the menu.

See "To Specify the Default Advanced Properties for Descriptors:" on page 2-13 for information on specifying default advanced properties.

Amending Descriptors After Loading

Some OracleAS TopLink features cannot be configured from the OracleAS TopLink Mapping 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 takes the descriptor as a single parameter. You can then send messages to the descriptor or any of its specific mappings to configure advanced features.

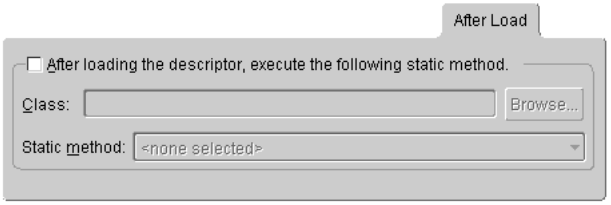
To a specify a method to execute after loading the descriptor:

1. Select a descriptor in the **Navigator** pane. Its properties appear in the **Editor** pane.

   If the **After load** advanced property is not visible for the descriptor, right-click the descriptor and choose **Set Advanced Properties** > **After Load** from pop-up menu or from the **Selected** menu.

2. Click the **After load** tab in the **Editor** pane.

Figure 4–17 After Load Tab



3. Use this table to enter data in each field:

Field	Description
After Loading	Specify whether the OracleAS TopLink Mapping Workbench should execute a method after loading the descriptor.

Field	Description
Class	Click **Browse** and choose the class of the method to execute.
Static Method	Use the **Static Method** drop-down list to choose the method to execute.

Specifying Events

Use the **Events** tab to specify a methods to execute when certain events occur.

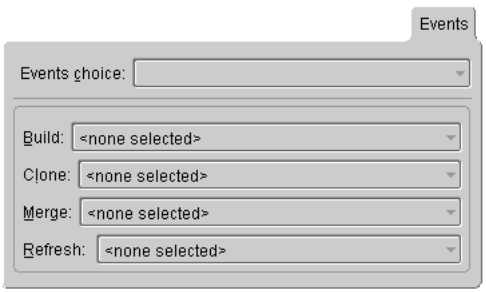
To specify an event method:

1. Select a descriptor in the **Navigator** pane. Its properties appear in the **Editor** pane.

   If the **Events** advanced property is not visible for the descriptor, then right-click the descriptor and choose **Set Advanced Properties** > **Events** from pop-up menu or from the **Selected** menu.

2. Click the **Event** tab in the **Editor** pane.

Figure 4–18 Events Tab



3. Use this table to enter data in each field:

Field	Description
Events Choice	Select an event for this descriptor.
Methods	Select a method for each event. **Note**: The methods available will vary, depending on the **Event**.

See "Supported Events" on page 4-64 for a complete list of events and methods.

Specifying Identity Mapping

The OracleAS TopLink Mapping Workbench specifies the default identity mapping for each descriptor in the project options (see "Working with Default Properties" on page 2-10). Use the **Identity** tab to specify identity map and existence checking information.

> **Note:** Changing the project's default identity policy does not affect descriptors that already exist in the project.

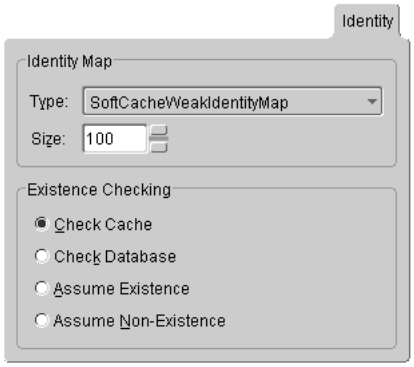
To specify an identity map for a descriptor:

1. In the **Navigator** pane, select a descriptor.

   If the **Identity** advanced property is not visible for the descriptor, right-click the descriptor and choose **Set Advanced Properties** > **Identity** from the pop-up menu or from the **Selected** menu.

2. Click the **Identity** tab.

Figure 4–19 Identity Tab



3. Use this table to enter data in each field:

Field	Description
Type	Use the **Type** drop-down list to choose the identity map (see Table 4–2 for details).
Size	Size of the identity map.

Field	Description
Existence Checking	Specify the method of existence checking.

Specifying Inheritance

Use the **Inheritance** tab to specify the descriptor's inheritance properties as either a root or subclass (branch class or leaf class).

> **Note:** When using an aggregate descriptor in an inheritance, *all* the descriptors in the inheritance tree must be aggregates. Aggregate and Class descriptors cannot exist in the same inheritance tree.

Creating a Root Class

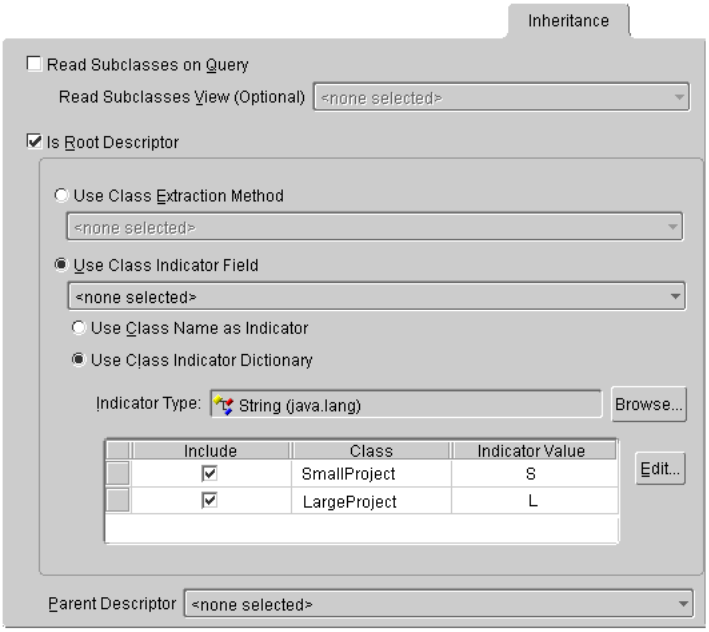
Use this procedure to create a root class.

To specify a root class:

1. In the **Navigator** pane, select the descriptor you wish to specify as the root.
2. Choose the **Inheritance** tab in the **Property** pane.

   If the **Inheritance** tab is not visible, right-click the descriptor and choose **Set Advanced Properties** > **Inheritance**.

Figure 4–20 Creating a Root Class



3. To instantiate the descriptor's subclasses when queried, select the **Read Subclasses on Query** checkbox. Select a database view to use for reading subclasses if desired.

Note: The view can be used for root or branch classes that have subclasses spanning multiple tables. The view must apply an outer-join or union all to the subclass tables.

---

4. Select the **Is Root Descriptor** checkbox.
5. You can use a class extraction method or a class indicator field to specify which class to instantiate on querying. Choose the option and select the appropriate method or field.
6. If you use a class indicator field, you can use the class name as the indicator, or you can use a class indicator dictionary. Choose which option you wish to use and specify the necessary information.

7. If you use an indicator dictionary, choose the indicator type and set the indicator values for each subclass.

   Note: A list of subclasses and their indicator values appears when the subclasses have set their parent descriptor. Abstract roots are not in the list.

8. If you want instances of the subclasses to be instantiated when the root class is queried, select the **Read Subclasses on Query** checkbox. Do not select this checkbox for leaf classes.

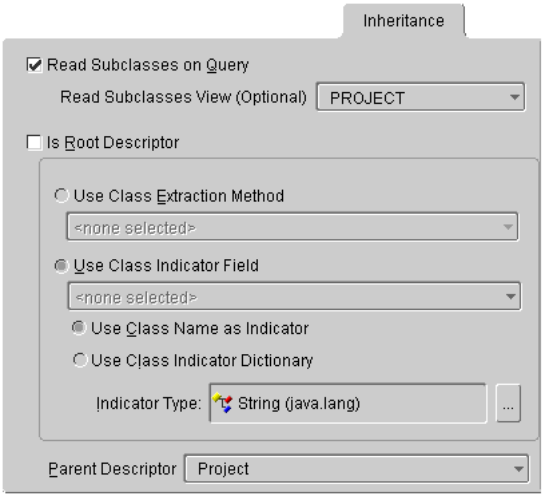
Creating Branch and Leaf Classes

After setting up the root class for inheritance, you must also specify properties for branch and leaf classes.

To create branch and leaf classes:

1. In the **Navigator** pane, select the descriptor for which to specify inheritance information.
2. If the **Inheritance** advanced property has not been added to the descriptor, right-click the descriptor and choose **Set Advanced Properties** > **Inheritance**.
3. Click the **Inheritance** tab of the **Editor** pane.
4. Ensure that **Is Root Descriptor** is not selected. The **Parent Descriptor** drop-down list is now enabled and the class indicator information is disabled.

Figure 4–21 Creating Branch and Leaf Classes



5. Choose the parent descriptor from the **Parent Descriptor** drop-down list. This may be the root class or a branch class.
6. Select the **Read Subclasses on Query** option if this is a branch class and you want its subclasses to be instantiated when it is queried. Choose a database view for reading subclasses, if desired. Do not select this checkbox for leaf classes.

Specifying Optimistic Locking

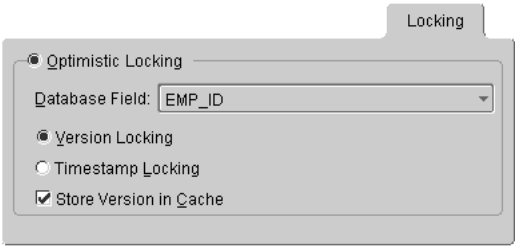
Use the **Locking** tab to specify whether the descriptor uses optimistic locking.

To specify a descriptor's locking policy:

1. In the **Navigator** pane, select a descriptor.

   If the **Locking** advanced property is not visible for the descriptor, right-click the descriptor and choose **Set Advanced Properties** > **Locking** from the pop-up menu or from the **Selected** menu.
2. Click the **Locking** tab.

Figure 4–22 Locking Tab



3. Use this table to enter data in each field:

Field	Description
Optimistic Locking	Specify that the descriptor uses optimistic locking.
Database Field	Select the correct database field used for optimistic locking.
Version Locking	Specify that the descriptor uses version locking.
Timestamp Locking	Specify that the descriptor uses timestamp locking.
Store Version in Cache	Specify whether you want to store the version information in the cache.

Specifying an Interface Alias

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.

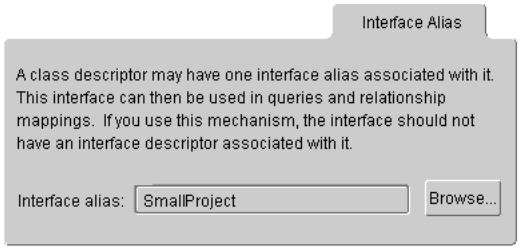
To specify an interface alias:

1. In the **Navigator** pane, select a descriptor.

   If the **Interface Alias** advanced property is not visible for the descriptor, right-click the descriptor and choose **Set Advanced Properties** > **Interface Alias** from pop-up menu or from the Selected menu.

2. Click the **Interface Alias** tab.

Figure 4–23 Interface Alias Tab



3. Click **Browse** and select an interface.

Working with Primary Keys

A primary key is a column (or a combination of columns) that contains a unique identifier for every record in the table. In the OracleAS TopLink Mapping Workbench, every table that stores persistent objects must have a primary key. Tables that require multiple columns to create an identifier use a *composite* primary key. Setting the primary key for a table also sets the primary key for the descriptor that uses the table.

The OracleAS TopLink Mapping Workbench implements primary keys using sequence numbers (see "Working with Sequencing" on page 4-36).

Each descriptor must provide mappings for its primary key. These mappings may be direct, transformation, or one-to-one. The OracleAS TopLink Mapping Workbench does not require you to define a primary key constraint in the database — only that the fields specified for the primary key are unique.

Note: Primary keys for classes in an inheritance hierarchy or for descriptors that map to multiple tables have special requirements. Refer to"Working with Inheritance" on page 4-39 and "Working with Multiple Tables" on page 4-48 for more information.

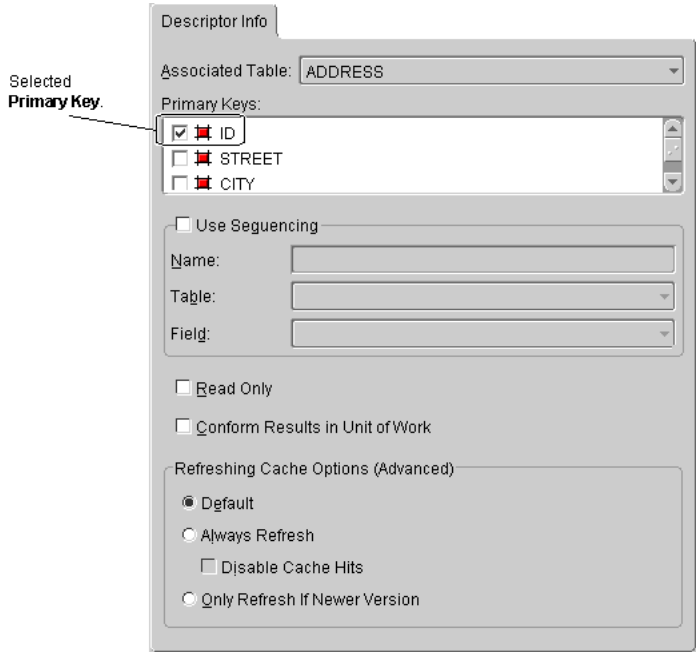
Setting a Primary Key for Descriptors

Use this procedure to set a primary key for a descriptor.

To set a primary key:

1. Select a descriptor in the **Navigator** pane. Its properties appear in the **Editor** pane.
2. Click the **Descriptor Info** tab.

Figure 4–24 Primary Keys



3. Select the field(s) to set as the primary key.

Working with Sequencing

Sequence numbers are artificial keys that uniquely identify the records in a table. When you define a sequence number field for a descriptor, the OracleAS TopLink Mapping Workbench automatically generates a new sequence number every time you insert a new record into the table.

Use the project's **Sequencing** tab (see Figure 2–5) or the **Sequencing** area of a descriptor's **Descriptor Info** tab (see Figure 4–1) to specify sequencing information

Database tables often use a sequence number as the primary key. The OracleAS TopLink Mapping Workbench can use the database's native support or a sequence table to maintain sequence numbers.

Tip: Oracle recommends using sequence numbers for primary keys because they are single, guaranteed, unique values.

Other data values may require composite primary keys to make up a unique value, which is less optimal. Additionally, non-artificial values may need to change, and this is not allowed for primary keys.

Using Sequence Numbers with Entity Beans

When implementing sequencing for Entity Beans, you must provide `create()` methods and the corresponding `ejbCreate()` and `ejbPostCreate()` methods for your bean home and bean class.

OracleAS TopLink creates the primary key value when you first insert the bean in the database. The key value is not passed as a parameter to the `create()` methods because they do not set the primary key value (the key is generated).

Note: Be careful when using transactions with these create methods. If you create an Entity Bean within a transaction and you use native sequencing in Sybase, SQL Server or Informix, then the bean's key is not initialized until the transaction commits and the bean is persisted to the database for the first time.

Using Native Sequencing

Oracle, Sybase, SQL Server, and Informix databases support native sequencing in which the DBMS generates the sequence numbers. However, the OracleAS TopLink Mapping Workbench must still tell the DBMS to assign sequence number values.

- For Oracle databases, create a SEQUENCE object in the database.
- For Sybase and SQL Server databases, set the primary key field to IDENTITY.
- For Informix databases, set the primary key field to use SERIAL.

Tip: If you use native sequencing in these databases, the OracleAS TopLink Mapping Workbench cannot support pre-allocation. Oracle recommends using the sequence table instead. Oracle databases support pre-allocation, but only if the sequence increment matches the pre-allocation size. See "Pre-allocating Sequence Numbers" on page 4-38 for more information.

Using Sequence Tables

If your database does not use native sequencing, you must manually create the sequence table (named SEQUENCE). Use this table to store each table, as illustrated below:

Field name	Field format	Description
SEQ_NAME	CHAR	Name of the sequence number
SEQ_COUNT	NUMERIC	Current value

After creating the table, you must initialize the table within the application. The value of the `SEQ_COUNT` field for each sequence should be zero (**0**), as in the following table.

SEQ_NAME	SEQ_COUNT
EMP_SEQ	0
PROJ_SEQ	0

Pre-allocating Sequence Numbers

To increase the speed of database inserts, obtain a block of sequence numbers (by setting an allocation size) instead of executing a corresponding `SELECT` statement to obtain the newly assigned sequence number each time you create an object.

OracleAS TopLink uses a default pre-allocation size of **50** when using a sequence table and **1** when using native sequencing.

- When using native sequencing in Sybase, SQL Server, or Informix databases, pre-allocation cannot be set — it is always **1**.
- When using native sequencing, you must set the pre-allocation size explicitly in the OracleAS TopLink Mapping Workbench.

- When using native sequencing in an Oracle database, you can use pre-allocation only if an `INCREMENT` is set on the Oracle Sequence object (not the `CACHE` option). This increment *must* match the pre-allocation size specified in the OracleAS TopLink Mapping Workbench. If the increment is set incorrectly, invalid and negative sequence numbers could be generated. The `CACHE` option specifies how many sequences are pre-allocated on the database server; the `INCREMENT` specifies the number that can be pre-allocated to the database client.

  Tip: Oracle recommends using sequence pre-allocation because of its performance and concurrency benefits.

Creating the Sequence Table on the Database

Normally, the database administrator defines the sequence table or sequencing object. However, you can use the OracleAS TopLink schema manager to define the sequence numbers using:

```
SchemaManager schemaManager = new SchemaManager(session);
schemaManager.createSequences();
```

You should execute this command only once. The `SchemaManager` creates a sequence entry for each registered descriptor.

Refer to the book *Oracle Application Server TopLink Application Developer's Guide* for more information on using the schema manager to create number information in the database.

Working with Inheritance

Inheritance describes how a child class inherits the characteristics of its parent class. OracleAS TopLink provides multiple methods to preserve the inheritance relationships. You can override mappings that have been specified in a superclass, or map attributes that have not been mapped at all in the superclass.

Note: When using an aggregate descriptor in an inheritance, *all* the descriptors in the inheritance tree must be aggregates. Aggregate and Class descriptors cannot exist in the same inheritance tree.

Using Inheritance with EJBs

Although inheritance is a standard tool in object-oriented modeling, the current EJB specification contains only general information regarding inheritance. You should fully understand the current EJB specification before implementing inheritance.

Caution: Use caution when employing inheritance. The next EJB specification may dictate inheritance guidelines not supported by the different servers.

Mapping Inherited Attributes in One Descriptor

If you are mapping only one class in an inheritance hierarchy, you can map attributes that it inherits from any of its superclasses.

To map attributes in one descriptor:

1. In the **Navigator** pane, select a descriptor.
2. Right-click the descriptor and choose **Map Inherited Attributes** > *specific location* from the pop-up menu. You can also choose **Selected** > **Map Inherited Attributes** from the menu.

   Map inherited attributes to:

   - Superclass
   - Root minus one
   - Selected class
3. Map the now visible attributes as though they belonged to this descriptor.

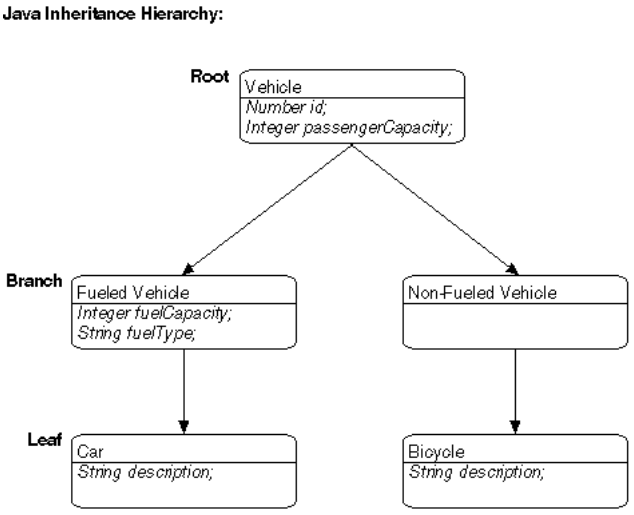
You can also do this if you have a common superclass that stores little or no persistent data. For example, if you were mapping subclasses of `java.rmi.RemoteObject`, each subclass could be mapped independently.

Supporting Inheritance Using One Table

Store classes with multiple levels of inheritance in a single table to optimize database access speeds.

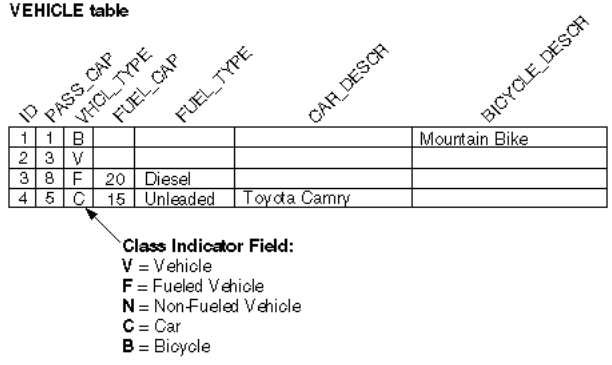
The following diagram illustrates the `Vehicle` object model.

Figure 4–25 Supporting Inheritance Using One Table



The entire inheritance hierarchy can share the same table, as in Figure 4–26. 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.

Figure 4–26 Inheritance Using a Superclass Table with Optional Fields

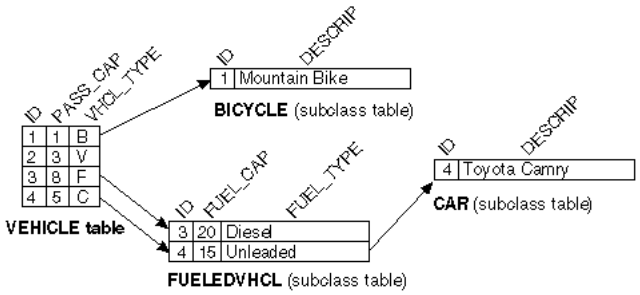


Supporting Inheritance Using Multiple Tables

For subclasses that require additional attributes, use multiple tables instead of a single superclass table to optimize storage space (because there are no unused fields in the database). However, this may affect performance because OracleAS TopLink must read from more than one table before it can instantiate the object. OracleAS TopLink first looks at the class indicator field to determine the class of object to create, then uses the descriptor for that class to read from the subclass tables.

Figure 4–27 illustrates the OracleAS TopLink implementation of the `FUELEDVHCL`, `CAR`, and `BICYCLE` tables. All objects are stored in the `VEHICLE` table. The secondary table stores `FueledVehicle`, `Car`, and `Bicycle` information.

Figure 4–27 Inheritance Using Separate Tables for Each Subclass



Note: Because `NonFueledVehicle` does not hold any attributes or relationships, it does not need a secondary table. For performance considerations, this design is inefficient because it requires multiple table fetching.

Finding Subclasses

An inheritance mapping for a root class must be able to locate its subclasses by using one of the following methods:

- Providing a *class indicator field*, which contains a key corresponding to its subclass.
- Including a *class extraction method*, which can be implemented in Java code; this is simply a method that returns a `java.lang.Class` object.
- Using class names directly in the class indicator field.

Providing a Class Indicator Field

Use a *class indicator field* in the table of the root class table to indicate which subclass should be instantiated. 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. Refer to "Working with Transformation Mappings" on page 5-10 for more information.

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. The following table illustrates the class indicator mapping from the `Vehicle` class containing four entries.

Table 4–1 Class Indicator Mapping from the Vehicle Class

Key	Value
F	FueledVehicle
N	NonFueledVehicle
C	Car
B	Bicycle

When working with hierarchies more than two levels deep, the class indicator field and the class indicator mapping can be in only the root class.

> **Note:** All concrete classes in the hierarchy must have a defined indicator value.

Understanding Root, Branch, and Leaf Classes in an Inheritance Hierarchy

OracleAS TopLink allows three types of classes in an inheritance hierarchy:

- 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 Example 4–25, "Supporting Inheritance Using One Table" on page 4-41 is a root class.

- *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 Example 4–25, "Supporting Inheritance Using One Table" on page 4-41 is a branch class.
- *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 Example 4–25, "Supporting Inheritance Using One Table" on page 4-41 is a leaf class.

Specifying Primary Keys in an Inheritance Hierarchy

OracleAS 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 tables that have different primary keys must define the mapping between the root primary key and the local one.

See "Specifying Multi-table Info" on page 4-49 for more information on primary keys in an inheritance hierarchy.

Mapping Inherited Attributes 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 subclasses that inherit the attribute are unaffected.

To view and map attributes inherited from a superclass:

1. In the **Navigator** pane, right-click a descriptor and choose **Map Inherited Attributes** > *selected class* from the pop-up 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 "Working with Mappings" on page 4-67 for more information.

If you leave inherited attributes unmapped, OracleAS TopLink uses the mapping (if any) from the superclass if the superclass's descriptor has been designated as the parent descriptor.

Working with Interfaces

An *interface* is a collection of method declarations and constants used by one or more classes of objects. Domain classes can implement interfaces or can reference existing interfaces. OracleAS TopLink supports interfaces in the following methods:

- In a *variable class relationship*, a domain object references another domain object or a collection of objects that implement a specific interface.
- A read query can be issued to query an interface.

Understanding Interface Descriptors

An *interface descriptor* is a descriptor whose reference class is an interface. Each domain class specified in OracleAS TopLink has a related descriptor. A descriptor is a set of mappings that describes how an object's data is represented in a relational database. It contains mappings from the class instance variable to the table's fields, as well as the transformation routines necessary for storing and retrieving attributes. The descriptor acts as the link between the Java object and its database representation.

An interface is a collection of abstract behavior that other classes can use. It is a purely Java object 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.

Note: You cannot create or edit interface descriptors in the OracleAS TopLink Mapping Workbench.

Here are the components defined in the interface descriptor:

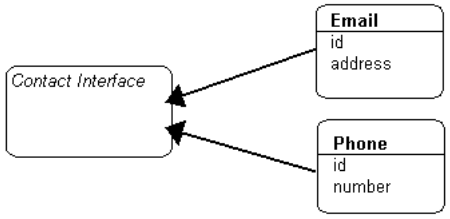
- The Java interface it describes
- The parent interface(s) it implements
- A list of abstract query keys

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. A read query on the interface results in reading one or more of its implementors.

The following illustration shows an interface implemented by two descriptors.

Figure 4–28 Classes Implement an Interface



Following is the sample code implementation for the descriptors for `Email` and `Phone`:

```
Descriptor descriptor = new Descriptor();
    descriptor.setJavaInterface(Contact.class);
    descriptor.addAbstractQueryKey("id");
    return descriptor;
Descriptor descriptor = new Descriptor();
    descriptor.setJavaClass(Email.class);
    descriptor.addDirectQueryKey("id", "E_ID");
    descriptor.getInterfacePolicy().addParentInterface(Contact.class);
    descriptor.setTableName("INT_EML");
    descriptor.setPrimaryKeyFieldName("E_ID");
    descriptor.setSequenceNumberName("SEQ");
    descriptor.setSequenceNumberFieldName("E_ID");
    descriptor.addDirectMapping("emailID", "E_ID");
    descriptor.addDirectMapping("address", "ADDR");
    return descriptor;
Descriptor descriptor = new Descriptor();
    descriptor.setJavaClass(Phone.class);
    descriptor.getInterfacePolicy().addParentInterface(Contact.class);
    descriptor.addDirectQueryKey("id", "P_ID");
    descriptor.setTableName("INT_PHN");
    descriptor.setPrimaryKeyFieldName("P_ID");
    descriptor.setSequenceNumberName("SEQ");
    descriptor.setSequenceNumberFieldName("P_ID");
    descriptor.addDirectMapping("phoneID", "P_ID");
    descriptor.addDirectMapping("number", "P_NUM");
```

```
    return descriptor;
```

If the `Contact` interface extended another interface, you would call the following method to set its parent:

```
descriptor.getInterfacePolicy().addParentInterface(Interface.class);
```

Single Implementor Interfaces

Use single implementor interfaces for applications where only the domain objects' interface is visible. Each domain class has its own unique interface, and no other domain class implements it. The references to other domain objects are also through interfaces.

In such applications, defining a descriptor for each interface would be expensive and may be unnecessary. OracleAS TopLink does not force you to define descriptors for such interfaces. The descriptors are defined for the domain classes, and the parent interface is set as usual.

During the initializing of a descriptor, the interface is given the descriptor of its implementor. This process allows queries on both the domain class and its interface. The only restriction is that each interface should have a unique implementor. In other words, a descriptor is not needed for an interface unless it has multiple implementors.

Implementing an Interface

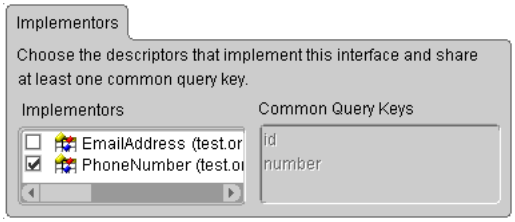
One-to-one mappings that reference interfaces that have multiple implementors are known as *variable* one-to-one mappings. See Chapter 6, "Understanding Relationship Mappings", and Chapter 4, "Understanding Descriptors" for more information.

Use this procedure to implement an interface.

To configure an interface descriptor:

1. In the **Navigator** pane, select an interface.
2. On the **Implementors** tab in the **Editor** pane, click the descriptors that implement this interface and share at least one common query key.

Figure 4–29 Implementors Tab



The **Common Query Keys** area displays all the query keys for the interface's implementors.

To specify a class descriptor as a single implementor of an interface:

1. In the **Navigator** pane, select the descriptor that will be the sole implementor of an interface.
2. If the **Interface Alias** advanced descriptor property is not visible for this descriptor, choose **Set Advanced Properties** > **Interface Alias** from the **Selected** menu or the pop-up menu to create the **Interface Alias** page.
3. Select the interface that will serve as an alias for this descriptor on the **Interface Alias** page. It is not necessary for this interface to have a descriptor in the project, and in fact, if an associated descriptor exists, it will be removed. Every instance of the interface will now be treated as an instance of this class as well.

Working with Multiple Tables

Descriptors can use multiple tables in mappings. Use multiple tables when:

- 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, OracleAS TopLink assumes that the primary key of the first, or primary, table is included in the additional tables, thereby joining the tables.

OracleAS TopLink also supports custom methods for joining tables.

Specifying Multi-table Info

Use the **Multi-table Info** tab to define multiple tables for a descriptor in the OracleAS TopLink Mapping Workbench.

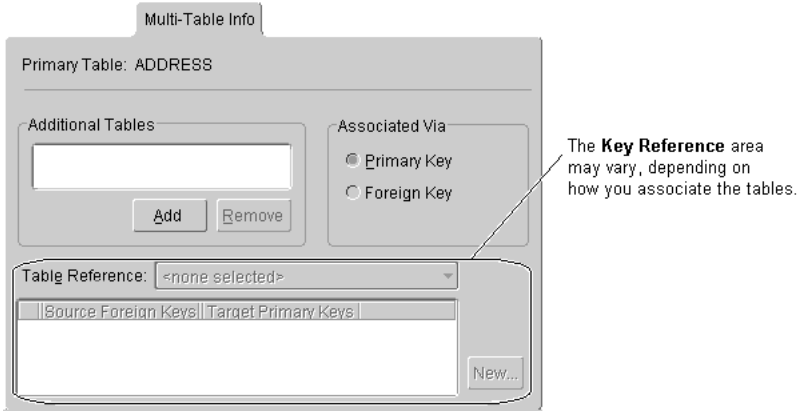
To associate multiple tables with a descriptor:

1. In the **Navigator** pane, select a descriptor.

   If the **Multi-table Info** advanced property is not visible for the descriptor, right-click the descriptor and choose **Set Advanced Properties** > **Multi-table Info** from pop-up menu or from the **Selected** menu.

2. Click the **Multi-table Info** tab.

Figure 4–30 Multi-table Info Tab

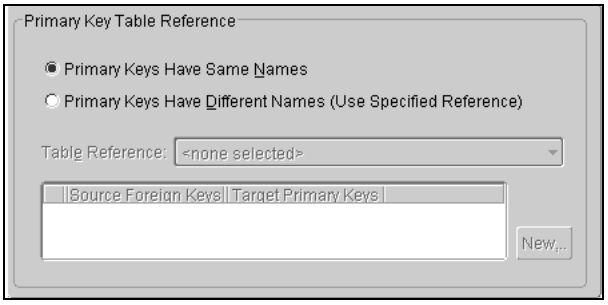


3. Use this table to enter data in each field:

Field	Description
Primary Table	The primary table for this descriptor. This field is for display only.
Additional Tables	Use the **Add** and **Remove** buttons to add or remove additional tables.
Associated Via	Specify if each **Additional Table** is associated by its **Primary** or **Foreign** key.

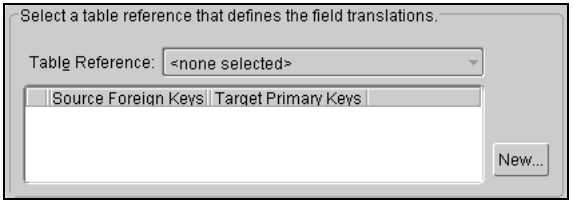
When associating an additional table via **Primary Key,** additional options appear on the Multi-table Info tab. Continue with "Primary Keys Match" on page 4-50 or "Primary Keys are Named Differently" on page 4-51 to assign the primary key.

Figure 4–31 Associating Multiple Tables via Primary Key



When associating a table via **Foreign Key**, additional options (shown in Figure 4–31) appear. You must choose a reference that relates the correct fields in the primary table to the primary keys in the selected table. Continue with "Tables are Related by Foreign Key in Primary Table" on page 4-51 to assign the foreign key.

Figure 4–32 Associating Multiple Tables via Foreign Key



Primary Keys Match

When associating a descriptor with multiple tables in which the primary key field names are identical, you do not have to specify any additional information. Select the tables from the list of available tables on the **Multi-table Info** tab. The OracleAS TopLink Mapping Workbench automatically selects the **Primary Keys Have the Same Names** option.

Primary Keys are Named Differently

If the primary keys of the additional table(s) are the same, but they are named differently, you must specify how they relate to the primary key(s) of the default/primary table.

1. Select the associated table, and select **Associated Via Primary Key**.
2. Select **Primary Keys Have Different Names**.
3. In the **Primary Key Reference** area (Figure 4–31), choose a table reference that relates how the primary keys of the primary table relate to the primary keys of the selected table. Use the drop-down list to select a primary key association.

Tables are Related by Foreign Key in Primary Table

If the primary keys of the additional table are not the same as the primary keys of the primary table, but are instead related to a different set of fields, you must set up a foreign key relation between the tables.

1. Select the associated table, and select **Associated Via Foreign Key**.
2. Use the drop-down list to choose a foreign key reference that relates the correct fields in the primary table to the primary keys in the selected table. Click **Browse** to create a reference.

Working with a Copy Policy

The OracleAS TopLink unit of work feature must be able to clone persistent objects. OracleAS TopLink supports two ways of copying objects:

- By default, an object's default constructor is called to create a copy.
- You can specify a method on the object to be used by OracleAS TopLink to perform the copy, such as `clone`.

Setting the Copy Policy

Use the **Copying** tab to specify how OracleAS TopLink copies objects. OracleAS TopLink supports the following methods:

- Using the object's default constructor to create a copy
- Specifying a method, such as `clone`

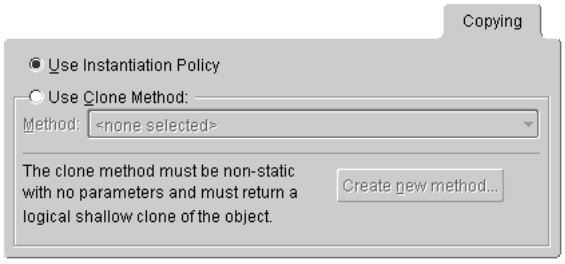
To specify a copy method:

1. Select a descriptor in the **Navigator** pane. Its properties appear in the **Editor** pane.

   If the **Copying** advanced property is not visible for the descriptor, right-click the descriptor and choose **Set Advanced Properties** > **Copying** from the pop-up menu or from the **Selected** menu.

2. Click the **Copying** tab in the **Editor** pane.

Figure 4–33 Copying Tab



3. Use this table to enter data in each field:

Field	Description
Use Instantiation Policy	Creates a new instance of the object using the descriptor's instantiation policy.
Use Clone Method	Specifies to call the `clone()` method of the object.
Method	Select the clone method from the drop-list. Click **Create New Method** to create a new method.

Working with Instantiation Policy

OracleAS TopLink supports several ways to instantiate objects:

- By default, the default constructor of the class instantiates a new instance.
- If the application requires that objects be instantiated in other ways, the instantiation method can be customized.

You can use custom Java code to override the instantiation policy. Refer to the book *Oracle Application Server TopLink Application Developer's Guide* for details.

Setting Instantiation Policy

Use the **Instantiation** tab to specify if objects are instantiated by the default constructor, a specific method, or a factory.

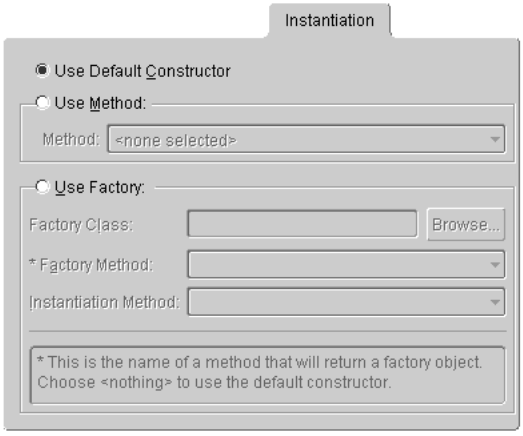
To set the instantiation policy:

1. In the **Navigator** pane, select a descriptor.

   If the Instantiation advanced property is not visible for the descriptor, right-click the descriptor and choose **Set Advanced Properties** > **Instantiation** from the pop-up menu or from the **Selected** menu.

2. Click the **Instantiation** tab.

Figure 4–34 Instantiation Tab



3. Use this table to enter data in each field:

Field	Description
Use Default Constructor	The default constructor of the class instantiates a new instance.
Use Method	Specify a **Method** to execute to create objects from the database.
Method	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	Refer to the book *Oracle Application Server TopLink Application Developer's Guide* for more information.

Field	Description
Factory Class	The class of the factory object that creates the new instances.
Factory Method	The message to be sent to obtain a factory object. Choose `<nothing>` to use the default constructor.
Instantiation Method	The method to be sent to the factory object to obtain a new instance that will be populated with data from the database.

Working with a Wrapper Policy

OracleAS TopLink allows you to 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 Enterprise JavaBeans 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` (the *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, OracleAS 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.

Use a *wrapper policy* to tell OracleAS 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, OracleAS 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 OracleAS TopLink descriptor.
- By default, no wrapper policy is used (the wrapper policy for a descriptor is null by default).
- Wrapper policies cannot be set using the OracleAS TopLink Mapping Workbench and can only be set using Java code.

> **Note:** Wrapper policies are advanced OracleAS TopLink options. Using a wrapper policy may not be compatible with some OracleAS TopLink Mapping Workbench features.

Setting the Wrapper Policy Using Java Code

The `Descriptor` class provides methods used in conjunction with the wrapper policy:

- `setWrapperPolicy(oracle.toplink.descriptors.WrapperPolicy)` can be invoked to provide a wrapper policy for the descriptor.
- `getWrapperPolicy()` returns the wrapper policy for a descriptor.

Refer to the book, *Oracle Application Server TopLink Application Developer's Guide*, for detailed information.

Working with Optimistic Locking

When using caching to provide performance benefits, you should also use a locking policy to manage database record modification in multi-user environments. Without a locking policy, it may be possible for users to see data that is no longer valid (sometimes called *stale* data) stored in the cache.

Databases typically support the following locking policies:

- Optimistic – All users have read access to the data. When a user attempts to write a change, the application checks to ensure the data has not changed since the last read. OracleAS TopLink supports multiple methods of optimistic locking.
- Pessimistic – The first user who accesses the data with the purpose of updating locks the data until completing the update. OracleAS TopLink supports pessimistic locking through `UnitOfWork` and `updateAndLockObject()`.
- No locking – The application does not verify that data is current.

Oracle recommends using optimistic locking to ensure that all users are working with valid data before committing changes. OracleAS TopLink supports multiple locking policies for optimistic locking:

- Version locking policies enforce optimistic locking by using version fields (or write lock fields) that are updated each time a record version field must be added to the table for this.
- Field locking policies do not require additional fields, but do require a `UnitOfWork` to be implemented.

Note: If a three-tier application is being built and objects are edited outside the context of a unit of work, then the write lock value is stored in the object and passed to the client. If it is only the server, then lock conflicts may be missed as clients update the same cache.

Using Version Locking Policies

For each of the following version locking policies, you must add a specific database field.

- For `VersionLockingPolicy`, add a numeric field.
- For `TimestampLockingPolicy`, add a timestamp field.

OracleAS TopLink records the version as it reads an object from a table. When the client attempts to write the object, OracleAS TopLink compares the version of the object with the version in the table record.

- If the versions are the same, the updated object is written to the table, and the version of both the table record and the object are updated.
- If the versions are different, the write is disallowed because another client must have updated the object since this client initially read it.

The two version locking policies have different ways of writing to the database:

- For `VersionLockingPolicy`, the number in the version field increments by one.
- For `TimestampLockingPolicy`, a new timestamp is inserted into the row (this policy can be configured to get the time from the server or locally).

For both policies, the values of the write lock field can be the writable mapping within the object.

If the value is stored in the identity map, then by default an attribute mapping is not required for the version field. If the application does map the field, it must make the mappings read-only to allow OracleAS TopLink to control writing the fields.

Using Field Locking Policies

The following locking policies, included in OracleAS TopLink, do not require any additional fields:

- `AllFieldsLockingPolicy`

- `ChangedFieldsLockingPolicy`
- `SelectedFieldsLockingPolicy`

All these policies compare the current values of certain mapped previous values. When using these policies, a `UnitOfWork` must be employed for updating the database. Each policy handles its field comparisons in specific way, as defined by the policy.

- Whenever an object using `AllFieldsLockingPolicy` is updated or deleted, all the fields in that table are compared in the `where` clause. If any value in that table has been changed since the object was read, the update or delete fails.

  Note: This comparison is only on a per table basis. If an update is performed on an object that is mapped to multiple tables (multiple table inheritance), then only the changed table(s) appear in the `where` clause.

- Whenever an object using `ChangedFieldsLockingPolicy` is updated, only the modified fields are compared. This allows for multiple clients to modify different parts of the same row without failure. Using this policy, a delete compares only on the primary key.
- Whenever an object using `SelectedFieldsLockingPolicy` is updated or deleted, a list of selected fields is compared in the statement. Updating these fields must be done by the application manually or through an event.

Whenever any update fails because optimistic locking has been violated, an `OptimisticLockException` is thrown. This should be handled by the application when performing any database modification The application must refresh the object and reapply its changes.

Specifying Advanced Optimistic Locking Policies

The OracleAS TopLink optimistic locking policies that "Working with Optimistic Locking" describes implement the `OptimisticLockingPolicy` interface, referenced throughout the OracleAS TopLink code. You can create more policies by implementing this interface and implementing the methods defined.

Use the **Locking** tab (see Figure 4–22) to specify locking policies for the OracleAS TopLink Mapping Workbench, or refer to the book *Oracle Application Server TopLink Application Developer's Guide* for more information.

Working with Identity Maps

OracleAS TopLink uses *identity maps* to cache objects for performance and maintain object identity. The OracleAS TopLink Mapping Workbench provides the following identity map types on the **Identity** tab (see Figure 4–19):

Table 4–2 Identity Maps

Identity Map	Description
Full identity map	Provides full caching and guaranteed identity. Caches all objects and does not remove them. This method may be memory intensive when many objects are read. Do not use on batch type operations.
Soft cache weak identity map (default with JDK 1.2, available since JDK 1.2)	Similar to the weak identity map except that it maintains a most-frequently-used sub-cache. The size of the sub-cache is proportional to the size of the identity map as specified by descriptor's `setIdentityMapSize()` method. The sub-cache uses soft references to ensure that these objects are garbage-collected only if the system is low on memory.
Hard cache weak identity map (available since JDK 1.2)	Identical to the soft cache weak identity map except that it uses hard references in the sub-cache. Use this identity map if soft references do not behave properly on your platform.
Weak identity map (available since JDK 1.2)	Similar to the full identity map except that the map holds the objects using weak references. This method allows full garbage collection and provides full caching and guaranteed identity.
Cache identity map	Furnishes caching and identity, but does not guarantee identity. A cache identity map maintains a fixed number of objects specified by the application. Objects are removed from the cache on a least-recently-used basis. This method allows object identity for the most commonly used objects.
No identity map	Does not preserve object identity and does not cache objects.

Identity Map Size

The default identity map size is **100**.

- For the *cache* identity map policy, the size indicates the maximum number of objects stored in the identity map.
- For the *full* identify map policy, the size determines the starting size of the map.
- For the *soft/hard cache weak identity map*, the most recently used sub-cache is proportional to the size.

Design Guidelines

Use the following guidelines when employing an identity map:

- If using a Java 2-compatible Virtual Machine (VM), objects with a long lifespan, and object identity are important, use a `SoftCacheWeakIdentityMap or HardCacheWeakIdentityMap` policy.
- If using a Java 2-compatible VM, and object identity is important but caching is not, use a `WeakIdentityMap` policy.
- If an object has a long lifespan or requires frequent access, or is important, use a `FullIdentityMap` policy.
- If an object has a short lifespan or requires frequent access, and identity is not important, use a `CacheIdentityMap` policy.
- If objects are discarded immediately after being read from the database, such as in a batch operation, use a `NoIdentityMap` policy.

> **Note:** The `NoIdentityMap` does not preserve object identity.

Using Object Identity

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

Maintaining object identity is extremely important when the application's object model contains circular references between objects. You must ensure that two are referencing each other directly, rather than copies of each other. Object identity is important when multiple parts of the application may be modifying the same object simultaneously.

Turn identity off when object identity is not important (for example, for read-only objects).

Caching Objects

Identity maps maintain client-side object caches, which increases application performance by minimizing the number of database reads.

When the cache fills up, OracleAS TopLink cleans up the cache based on the identity map policy.

Working with Query Keys

Use a *direct query key* as an alias for a field name. Query keys allow OracleAS TopLink expressions to refer to a field using Java attribute names (such as `firstName`), rather than DBMS-specific names (such as `F_NAME`).

Use query keys to:

- Enhance code readability when defining OracleAS TopLink expressions.
- Increase portability by making code independent of the database schema. If you rename a field, the query key could be redefined without changing any code that references it.
- Interface descriptors define only common query keys shared by implementors; the fields aliased could have different names in the implementor's tables.

Automatically-generating Query Keys

OracleAS TopLink automatically defines direct query keys for all direct mappings and has a special query key type for each mapping. Typically, use query keys to access fields that do not have direct mappings, such as the *version* field used for optimistic locking or the *type* field used for inheritance.

OracleAS TopLink displays automatically generated queries in the **Query Keys** tab of the **Editor** pane (see Figure 4–15). You cannot change these keys.

Example 4–3 Automatically Generated Query Key

For example, consider the `Employee` class in the OracleAS TopLink tutorial: When you define a direct-to-field mapping from the `Employee` class (attribute `firstName`) to the EMPLOYEE table (field F_NAME) you get a query key for free — it is automatically generated.

The following code example illustrates using an automatically generated query key within the OracleAS TopLink expression framework.

```
Vector employees = session.readAllObjects(Employee.class, new
ExpressionBuilder().get("firstName").equal("Bob"));
```

Using Query Keys in Interface Descriptors

Interface descriptors are defined only with query keys that are shared among their implementors. In the descriptor for the interface, only the name of the query key is specified.

In each implementor descriptor, the key must be defined, and with appropriate field, from one of the implementor descriptor's tables.

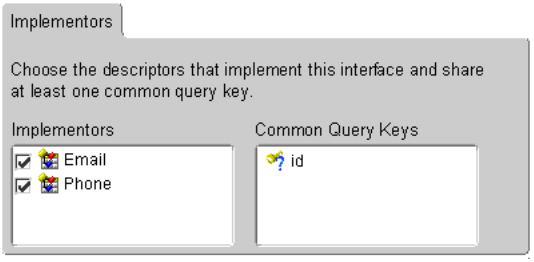
Doing this ensures that a single query key can be used to specify foreign key information from a descriptor that contains a mapping to the interface, even if the field names differ.

Consider an `Employee` that contains a contact, of type `Contact`. `Contact` is an interface with two implementors: `Phone` and `EmailAddress`. Each class has two attributes. The following figure illustrates the generated keys:

Figure 4–35 Auto-generated Query Keys for Phone and Email

Note: 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, indicate that the `id` query key must be defined for each of the implementors, as Figure 4–36 illustrates.

Figure 4–36 Contact interface Descriptor with Common Query Key id



Note: If either of the implementor classes did not have the `id` query key defined, that descriptor would be flagged 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. For example, you can now create a 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.

See "Working with Interfaces" on page 4-45 and "Working with Relationship Mappings" on page 6-2 for more information.

Relationship Query Keys

OracleAS TopLink supports query keys for relationship mappings and automatically defines them for all relationship mappings. You can use these keys to join across a relationship. One-to-one query keys define a joining relationship and are accessed through the `get()` method in expressions.

One-to-many and many-to-many query keys define a distinct join across a collection relationship and are accessed through the `anyOf()` method in expressions. You can also define relationship query keys manually if `mapping` does not exist for the relationship. The relationship defined by the query key is data-level expressions.

Example 4–4 One-to-one Query Key

The following code example illustrates using a one-to-one query key within the OracleAS TopLink expression framework

```
ExpressionBuilder employee = new ExpressionBuilder();
Vector employees = session.readAllObjects(Employee.class,
employee.get("address").get("city").equal("Ottawa"));
```

Defining Relationship Query Keys by Amending a Descriptor

Relationship query keys are not supported directly in the OracleAS TopLink Mapping Workbench. To define a relationship query key, you must specify and write an amendment method. Register query keys by sending the `addQueryKey()` message.

Example 4–5 Defining One-to-one Query Key Example

The following code example illustrates how to define a one-to-one query key.

```
// Static amendment method in Address class, addresses do not know their owners
```

```
in the object-model, however you can still query on their owner if a
user-defined query key is defined
public static void addToDescriptor(Descriptor descriptor)
{
OneToOneQueryKey ownerQueryKey = new OneToOneQueryKey();
ownerQueryKey.setName("owner");
ownerQueryKey.setReferenceClass(Employee.class);
ExpressionBuilder builder = new ExpressionBuilder();
ownerQueryKey.setJoinCriteria(builder.getField("EMPLOYEE.ADDRESS_
ID").equal(builder.getParameter("ADDRESS.ADDRESS_ID")));
descriptor.addQueryKey(ownerQueryKey);
}
```

Working with Events

Use the event manager to specify specific events to occur whenever OracleAS TopLink performs a read, update, delete, or insert on the database.

> **Note:** OracleAS TopLink uses the Java event model.

Applications can receive descriptor events in the following ways:

- Implement the `DescriptorEventListener` interface
- Subclass the `DescriptorEventAdapter` adapter class
- Register an event method with a descriptor

Objects that implement the `DescriptorEventListener` interface can be registered with the descriptor event manager to be notified when any event occurs for that descriptor.

Alternately, you may wish to use the `DescriptorEventAdapter` class if your application does not require all the methods defined in the interface. The `DescriptorEventAdapter` implements the `DescriptorEventListener` interface and defines an empty method for each method in the interface. To use the adapter, first subclass it, then register your new object with the descriptor event manager.

You can use descriptor events in many ways, including:

- Synchronizing the persistent objects with other systems, services and frameworks
- Maintaining non-persistent attributes of which OracleAS TopLink is not aware

- Notifying other parts of the application when the persistent state of objects is changed
- Performing complex mappings or optimizations not directly supported by OracleAS TopLink mappings

Use the descriptor's **Event** tab (see Figure 4–18) to specify events for a descriptor.

Example 4–6 Event Example

To invoke a method called `postBuild()` for an `Employee` object, the `postBuild()` method must be implemented in the `Employee` class. This method must also accept one parameter that is an instance of `DescriptorEvent` fully qualified with a package name.

Registering an Event with a Descriptor

A persistent class can register a public method as an event method. A descriptor calls the event method when a particular database operation occurs.

Event methods:

- Must be public so that OracleAS TopLink can call them
- Must return void
- Must take a `DescriptorEvent` as a parameter

Example 4–7 Registering an Event

The following code illustrates an event method definition:

```
public void myEventHandler(DescriptorEvent event);
```

Supported Events

Events the `DescriptorEventManager` supports include:

Post-X Methods:

- Post-Build — occurs after an object is built from the database.
- Post-Clone — occurs after an object has been cloned into a unit of work.
- Post-Merge — occurs after an object has been merged from a unit of work.
- Post-Refresh — occurs after an object is refreshed from the database.

Updating Methods:

- Pre-Update — occurs before an object is updated in the database. This may be called in a unit of work even if the object has no changes and does not require an update.
- About-to-Update — occurs when an object's row is updated in the database. This is called only if the object has changes in the unit of work.
- Post-Update — occurs after an object is updated in the database. This may be called in a unit of work even if the object has no changes and does not require an update.

Inserting Methods:

- Pre-Insert — occurs before an object is inserted in the database.
- About-to-Insert — occurs when an object's row is inserted in the database.
- Post-Insert — occurs after an object is inserted in the database.

Writing Methods:

- Pre-Write — occurs before an object is inserted or updated in the database. This occurs before Pre-Insert/Update.
- Post-Write — occurs after an object is inserted or updated in the database. This occurs after Pre-Insert/Update.

Deleting Methods:

- Pre-Delete — occurs before an object is deleted from the database.
- Post-Delete — occurs after an object is deleted from the database.

Working with Finders

In OracleAS TopLink, use named queries to represent SQL or EJB QL finders to use in database accesses. You can create these finders within the OracleAS TopLink Mapping Workbench.

When you create a finder for an EJB, the OracleAS TopLink Mapping Workbench creates a named query and populates the descriptor alias with information from the `ejb-jar.xml` file.

Reserved finders are valid for projects with CMP persistence.

Working with Object-relational Descriptors

The object-relational paradigm extends traditional relational databases with object-oriented functionality. 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 mainly concerned with extending the database data structures and the SQL language (SQL 3).

The new features include:

- Structures or Object-types can be defined and stored on the database
- Collections/Arrays can be defined and stored on the database
- Structures/Object-types can have system-generated ObjectIDs
- Structures/Object-types can reference other structures through References or aggregation
- SQL 3, an extension to the SQL language that supports querying and manipulating the new object-types

Coinciding with object-relational changes, most database vendors are also extending their server architectures to support features such as:

- Embedded server-side Java VMs
- Java stored procedures
- CORBA, HTML and EJB support in the database

This section describes how the object-relational features affect OracleAS TopLink descriptors and mappings. The server architecture changes are discussed in the book *Oracle Application Server TopLink Application Developer's Guide*.

Effect on OracleAS TopLink

Object-relational databases introduce several new features that allow more complex data to be stored and accessed. One advantage of object-relational databases is that the differences between the object model and data model can be reduced to the point that the two are almost identical. Although this makes the object-relational mapping process easier, it does not reduce the need for a persistence framework such as OracleAS TopLink. Although the JDBC standard has been improved to take advantage of object-relational features in JDBC 2.0, it still remains a low-level database interface. On top of JDBC, frameworks such as OracleAS TopLink can provide applications with much more sophisticated functionality, including units of work, identity maps, expressions, querying, complex mappings, three-tier and enterprise application support.

OracleAS TopLink provides object-relational support through a new type of descriptor object and several new types of mappings. See Chapter 7, "Understanding Object-Relational Mappings" for more information.

Databases OracleAS TopLink Supports

OracleAS TopLink supports any JDBC 2.0 driver that complies with JDBC's 2.0 object-relational extensions. Contact your database and JDBC vendor to determine which object-relational features they support.

Defining Object-Relational Descriptors

The OracleAS TopLink Mapping Workbench does not currently support the object-relational descriptor and mappings. Support will be added to the OracleAS TopLink Mapping Workbench in future releases.

You should be able to import most of the simple object-relational table structures into OracleAS TopLink. In addition, you can define the standard non-object-relational descriptor properties and mappings. You can use amendment methods to add any object-relational mappings and features to the descriptors.

Working with Mappings

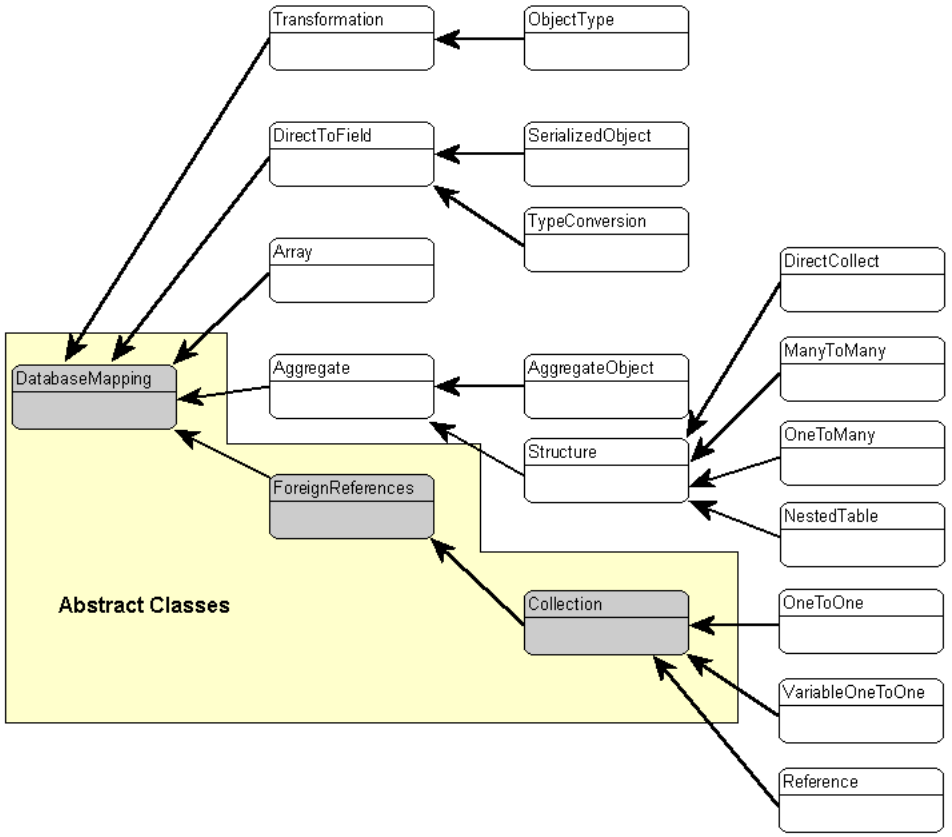
In OracleAS TopLink, mappings define how an object's attributes are represented in the database.

- Direct mappings define how a persistent object refers to objects that do not have descriptors (for example, the JDK classes and primitives). See Chapter 5, "Understanding Direct Mappings" for details.

- Relationship mappings define how a persistent object refers to other persistent objects. See Chapter 6, "Understanding Relationship Mappings", and Chapter 7, "Understanding Object-Relational Mappings", for details.

All the mapping classes are derived from the `DatabaseMapping` class, as Figure 4–37 illustrates.

Figure 4–37 Mapping Classes Hierarchy

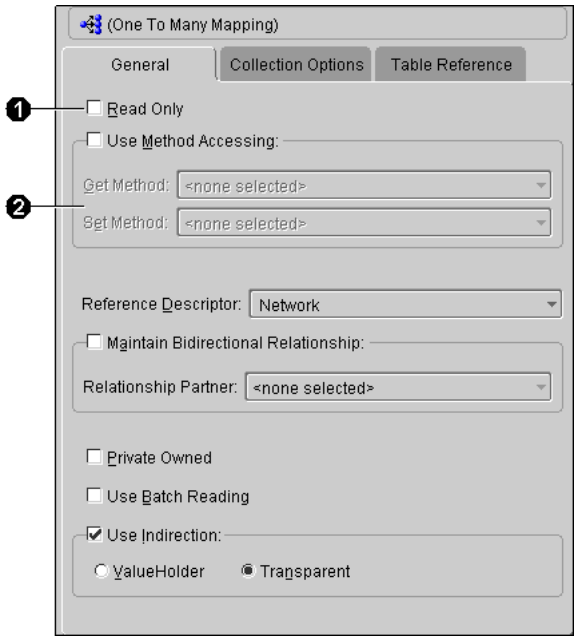


Working with Common Mapping Properties

OracleAS TopLink associates each mapping with the attribute whose persistence it describes. To create a mapping in the OracleAS TopLink Mapping Workbench, select the attribute to map from the **Navigator** pane and then click the appropriate button on the mapping toolbar (see "Mapping Toolbar" on page 1-7).

Use the mapping's **Editor** pane to enter specific information for the mapping. Some mappings require more information that others and have multiple tabs in the **Editor** pane.

Figure 4–38 Sample Properties for a Mapping



Mapping properties called out in Figure 4–38:

1. Specify if read-only.
2. Specify access method.

Specifying Direct Access and Method Access

By default, OracleAS TopLink uses direct access to access public attributes. Alternatively, you can use accessor 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.

Starting with JDK 1.2, the Java Core Reflection API provides a means to suppress default Java language access control checks when using reflection. OracleAS TopLink uses reflection to access the application's persistent objects. This means that if you are using a VM that supports the API, then OracleAS TopLink can access an attribute directly, regardless of its declared visibility.

Note: Private variable access under JDK 1.2 requires you to enable the security setting. Consult the JDK documentation for more information.

Oracle recommends using direct access whenever possible to improve performance and avoid executing any application-specific behavior while building objects.

Setting the Access Type

Use the **General** tab of the mapping **Editor** pane (see Figure 4–38) to set the access type as direct or method-based

To change the default access type used by all new mappings, use the **Defaults** tab on the project **Editor** pane. See "Working with Default Properties" on page 2-10 for more information.

Note: If you change the access default, existing mappings retain their current access settings, but new mappings will be created with the new default.

Specifying Read-Only Settings

Use the **Read Only** check-box on the **General** tab of the mapping **Editor** pane (see Figure 4–38) to set a mapping to be read only. OracleAS TopLink will not consider attributes associated with read-only mappings during update and delete operations.

Because these operations are not actually performed for the mapping, any processes dependent on these operations (such as custom SQL or descriptor events) are not called for read-only. The attributes are still used for read operations.

Note: The primary key mappings must 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 and the class indicator is part of the primary key.

Defaulting Null Values

Direct mappings include a `nullValue` attribute. Use this attribute to convert database null values to application-specific values (if the application does not allow null values). This applies when typed as primitives. The null value must be set to the desired value, not the database value.

Null values translate in two directions: from null values read from the database to the specified value, and from the specified value back to null when writing or querying. You can also use OracleAS TopLink to set global default null values on a per-class basis. For more information, refer to the book *Oracle Application Server TopLink Application Developer's Guide*.

Click the **Use Default Value when Database Field is Null** option on the **General** tab (see Figure 4–38) and the **Type** and **Value** drop-down lists to specify the null value.

Note: You must specify the **Type** and **Value** in the mapping form.

Maintaining Bidirectional Relationships

Select the **Maintain Bidirectional Relationship Only** option on the **General** tab of the mapping **Editor** pane (see Figure 4–38) to maintain a bidirectional relationship for a one-to-one or one-to-many mapping. You can also specify the relationship partner.

Specifying Field Names and Multiple Tables

When defining mappings in code, OracleAS TopLink assumes all mappings are in the first table specified by the descriptor's `setTableName()` or `addTableName()` method. If the persistent class stores information in multiple tables, any messages sent that require field names should be implemented to pass fully qualified names (that include the table name). Use the following syntax to fully qualify a field:

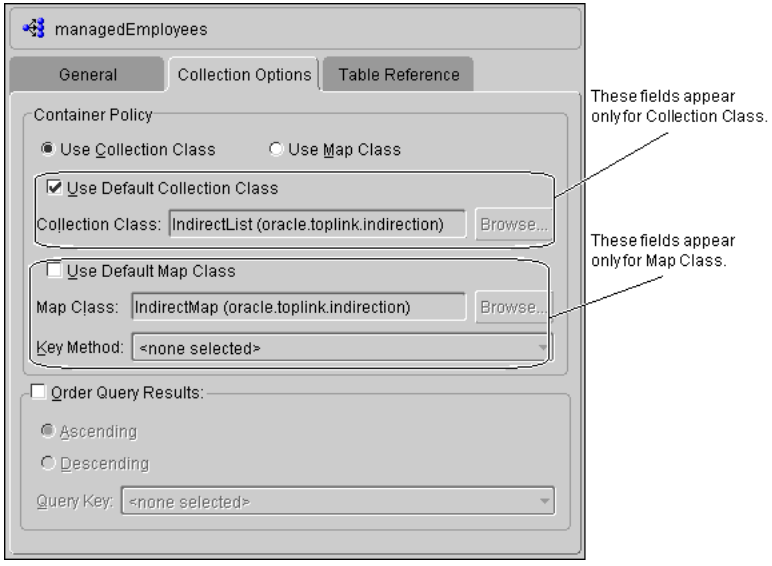
```
someMessage("tablename.fieldname");
```

Specifying Collection Properties

Some relationship mapping types (direct collection, one-to-many, and many-to-many) contain a **Collection Options** tab to allow you to specify collection options.

OracleAS TopLink can populate a collection in ascending or descending order, upon your specification. Query keys are automatically created for and with the same name as all attributes mapped as direct-to-field, type conversion, object type, and serialized object mappings.

Figure 4–39 Collection Options



Use this table to enter data in each field:

Field	Description
Container Policy	
Collection or Map Class	Select the collection or map class to use for this collection mapping.
Use Default Container Class	If **Container Policy** = **Use Collection Class**, select the default **Collection Class** for the mapping.
Use Default Map Class	If **Container Policy** = **Use Map Class**, select the default **Map Class** and **Key Method** for the mapping.
Order Query Results	Specify how the collection results are sorted for queries.

Specifying Mapping information in ejb-jar.xml File

For 2.0 CMP projects, the `ejb-jar.xml` files stores information on bean-to-bean relationships (mappings) in the `<relationship>` element. By updating this information in the `ejb-jar.xml`, the OracleAS TopLink Mapping Workbench creates new mappings. You can then update the mapping information (such as reference tables).

If the information does not exist in the `ejb-jar.xml` file, you can build the mappings in the OracleAS TopLink Mapping Workbench, then write the information to the file. See "Writing to the ejb-jar.xml File" on page 2-19 for more information.

5

Understanding Direct Mappings

In Oracle Application Server TopLink, direct mappings define how a persistent object refers to objects without descriptors, such as the JDK classes and primitives.

You can create the following direct mappings in OracleAS TopLink:

- Direct-to-field mappings – Map a Java attribute directly to a database field (see "Working with Direct-to-Field Mappings" on page 5-2).
- Type conversion mappings – Map Java values with simple type conversions, such as from *character* to *string* (see "Working with Type Conversion Mappings" on page 5-5).
- Object type mappings – Use an association to map values to the database (see "Working with Object Type Mappings" on page 5-6).
- Serialized object mappings – Map serializable objects, such as multimedia objects, to database BLOB fields (see "Working with Serialized Object Mappings" on page 5-9).
- Transformation mappings – Allow you to create custom mappings where one or more fields can be used to create the object be stored in the attribute (see "Working with Transformation Mappings" on page 5-10).

Working with Direct Mappings

There are two basic ways of storing object attributes directly in a database 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.

OracleAS TopLink furnishes the following classes of direct mappings:

- Direct-to-field
- Type conversion
- Object type
- Transformation
- Serialized object

If the application's objects contain attributes that cannot be represented as direct-to-field, type conversion, or object-type mappings, then the application must provide transformation routines for saving the attributes.

If a direct-to-field mapping cannot be used to perform the desired conversion, try type conversion and object type mappings before attempting to define a custom transformation mapping.

Working with Direct-to-Field Mappings

Direct-to-field mappings map a Java attribute directly to a value database column. When the application writes a Java instance to the database, it stores the value of the attribute in a field of the table column. OracleAS TopLink supports the following types:

- **`java.lang`**: `Boolean`, `Float`, `Integer`, `String`, `Double`, `Long`, `Short`, `Byte`, `Byte[ ]`, `Character`, `Character[ ]`; all of the primitives associated with these classes
- **`java.math`**: `BigInteger`, `BigDecimal`
- **`java.sql`**: `Date`, `Time`, `Timestamp`
- **`java.util`**: `Date`, `Calendar`

While reading, direct-to-field mappings perform some simple one-data conversions, as described in Table 5–1. You must use other direct mappings for two-way or more complete conversions.

Table 5–1 Type Conversions Provided by Direct-to-Field Mappings

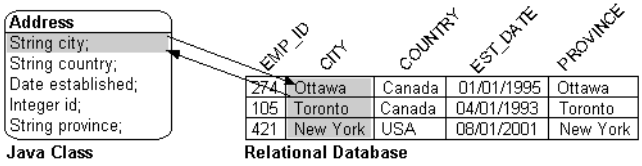
Java type	Database type
Integer, Float, Double, Byte, Short, BigDecimal, BigInteger, int, float, double, byte, short	NUMBER, NUMERIC, DECIMAL, FLOAT, DOUBLE, INT, SMALLINT, BIT, BOOLEAN
Boolean, boolean	BOOLEAN, BIT, SMALLINT, NUMBER, NUMERIC, DECIMAL, FLOAT, DOUBLE, INT
String	VARCHAR, CHAR, VARCHAR2, CLOB, TEXT, LONG, LONG VARCHAR, MEMO
nSting	NVARCHAR2 (applies to Oracle9)
nClob	NCLOB (applies to Oracle9)
Character, char	CHAR
nChararcter	NCHAR (applies to Oracle9)
byte[ ]	BLOB, LONG RAW, IMAGE, RAW, VARBINARY, BINARY, LONG VARBINARY
Time	TIME
sql.Date	DATE (only applies to DB2)
Timestamp, util.Date, Calendar	TIMESTAMP (only applies to DB2)
sql.Date, Time, Timestamp, util.Date, Calendar	DATE, DATETIME (applies to Oracle, Sybase, SQL Server)

Direct-to-field mappings also allow you to specify a **null** value. This may be required if primitive types are used in the object, and the database field allows null values.

Example 5–1 Direct-to-Field Mapping Example

Figure 5–1 illustrates a direct-to-field mapping between the Java attribute `city` and the relational database column `CITY`. Similarly, direct-to-field mappings could be defined from `country` to `COUNTRY`, `id` to `ADDRESS_ID`, `established` to `EST_DATE`, and `province` to `PROVINCE`.

Figure 5–1 Direct-to-Field Mapping



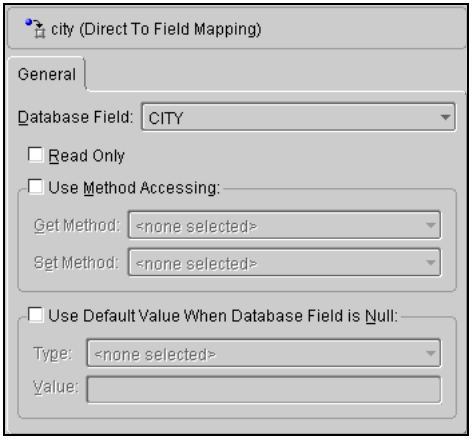
Creating Direct-to-Field Mappings

Use this procedure to create a basic direct-to-field mapping to map a Java attribute directly to a value in a database.

To create a direct-to-field mapping:

1. Select the attribute to be mapped from the **Navigator** pane.
2. Click the **Direct to Field Mapping** button on the mapping toolbar.
3. From the **Database Field** drop-down list on the **General** tab on the **Editor** pane, choose the appropriate database field.
4. Select the **Use Default Value When Database Field is Null** option to specify a default **Type** and **Value** to use if the database field is null.

Figure 5–2 Direct-to-field Mapping Properties



You can also specify:

- Read-only attributes – See "Specifying Read-Only Settings" on page 4-70
- Access methods – See "Specifying Direct Access and Method Access" on page 4-70
- Null values – See "Defaulting Null Values" on page 4-71

Working with Type Conversion Mappings

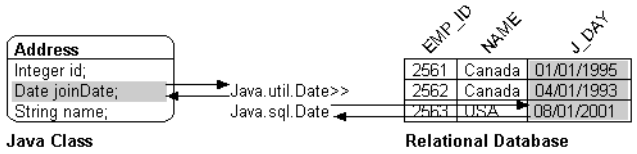
Type conversion mappings explicitly map a database type to a Java type. For example, a `Number` in the database can be mapped to a `String` in Java, or a `java.util.Date` in Java can be mapped to a `java.sql.Date` in the database.

For Oracle9 databases, OracleAS TopLink supports NCHAR, NAVRCHAR2, and NCLOB database types. Use the `Ncharacter`, `NString`, and `NClob` target types, respectively.

Example 5–2 Type Conversion Mapping Example

Figure 5–3 illustrates a type conversion mapping. 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 5–3 Type Conversion Mappings



Creating Type Conversion Mappings

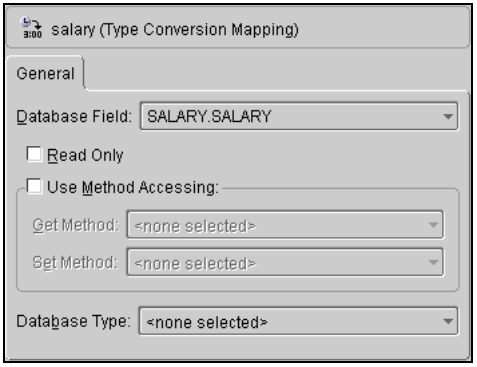
Use this procedure to create a type conversion mapping.

To create a type conversion mapping:

1. Select the attribute to be mapped from the **Navigator** pane.
2. Click the **Type Conversion Mapping** button on the mapping toolbar.

3. From the **Database field** and **Database type** drop-down lists on the **General** tab in the **Editor** pane, choose the appropriate database field and database type.

Figure 5–4 Type Conversion Mapping Properties



You can also specify:

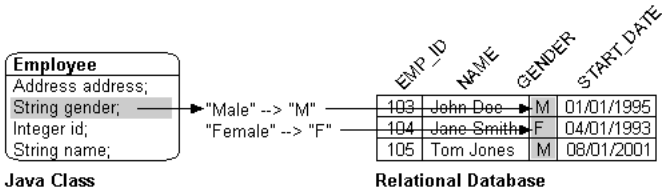
- Read-only attributes – See "Specifying Read-Only Settings" on page 4-70
- Access methods – See "Specifying Direct Access and Method Access" on page 4-70

Working with Object Type Mappings

Object type mappings match a fixed number of database values to Java objects. Use these mappings when the values in the database differ from those in Java. Object types mappings are similar to direct-to-field mappings in all other respects.

The following figure illustrates an object type mapping between the `Employee` attribute `gender` and the relational database column `GENDER`. If the `gender` value in the Java class = `Male`, the system stores it in the `GENDER` database field as `M`; `Female` is stored as `F`.

Figure 5–5 Object Type Mappings



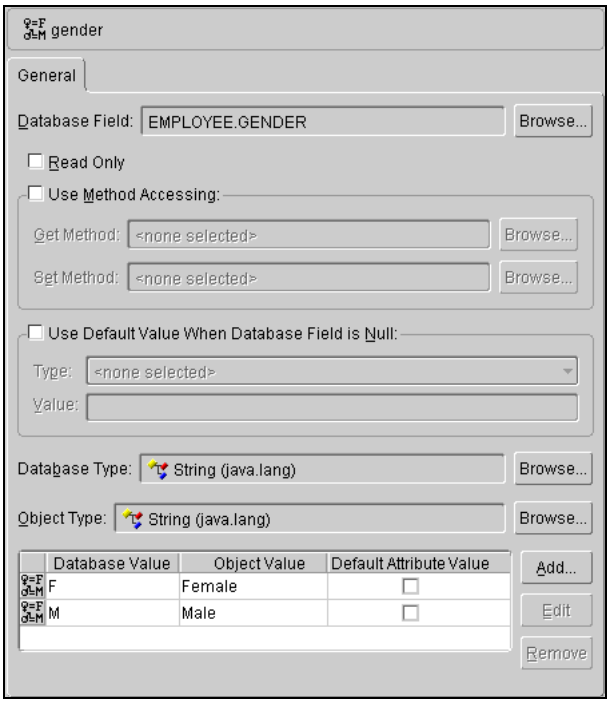
Creating Object Type Mappings

Use this procedure to create an object type mapping between an attribute and a database column.

To create a basic object type mapping:

1. In the **Navigator** pane, select the attribute to be mapped.
2. Click the **Object-Type Mapping** button on the mapping toolbar. The Object type mapping tab appears in the **Editor** pane.

Figure 5–6 Object Type Mapping General Properties



3. Choose the appropriate database field in the **Database Field** drop-down list.
4. Select **Use Default Value When Database Field is Null** to specify a default **Type** and **Value** to use if the database field is null.
5. Set the database type from the **Database Type** drop-down list and the Java type from the **Object type** drop-down list.
6. Click on **Add** to add **Database Value** and **Object Value** pairs to the table. Select the **Default Attribute Value** option for the value to use as the default.

   To remove a database value, select the value and click **Remove**.

You can also specify:

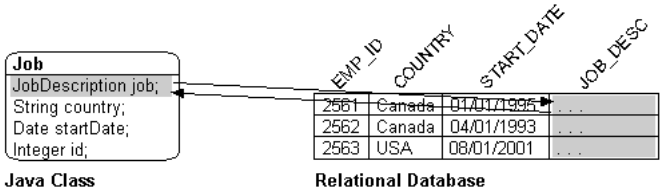
- Read-only attributes – See "Specifying Read-Only Settings" on page 4-70
- Access methods – See "Specifying Direct Access and Method Access" on page 4-70

Working with Serialized Object Mappings

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

As with direct-to-field mappings, serialized object mappings require you to specify an attribute and field, as shown in the following illustration.

Figure 5–7 Serialized Object Mappings



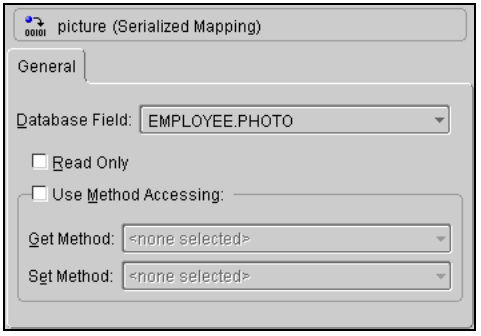
Creating Serialized Object Mappings

Use this procedure to create serialized object mappings.

To create a serialized object mapping:

1. In the **Navigator** pane, select the attribute to be mapped.
2. Click the **Serialized Mapping** button on the mapping toolbar.

Figure 5–8 Serialized Object Mapping Properties



3. Choose the appropriate database field in the **Database Field** drop-down list.

You can also specify:

- Read-only attributes – See "Specifying Read-Only Settings" on page 4-70
- Access methods – See "Specifying Direct Access and Method Access" on page 4-70

Working with Transformation Mappings

Use transformation mappings for specialized translations between how a value is represented in Java and in the database.

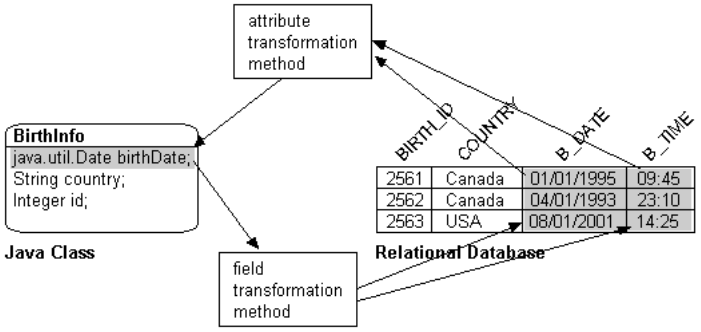
Tip: Use transformation mappings only when mapping multiple fields into a single attribute. Because of the complexity of transformation mappings, it is often easier to perform the transformation with `get/set` methods of a direct-to-field mapping.

Often, a transformation mapping is appropriate when values from multiple fields are used to create an object. This type of mapping requires that you provide an *attribute transformation method* that is invoked when reading the object from the database. This method must have at least one parameter that is an instance of `DatabaseRow`. In your attribute transformation method, you can send the `get()` message to the `DatabaseRow` to get the value in a specific column. Your attribute transformation method can specify a second parameter, when it is an instance of `Session`. The `Session` performs queries on the database to get additional values needed in the transformation. The method should *return* the value to be stored in the attribute.

Transformation mappings also require a *field transformation method* for each field, to be written to the database when the object is saved. The transformation methods are specified in a dictionary associating each field with a method. The method returns the value to be stored in that field.

Figure 5–9 illustrates a transformation mapping. The values from the B_DATE and B_TIME fields are used to create a `java.util.Date` to be stored in the `birthDate` attribute.

Figure 5–9 Transformation Mappings



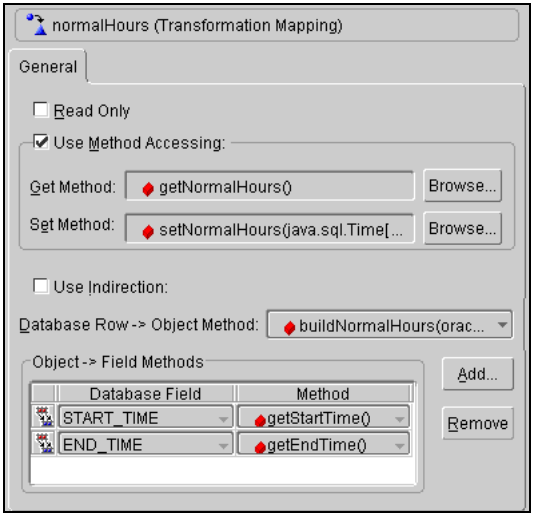
Creating Transformation Mappings

Use this procedure to create transformation mappings in the OracleAS TopLink Mapping Workbench.

To create a transformation mapping:

1. In the **Navigator** pane select the attribute to be mapped.
2. Click the **Transformation Mapping** button from the **Mapping** toolbar.

Figure 5–10 Transformation Mapping Tab



3. Use the **Database Row --> Object Method** drop-down list to select a method to convert the database row into an object.

   Note: The method must have the parameter `(DatabaseRow)` or parameters `(DatabaseRow, Session)`.

4. Click **Add** to add field transformation methods to the descriptor.

   To remove a transformation method, select the method and click **Remove**.

5. Select the **Use indirection** option to specify if the creation of the target object requires extensive computational resources. If selected, OracleAS TopLink uses indirection objects. See "Working with Indirection" on page 6-5 for more information.

6. After specifying the details of the mapping, create the attribute field transformation methods in the associated Java class (see Example 5–3).

You can also specify:

- Read-only attributes – See "Specifying Read-Only Settings" on page 4-70
- Access methods – See "Specifying Direct Access and Method Access" on page 4-70

Example 5–3 Transformation Mapping Code Example

The following code example illustrates the methods required for a transformation mapping:

```
// Get method for the normalHours attribute since method access indicated
access public Time[] getNormalHours()
{
    return normalHours;
}
// Set method for the normalHours attribute since method access indicated
access public void setNormalHours(Time[] theNormalHours)
{
    normalHours = theNormalHours;
}
// Create attribute transformation method to read from the database row
//** Builds the normalHours Vector. IMPORTANT: This method builds the value but
does not set it. The mapping will set it using method or direct access as
defined in the descriptor. */
public Time[] getNormalHoursFromRow(DatabaseRow row)
```

```
{
    Time[] hours = new Time[2];
    hours[0] = (Time)row.get("START_TIME");
    hours[1] = (Time)row.get("END_TIME");
    return hours;
}
// Define a field transformation method to write out the start time. Return the
first element of the normalHours attribute.
public java.sql.Time getStartTime()
{
    return getNormalHours()[0];
}
// Define a field transformation method to write out the end time. Return the
last element of the normalHours attribute.
public java.sql.Time getEndTime()
{
    return getNormalHours()[1];
}
```

Specifying Advanced Features Available by Amending the Descriptor

In OracleAS TopLink, transformation mappings do not require you to specify an attribute.

A field can be mapped from a computed value that does not map to a logical attribute. This, in effect, constitutes a write-only mapping. In the OracleAS TopLink Mapping Workbench, all mappings are associated with an attribute before any other information can be specified. Therefore, to use a write-only mapping, you must build it by amending the descriptor. The mapping itself has no attribute name, `get` and `set` methods, or attribute method. In your amendment method, create an instance of `TransformationMapping` and send `addFieldTransformation()` message for each field to be written.

Example 5–4 Descriptor Amendment Examples

The following code example illustrates creating a write-only transformation mapping and adding it to the descriptor.

```
public static void addToDescriptor(Descriptor descriptor) {
// Create a Transformation mapping and add it to the descriptor.
TransformationMapping transMapping = new
transMapping.addFieldTransformation("WRITE_DATE",
"descriptor.addMapping(transMapping);
}
```

The following example illustrates how to create a one-way transformation mapping by using the inheritance indicator field of the primary key. Map the class as normal, including the other part of the primary key, and the inheritance through the type field.

Note: The OracleAS TopLink Mapping Workbench displays a neediness error, because the class indicator field part of the primary key is not mapped. Use the following code to create an amendment method to map the indicator field.

Create an amendment method for the class:

```
public void addToDescriptor(Descriptor descriptor) {
    TransformationMapping keyMapping = new TransformationMapping();
    keyMapping.addFieldTranslation("PROJECT.PROJ_TYPE",
        "getType");descriptor.addMapping(keyMapping);}
```

Define the `getType` method on the class to return its type value:

```
Project>>public abstract String getType();
LargeProject>>public String getType() { return "L"; }
SmallProject>>public String getType() { return "S"; }
```

Refer to "Amending Descriptors After Loading" on page 4-27 for more information.

6

Understanding Relationship Mappings

Relational mappings define how persistent objects reference other persistent objects. Oracle Application Server TopLink furnishes the following relationship mappings:

- Direct collection mappings — Map Java collections of objects that do not have descriptors (see "Working with Direct Collection Mappings" on page 6-29).
- Aggregate object mappings — Strict one-to-one mappings that require both objects to exist in the same database row (see "Working with Aggregate Object Mappings" on page 6-15).
- One-to-one mappings — Map a reference to another persistent Java object to the database (see "Working with One-to-One Mappings" on page 6-20).
- Variable one-to-one mappings — Map a reference to an interface to the database (see "Working with Variable One-to-One Mappings" on page 6-24).
- One-to-many mappings — Map Java collections of persistent objects to the database (see "Working with One-to-Many Mappings" on page 6-32).
- Aggregate collection mappings — Map Java collections of persistent objects to the database (see "Working with Aggregate Collection Mappings" on page 6-31).
- Many-to-many mappings — Use an association table to map Java collections of persistent objects to the database (see "Working with Many-to-Many Mappings" on page 6-35).

Oracle Application Server TopLink also provides object-relational relationship mappings (see Chapter 5, "Understanding Direct Mappings", and Chapter 7, "Understanding Object-Relational Mappings").

All OracleAS TopLink relationship mappings are unidirectional, from the class being described (the *source* class) to the class with which it is associated (the *target*

class). The target class does not have a reference to the source class in a unidirectional relationship.

To implement a bidirectional relationship (classes that reference each other), use two unidirectional mappings with the sources and targets reversed.

Working with Relationship Mappings

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.

Specifying Private or Independent Relationships

In OracleAS 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 are public objects that exist independently. Destroying one object does not necessarily imply the destruction of the other.

Aggregate object mappings are private by default, because the target object shares the same row as the source object. One-to-one, one-to-many, and many-to-many mappings can be independent or private, depending upon the application. Normally, many-to-many mappings are independent by definition; however, because a many-to-many mapping can be used to implement a logical one-to-many without requiring a back reference in the target to the source, OracleAS TopLink allows many-to-many mappings to be private as well as independent.

Tip: OracleAS TopLink automatically supports 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 because are also removed. Be aware of this when creating new systems, since it may affect both the behavior and the performance of your application.

Working with Foreign Keys

OracleAS TopLink uses *references* to maintain foreign key information. OracleAS TopLink defines the reference as a property of the table containing the foreign key. This may or may not correspond to an actual constraint that exists on the database.

If you import tables from the database, OracleAS TopLink creates references that correspond to existing database constraints (if the driver supports this). You can also define any number of references in the OracleAS TopLink Mapping Workbench without creating similar constraints on the database.

OracleAS TopLink uses these references when defining relationship mappings and descriptors' multiple table associations.

Understanding Foreign Keys

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

Relationship mappings use foreign keys to find information in the database so that the target object(s) can be instantiated. For example, if every `Employee` has an attribute address that contains an instance of `Address` (which has its own descriptor and table) then, the one-to-one mapping for the address attribute would specify foreign key information to find an address for a particular `Employee`.

OracleAS TopLink classifies foreign keys into two categories in mappings — **foreign keys** and **target foreign keys**:

- In a *foreign key,* the key is found in the table associated with the mapping's own descriptor. In the previous example, a foreign key to `ADDRESS` would be in the `EMPLOYEE` table.
- In a *target foreign key*, the reference is from the target object's table back to the key from the mapping's descriptor's table. In the previous example, the `ADDRESS` table would have a foreign key to `EMPLOYEE`.

> **Caution:** Make sure you fully understand the distinction between *foreign key* and *target foreign key* before defining a mapping.

Specifying Foreign Keys

If you import tables from the database, OracleAS TopLink creates references that correspond to existing database constraints (if supported by the driver). You can also define references in OracleAS TopLink without creating similar constraints on the database.

To display existing references for a table, use the **References** tab (see Figure 3–8). References that contain the **On Database** option will create a constraint that corresponds to the references.

> **Note:** Your database driver must support this.

To create a foreign key:

1. Choose a database table in the **Navigator** pane that will contain the foreign key.
2. Click the **References** tab in the **Editor** pane (see Figure 3–8).
3. Select a reference table. See "Creating Table References" on page 3-11 for more information.
4. Add a key pair for the reference. See "Creating Field References" on page 3-12 for more information.

   Use the **Source Field** and **Target Field** drop-down lists to choose the appropriate fields on the source and target tables.

   Repeat step 4 for each foreign key field.

Working with a Container Policy

A container policy specifies the concrete class OracleAS TopLink should use when reading target objects from the database. You can specify a container policy for collection mappings (`DirectCollectionMapping`, `OneToManyMapping`, and `ManyToManyMapping`) and for read-all queries (`ReadAllQuery`).

Starting with JDK 1.2, the collection mappings can use any concrete class that implements either the `java.util.Collection` interface or the `java.util.Map interface`.

When using OracleAS TopLink with JDK 1.2 (or later), you can map object attributes declared as `Collection` or `Map`, or any subinterface of these two interfaces, or as a class that implements one of these two interfaces. You must specify in the mapping the concrete container class to be used. When OracleAS

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. By default, a collection mapping's container class is `java.util.Vector`.

Read-all queries also require a container policy to specify how the result objects are to be returned. The default container is `java.util.Vector`.

Container policies cannot be used to specify a custom container class when using indirect containers.

Overriding the Default Container Policy

For collection mappings, you can specify the container class in the OracleAS TopLink Mapping Workbench (see "Working with Direct Collection Mappings" on page 6-29).

To set the container policy without using the OracleAS TopLink Mapping Workbench, the following API is available for both `CollectionMapping` and `ReadAllQuery`:

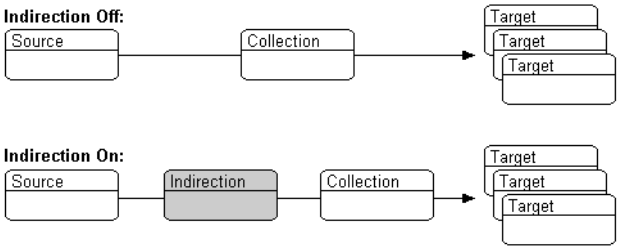
- `useCollectionClass(Class)` – Specifies the concrete `Collection` class to use as a container for the objects in the collection. In JDK 1.2, the class must implement the `java.util.Collection` interface.
- `useMapClass(Class, String)` – Specifies the concrete Map class to use as a container for the objects in the collection. In JDK 1.2, the class must implement the `java.util.Map interface`.

  Also specified is the name of the zero argument method whose result, when called on the target object, is used as the key in the `Hashtable` or `Map`. This method must return an object that is a valid key in the `Hashtable` or `Map`.

Working with Indirection

Using indirection objects can improve the performance of OracleAS TopLink object relationships. An indirection object takes the place of an application object so that the application object is not read from the database until it is needed (see Figure 6–1).

Figure 6–1 OracleAS TopLink indirection



Without indirection, when OracleAS TopLink retrieves a persistent object, it also retrieves all the objects referenced by that object. This can result in lower performance for some applications. Using indirection allows OracleAS TopLink to create "stand-ins" for related objects, resulting in significant performance improvements, especially when the application is interested only in the contents of the retrieved object rather than the objects to which it is related.

Understanding Indirection

Indirection is available for transformation mappings and for direct collection, one-to-one, one-to-many, and many-to-many relationship mappings.

You can enable or disable indirection for each mapping individually. By default, indirection is enabled for relationship mappings and disabled for transformation mappings. Indirection should be enabled only for transformation mappings if the execution of the transformation method is a resource-intensive task, such as accessing the database.

- *Indirection disabled*: An indirection object is not used. Whenever an object is retrieved from the database, all the objects associated with it through the mapping are also read.
- *Indirection enabled*: A value holder is used to represent the entire relationship. When an object is retrieved from the database, a value holder is created and stored in the attribute corresponding to the mapping. The first time the value holder is accessed, it retrieves the **related object from the database**.

In addition to this standard version of indirection, collection mappings (direct collection, one-to-many, and many-to-many) can use indirect **containers**.

Using 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.

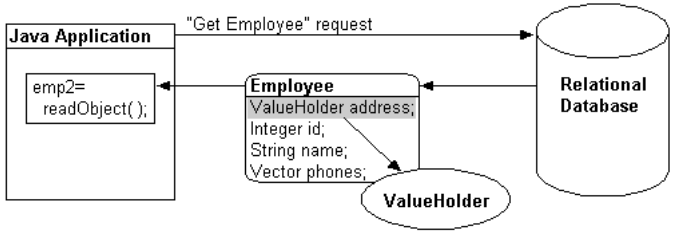
When using method access, the `get` and `set` methods specified for the mapping must access an instance of `ValueHolderInterface`, rather than the object referenced by the value holder.

To obtain the object that the value holder replaces, use the `getValue()` and `setValue()` methods of the `ValueHolderInterface` class. A convenient way of using these methods is to hide the `getValue` and `setValue` methods of the `ValueHolderInterface` inside `get` and `set` methods, as in the following example.

Example 6–1 Value Holder Indirection

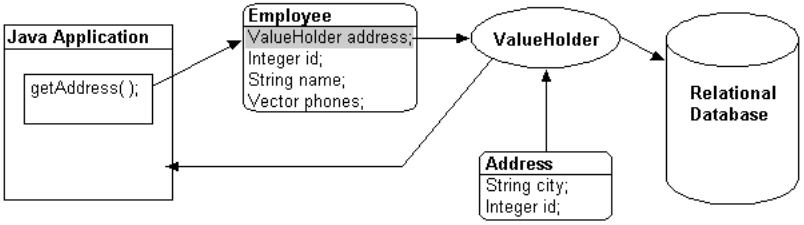
The following figure illustrates the `Employee` object being read from the database. The `Address` object is not read and will not be created unless it is accessed.

Figure 6–2 Address Object Not Read



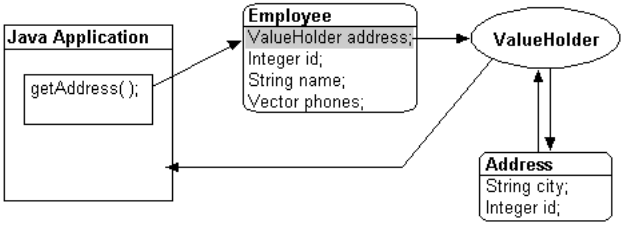
The first time the address is accessed, as in the following figure, the `ValueHolder` reads and returns the `Address` object.

Figure 6–3 Initial Request



Subsequent requests for the address do not access the database, as shown in the following figure.

Figure 6–4 Subsequent Requests



Specifying Indirection

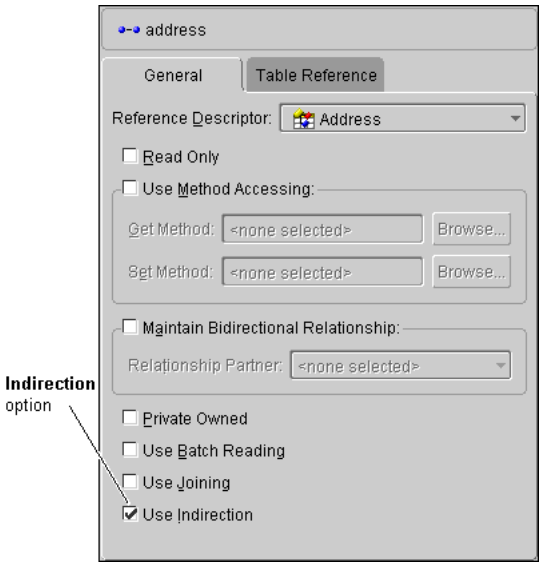
Use this procedure to specify that a mapping uses indirection.

To specify indirection:

1. In the **Navigator** pane, select the mapping to be mapped, and click the appropriate button on the mapping toolbar.

   The mapping tab appears in the **Editor** pane.

Figure 6–5 Sample Mapping Properties



2. On the **General** tab, click **Use Indirection** to specify that the mapping uses indirection.

Changing Java Classes to Use Indirection

Attributes using indirection must conform to the `ValueHolderInterface`. You can change your attribute types in the Class Editor without re-importing your Java classes. Ensure that you change the attribute types in your Java code as well. Attributes typed incorrectly will be marked as deficient.

In addition to changing the attribute's type, you may also need to change its accessor methods. If you use method access, OracleAS TopLink requires accessors to the *indirection* object itself, so your get method returns an instance that conforms to `ValueHolderInterface`, and your set method accepts one argument that conforms to the same. If the instance variable returns a Vector instead of an object, then the value holder should be defined in the constructor as follows:

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

In any case, the application uses the `getAddress()` and `setAddress()` methods to access the `Address` object. With indirection, OracleAS TopLink uses the `getAddressHolder()` and `setAddressHolder()` methods when saving and retrieving instances to and from the database.

Refer to the book *Oracle Application Server TopLink Application Developer's Guide* for details.

Example 6–2 Indirection

The following code illustrates the `Employee` class using indirection with method access for a one-to-one mapping to `Address`.

The class definition is modified so that the address attribute of `Employee` is a `ValueHolderInterface` instead of an `Address`, and appropriate get and set methods are supplied.

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

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

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

Working with Transparent Indirection

Transparent indirection allows you to declare any relationship attribute of a persistent class that holds a collection of related objects as a `java.util.Collection`, `java.util.Map`, `java.util.Vector`, or `java.util.Hastable`. OracleAS 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`.

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

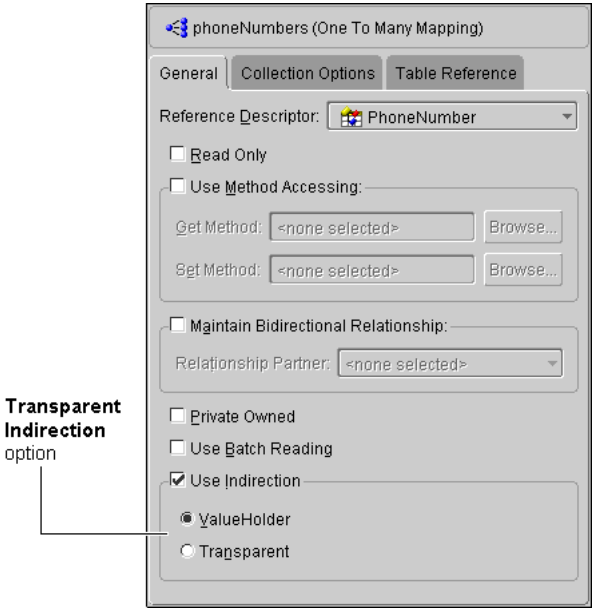
Specifying Transparent Indirection

Use this procedure to use transparent indirection.

Using Transparent Indirection:

1. In the **Navigator** pane, select the attribute. The mapping tab appears in the **Editor** pane.

Figure 6–6 Sample Mapping Properties



2. On the **General** tab, click the **Use Indirection** option for attributes that use indirection.
3. Select the **Transparent** indirection option.

Working with Proxy Indirection

Introduced in JDK 1.3, the Java class `Proxy` enables you to use dynamic proxy objects as stand-ins for a defined interface. Certain OracleAS TopLink mappings (`OneToOneMapping`, `VariableOneToOneMapping`, `ReferenceMapping`, and `TransformationMapping`) can be configured to use proxy indirection, which gives you the benefits of OracleAS TopLink indirection without the need to include OracleAS TopLink classes in your domain model. Proxy indirection is to one-to-one relationship mappings as indirect containers are to collection mappings.

Although the OracleAS TopLink Mapping Workbench does not support proxy indirection, you can use the `useProxyIndirection` method in an amendment method.

To use proxy indirection, your domain model must satisfy 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, then the `get()` and `set()` methods must use the interface.

Example 6–3 Proxy indirection Examples

The following code illustrates an `Employee->Address` one-to-one relationship.

```
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 6–3, both the `EmployeeImpl` and the `AddressImpl` classes implement public interfaces (`Employee` and `Address` respectively). Therefore, because the `AddressImpl` 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 below:

```
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 insert. You still use the following code:

```
Employee emp = new EmployeeImpl("John Smith");
Address add = new AddressImpl("123 Main St.");
emp.setAddress(add);
```

Implementing Proxy Indirection in Java

To enable proxy indirection in Java code, use the following API for `ObjectReferenceMapping`:

- `useProxyIndirection()` – Indicates that OracleAS TopLink should use proxy indirection for this mapping. When the source object is read from the database, a proxy for the target object is created and used in place of the "real" target object. When any method other than `getString()` is called on the proxy, the "real" data will be read from the database.

Example 6–4 Proxy indirection Example

The following code example illustrates using proxy indirection.

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

Optimizing for Queries

You can configure query optimization on any relationship mappings. The optimization requires fewer database calls to read a set of objects from the database. You can configure query optimization on a descriptor's mappings to affect all queries for that class, resulting in a significant system performance gain without changing any application code. Queries can also be optimized on a per-query basis. For more information, see the *Oracle Application Server TopLink Application Developer's Guide*.

OracleAS TopLink provides two query optimization features on mappings: joining and batch reading.

- Joining can be used only on one-to-one mappings. Joining joins the two related classes tables to read all the data in a single query. This feature should be used only if it is known that the target object is always required with the source object, or indirection is not used.

- Batch reading can be used in most of the relational mappings, including direct collection mappings, one-to-one mappings, aggregate collection mappings, one-to-many mappings, and many-to-many mappings. This feature should be used only if it is known that the related objects are always required with the source object.

Example 6–5 Query Optimization Using Joining

The following code example illustrates using joining for query optimization.

```
// Queries for Employee are configured to always join Address
OneToOneMapping addressMapping = new OneToOneMapping();
addressMapping.setReferenceClass(Address.class);
addressMapping.setAttributeName("address");
addressMapping.useJoining();
addressMapping.privateOwnedRelationship();
```

Example 6–6 Query Optimization Using Batching

The following code example illustrates using batch for query optimization.

```
// Queries on Employee are configured to always batch read Address
OneToManyMapping phoneNumbersMapping = new OneToManyMapping();
phoneNumbersMapping.setReferenceClass("
PhoneNumber.class")
phoneNumbersMapping.setAttributeName("phones");
phoneNumbersMapping.useBatchReading();
phoneNumbersMapping.privateOwnedRelationship();
```

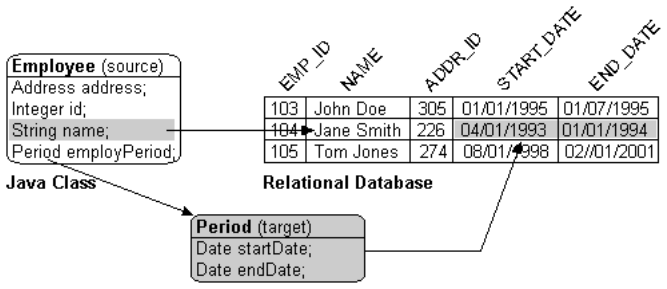
Working with Aggregate Object Mappings

Two objects are related by aggregation if there is a strict one-to-one relationship between the objects, and all the attributes of the second object can be retrieved from the same table(s) as the owning object. This means that if the source (parent) object exists, then the target (child or owned) object must also exist, as illustrated in Figure 6–7.

Aggregate objects are privately owned and should not be shared or referenced by other objects.

Note: When using an aggregate descriptor in an inheritance, *all* the descriptors in the inheritance tree must be aggregates. Aggregate and Class descriptors cannot exist in the same inheritance tree.

Figure 6–7 Aggregate Object Mapping



To implement an aggregate object mapping:

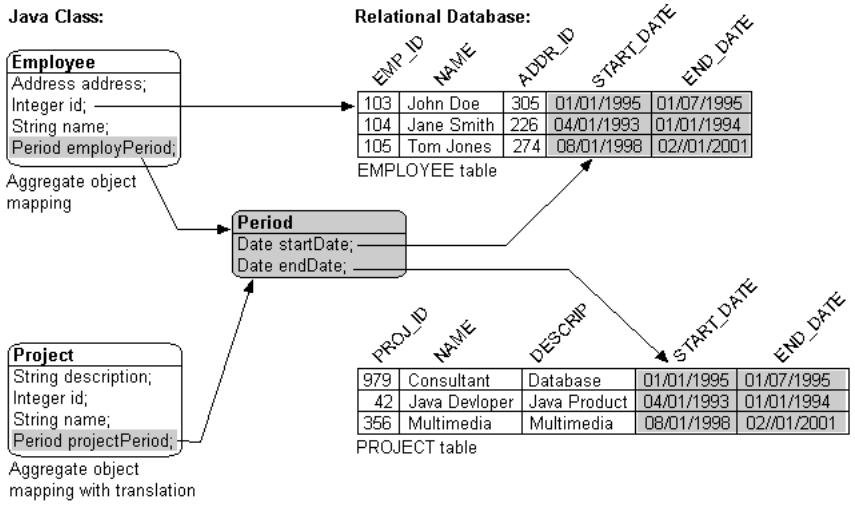
- The descriptor of the target class must declare itself to be an aggregate object. Because all its information comes from its parent's table(s), the target descriptor does not have a specific table associated with it. You must, however, choose one or more candidate table(s) from which you can use fields in mapping the target. In the example above, you could choose the `EMPLOYEE` table so that the `START_DATE` and `END_DATE` fields are available during mapping.
- The descriptor of the source class must add an aggregate object mapping that specifies the target class. In the example above, the `Employee` class has an attribute called `employPeriod` that would be mapped as an aggregate object mapping with `Period` as the reference class. The source class must ensure that its table has fields that correspond to the field names registered with the target class.
- If a source object has a null target reference, OracleAS TopLink writes NULLs to the aggregate database fields. When the source is read from the database, it can handle this null target in one of two ways:
  - Create an instance of the object with all its attributes equal to null.
  - Put a null reference in the source object without instantiating a target. (This is the default method of handling null targets.)

Target objects can also have multiple sources, hence the need to choose a candidate table during its mapping. This allows different source types to store the same target information within their tables. Each source class must have table fields that correspond to the field names registered with the target class. If one of the source tables has different field names than the names registered with the target class, the source class must translate the field names.

In Figure 6–8:

- The `Period` class has a direct-to-field mapping between `startDate` and `START_DATE`.
- The `Employee` class can use the `Period` class as a normal aggregate to write to its `START_DATE` column.
- The `PROJECT` table does not have a field called `START_DATE`, so the `Project` descriptor must provide a field translation on its aggregate object mapping from `START_DATE` to `S_DATE`. (If the `PROJECT` table had a `START_DATE` column, this field translation would be unnecessary.)

Figure 6–8 Aggregation with Multiple Source Classes



Aggregate target classes not shared among multiple source classes can have any type of mapping, including other aggregate object mappings.

Aggregate target classes shared with multiple source classes cannot have one-to-many or many-to-many mappings.

Other classes cannot reference the aggregate target with one-to-one, one-to-many, or many-to-many mappings. If the aggregate target has a one-to-many relationship with another class, the other class must provide a one-to-one relationship back to the aggregate's parent class, instead of the aggregate child. This is because the source class contains the table and primary key information of the aggregate.

Aggregate descriptors can make use of inheritance. The subclasses must also be declared as aggregate and be contained in the source's table. See "Working with Inheritance" on page 4-39 for more information.

Creating a Target Descriptor

Use this procedure to create a target descriptor to employ with an aggregate mapping. You must configure the target before specifying field translations in the parent descriptor.

To create the target descriptor:

1. In the **Navigator** pane, right-click the target descriptor and choose **Aggregate** from the pop-up menu. The descriptor's icon in the **Navigator** pane changes to an Aggregate Descriptor  .

   You can also choose **Selected** > **Aggregate** from the menu or by clicking the **Aggregate Descriptor** button  .

2. Map the attributes, specifying all but field information.

   - For a one-to-one mapping, pick a reference between a table in the target descriptor and a table in a descriptor that will have a mapping to this aggregate target. If this aggregate target will be mapped to multiple source descriptors, pick a reference whose foreign key field(s) will be in the tables of one of the source descriptors.
   - For a one-to-many mapping or a many-to-many mapping, pick a reference whose foreign key field(s) will be in the referenced descriptor's tables and whose primary key field will be in the source descriptor's tables.

3. Continue with "Creating an Aggregate Object Mapping" on page 6-19 to create the aggregate mapping.

Creating an Aggregate Object Mapping

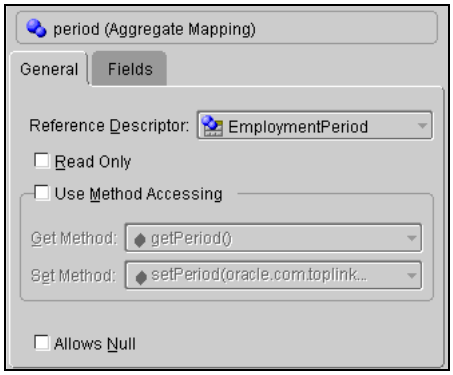
Use this procedure to create an aggregate object mapping. You must also create a target descriptor to use with the aggregate mapping.

To create an aggregate object mapping:

1. In the **Navigator** pane, select the mapping to be mapped and click the **Aggregate Mapping** button on the mapping toolbar.

   The Aggregate mapping tab appears in the **Editor** pane.

Figure 6–9 Aggregate Mapping General Tab

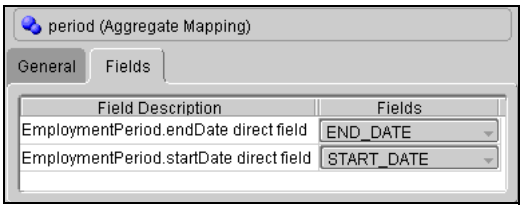


2. Use the **Reference Descriptor** drop-down list on the **General** tab to choose a reference descriptor.

   Note: You can select only aggregate descriptors. See "Creating a Target Descriptor" on page 6-18 for details.

3. You can also specify:
   - Read-only attributes – See "Specifying Read-Only Settings" on page 4-70.
   - Access methods – See "Specifying Direct Access and Method Access" on page 4-70.
   - Null values – See "Defaulting Null Values" on page 4-71.
4. Click the **Fields** tab to specify field information for the target descriptor's mapping.

Figure 6–10 Aggregate Mapping Fields Tab



5. Use this table to enter data in each field:

Field	Description
Field Description	Available fields from the reference descriptor. These fields are for display only and cannot be changed on this tab.
Field	Use the drop-down list to choose a field to use for the mapping for each field description.

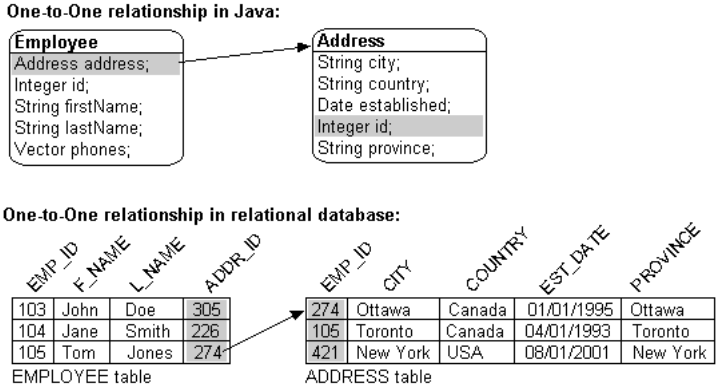
Working with One-to-One Mappings

One-to-one mappings represent simple pointer references between two Java objects. In Java, a single pointer stored in an attribute represents the mapping between the source and target objects. Relational database tables implement these mappings using foreign keys.

Figure 6–11 illustrates a one-to-one relationship from the `address` attribute of an `Employee` object to an `Address` object. To store this relationship in the database, create a one-to-one mapping between the `address` attribute and the `Address` class. This mapping stores the `id` of the `Address` instance in the `EMPLOYEE` table when the `Employee` instance is written. It also links the `Employee` instance to the `Address` instance when the `Employee` is read from the database. Because an `Address` does not have any references to the `Employee`, it does not have to provide a mapping to `Employee`.

For one-to-one mappings, the source table normally contains a foreign key reference to a record in the target table. In Figure 6–11, the ADDR_ID field of the EMPLOYEE table is a foreign key.

Figure 6–11 One-to-One Mappings



You can also implement a one-to-one mapping where the target table contains a foreign key reference to the source table. In the example, the database design would change such that the `ADDRESS` row would contain the `EMP_ID` to identify the `Employee` to which it belonged. In this case, the target must also have a relationship mapping to the source.

The update, insert and delete operations, which are normally done for the target before the source, for privately owned one-to-one relationships, are performed in the opposite order when the target owns the foreign key. Target foreign keys normally occur in bidirectional one-to-one mappings, because one side has a foreign key and the other shares the same foreign key in the other's table.

Target foreign keys can also occur when large cascaded composite primary keys exist (that is, one object's primary key is composed of the primary key of many other objects). In this case it is possible to have a one-to-one mapping that contains both foreign keys and target foreign keys.

In a foreign key, OracleAS TopLink automatically updates the foreign key value in the object's row. In a target foreign key, it does not. In OracleAS TopLink, the **Target Foreign Key** checkbox includes a checkmark when a target foreign key relationship is defined.

When mapping a relationship, you must understand these differences between a foreign key and a target foreign key, to ensure that the relationship is defined correctly.

In a bidirectional relationship where the two classes in the relationship reference each other, only one of the mappings should have a foreign key. The other mapping should have a target foreign key. If one of the mappings in a bidirectional

relationship is a one-to-many mapping, see "Working with Variable One-to-One Mappings" on page 6-24 for details.

Creating One-to-One Mappings

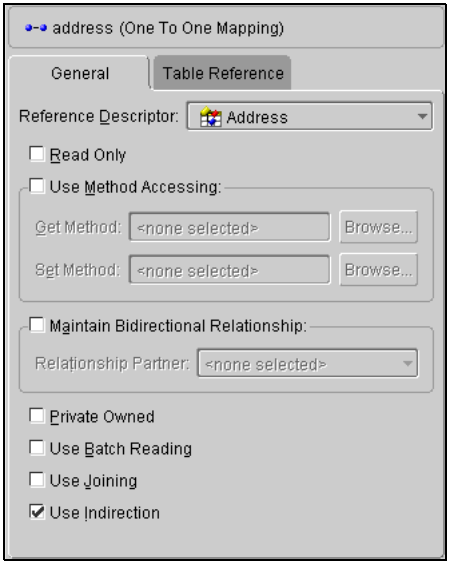
Use this procedure to create a one-to-one mapping.

To create a one-to-one mapping:

1. In the **Navigator** pane, select the mapping to be mapped and click the **One-to-One Mapping** button on the mapping toolbar.

   The One-to-one mapping tab appears in the **Editor** pane.

Figure 6–12 One-to-One Mapping General Properties

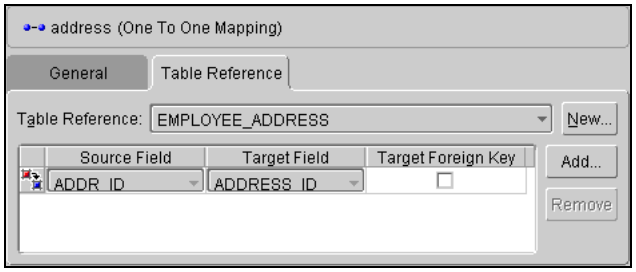


2. Enter the required information on the **General** tab (see "Working with Common Mapping Properties" on page 4-69).
3. You can also specify:
   - Bidirectional relationships – See "Maintaining Bidirectional Relationships" on page 4-71
   - Read-only attributes – See "Specifying Read-Only Settings" on page 4-70

- Access methods – See "Specifying Direct Access and Method Access" on page 4-70
- Null values – See "Defaulting Null Values" on page 4-71

4. Click the **Table Reference** tab to choose the reference.

Figure 6–13 One-to-One Mapping Table Reference Properties



5. Use this table to enter data in each field:

Field	Description
Table Reference	Use the drop-down list to choose a table reference for the mapping. Click **New** to create a new table
Key Pairs	
Source Field	Use the drop-down list to choose a field from the source table.
Target Field	Use the drop-down list to choose a field from the target table.
Target Foreign Key	Specify if the relationship is a target foreign key.

Specifying Advanced Features Available by Amending the Descriptor

One-to-one target objects mapped as **Privately Owned** are, by default, verified before deletion or update outside of a unit of work.

Verification is a check for the previous value of the target and is accomplished through joining the source and target tables. Inside a unit of work, verification is accomplished by obtaining the previous value from the back-up clone, so this setting is not used because a database read is not required. You may wish to disable verification outside of a unit of work for performance reasons and can do so by sending the `setShouldVerifyDelete()` message to the mapping in an amendment method written for the descriptor, as follows:

```
public static void addToDescriptor(Descriptor descriptor){
//Find the one-to-one mapping for the address attribute
OneToOneMapping addressMapping=(OneToOneMapping)
descriptor.getMappingForAttributeName("address");
addressMapping.setShouldVerifyDelete(false);
}
```

Working with Variable One-to-One Mappings

Variable class relationships are similar to polymorphic relationships except that in this case the target classes are not related through inheritance (and thus not good candidates for an abstract table), but through an interface.

To define variable class relationships in OracleAS TopLink Mapping Workbench, use the variable one-to-one mapping selection, but choose the interface as the reference class. This makes the mapping a variable one-to-one. When defining mappings in Java code, use the `VariableOneToOneMapping` class.

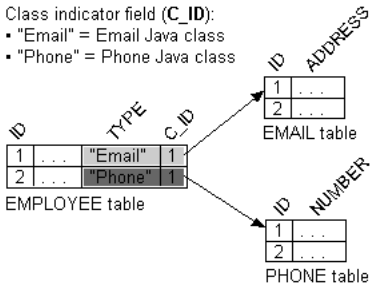
OracleAS TopLink supports variable relationships only in one-to-one mappings. It handles this relationship in two ways:

- Through the class indicator field
- Through unique primary key values among target classes implementing the interface

Specifying Class Indicator

A source table has an indicator column that specifies the target table through the class indicator field, as show in Figure 6–14. The EMPLOYEE table has a TYPE column that indicates the target for the row (either PHONE or EMAIL).

Figure 6–14 Class indicator Field



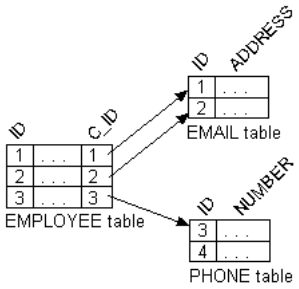
The principles of defining such a variable class relationship are similar to defining a normal one-to-one relationship, except:

- The reference class is a Java *interface*, not a Java *class*. However, the method to set the interface is identical.
- You must specify a type indicator field.
- You specify the class indicator values on the mapping so that mapping can determine the class of object to create.
- You must specify the foreign key names and the respective abstract query keys from the target interface descriptor.

Specifying Unique Primary Key

As Figure 6–15 illustrates, the value of the foreign key in the source table mapped to the primary key of the target table is unique. No primary key values among the target tables are the same, so primary key values are not unique just in the table, but also among the tables.

Figure 6–15 Unique primary key



Because there is no indicator stored in the source table, OracleAS TopLink cannot determine to which target table the foreign key value is mapped. Therefore, OracleAS TopLink reads through all the target tables until it finds an entry in one of the target tables. This is an inefficient way of setting up a relation model, because reading is expensive. The class indicator is much more efficient and it reduces the number of reads performed on the tables to get the data. In the class indicator method, OracleAS TopLink knows exactly which target table to look into for the data.

The principles of defining such a variable class relationship are similar to defining class indicator variable one-to-one relationships, except:

- A type indicator field is not specified.
- The class indicator values are not specified.

The type indicator field and its values are not needed, because OracleAS TopLink goes through all the target tables until data is finally found.

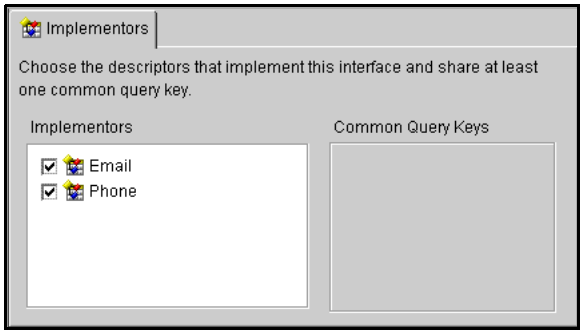
Creating Variable One-to-One Mappings

Use this procedure to create a variable one-to-one mapping. You must configure the target descriptor before defining the mapping.

To create a variable one-to-one mapping:

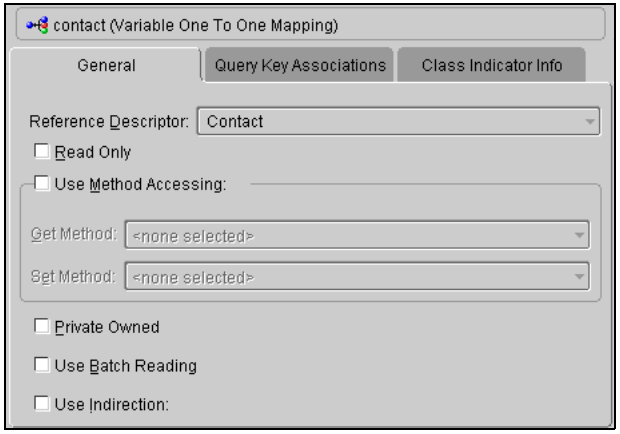
1. In the **Navigator** pane, select the interface descriptor that will be referenced.
2. On the **Implementors** tab, choose all descriptors that implement this interface and share a common query key. You may need to create query keys for some or all of these descriptors.

Figure 6–16 Implementors Tab



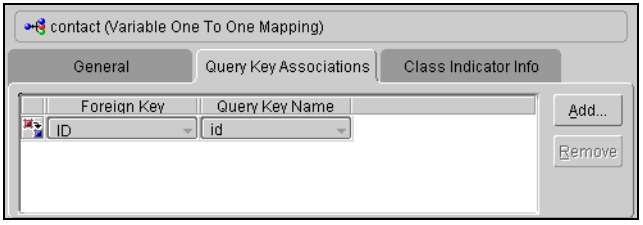
3. In the **Navigator** pane, select the attribute to be mapped as a variable one-to-one mapping and click the **Variable One-to-One Mapping** button on the mapping toolbar.
4. Select the **General** tab.

Figure 6–17 Variable One-to-One Mapping General Properties



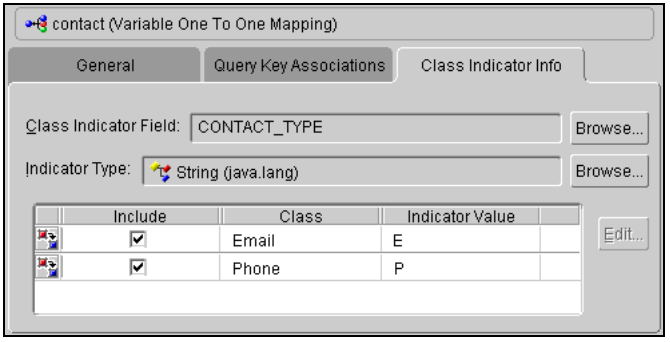
5. Use the **Reference Descriptor** drop-down list to choose a reference descriptor. The OracleAS TopLink Mapping Workbench displays only interface descriptors.
6. Enter any other required information on the **General** tab (see "Working with Common Mapping Properties" on page 4-69).
7. Select the **Query Key Associations** tab.

Figure 6–18 Variable One-to-One Mapping Query Key Associations Properties



8. Specify fields in the source descriptor's tables to use for common query keys.
9. Select the **Class Indicator Info** tab.

Figure 6–19 Variable One-to-One Mapping Class Indicator Info Tab



10. Use this table to enter data in each field.

Field	Description
Class Indicator Field	Use the drop-down list to choose a field to use as a class indicator. To use unique primary keys (no class indicator values), choose **<none selected>**.
Indicator Type	Use the drop-down list to choose the Java type for the **Class Indicator Field**.
Class information:	
Include	Specify to use this class for the mapping.
Class	Name of the class. This field is for display only.
Indicator Value	Value used by this class.

Note: If the class does not appear in the **Class Information** table, you must add the class in the interface descriptor. See "Implementing an Interface" on page 4-47 for more information.

Working with Direct Collection Mappings

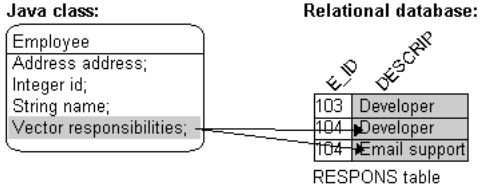
Direct collection mappings store collections of Java objects that are not OracleAS TopLink-enabled. The object type stored in the direct collection is typically a Java type, such as `String`.

It is also possible to use direct collection mappings to map a collection of non-`String` objects. For example, it is possible to have an attribute that contains a collection of `Integer` or `Date` instances. The instances stored in the collection can be any type supported by the database and has a corresponding wrapper class in Java.

Support for primitive data types such as `int` is not provided because Java vectors hold only objects.

Figure 6–20 illustrates how a direct collection is stored in a separate table with two fields. The first field is the reference key field, which contains a reference to the primary key of the instance owning the collection. The second field contains an object in the collection and is called the direct field. There is one record in the table for each object in the collection.

Figure 6–20 Direct Collection Mappings



Note: The **responsibilities** attribute is of type Vector. When using JDK 1.2, it is possible to use a Collection interface (or any class that implements the Collection interface) for declaring the collection attribute. See "Working with a Container Policy" on page 6-4 for details.

Maps are not supported for direct collection because there is no key value.

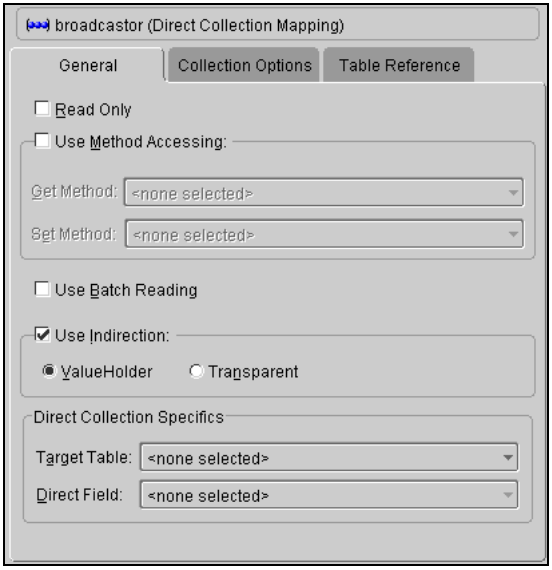
Creating Direct Collection Mappings

Use this procedure to create a direct collection mapping.

To create a direct collection mapping:

1. Select the attribute to be mapped from the **Navigator** pane.
2. Click the **Direct Collection Mapping** button on the mapping toolbar.

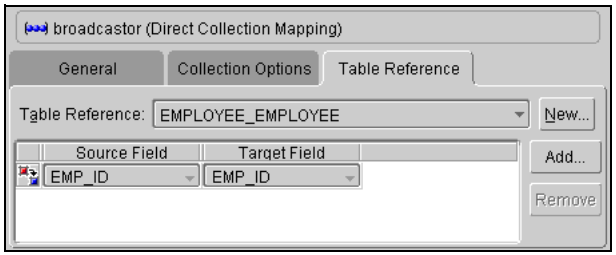
Figure 6–21 Direct Collection Mapping General Properties



3. Use the **Target Table** and **Direct Field** drop-down lists to specify the appropriate information.
4. Enter any other required information on the **General** tab (see "Working with Common Mapping Properties" on page 4-69).
5. Click the **Collection Options** tab to specify collection information for this mapping. See "Specifying Collection Properties" on page 4-72 for more information.

6. Click the **Table Reference** tab to specify foreign key information for this mapping. See "Creating Table References" on page 3-11 for more information.

Figure 6–22 Direct Collection Mapping Table Reference Properties



7. Choose the appropriate reference that relates the target table to the tables associated with the source descriptor.

Working with Aggregate Collection Mappings

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

> **Caution:** The OracleAS TopLink Mapping Workbench does not directly support aggregate collections. You must use an amendment method (see "Amending Descriptors After Loading" on page 4-27) or manually edit the project source to add the mapping.

To implement an aggregate collection mapping:

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

Aggregate collection descriptors can use inheritance. You must also declare subclasses as aggregate collection. The subclasses can have their own mapped tables, or share the table with their parent class. See "Working with Inheritance" on page 4-39 for more information on inheritance.

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

Note: For information on using collection classes other than `Vector` with aggregate collection mappings, see the book *Oracle Application Server TopLink Application Developer's Guide*.

Working with One-to-Many Mappings

One-to-many mappings are used to represent the relationship between a single source object and a collection of target objects. They are a good example of something that is simple to implement in Java using a `Vector` (or other collection types) of target objects, but difficult to implement using relational databases.

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

Note: See "Working with a Container Policy" on page 6-4 for information on using collection classes other than `Vector` with one-to-many mappings.

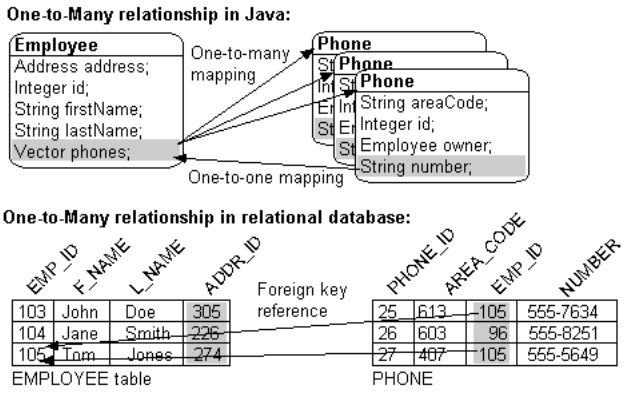
The purpose of creating this one-to-one mapping in the target is so that the foreign key information can be written when the target object is saved. Alternatives to the one-to-one mapping back reference include:

- Use a direct-to-field mapping to map the foreign key and maintain its value in the application. Here the object model does not require a back reference, but the data model still requires a foreign key in the target table.
- Use a many-to-many mapping to implement a logical one-to-many. This has the advantage of not requiring a back reference in the object model and not requiring a foreign key in the data model. In this model the many-to-many relation table stores the collection. It is possible to put a constraint on the join table to enforce that the relation is a logical one-to-many relationship.

Example 6–7 One-to-Many Mapping

One-to-many mappings must put the foreign key in the target table, rather than the source table. The target class should also implement a one-to-one mapping back to the source object, as illustrated in the following figure.

Figure 6–23 One-to-Many Relationships



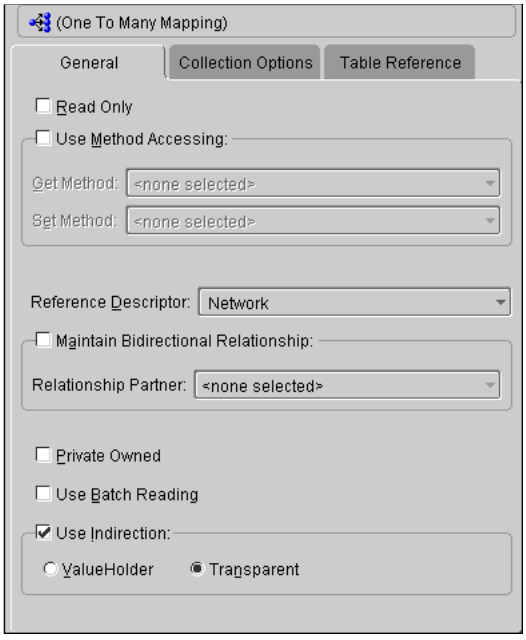
Creating One-to-Many Mappings

Use this procedure to create a one-to-many mapping in the OracleAS TopLink Mapping Workbench.

To create a one-to-many mapping:

1. Select the attribute to be mapped from the **Navigator** pane.
2. Click the **One-to-Many Mapping** button on the mapping toolbar.

Figure 6–24 One-to-many mapping General Properties



3. Use the **Reference Descriptor** drop-down list to choose the reference for this descriptor.
4. You can also specify:
   - Bidirectional relationships – See "Maintaining Bidirectional Relationships" on page 4-71.
   - Read-only attributes – See "Specifying Read-Only Settings" on page 4-70.
   - Access methods – See "Specifying Direct Access and Method Access" on page 4-70.
   - Null values – See "Defaulting Null Values" on page 4-71.
5. Click the **Table References** tab to specify foreign key information for this mapping. See "Creating Table References" on page 3-11 for more information.

Working with Many-to-Many Mappings

Many-to-many mappings represent the relationships between a collection of source objects and a collection of target objects. They require the creation of an intermediate table for managing the associations between the source and target records. Figure 6–25 illustrates a many-to-many mapping in Java and in relational database tables.

Many-to-many mappings are implemented using a relation table. This table contains columns for the primary keys of the source and target tables. Composite primary keys require a column for each field of the composite key. The intermediate table must be created in the database before using the many-to-many mapping.

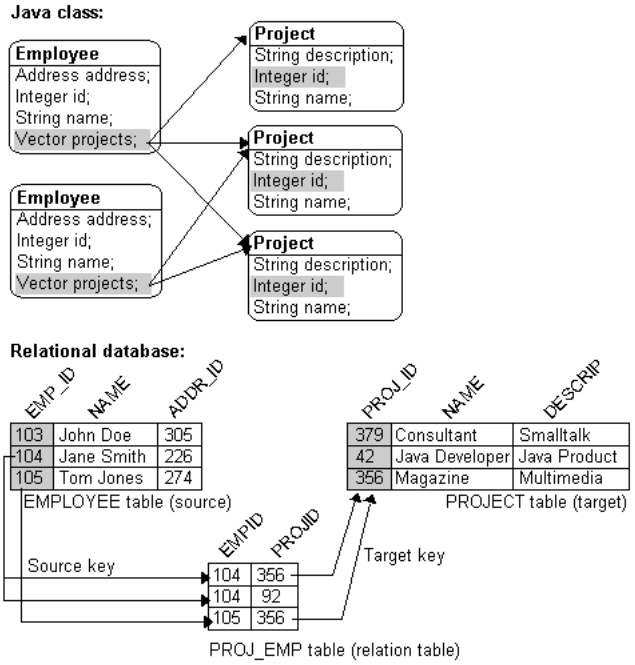
The target class does not have to implement any behavior for the many-to-many mappings. If the target class also creates a many-to-many mapping back to its source, then it can use the same relation table, but one of the mappings must be set to read-only. If both mappings write to the table, they can cause collisions.

Note: See "Working with a Container Policy" on page 6-4 for information on using collection classes other than Vector with one-to-many mappings.

Indirection is enabled by default in a many-to-many mapping, which requires that the attribute have the `ValueHolderInterface` type or transparent collections.

The following figures illustrate a many-to-many relationship in both Java and a relational database.

Figure 6–25 Many-to-many Relationships



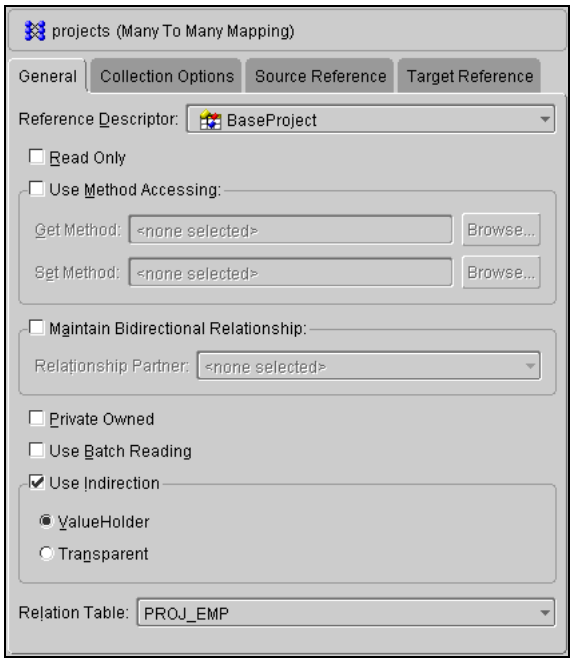
Creating many-to-many Mappings

Use this procedure to create a many-to-many mapping.

To create a many-to-many mapping:

1. In the **Navigator** pane, select the attribute to be mapped.
2. Click the **Many-to-Many Mapping** button on the mapping toolbar.

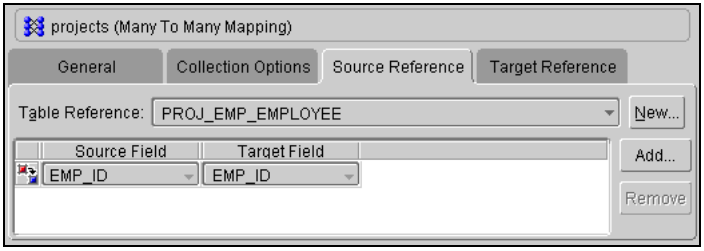
Figure 6–26 Many-to-Many Mapping General Properties



3. Use the **Reference Descriptor** drop-down list to choose the reference descriptor for this mapping.
4. Use the **Relation Table** drop-down list to choose the relation table.
5. Use the **Maintain Bidirectional Relationship** option to select a **Relationship Partner** for this mapping. See "Maintaining Bidirectional Relationships" on page 4-71 for more information.
6. Modify any other properties, as needed. See "Working with Common Mapping Properties" on page 4-69 for more information.
7. Click the **Collection Options** tab to specify how the source descriptor relates to the relation table. See "Specifying Collection Properties" on page 4-72 for more information.

8. Click the **Source Reference** tab to specify how the source descriptor relates to the relation table.

Figure 6–27 Many-to-Many Mapping Source Reference Properties



9. Use the **Table Reference** drop-down list to choose a reference whose foreign key is in the relation table and that points to a table associated to the source descriptor. See "Creating Table References" on page 3-11 for more information.
10. Click the **Target Reference** tab to specify how the reference descriptor relates to the relation table.
11. Choose a reference whose foreign key is in the relation table and that points to a table associated to the reference descriptor. See "Creating Table References" on page 3-11 for more information.

Specifying Advanced Features by Amending the Descriptor

OracleAS TopLink can populate a collection in ascending or descending order, upon your specification. To do this, specify and write an amendment method, sending the `addAscendingOrdering()` or `addDescendingOrdering()` to the many-to-many mapping. Both messages expect a string as a parameter, which indicates what attribute from the target object is used for the ordering. This string can be an attribute name or query key from the target's descriptor. Query keys are automatically created for and with the same name as all attributes mapped as direct-to-field, type conversion, object type, and serialized object mappings.

Example 6–8 Descriptor Amendment Example

The following code example illustrates returning an Employee's projects in ascending order, according to their descriptions:

```
public static void addToDescriptor(Descriptor descriptor)
{
//Find the Many-to-Many mapping for the projects attribute
```

```
ManyToManyMapping projectsMapping=(ManyToManyMapping)
descriptor.getMappingForAttributeName("projects");
projectsMapping.addAscendingOrdering("description");
}
```

Working with Custom Relationship Mappings

Just as a descriptor's query manager generates the default SQL code used for database interaction, relationship mappings also generate query information.

As with the queries used by a descriptor's query manager, you can customize relationship mappings using SQL strings or query objects. Refer to "Specifying Queries and Named Finders" on page 4-12 for more information on customizing queries and the syntax that OracleAS TopLink supports.

To customize the way a relationship mapping generates SQL, use any of the following methods:

- selection — All relationship mappings can use the `setSelectionCriteria()`, `setSelectionSQLString()`, and `setCustomSelectionQuery()` methods of the mapping to customize the selection criteria.
- insert — Many-to-many and direct collection mappings can use the `setInsertSQLString()` or `setCustomInsertQuery()` methods of the mapping to customize the insertion criteria.
- delete all — Many-to-many, direct collection, and one-to-many mappings can use the `setDeleteAllSQLString()` and `setCustomDeleteAllQuery()` methods of the mapping to customize the deletion criteria.
- delete —Many-to-many mappings can use the `setDeleteSQLString()` and `setCustomDeleteQuery()` methods of the mapping to customize the deletion criteria.

A query object that specifies the search criteria must be passed to each of these methods. Because search criteria for these operations usually depend on variables at runtime, the query object must usually be created from a parameterized expression, SQL string, or stored procedure call.

See the book *Oracle Application Server TopLink Application Developer's Guide* for more information on defining parameterized queries and stored procedure calls.

Creating Custom Mapping Queries in Java Code

The following example illustrates selection customization with a parameterized expression using `setSelectionCriteria()`, and deletion customization using `setDeleteAllSQLString()`. Because the descriptor is passed as the parameter to this amendment method, which has been specified to be called after the descriptor is loaded in the project, you must locate each mapping for which you wish to define a custom query.

Example 6–9 Custom Mapping Example

The following code illustrates adding a custom query to two different mappings in the `Employee` descriptor.

```
// Amendment method in Employee class
public static void addToDescriptor(Descriptor descriptor)
{

//Find the one-to-one mapping for the address attribute
OneToOneMapping addressMapping=(OneToOneMapping)
descriptor.getMappingForAttributeName("homeaddress");

//Create a parameterized Expression and register it as the default selection
criterion for the mapping.
ExpressionBuilder builder = new ExpressionBuilder();
addressMapping.setSelectionCriteria(builder.getField("ADDRESS.ADDRESS_
ID").equal(builder.getParameter("EMP.ADDRESS_
ID")).and(builder.getField("ADDRESS.TYPE").equal("home")));

// Get the direct collection mapping for responsibilitiesList.
DirectCollectionMapping directCollection=(DirectCollectionMapping)
descriptor.getMappingForAttributeName("responsibilitiesList");
directCollection.setDeleteAllSQLString("DELETE FROM RESPONS WHERE EMP_ID = #EMP_
ID");
}
```

7

Understanding Object-Relational Mappings

Relational mappings define how persistent objects reference other persistent objects. Oracle Application Server TopLink supports the following object relational mapping types:

- *Array mappings* are similar to direct collection mappings, but map to object-relational array data-types (the Array type in JDBC 2.0 and the VARRAY type in Oracle8*i*). Use array mappings to map a collection of primitive data. See "Working with Array Mappings" on page 7-2 for more information.
- *Object array mappings* are similar to array mappings, but map to object-relational array data types. See "Working with Object Array Mappings" on page 7-4 for more information.
- *Structure mappings* are similar to aggregate object mappings, but map to object-relational aggregate structures (the Struct type in JDBC 2.0 and the OBJECT TYPE in Oracle8*i*). See "Working with Structure Mappings" on page 7-5 for more information.
- *Reference mappings* are similar to one-to-one mappings, but map to object-relational references (the Ref type in JDBC 2.0 and the REF type in Oracle8*i*). See "Working with Reference Mappings" on page 7-7 for more information.
- *Nested table mappings* are similar to many-to-many mappings, but map to object-relational nested tables (the NESTED TABLE type in Oracle8*i*). See "Working with Nested Table Mappings" on page 7-9 for more information.

These mappings allow for an object model to be persisted into an object-relational data-model. Currently the OracleAS TopLink Mapping Workbench does not support object-relational mappings—they must be defined in code or through amendment methods. See "Working with Object-relational Descriptors" on page 4-66 for more information.

Working with Object-Relational Mappings

Object-relational mappings allow for an object model to be persisted into an object-relational data-model. The OracleAS TopLink Mapping Workbench does not directly support these mappings—you must define them in code through amendment methods.

OracleAS TopLink supports the following object-relational mappings:

- Array
- Object array
- Structure
- Reference
- Nested table

Working with Array Mappings

In an object-relational data-model, structures can contain *arrays* (collections of other data types). These arrays can contain primitive data types or collections of other structures. OracleAS TopLink stores the arrays with their parent structure in the same table.

All elements in the array must be the same data type. The number of elements in an array controls the size of the array. An Oracle database allows arrays of variable sizes (called Varrays).

Oracle8*i* offers two collection types:

- Varray – Used to represent a collection of primitive data or aggregate structures.
- Nested table – Similar to varrays except they store information in a separate table from the parent structure's table

OracleAS TopLink supports arrays of primitive data through the `ArrayMapping` class. This is similar to `DirectCollectionMapping` – it represents a collection of primitives in Java. However, the `ArrayMapping` class does not require an additional table to store the values in the collection.

OracleAS TopLink supports arrays of aggregate structures through the `ObjectArrayMaping` class.

OracleAS TopLink supports nested tables through the `NestedTableMapping` class.

Implementing Array Mappings in Java

Array mappings are instances of the `ArrayMapping` class. You must associate this mapping to an attribute in the parent class. OracleAS TopLink requires the following elements for an array mapping:

- Attribute being mapped – Set by sending the `setAttributeName( )` message.
- Field being mapped – Set by sending the `setFieldName( )` message.
- Name of the array – Set by sending the `setStructureName( )` message.

Table 7–1 summarizes all array mapping properties:

Example 7–1 Array Mapping

The following code example illustrates creating an array mapping for the `Employee` source class and registering it with the descriptor

```
// Create a new mapping and register it with the source descriptor.
ArrayMapping arrayMapping = new ArrayMapping();
arrayMapping.setAttributeName("responsibilities");
arrayMapping.setStructureName("Responsibilities_t");
arrayMapping.setFieldName("RESPONSIBILITIES");
descriptor.addMapping(arrayMapping);
```

Reference

The following table summarizes all array mapping properties. In the Method Names column, arguments are **bold**, methods are not.

Table 7–1 Properties for ArrayMapping methods

Property	Default	Method Names
Attribute to be mapped *	not applicable	`setAttributeName(String name)`
Set parent class *	not applicable	`setReferenceClass(Class referenceClass)`
User-defined data type *	not applicable	`setStructureName(String Structurename)`
Field to be mapped *	not applicable	`setFieldName(String fieldName)`

* Required property

Table 7–1 Properties for ArrayMapping methods (Cont.)

Property	Default	Method Names
Method access	direct access	`setGetMethodName(String name)`
		`setSetMethodName(String name)`
Read only	read / write	`readWrite()`
		`readOnly()`
		`setIsReadOnly(boolean readOnly)`

* Required property

Working with Object Array Mappings

In an object-relational data-model, object arrays allow for an array of object types or structures to be embedded into a single column in a database table or an object table. OracleAS TopLink supports object array mappings to define a collection-aggregated relationship in which the target objects share the same row as the source object.

Implementing Object Array Mappings in Java

Object array mappings are instances of the `ObjectArrayMapping` class. You must associate this mapping to an attribute in the parent class. OracleAS TopLink requires the following elements for an object array mapping:

- Attribute being mapped – Set by sending the `setAttributeName( )` message.
- Field being mapped – Set by sending the `setFieldName( )` message.
- Name of the array – Set by sending the `setStructureName( )` message.

Use the optional `setGetMethodName( )` and `setSetMethodName( )` messages to access the attribute through user-defined methods rather than directly. See "Specifying Direct Access and Method Access" on page 4-70 for more information. Table 7–2 summarizes all object array mapping properties.

Example 7–2 Object Array Mapping

The following code example illustrates creating an object array mapping for the `Insurance` source class and registering it with the descriptor.

```
// Create a new mapping and register it with the source descriptor.
ObjectArrayMapping phonesMapping = new ObjectArrayMapping();
phonesMapping.setAttributeName("phones");
```

```
phonesMapping.setGetMethodName("getPhones");
phonesMapping.setSetMethodName("setPhones");
phonesMapping.setStructureName("PHONELIST_TYPE");
phonesMapping.setReferenceClass(Phone.class);
phonesMapping.setFieldName("PHONES");
descriptor.addMapping(phonesMapping);
```

Reference

The following table summarizes all object array mapping properties. In the Method Names column, arguments are **bold**, methods are not.

Table 7–2 Properties for ObjectArrayMapping Methods

Property	Default	Method Names
Attribute to be mapped *	not applicable	`setAttributeName(String name)`
Set parent class *	not applicable	`setReferenceClass(Class referenceClass)`
User-defined data type *	not applicable	`setStructureName(String structureName)`
Field to be mapped *	not applicable	`setFieldName(String fieldName)`
Method access	direct access	`setGetMethodName(String name)`
		`setSetMethodName(String name)`
Read only	read / write	`readWrite()`
		`readOnly()`
		`setIsReadOnly(boolean readOnly)`

* Required property

Working with Structure Mappings

In an object-relational data-model, structures are user defined data-types or object-types. This is similar to a Java class—it defines attributes or fields in which each attribute is either:

- A primitive data type
- Another structure
- Reference to another structure

OracleAS TopLink maps each structure to a Java class defined in your object model and defines a descriptor for each class. A `StructureMapping` class maps nested structures, similar to an `AggregateObjectMapping` class. However, the structure mapping supports null values and shared aggregates without requiring additional settings (because of the object-relational support of the database).

Implementing Structure Mappings in Java

Structure mappings are instances of the `StructureMapping` class. You must associate this mapping to an attribute in each of the parent classes. OracleAS TopLink requires the following elements for structure mapping:

- Attribute being mapped – Set by sending the `setAttributeName( )` message.
- Field being mapped – Set by sending the `setFieldName( )` message.
- Target (child) class – Set by sending the `setReferenceClass( )` message.

Use the optional `setGetMethodName( )` and `setSetMethodName( )` messages to access the attribute through user-defined methods, rather than directly. See "Specifying Direct Access and Method Access" on page 4-70 for more information.

You must make the following changes to the target (child) class descriptor:

- Send the `descriptorIsAggregate()` message to indicate it is not a root level.
- Remove table or primary key information.

Table 7–3 summarizes all structure mapping properties:

Example 7–3 Structure Mapping Examples

The following code example illustrates creating a structure mapping for the `Employee` source class and registering it with the descriptor

```
// Create a new mapping and register it with the source descriptor.
StructureMapping structureMapping = new StructureMapping();
structureMapping.setAttributeName("address");
structureMapping.setReferenceClass(Address.class);
structureMapping.setFieldName("address");
descriptor.addMapping(structureMapping);
```

The following code example illustrates creating the descriptor of the `Address` aggregate target class. The aggregate target descriptor does not need a mapping to its parent, or any table or primary key information.

```
// Create a descriptor for the aggregate class. The table name and primary key
```

```
are not specified in the aggregate descriptor.
ObjectRelationalDescriptor descriptor = new ObjectRelationalDescriptor ();
descriptor.setJavaClass(Address.class);
descriptor.setStructureName("ADDRESS_T");
descriptor.descriptorIsAggregate();
// Define the field ordering
descriptor.addFieldOrdering("STREET");
descriptor.addFieldOrdering("CITY");
...
// Define the attribute mappings or relationship mappings.
...
```

Reference

The following table summarizes all structure mapping properties. In the Method Names column, arguments are **bold**, methods are not.

Table 7–3 Properties for StructureMapping Methods

Property	Default	Method Names
Attribute to be mapped *	not applicable	`setAttributeName(String name)`
Set parent class *	not applicable	`setReferenceClass(Class aClass)`
Field to be mapped *	not applicable	`setFieldName(String fieldName)`
Method access	direct access	`setGetMethodName(String name)`
		`setSetMethodName(String name)`
Read only	read / write	`readWrite()`
		`readOnly()`
		`setIsReadOnly(boolean readOnly)`

* Required property

Working with Reference Mappings

In an object-relational data-model, structures reference each other through *refs*—not through foreign keys (as in a traditional data-model). Refs are based on the target structure's `ObjectID`.

OracleAS TopLink supports refs through the `ReferenceMapping` class. They represent an object reference in Java, similar to a `OneToOneMapping`. However, the reference mapping does not require foreign key information.

Implementing Reference Mappings in Java

Reference mappings are instances of the `ReferenceMapping` class. You must associate this mapping to an attribute in the source class. OracleAS TopLink requires the following elements for a reference mapping:

- Attribute being mapped – Set by sending the `setAttributeName( )` message.
- Field being mapped – Set by sending the `setFieldName( )` message.
- Target class – Set by sending the `setReferenceClass( )` message.

Use the optional `setGetMethodName( )` and `setSetMethodName( )` messages to access the attribute through user-defined methods rather than directly. See "Specifying Direct Access and Method Access" on page 4-70 for more information.

Table 7–4 summarizes all reference mapping properties.

Example 7–4 Reference Mapping

The following code example illustrates creating a reference mapping for the `Employee` source class and registering it with the descriptor.

```
// Create a new mapping and register it with the source descriptor.
ReferenceMapping refrenceMapping = new ReferenceMapping();
referenceMapping.setAttributeName("manager");
referenceMapping.setReferenceClass(Employee.class);
referenceMapping.setFieldName("MANAGER");
descriptor.addMapping(refrenceMapping);
```

Reference

The following table summarizes all reference mapping properties. In the Method Names column, arguments are **bold**, methods are not.

Table 7–4 Properties for ReferenceMapping Methods

Property	Default	Method Names
Attribute to be mapped *	not applicable	`setAttributeName(String name)`
Set parent class *	not applicable	`setReferenceClass(Class aClass)`
Field to be mapped *	not applicable	`setFieldName(String fieldName)`
Method access	direct access	`setGetMethodName(String name)` `setSetMethodName(String name)`
Indirection	use indirection	`useBasicIndirection()` `dontUseIndirection()`

Privately owned relationship	independent	`independentRelationship()`
		`privateOwnedRelationship()`
		`setIsPrivateOwned(boolean isPrivateOwned)`

Read only	read / write	`readWrite()`
		`readOnly()`
		`setIsReadOnly(boolean readOnly)`

* Required property

Working with Nested Table Mappings

Nested table types model an unordered set of elements. These elements may be built-in or user-defined types. You can view a nested table as a single-column table or, if the nested table is an object type, as a muticolumn table (with a column for each attribute of the object type).

Typically, nested tables represent a one-to-many or many-to-many relationship of references to another independent structure. They support querying and joining better than Varrays that are inlined to the parent table.

OracleAS TopLink supports nested table through the `NestedTableMapping` class. They represent a collection of object references in Java, similar to a `OneToManyMapping` or `ManyToManyMapping`. However, the nested table mapping

does not require foreign key information (as with a one-to-many mapping) or the relational table (as with a many-to-many mapping).

Implementing Nested Table Mappings in Java

Nested table mappings are instances of the `NestedTableMapping` class. This mapping is associated to an attribute in the parent class. The following elements are required for a nested table mapping to be viable:

- The attribute being mapped, which is set by sending the `setAttributeName()` message
- The field being mapped, which is set by sending the `setFieldName()` message
- The name of the array structure, which is set by sending the `setStructureName()` message

Use the optional `setGetMethodName()` and `setSetMethodName()` messages to allow OracleAS TopLink to access the attribute through user-defined methods, rather than directly. See "Specifying Direct Access and Method Access" on page 4-70 for more information.

Table 7–5 summarizes all nested table mapping properties.

Example 7–5 Nested Table

The following code example illustrates creating a nested table mapping for the Insurance source class and registering it with the descriptor.

```
// Create a new mapping and register it with the source descriptor.
NestedTableMapping policiesMapping = new NestedTableMapping();
policiesMapping.setAttributeName("policies");
policiesMapping.setGetMethodName("getPolicies");
policiesMapping.setSetMethodName("setPolicies");
policiesMapping.setReferenceClass(Policy.class);
policiesMapping.dontUseIndirection();
policiesMapping.setStructureName("POLICIES_TYPE");
policiesMapping.setFieldName("POLICIES");
policiesMapping.privateOwnedRelationship();
policiesMapping.setSelectionSQLString("select p.* from policyHolders ph,
table(ph.policies) t, policies p where ph.ssn=#SSN and ref(p) = value(t)");
descriptor.addMapping(policiesMapping);
```

Reference

The following table summarizes all nested table mapping properties. In the Method Names column, arguments are **bold**, methods are not.

Table 7–5 Properties for NestedTableMapping Methods

Property	Default	Method Names
Attribute to be mapped *	not applicable	`setAttributeName(String name)`
Set parent class *	not applicable	`setReferenceClass(Class referenceClass)`
User-defined data type *	not applicable	`setStructureName(String structureName)`
Field to be mapped *	not applicable	`setFieldName(String fieldName)`
Method access	direct access	`setGetMethodName(String name)` `setSetMethodName(String name)`

Indirection	use indirection	`useIndirection()`
		`dontUseIndirection()`
		`setUsesIndirection(boolean usesIndirection)`

Privately owned relationship	independent	`independentRelationship()`
		`privateOwnedRelationship()`
		`setIsPrivateOwned(Boolean isPrivateOwned)`

Read only	read / write	`readWrite()`
		`readOnly()`
		`setIsReadOnly(boolean readOnly)`

* Required property

8

Understanding the OracleAS TopLink Sessions Editor

Use the Oracle Application Server TopLink Sessions Editor to easily manage your `sessions.xml` file information. You can work with this file outside of the OracleAS TopLink Mapping Workbench, if necessary.

Use the `sessions.xml` file to configure one or more sessions for the OracleAS TopLink project, and associate the sessions with the project. This approach allows you to configure the following elements for each project:

- Database (JDBC) login information different from that used during development (for example, external data sources for the host application server's connection pools).
- JTA/JTS transaction usage
- Cache synchronization
- Session Broker (multiple databases appearing as a single OracleAS TopLink session)

Refer to the *Oracle Application Server TopLink Application Developer's Guide* for more information.

Starting the OracleAS TopLink Sessions Editor

Use one of the following methods to start the OracleAS TopLink Sessions Editor:

- For Windows environments: From the **Start** menu, choose **Program Files** > **OracleAS TopLink** > **Sessions Editor**.
- For non-Windows environments: Execute the *<ORACLE_HOME>*`\toplink\bin\sessionsEditor.sh` file.

The splash screen appears, followed by the OracleAS TopLink Sessions Editor window.

Working with the OracleAS TopLink Sessions Editor

The OracleAS TopLink Sessions Editor window contains similar parts as the OracleAS TopLink Mapping Workbench window (see Figure 1–2). The **Navigator** pane contains the project tree for all open projects (see "Using the Navigator Pane" on page 8-2). Click the plus or minus (+ or –) symbol next to an object (or double-click the object) to expand and collapse the tree. When you select an object in the **Navigator** pane, its properties appear in the **Editor** pane.

Using the Navigator Pane

The OracleAS TopLink Sessions Editor displays each configuration's session and connection pool information in the **Navigator** pane on the left side of the editor.

Figure 8–1 Sample Navigator Pane

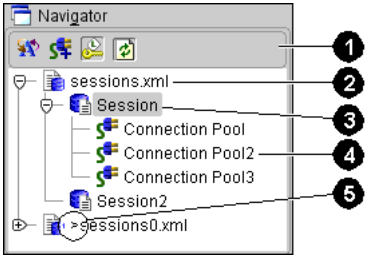


Figure 8–1 calls out the following user-interface components:

1. Navigator toolbar (see "Using the Toolbars" on page 1-5)
2. Configuration (refers to the `<toplink-configuration>` tag)

3. Session (refers to the `<session>` tag)
4. Connection pool (refers to the `<connection-pool>` tag)
5. Change indicator (**>**) – Appears beside elements that have been changed, but not yet saved.

Click the **+** or **–** symbol next to the item, or double-click the item name, to expand and collapse an item.

When you select an item in the **Navigator** pane, its properties appear in the **Editor** pane (see "Using the Editor Pane" on page 1-10).

You can perform specific functions for an item by right-clicking the item in the **Navigator** pane and choosing the function from the pop-up menu. (see "Pop-Up Menus" on page 1-5).

If a session element contains an error, a warning icon ⚠ appears beside the descriptor's icon in the **Navigator** pane, and a message displays in the status bar detailing the error.

Renaming Elements

To rename an element, select the element in the **Navigator** pane, then click the **Rename** button. The Rename dialog box appears.

Understanding Configurations

The OracleAS TopLink sessions configuration file (named `sessions.xml`, by default) is an XML (extensible markup language) file that contains all sessions associated with a project. Each OracleAS TopLink project belongs to an OracleAS TopLink *session*. To deploy beans that belong to different projects, add an appropriate OracleAS TopLink session (see "Working with Sessions" on page 8-7) to the configuration.

Refer to the book *Oracle Application Server TopLink Application Developer's Guide* for a sample `sessions.xml` configuration file.

Working with Configurations

The OracleAS TopLink Sessions Editor displays configurations and their contents in the **Navigator** pane. When you select a configuration, its attributes display in the **Editor** pane.

Creating New Configurations

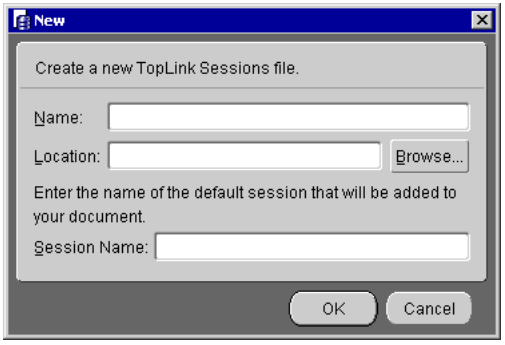
Use this procedure to create a new OracleAS TopLink session configuration.

To Create a New Configuration:

1. Click the **Create New Configuration** button on the toolbar. The Create New Session Configuration dialog box appears.

   You can also choose **File** > **New** from the menu.

Figure 8–2 New Sessions File



2. Enter the configuration **Name** (`sessions.xml` by default), select a directory **Location** in which to save the file, enter a default Session Name, and click **OK**.

   The OracleAS TopLink Sessions Editor appears, showing the configuration name in the **Navigator** pane.

3. Select the configuration element in the **Navigator** pane. The session's **General** tab appears in the **Editor** pane (see Figure 8–5 on page 8-8).

Opening Existing Configurations

Use this procedure to open an existing session configuration.

To Open an Existing Configuration:

Click the **Open Configuration** button on the toolbar. The Create New Session Configuration dialog box appears.

You can also choose **File** > **Open** from the menu.

Saving Configurations

The OracleAS TopLink Sessions Editor does not automatically save your project. Be sure to save your project often to avoid losing data.

To Save Your Configuration(s):

Click the **Save** button or **Save All** button to save your project(s).

You can also choose **File** > **Save**, or **File** > **Save All** from the menu.

To Save Your Configuration with a Different Name or Location:

1. Click the **Save As** button or choose **File** > **Save As**. The Save As dialog box appears.
2. Browse to the directory in which to save the project. In the **File Name** field, type the name of the project and click **Save**.

> **Caution:** Do not rename the `.xml` file outside of the OracleAS TopLink Sessions Editor. Always rename a project by using the **Save As** option.

Working with Session Brokers

Use session brokers to allow OracleAS TopLink to access multiple databases. With a session broker, you can store the objects within one application on multiple databases.

To Create a Session Broker

1. Select the configuration element in the **Navigator** pane. The session's **General** tab appears in the **Editor** pane.

Figure 8–3 Configuration — General Property Sheet

2. Click the **Sessions Broker** tab. The Session Broker tab of the Session property sheet appears.

If the **Session Broker** tab is not visible for the session, click the **Session Broker** button on the Navigator toolbar. You can also create brokers by right-clicking on the configuration element in the Navigator pane and selecting **Advanced** > **Session Brokers** from the pop-up menu.

Figure 8–4 Configuration – Session Broker Property Sheet

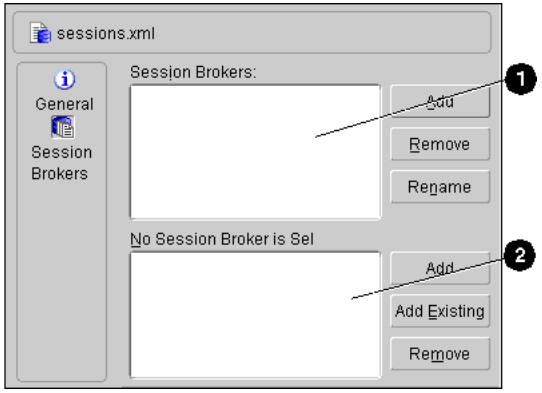


Figure 8–4 calls out the following user-interface components:

1. Session Broker area
2. Sessions area (for currently selected session broker)

3. Click **Add** in the **Session Broker** area to create a new session broker.

   To delete a session broker, select the session broker and click **Remove**. To rename a session broker, select the session broker and click **Rename** (see "Renaming Elements" on page 8-3).

   Note: Each session broker must contain at least one session.

4. To create a new session, select a session broker, then click **Add** in the **Session** area to create a new session for the selected session broker.

   To delete a session, select the session and click **Remove**. To rename a session, select the session and click **Rename** (see "Renaming Elements" on page 8-3).

Working with Sessions

A OracleAS TopLink session describes how OracleAS TopLink communicates with the datasource at runtime. When using the OracleAS TopLink Sessions Editor to configure the session options, choose **Default** to use the values specified in the Oracle Application Server TopLink Foundation Library. Refer to the *Oracle Application Server TopLink Application Developer's Guide* for more information.

To create a new session:

Use one of the following methods to create a new session:

- Right-click on a configuration element in the **Navigator** pane and choose **New** > **Session** from the pop-up menu.
- Select a configuration element in the **Navigator** pane and click the **Session**  button.
- Create a new session for the session broker (see "Working with Session Properties" on page 8-7).

Working with Session Properties

Each session contains tabs and specific properties. By default, sessions contain the following properties:

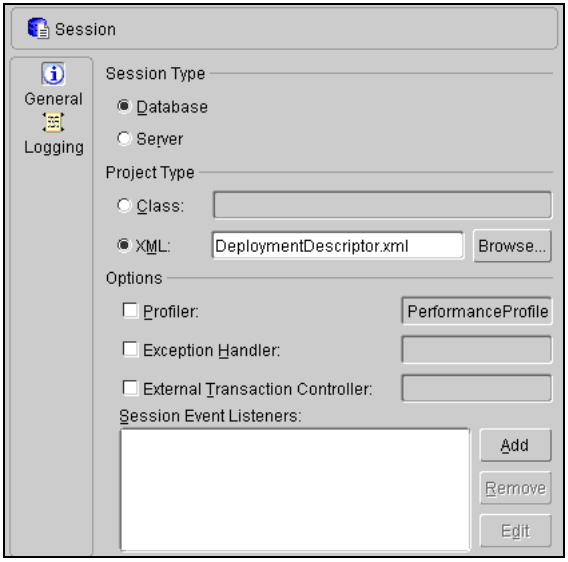
- General
- Logging

In addition, you can specify advanced properties (see "Working with Advanced Session Properties" on page 8-10) for each session.

Setting General Properties

1. Select the session element in the **Navigator** pane. The OracleAS TopLink Sessions Editor displays the session's properties in the **Editor** pane.
2. Click the **General** tab. The General property sheet appears.

Figure 8–5 Session — General Property Sheet



3. Use this table to enter data in each field:

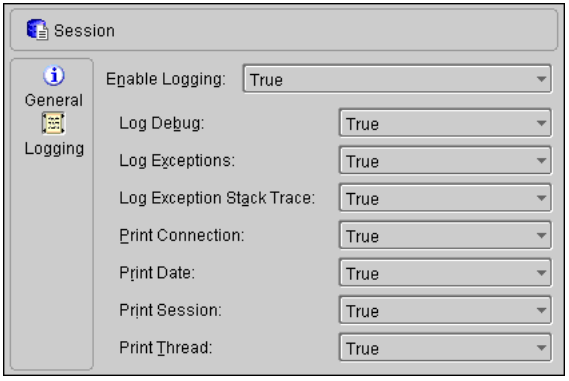
Field	Description
Session Type	Specify whether this is a **Database** or **Server** session (the `<session-type>` tag).
Project Type	Specify whether this is a **Class** or **XML** project (the `<project-class>` or `<project-xml>` tag).
Profiler	Specify the profiler class (the `<profiler-class>` tag).
Exception Handler	Specify the exception handler class (the `<exception-handler-class>` tag).
External Transaction Controller	Specify whether OracleAS TopLink database calls are synchronized with the container's transaction manager (the `<uses-external-transaction-controller>` tag).
Event Listeners	Name of the event listener(s) registered with the session (the `<event-listener-class>` tag). Click **Add** to add a new event listener.

Refer to the *Oracle Application Server TopLink Application Developer's Guide* for more information.

Setting Logging Properties

1. Select the session element in the **Navigator** pane. The OracleAS TopLink Sessions Editor displays the session's properties in the **Editor** pane.
2. Click the **Logging** tab. The Logging property sheet appears.

Figure 8–6 Session — Logging Property Sheet



3. Use this table to enter data in each field:

Field	Description
Logging	Use these fields to specify the session logging properties. Select **Default** to use the logging properties specified by OracleAS TopLink.
Enable Logging	Specify whether the session logs messages (the `<enable-logging>` tag).
Logging Options	Specify whether the session logs individual message types (such as the `<log-debug>` tag).

Refer to the *Oracle Application Server TopLink Application Developer's Guide* for more information.

Working with Advanced Session Properties

OracleAS TopLink sessions may contain advanced properties to specify login and clustering (cache synchronization) information. Select one of the following methods to display the advanced properties:

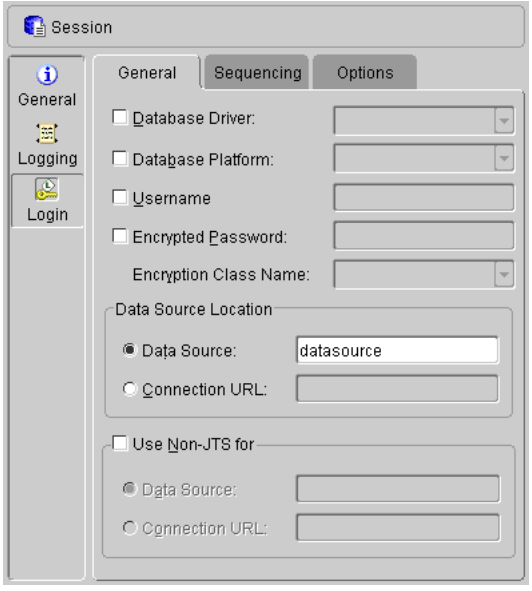
- Right-click the session in the **Navigator** pane and choose **Advanced** > **Login** *or* **Clustering** from the pop-up menu.
- Select the session in the Navigator pane and click **Login** or **Clustering** on the Navigator pane toolbar.

Setting Login Properties

1. Select the session element in the **Navigator** pane. The OracleAS TopLink Sessions Editor displays the session's properties in the **Editor** pane.
2. Click the **Login** tab. The General tab of the Login property sheet appears.

   If the **Login** advanced property is not visible for the session, click the **Login** button on the Navigator toolbar.

Figure 8–7 Login Property Sheet—General Tab

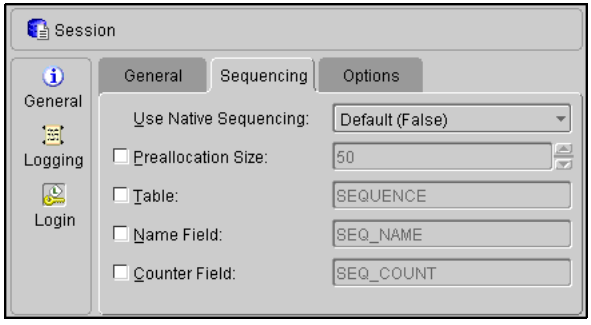


3. Use this table to enter data in each field:

Field	Description
Database Driver	Name of the database driver (the `<driver-class>` tag).
Database Platform	Name of the database platform (the `<platform-class>` tag).
Username	Name used to log into the database (the `<user-name>` tag).
Encrypted Password	Password of the **Username** used to log into the database (the `<encrypted-password>` tag).
Encryption Class Name	When using an **Encrypted Password**, select the specific encryption class (the `<encryption-class-name>` tag).
Data Source Location	
Data Source	Name of the data source used by the session to connect to the database (the `<datasource>` tag).
Connection URL	URL used by the session to connect to the database (the `<connection-url>` tag).
Use Non-JTS for	Specify if the session requires a non-JTS connection. **Note**: Normally, use this option for an application server when using cache synchronization.
Data Source	Name of the non-JTS connection (the `<non-jts-datasource>` tag).
Connection URL	URL used by the non-JTS connection (the `<non-jts-connection-url>` tag).

4. Click the **Sequencing** tab. The Sequencing tab appears.

Figure 8–8 Login Property Sheet—Sequencing Tab

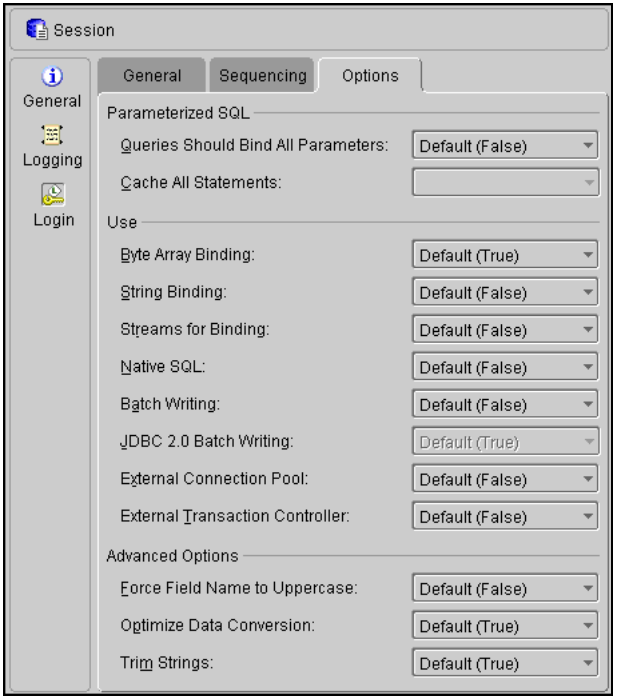


5. Use this table to enter data in each field:

Field	Description
Use Native Sequencing	Specify whether the login uses the database's native sequencing (the `<uses-native-sequencing>` tag).
Preallocation Size	Number of sequences retrieved from the database (the `<sequence-preallocation-size>` tag).
Table	Name of the sequence table (the `<sequence-table>` tag).
Name Field	Field in the sequence **Table** that contains the sequence name (the `<sequence-name-field>` tag).
Counter Field	Field in the sequence **Table** that contains the sequence counter (the `<sequence-counter-field>` tag).

6. Click the **Options** tab. The Options Login property sheet appears.

Figure 8–9 Login Property Sheet – Options Tab



7. Use this table to enter data in each field:

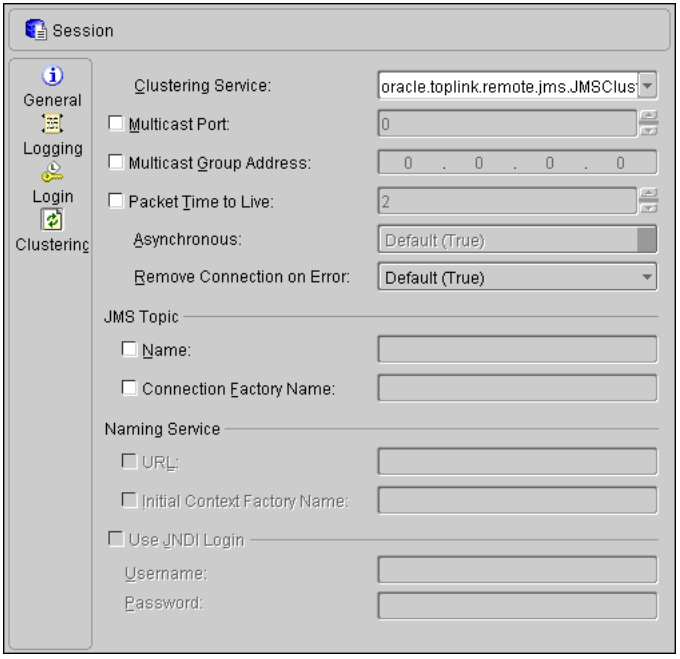
Field	Description
Queries Should Bind All Parameters	Specify whether all queries should bind all parameters (the `<should-bind-all-parameters>` tag).
Cache All Statements	Specify whether OracleAS TopLink caches all statements (the `<should-cache-all-statements>` tag).
Use	
Byte Array Binding	Specify whether OracleAS TopLink binds byte arrays (the `<uses-byte-array-binding>` tag).
String Binding	Specify whether OracleAS TopLink binds strings (the `<uses-string-binding>` tag).
Streams for Binding	Specify whether OracleAS TopLink uses streams when binding byte arrays (the `<uses-streams-for-binding>` tag).
Native SQL	Specify whether OracleAS TopLink uses native SQL for the connection (the `<uses-native-sql>` tag).
Batch Writing	Specify whether the connection batches statements for writing (the `<uses-batch-writing>` tag).
JDBC 2.0 Batch Writing	Specify whether the connection uses JDBC 2.0 batch writing for statements (the `<uses-jdbc20-batch-writing>` tag).
External Connection Pool	Specify whether the connection uses external connection pooling (the `<uses-external-connection-pool>` tag).
External Transaction Controller	Specify whether the session uses an external transaction controller (the `<uses-external-transaction-controller-pool>` tag).
Other Options	
Force Field Name to Uppercase	Specify whether OracleAS TopLink converts all field names to uppercase when generating SQL (the `<should-force-field-names-to-uppercase>` tag).
Optimize Data Conversion	Specify whether OracleAS TopLink optimizes data conversions (the `<should-optimize-data-conversion>` tag). **Note:** Selecting this option may affect the how OracleAS TopLink converts dates.
Trim Strings	Specify whether OracleAS TopLink removes white space (blanks) from strings (the `<should-trim-strings>` tag).

Setting Clustering Properties

1. Select the session element in the **Navigator** pane. The OracleAS TopLink Sessions Editor displays the session's properties in the **Editor** pane.
2. Click the **Clustering** tab. The Clustering property sheet appears.

   If the **Clustering** advanced property is not visible for the session, click the **Clustering** button  in the Navigator toolbar.

Figure 8–10 Clustering Property Sheet



3. Use this table to enter data in each field:

Field	Description
Clustering Service	Name of the clustering service used for cache synchronization (the `<clustering-service>` tag).
Multicast Port	Multicast port used by the clustering service to listen for new sessions (the `<multicast-port>` tag).

Field	Description
Multicast Group Address	IP address used by the clustering service to listen for new sessions (the `<multicast-group-address>` tag).
Packet Time to Live	The maximum number of hops a packet will be broadcast (the `<packet-time-to-live>` tag).
Asynchronous	Specify whether the cache manager does not require that all sessions be synchronized before returning (the `<is-asynchronous>` tag).
Remove Connection on Error	Specify whether OracleAS TopLink removes the connection from the session in case of a configuration error (the `<should-remove-on-error>` tag).
JMS Topic	
Name	Specify the name of the JMS topic (the `<jms-topic-name>` tag).
Connection Factory Name	Specify the name of the JMS topic connection factory (the `<jms-topic-connection-factory-name>` tag).
Naming Service	
URL	URL of the global namespace for the cache synchronization (the `<naming-service-url>` tag). **Note**: Normally, this is the URL of the JNDI service.
Initial Context Factory Name	Name of the element used for synchronization (the `<naming-service-initial-context-factory-name>` tag).
Use JNDI Login	Specify whether cache synchronization uses a JNDI service.
Username	Name used to log into the JNDI service (the `<jndi-user-name>` tag).
Password	Password used to log into the JNDI service (the `<jndi-password>` tag).

Refer to the *Oracle Application Server TopLink Application Developer's Guide* for more information.

Working with Connection Pools

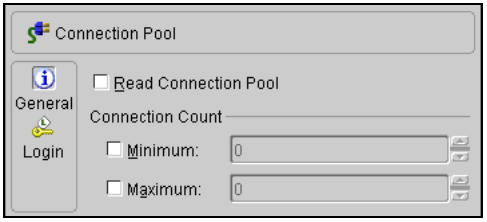
OracleAS TopLink uses connection pools to manage and share the connections used by the server and client sessions among multiple clients. This feature reduces the number of connections required by the application server and allows the server to support many clients.

To create a new connection pool, right-click a session element in the **Navigator** pane and choose **New** > **Connection Pool** from the pop-up menu, or click the **Connection Pool** button.

Setting General Properties

1. Select the connection pool element in the **Navigator** pane. The OracleAS TopLink Sessions Editor displays the connection pool's properties in the **Editor** pane.
2. Click the **General** tab. The General property sheet appear.

Figure 8–11 Connection Pool—General Property Sheet



3. Use this table to enter data in each field:

Field	Description
Read Connection Pool	Specifies whether OracleAS TopLink uses a read connection pool (the `<is-read-connection-pool>` tag).
Connection Count	Specify the **Minimum** and **Maximum** read connections in the OracleAS TopLink connection pool (the `<min-connections>` and `<max connections>` tags).

Refer to the *Oracle Application Server TopLink Application Developer's Guide* for more information.

Setting Login Properties

1. Select the connection pool element in the **Navigator** pane. The OracleAS TopLink Sessions Editor displays the connection pool's properties in the **Editor** pane.
2. Click the **Login** tab. The General tab of the Login property sheet appears. See "Setting Login Properties" on page 8-10

Working with the Source

Use this procedure to display the source code in the `sessions.xml` configuration at any time.

To View the Source:

Select a configuration in the **Navigator** pane and choose **View** > **Source** from the menu. The Sessions Configuration Editor opens the configuration in your editor.

If you have not selected a text editor (see "General Preferences" on page 1-11), the OracleAS TopLink Sessions Editor prompts you select a text editor.

A

Object Model Requirements

Oracle Application Server TopLink requires that classes must meet certain minimum requirements before they can become *persistent*. OracleAS TopLink also provides alternatives to most requirements. OracleAS TopLink uses a non-intrusive approach employing a meta-data architecture that allows for almost no object model intrusions.

This section summarizes OracleAS TopLink's object model requirements. Unlike other products, OracleAS TopLink *does not* require any of the following:

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

Persistent Class Requirements

The attribute requirements vary, depending on your Java version. When employing Java 2, you can use direct access on private or protected attributes. Refer to Chapter 4, "Understanding Descriptors" for more information on direct and method access.

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

In Java 2, OracleAS TopLink provides *transparent* indirection for `Collection` and `List` attribute types for any collection mappings. Using transparent indirection does not require the usage of `ValueHolderInterface` or any other object model requirements.

Refer to Chapter 6, "Understanding Relationship Mappings" for more information on indirection and transparent indirection.

Constructor Requirements

By default, OracleAS TopLink uses and requires default (zero argument) constructors to create objects from the database. It is also possible to instruct OracleAS TopLink to use a different constructor, static method, or factory. Refer to "Working with Instantiation Policy" on page 4-52 for more information.

Remote Session Requirements

If you employ the OracleAS TopLink Remote Session, all persistent classes to be used remotely must implement the `Serializable` interface.

B

Tutorials

This section contains the following tutorials that illustrate how to use Oracle Application Server TopLink Mapping Workbench.

- Introductory Tutorial
- Advanced Tutorial
- Completing the Tutorials

Before You Begin

The tutorials require classes that are built from the OracleAS TopLink Three Tier example project. This example project is included as part of the complete OracleAS TopLink installation. You must build and run the example before beginning the tutorial. Refer to *<ORACLE_HOME>*`\toplink\doc\examples.htm` for complete information.

Introductory Tutorial

In the introductory tutorial, you will create mappings from a simple database three-tier application using OracleAS TopLink Mapping Workbench. You will learn how to:

- Create a new project
- Enable and add Java classes (provided with the tutorial)
- Create and import database tables
- Associate descriptors to tables
- Implement mappings

By the end of the tutorial, you will be able to store data from a Java class into a relational database and access existing database information from Java classes.

Overview

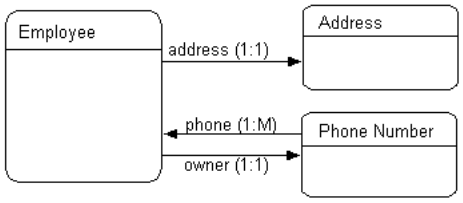
This tutorial project manages the employee database at the ACME Leisure Company. The system tracks each employee's name, address, and phone number.

The system uses these classes:

- `Employee` – Represents both full-time ACME employees and temporary contractors working on ACME projects. It includes the employees' personal information as well as references to their home addresses and phone numbers.
- `Address` – Represents the employee's home address. The class contains country, street, city, province, and postal code information.
- `PhoneNumber` – Contains the telephone number(s) for each employee and contractor (number, area code, and type information). The class also includes a reference to the employee who owns the phone number.

Figure B–1 illustrates the object model for this system.

Figure B–1 Simple ACME Object Model



Creating the Database Schema

The ACME employee system stores the employee data in three database tables. Later in this tutorial, you will be able to create these tables from the OracleAS TopLink Mapping Workbench or import them from your database application.

> **Note:** The column types listed here are generic; the actual column types depend on the database used.

Table B–1 EMPLOYEE Table

Column Name	Column Type	Details
EMP_ID	NUMERIC(15)	Primary key
F_NAME	VARCHAR(40)	
L_NAME	VARCHAR(40)	
ADDRESS_ID	NUMERIC(15)	

Table B–2 ADDRESS Table

Column Name	Column Type	Details
ADDRESS_ID	NUMERIC(15)	Primary key
COUNTRY	VARCHAR(80)	
STREET	VARCHAR(80)	
CITY	VARCHAR(80)	
PROVINCE	VARCHAR(80)	
P_CODE	VARCHAR(20)	

Table B–3 PHONE Table

Column Name	Column Type	Details
EMP_ID	NUMERIC(15)	Primary key
AREA_CODE	CHAR(3)	
P_NUMBER	CHAR(7)	
TYPE	VARCHAR(15)	Primary key

After creating these ACME database tables, you are ready to begin the tutorial.

Creating a New Project

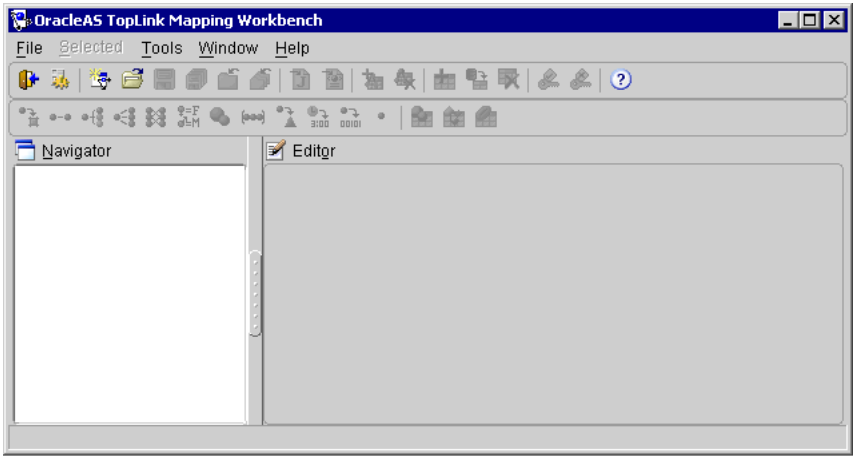
OracleAS TopLink Mapping Workbench stores project information in the `.mwp` file and associated folders. Always start an OracleAS TopLink Mapping Workbench project in a new folder.

To create a new project:

1. Start OracleAS TopLink Mapping Workbench. From the Windows **Start** menu, select **Programs** > **OracleAS TopLink** > **Mapping Workbench**.

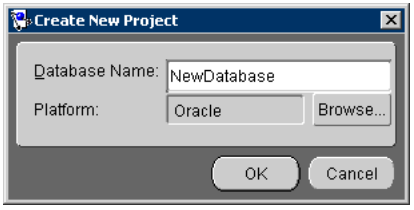
   The splash screen appears, followed by the OracleAS TopLink Mapping Workbench window.

Figure B–2 OracleAS TopLink Mapping Workbench



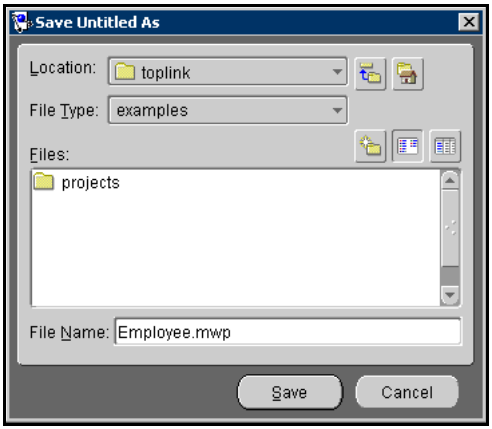
2. Click the **New Project** button on the toolbar or select **File** > **New Project** from the menu. The Create a New Project dialog box appears.

Figure B–3 Create New Project



3. From the Create New Project dialog box:
   - In the **Database Name** field, type `INTRO_TUTORIAL_DB`.
   - In the **Platform** field, click **Browse** button and select the appropriate database platform. Contact your database administrator if you need additional information.
4. Click **OK**. The Save As dialog box appears.

Figure B–4 Save As

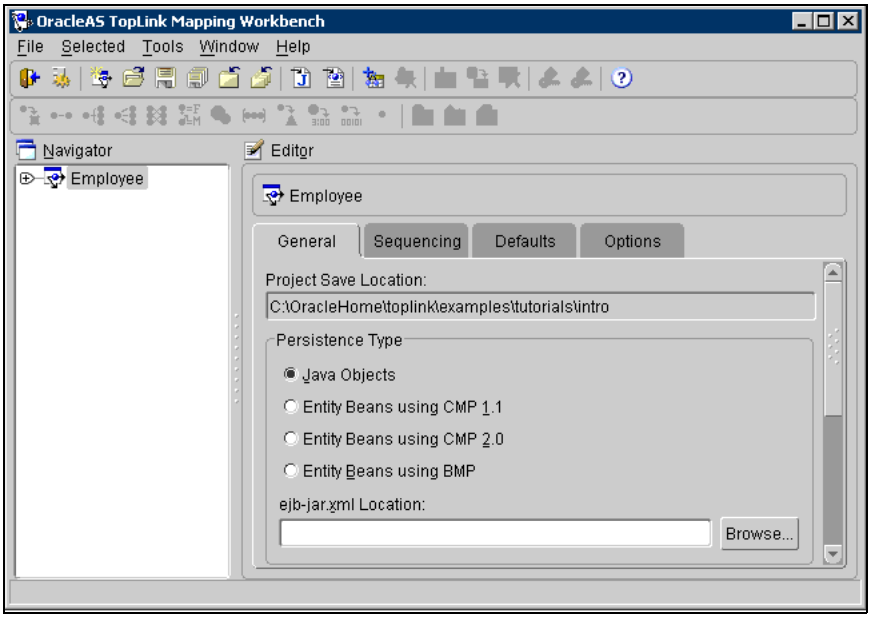


5. Select the folder in which to save the **Employee** project.

> **Note:** Always use a new folder to save a project.

6. In the **File Name** field, type `Employee.mwp`.
7. Click **Save** to save your work and return to the OracleAS TopLink Mapping Workbench window.

Figure B–5 OracleAS TopLink Mapping Workbench



8. Click **Save** on the toolbar or select **File** > **Save** to save the project.

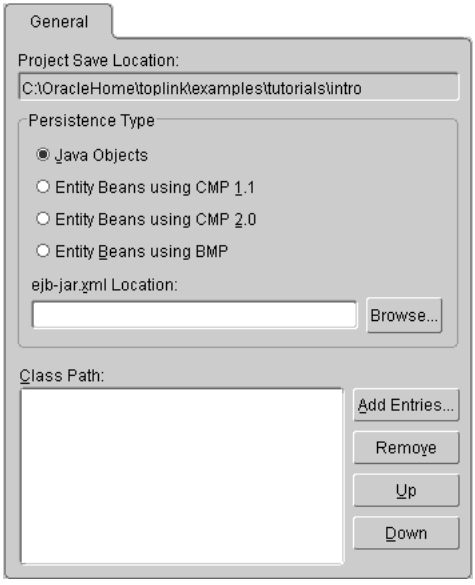
> **Note:** OracleAS TopLink Mapping Workbench does not automatically save your work; remember to save periodically.

Setting the Project's Class Path

Each OracleAS TopLink project uses a class path—a set of directories, `.jar` files, and `.zip` files—when importing Java classes and defining object types. In this tutorial, you will use information from the OracleAS TopLink three tier example project. Refer to *<ORACLE_HOME>*`\toplink\doc\examples.htm` for complete information.

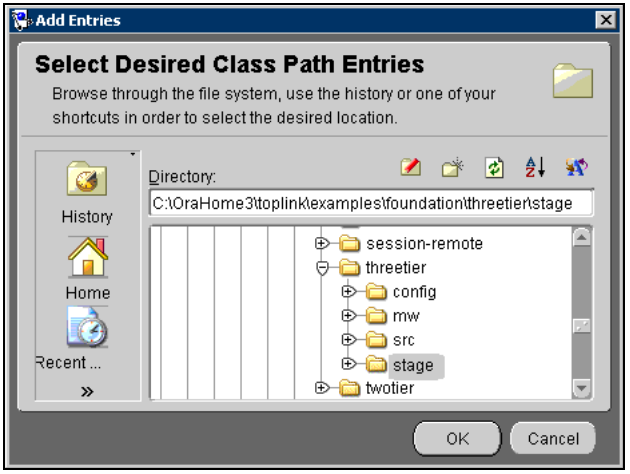
1. In the OracleAS TopLink Mapping Workbench, highlight the `Employee` project in the **Navigator** pane.
2. In the **Editor** pane on the right-hand side of the OracleAS TopLink Mapping Workbench window, click the **General** tab.

Figure B–6 General Tab



3. Click **Add Entries**. The Add Entry dialog box appears.

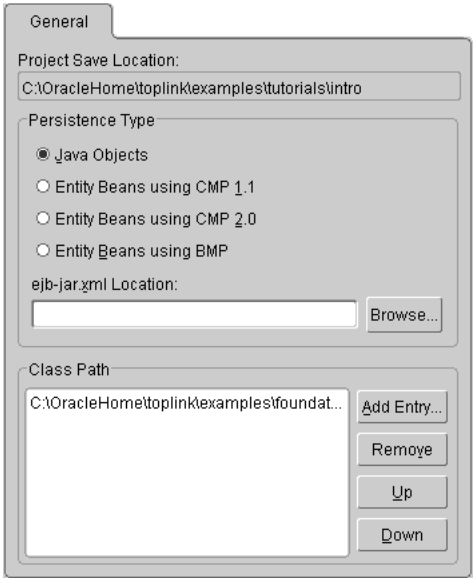
Figure B–7 Add Entries



4. Browse to the <*ORACLE_HOME*>\toplink\examples\foundation\threetier\stage directory and click **OK**.

Note: If the `stage` directory does not exist, you must create the OracleAS TopLink Three Tier example project. See "Before You Begin" on page B-1 for more information.

Figure B–8 General Tab with Class Path Information



5. Click **Save** on the toolbar or choose **File** > **Save** to save the project.

Enabling Your Java Classes

The `Employee` model uses three classes:

- `Employee` class has a name attribute and privately owned `PhoneNumber` and `Address` relationships.
- `PhoneNumber` class has attributes describing the phone number information and a relationship that describes the owner of the `PhoneNumber`.
- `Address` class has attributes describing the employee's mailing address.

You must enable the `Employee`, `PhoneNumber`, and `Address` classes (provided in the `examples.session.threetier.model package`) for this tutorial, as "Generating the Class Definitions" on page B-11 describes.

Table B–4 shows how the classes relate to the database tables.

Table B–4 Employee Classes and Database Tables

Column	Class Attribute	Database Type	Java Type
EMPLOYEE	Employee		
EMP_ID	id	NUMERIC(15)	BigDecimal
NAME	name	CHAR(40)	String
ADDRESS_ID	address	NUMERIC(15)	Address
not applicable	phoneNumbers	*not applicable*	Vector
ADDRESS	Address		
ADDRESS_ID	id	NUMERIC(15)	BigDecimal
COUNTRY	country	VARCHAR(80)	String
STREET	street	VARCHAR(80)	String
CITY	city	VARCHAR(80)	String
PROVINCE	province	VARCHAR(80)	String
P_CODE	postalCode	VARCHAR(20)	String
PHONE	PhoneNumber		
AREA_CODE	areaCode	CHAR(3)	String
P_NUMBER	number	CHAR(7)	String
EMP_ID	owner	NUMERIC(15)	Employee
TYPE	type	VARCHAR(15)	String

Note: Supplying each of the class members in OracleAS TopLink-enabled classes with accessor methods is good programming practice. This tutorial provides the `get` and `set` methods for each attribute. For example, the `Employee` class should have an `addPhoneNumber()` method to allow a new `PhoneNumber` to store a reference to its parent.

Example B–1 Accessor Method

The following code example illustrates providing accessor methods.

```
// addPhoneNumber method of the Employee class allows the phoneNumber to set a
reference to the Employee that owns it.
```

```
public void addPhoneNumber(PhoneNumber phoneNumber)
{
    getPhoneNumbers().addElement(phoneNumber);
    phoneNumber.setOwner(this);
}
```

Generating the Class Definitions

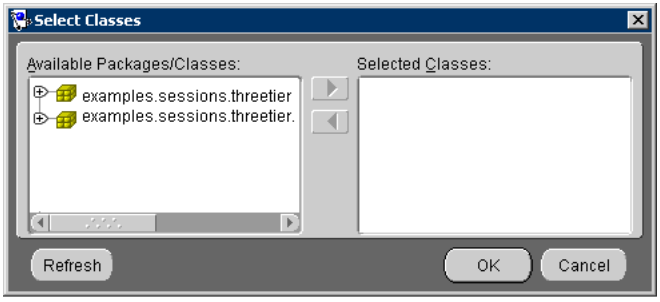
You must generate an OracleAS TopLink descriptor for each Java class in the project.

To create descriptors from the class definition file:

1. From the **Navigator** pane, click the `Employee` project.
2. Click the **Add or Refresh Classes** button  or choose **Selected** > **Add or Refresh Classes** from the menu. The Select Classes dialog box appears.

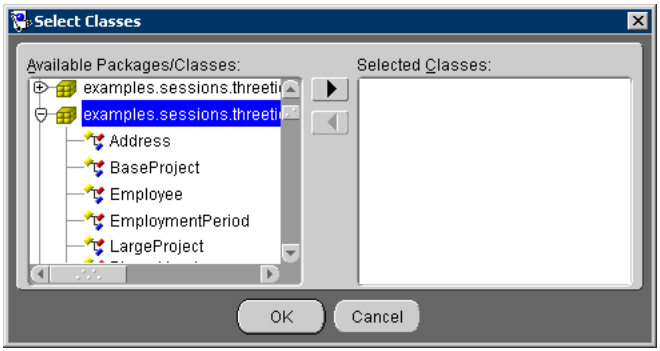
   You can also create descriptors by right-clicking on the project in the **Navigator** pane and selecting **Add or Refresh Classes** from the pop-up menu.

Figure B–9 Select Classes



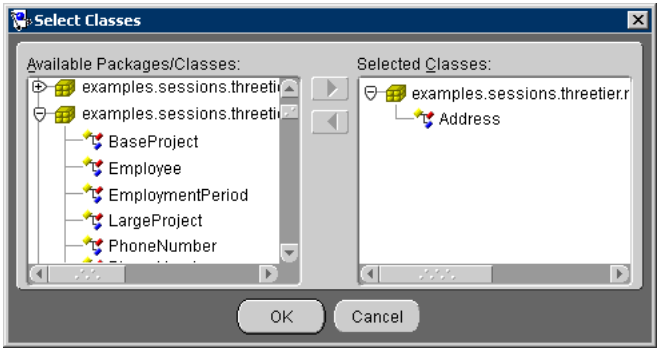
3. Locate the `examples.sessions.threetier.model` package. Click the plus sign (⊞) to expand that package (or double-click the name to expand the package).

Figure B–10 Demo Classes



4. Highlight the `Address` class and click the  button. OracleAS TopLink copies the `Address` class to the **Selected Classes** pane.

Figure B–11 Selected Class



5. Repeat Step 4 for `Employee` and `PhoneNumber` classes in that package. (Or, highlight both classes using **Shift+click** or **Ctrl+click** as necessary, and click  once to import all the remaining classes.)
6. Click **OK** to import the classes. OracleAS TopLink creates a descriptor for each class and an unmapped mapping for each attribute. The descriptors and their attributes appear in the **Navigator** pane.

Note: If an error occurs, check that the given classes are included in the class path and that JDK has been installed correctly.

Figure B–12 OracleAS TopLink Mapping Workbench with Employee Project

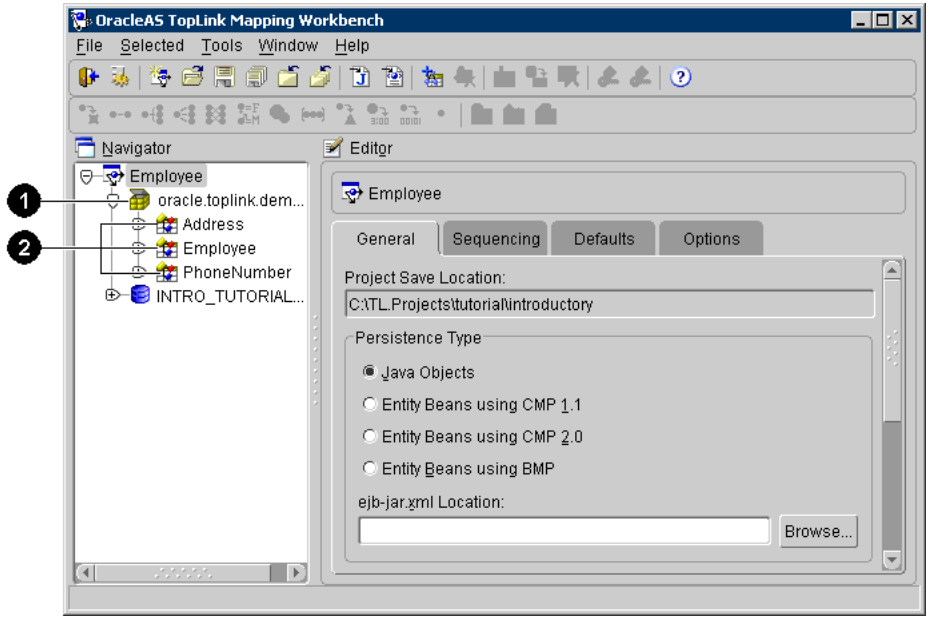


Figure B–12 identifies the following user-interface elements:

1. Package
2. Class/descriptors

7. Save your changes. Click **Save** or choose **File** > **Save** from the menu.

Logging Into the Database

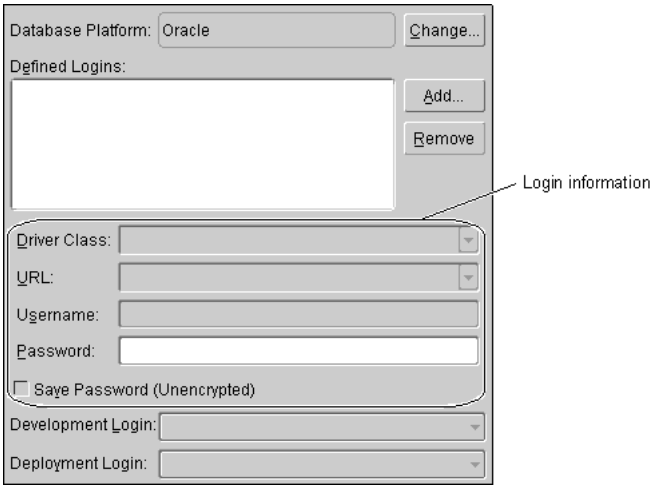
You can enter database table information directly from OracleAS TopLink Mapping Workbench or import the tables from the database. You must log into the database to obtain the table information and to generate table files.

Note: You must include your database driver on the system classpath – not the OracleAS TopLink Mapping Workbench project classpath.

---

1. Select the **INTRO_TUTORIAL_DB** database object in the **Navigator** pane. The Database **Editor** property sheet appears in the **Editor** pane of the OracleAS TopLink Mapping Workbench window.

Figure B–13 Database Properties



2. Click **Add** to create a new database login.
3. In the **Defined Login** area, select the newly added login. Contact your database administrator for the necessary database login information (driver, URL, username, and password).
4. Click the **Log In to Database** button or choose **Selected** > **Log In to Database** from the menu. The database icon changes to .

Note: OracleAS TopLink Mapping Workbench supports connecting to the database through JDBC. Make sure you have installed, configured, and tested your JDBC driver before attempting to connect.

If OracleAS TopLink Mapping Workbench is unable to connect to the database, contact your database administrator to ensure that the database login parameters have been entered correctly and your JDBC driver has been installed correctly. If problems persist, test your driver connectivity. See the Appendix C, "Troubleshooting" for details.

Creating Tables

You can enter database table information (as "Creating the Database Schema" on page B-3 specifies) directly from OracleAS TopLink Mapping Workbench or import the tables from the database.

- To create the tables from OracleAS TopLink Mapping Workbench, continue with "Creating Tables Using the OracleAS TopLink Mapping Workbench" on page B-15.
- To create the tables by importing from the database, continue with "Importing Tables from the Database" on page B-17.

Creating Tables Using the OracleAS TopLink Mapping Workbench

Use this procedure to create the database tables from the OracleAS TopLink Mapping Workbench.

Caution: Do not use this procedure if you plan to import the tables from a database.

Creating the Table Definitions

Use this procedure to create table definitions using the OracleAS TopLink Mapping Workbench. Later, you can create the actual tables on the database.

To create the tables:

1. Select the database in the **Navigator** pane and click the **Add New Table** button  or right-click the database in the **Navigator** pane, and choose **Add New Table.** The New Table dialog box appears.
2. Type `ADDRESS` for the table name, and click **OK**.

Note: Leave the **Catalog** and **Schema** fields blank.

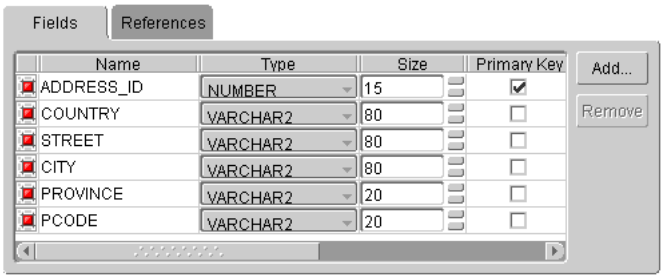
---

3. Click the **Fields** tab in the **Editor** pane.

4. Click **Add** to add each database field for the ADDRESS table. Refer to the tables in "Creating the Database Schema" on page B-3 for the correct field information. Be sure to indicate the primary key(s) for each table.

> **Note:** Use the scroll bar to view additional fields for each database field (such as the **Primary Key**). See "Preparing the Primary Keys" on page B-21 for more information.

Figure B–14 Database Fields Tab



5. Save your changes. Click the **Save Project** button or choose **File** > **Save** from the menu.

Repeat this procedure for the EMPLOYEE and PHONE tables.

Creating the Tables on the Database

After defining the tables in OracleAS TopLink Mapping Workbench, you can automatically create the tables on the database.

To create tables on the database:

1. Right-click one of the database tables in the **Navigator** pane, and choose **Create on Database** > **All Tables** from the pop-up menu. The system displays a message indicating that the three tables were created.

> **Note:** To use the **Create on Database** option, you must be logged into the database.

If the Confirm Replace dialog box appears, it means that an existing table on the database has the same name. Check with your database administrator before replacing any table. The existing table may have been created by:

- Someone else doing the tutorial previously (in which case you could click the **Yes to All** button safely)
- Someone using the same database name for a business project (in which case you should not replace it)

2. Click **OK** to close the dialog box and return to the OracleAS TopLink Mapping Workbench window.
3. On the toolbar, click the **Logout of Database** button  or choose **Selected** > **Log Out** from the menu.
4. Save your changes. Click the **Save Project** button  or choose **File** > **Save** from the menu.

Continue with "Mapping Classes and Tables in the Descriptor" on page B-19.

Note: OracleAS TopLink Mapping Workbench can generate Data Definition Language (DDL) creation scripts that can be used to create tables on the desired database.

If the table creation fails, there might have been a problem with the DDL, or you may not have permission to create tables on the database. Make sure you set the target database to the correct database platform on login. Because the DDL may not be compatible with some databases, you may have to edit the generated DDL and execute the DDL manually.

In addition, ensure that the **Database Platform** field on the Database property sheet (see Figure B–13 on page B-14) matches your database driver information.

Importing Tables from the Database

Use this procedure if you have already created tables in your database and want to import these tables directly into the OracleAS TopLink Mapping Workbench.

Caution: Do not use this procedure if you plan to create the tables directly from OracleAS TopLink Mapping Workbench.

To import tables from the database:

1. Click the **Login to Database** button or choose **Selected** > **Login** from the menu.
2. Select the database in the **Navigator** pane and click the **Add or Update Existing Tables from Database** button, or right-click the database and choose **Add or Update Existing Tables From Database** from the pop-up menu.

   The Import tables from database dialog box appears.

Figure B–15 Import Tables from Database

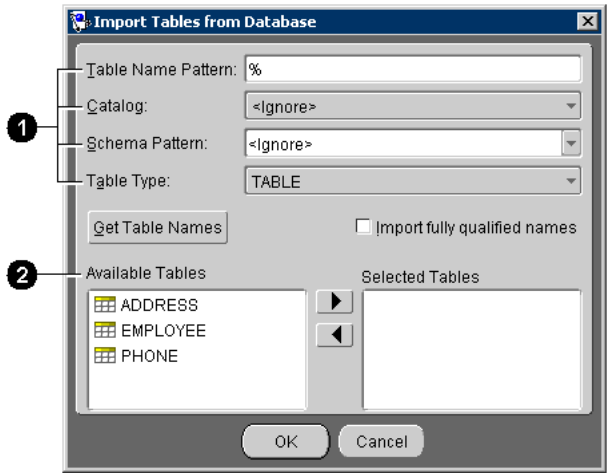


Figure B–15 calls out the following user-interface elements:

1. Use the filters to specify database tables to select for import.
2. The **Available Tables** displays the database tables that match the filter.

3. Click the **Get Table Names** button to display all tables in the database.

> **Note:** You can use the table filters to specify which database tables are available for import. For this tutorial, leave the filters as their default values.

4. Select the `ADDRESS` table in the **Available Tables** pane and then click the button. The `ADDRESS` table moves to the **Selected Tables** pane.

5. Repeat Step 4 for the `EMPLOYEE` and `PHONE` tables.
6. Click **OK** to add the selected tables to the **Employee** project.
7. To display the details of the imported tables, select a table in the **Navigator** pane and click the **Fields** tab in the **Editor** pane.
8. Click the **Log Out of Database** button  or choose **Selected** > **Log Out of Database** from the menu.
9. Save your changes. Click the **Save Project** button  or choose **File** > **Save** from the menu.

Continue with "Mapping Classes and Tables in the Descriptor" on page B-19.

Mapping Classes and Tables in the Descriptor

When you create a new project and generate class definitions, OracleAS TopLink Mapping Workbench automatically creates descriptors. However, these descriptors do not contain any information about how the classes are associated with the tables. This section describes how to store associations in a descriptor, which can then be used by a Java application to make the classes persistent.

This section contains procedures to map the classes to tables for the ACME project. After mapping the descriptors, you can access the database from a Java application.

Mappings

The OracleAS TopLink *mapping* describes the way an attribute is stored in, and retrieved from, a database. For example, the `name` attribute of the `Employee` class maps to the `NAME` column of the `EMPLOYEE` table.

Descriptors

A *descriptor* stores the class-to-table mappings for a class. OracleAS TopLink Mapping Workbench stores the descriptors in XML files in the `Descriptor` directory. At run time, OracleAS TopLink creates instances of the `Descriptor` class for each of the descriptor files and stores them in a database session.

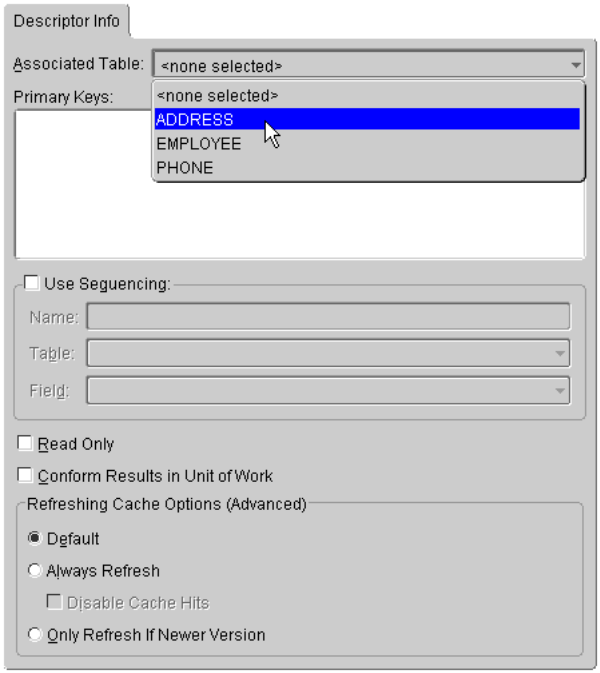
Mapping Classes to Tables

Use this procedure to associate the Java classes with database tables.

To map Java classes to a table:

1. Select the `Address` descriptor from the **Navigator** pane.
2. Click the **Descriptor Info** tab of the **Editor** pane.
3. In the **Associated Table** drop-down menu, choose the `Address` table.

Figure B–16 Descriptor Info Tab



Note: A warning message appears in the status bar, indicating that the primary key fields are unmapped. This is addressed later in the tutorial.

---

4. Repeat steps 1 – 3 to map:
   - `Employee` class to the `EMPLOYEE` table
   - `PhoneNumber` class to the `PHONE` table
5. Save your changes. Click the **Save Project** button or choose **File** > **Save** from the menu.

Although you have mapped the descriptors to specific database tables, the class *attributes* have not yet been mapped to the tables' columns. You will map the attributes later in this tutorial.

Preparing the Primary Keys

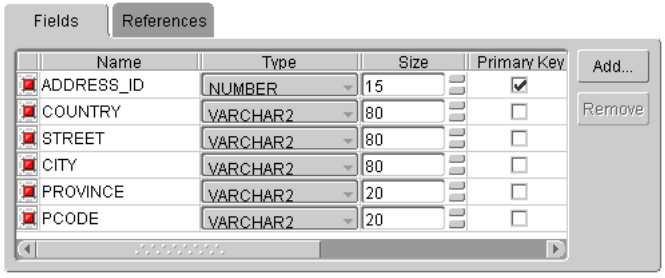
A table's primary key is the field (or fields) used to uniquely identify its records. The `PHONE` table has a compound primary key (`EMP_ID` and `TYPE` fields).

Database tables often use a *sequence number* as the primary key. Sequence numbers are sequential, artificially generated fields, outside of the problem domain, that uniquely identify a record. OracleAS TopLink supports sequence numbers through the database's native support, such as in Oracle and Sybase, or by maintaining a sequence table. If sequence numbers are enabled for a class, they are generated and incremented whenever an object of that class is inserted into a database.

To specify the primary key:

1. In the **Navigator** pane, select the `ADDRESS` database table.
2. On the **Fields** tab of the **Editor** pane, make sure the `ADDRESS_ID` column is selected as a **Primary Key**.

Figure B–17 Database Table Fields Tab



3. Repeat Step 1 – 2 for the other tables:
   - EMPLOYEE table – Set the EMP_ID field as the primary key.
   - PHONENUMBER table – Set both the EMP_ID and TYPE fields as primary keys.
4. Save your changes. Click the **Save Project** button or choose **File** > **Save** from the menu.

Setting the Sequence Table

The ACME system uses sequence numbers for the EMPLOYEE and ADDRESS tables. You must explicitly create a sequence table, then apply it to your project. You must also create the sequence table on the database.

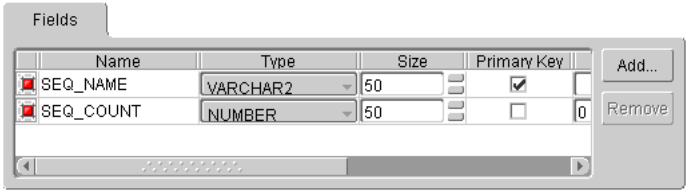
To create a sequence table:

1. Select the database in the **Navigator** pane, and log into the database by clicking the **Login** button or by right-clicking on the database in the **Navigator** and choosing **Login** from the pop-up menu.
2. Select **Database** in the **Navigator** pane and click the **Add New Table** button . The New Table dialog box appears.

Figure B–18 New Table

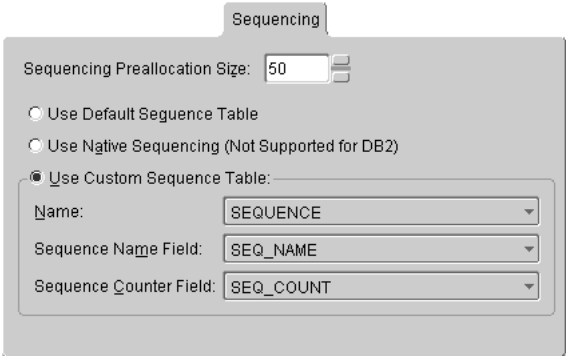
3. Create a table named SEQUENCE. Leave **Catalog** and **Schema** blank, unless required by your specific database.
4. Add the following fields to the table:
   - SEQ_NAME (VARCHAR2 type, size 38)
   - SEQ_COUNT (NUMBER type, size 38)

Figure B–19 Database Table Fields Tab



5. Set the `SEQ_NAME` field as the primary key.
6. Log out of the database by right-clicking the **Database** in the **Navigator** pane and choosing **Log Out** from the pop-up menu.
7. Select the **Project** in the **Navigator** pane.
8. Select the project's **Sequencing** tab in the **Editor** pane.

Figure B–20 Sequencing Tab



9. Select **Use Custom Sequence Table** and use the drop lists to choose the **Name**, **Sequence Name Field**, and **Sequence Counter Field**, as Figure B–20 illustrates.
10. Save your changes. Click the **Save Project** button or choose **File** > **Save** from the menu.

Note: You will set the individual sequence names for each table later.

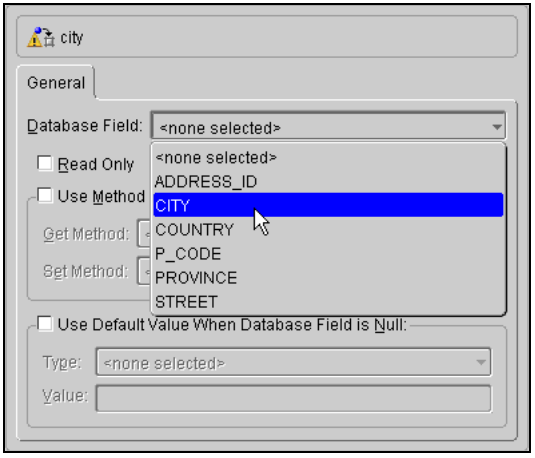
Implementing Direct-to-Field Mappings

The `Address` class does not reference any other classes. Its attributes map directly to database fields as a *direct-to-field* mapping.

To map the Address class attributes directly to the ADDRESS columns:

1. Expand the `Address` descriptor in the **Navigator** pane.
2. Click the `city` attribute.
3. Click the **Direct-to-Field** mapping button on the mapping toolbar. The Direct-to-Field mapping tab appears in the **Editor** pane.
4. Use the **Database Field** drop-down list to choose the `CITY` field.

Figure B–21 Direct-to-Field Mapping General Tab



5. Repeat steps 2 – 4 to map the remaining attributes in the `ADDRESS` table.
   - Map the `COUNTRY` attribute to `COUNTRY` field.
   - Map the `ID` attribute to `ADDRESS_ID` field.

     Note: When you map the `ADDRESS_ID` field (the primary key), the OracleAS TopLink Mapping Workbench removes the warning icon for the `Address` descriptor.

   - Map the `POSTALCODE` attribute to `P_CODE` field.

- Map the PROVINCE attribute to PROVINCE field.
- Map the STREET attribute to STREET field.

6. Save your changes. Click the **Save Project** button  or choose **File** > **Save** from the menu.

Setting the Sequence Name

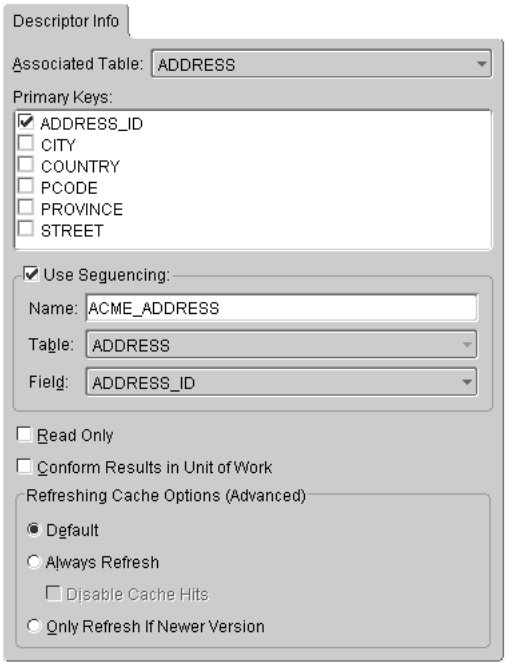
The Address and Employee classes use nonnative sequencing for primary keys.

Note: The sequence name is the value of a row stored in the sequence table. When you create tables for your own projects, you must insert this row into the table using the OracleAS TopLink SchemaManager.

To set the sequencing for the Address and Employee classes:

1. Select the **Address** descriptor in the **Navigator** pane.
2. Select the **Descriptor Info** in the **Editor** pane.
3. Select the **Use Sequencing** check box.
4. In the **Name** field, type ACME_ADDRESS and use the drop lists to choose the **Table** and **Field**, as in Figure B–22.

Figure B–22 Descriptor Info Tab



5. Save your changes. Click the **Save Project** button or choose **File** > **Save** from the menu.
6. Repeat steps 1 – 4 to set the sequencing for the `Employee` class. Use `ACME_EMPLOYEE` as the **Name**, and choose `EMPLOYEE` and `EMP_ID` from the **Table** and **Field** drop-down lists, respectively.
7. Save your changes. Click the **Save Project** button or choose **File** > **Save** from the menu

When the descriptors are registered with a database session, this information is entered into the `SEQUENCE` table. The session tracks the sequencing information.

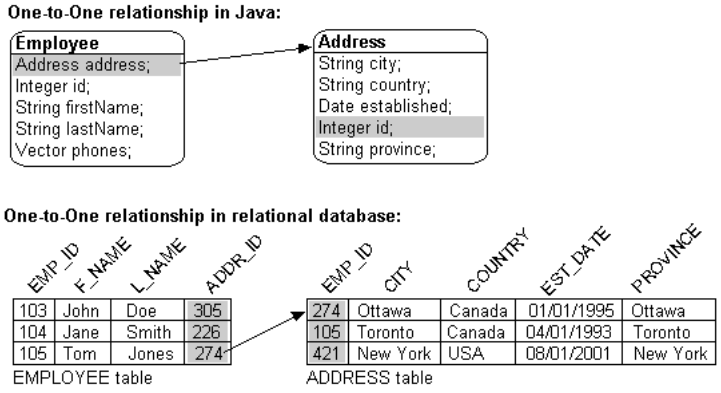
Creating One-to-One Mappings Between Objects

In the `Employee` class, the `name` and `id` attributes map directly to the `EMPLOYEE` table columns. The `phoneNumbers` and `address` attributes refer to other Java classes, rather than referring directly to columns in the database.

1. Map the `firstName` and `lastName` attributes as a direct-to-field mapping to the `F_NAME` and `L_NAME` fields, respectively.
2. Map the `id` attribute as a direct-to-field mapping to the `EMP_ID` field (the primary key).

Only one home address is associated with each employee, so the `address` attribute requires a *one-to-one mapping* with the `Address` class. Figure B–23 illustrates a sample one-to-one mapping.

Figure B–23 One-to-One Mappings

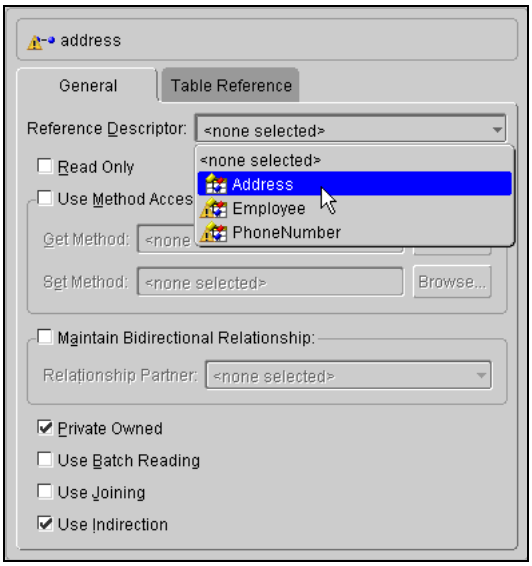


To create a one-to-one mapping

1. Select the Employee's `address` attribute in the **Navigator** pane, then click the **One-to-one Mapping** button on the mapping toolbar.

   The **Editor** pane displays the appropriate information for a one-to-one relationship to be specified.

Figure B–24 One-to-One Mapping General Tab



2. Use the **Reference Descriptor** drop-down list in the to select `Address` as the reference descriptor.
3. Select the **Use Indirection** check box.
4. Ensure that the **Private Owned** check box is enabled.

   This allows the `Address` object to be created, updated, or deleted automatically when the Employee owning it is changed.

Foreign Key References

One-to-one mappings use the relational database concept of *foreign keys* to access other classes stored in the database. You must specify the foreign key information in the descriptor so that OracleAS TopLink knows how to search for a referenced object, as Figure B–23 shows.

1. Click the **Table Reference** tab.
2. Create a new table reference by clicking the **New** button.
3. In the New Reference dialog box, create a reference whose:
   - Name is `EMPLOYEE_ADDRESS`
   - Source table is `EMPLOYEE`

- Target table is ADDRESS

Note: If you leave the **Name** field blank, OracleAS TopLink automatically builds the name as <SourceTable>_<TargetTable>.

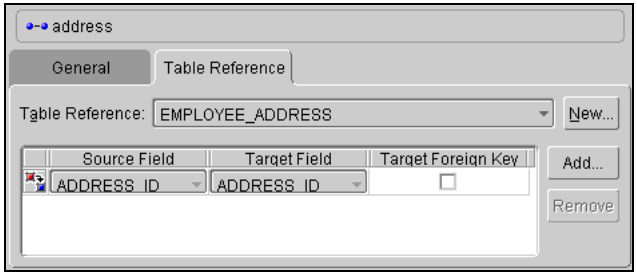
Select the **On Database** option if you want to create the reference on the database when you create the tables. OracleAS TopLink does not require that you actually have the constraint on the database, but you may wish to do this for other reasons. Consult your database administrator for more information.

Note: The mapping is **from** the EMPLOYEE table, ADDRESS_ID *attribute* **to** the ADDRESS *table*.

---

4. Choose EMPLOYEE_ADDRESS from the **Table Reference** drop-down list.
5. Click the **Add** button to define the foreign key fields.
   - In the **Source Field** column, choose ADDRESS_ID (foreign key).
   - In the **Target Field** column, choose ADDRESS_ID (primary key).
   - Leave the **Target Foreign Key** option unchecked.

Figure B–25 One-to-One Mapping Table Reference Tab

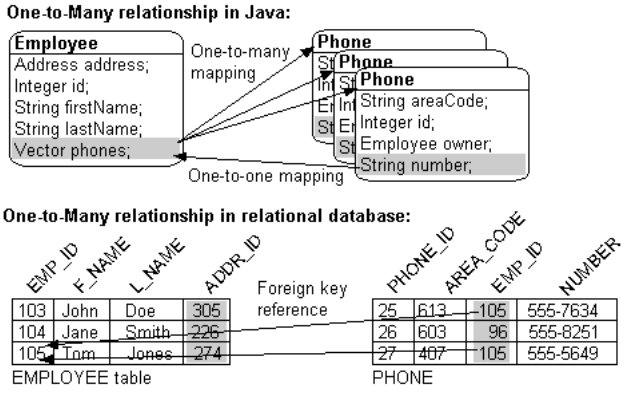


6. Save your changes. Click the **Save Project** button or choose **File** > **Save** from the menu.

Creating One-to-Many Mappings

To map an attribute to a Java collection such as a `Vector`, the application must make a one-to-many mapping for the class owning the collection, and a one-to-one mapping back from the class being referenced. The one-to-one mapping in the referenced class is implemented as a foreign key to the source class.

Figure B–26 One-to-Many Mappings



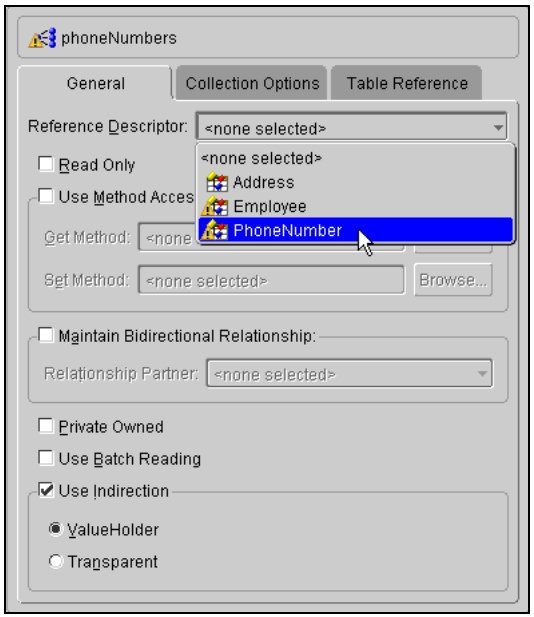
In this tutorial, the Employee project requires:

- A one-to-many mapping from the `phoneNumbers` attribute of the `Employee` class to the `PhoneNumber` class
- A one-to-one mapping from the `owner` attribute of the `PhoneNumber` class back to the `Employee` class

To map the phoneNumbers attribute:

1. Expand the `Employee` class in the **Navigator** pane.
2. Select the `phoneNumbers` attribute.
3. Click the **One-to-many Mapping** button on the mapping toolbar. The **Editor** pane changes to allow you to specify the appropriate information for a one-to-many relationship.
4. Use the **Reference Descriptor** drop-down list to choose `PhoneNumber`.

Figure B–27 One-to-Many Mapping General Tab



5. Click the **Table Reference** tab, and add a new reference by clicking **New**.
   - Create a new reference named `PHONE_EMPLOYEE` with a source table of `PHONE` and a target table of `EMPLOYEE` and click **OK**.
   - In the **Table Reference** drop-down list, choose the `PHONE_EMPLOYEE`.
   - Click the **Add** button on the **Table Reference** tab to add a foreign key relationship. Set the **Source** (foreign key) field to `EMP_ID` and the **Target** (primary key) field to `EMP_ID`.

Figure B–28 One-to-Many Mapping Table Reference Tab

6. Save your changes. Click the **Save Project** button  or choose **File** > **Save** from the menu.

> **Note:** Leave the remaining attributes of the `Employee` descriptor as unmapped. You will use them in the Advanced tutorial.

To map the PhoneNumber class to the Employee class:

After mapping the `Employee` descriptor, use this procedure to map the one-to-one back reference:

1. Map the `owner` attribute of the `PhoneNumber` descriptor as a one-to-one mapping to the `Employee` class (refer to "Creating One-to-One Mappings Between Objects" on page B-27).
2. Select `EMPLOYEE` as the **Reference Descriptor**.

> **Note:** You do not need to create a new table reference. Select the same `PHONE_EMPLOYEE` reference you created when you mapped the one-to-many from `Employee` to `PhoneNumber`.

3. Map the remaining attributes in the `Phone` descriptor as direct-to-field mappings (refer to "Implementing Direct-to-Field Mappings" on page B-24).
4. Remove all unmapped attributes in the `Employee` descriptor. Right-click the attribute and choose **Remove** from the pop-up menu.

   You can also remove attributes by choosing **Selected >Remove** from the menu.

Setting Up Database Sessions

A *database session* in OracleAS TopLink represents an application's dialog with a relational database. The `DatabaseSession` class keeps track of the following information:

- Project and login – contains the login and configuration information about the session
- Descriptors — maintain the associations between tables and persistent classes
- Identity maps — cache and maintain identity

- The database accessor — handles low-level communication between the session and the relational database

An application uses the session to log in to the database and perform read and write operations on the objects stored therein. The session's lifetime is normally the same as the lifetime of the application.

OracleAS TopLink includes a test class so that you can test the descriptor mappings that you have created for this introductory tutorial. This class, `Demo`, among other things, tests the validity of the descriptors and logs into the database.

Logging Into a Database

To log into a database, an application must first read the project file into a `Project` instance. The Project creates the `DatabaseSession` and connects through login. The OracleAS TopLink Mapping Workbench can generate the project file (`Employee.xml` in this example), including the login information. The following code fragment illustrates this approach.

Example B–2 Logging in and Creating a Project Example Code

The following code example illustrates creating the `EMPLOYEE` project.

```
...
    import oracle.toplink.sessions.*;
    ...
    Project builderProject =
oracle.toplink.tools.workbench.XMLProjectReader.read("C:\\oracle\\toplink\\
tutorials\\intro\\Employee.xml");
    DatabaseSession session = builderProject.createDatabaseSession();
    session.login(); // or, session.login(userName, password);
    ...
```

See the `loginToDatabase()` method, in the `Demo` class, for a complete method.

Creating the Tables in Code

You can use OracleAS TopLink Mapping Workbench to create database tables. OracleAS TopLink can also create tables using the `SchemaManager` class. To use this method of creating tables, you must have already obtained a valid login.

The following example illustrates how to create the `EMPLOYEE` table after having logged into the database. The method `createTables()` on the `Demo` class contains sample code that uses the schema manager to create all the required tables for the introductory tutorial.

Example B–3 Creating Tables

The following code example illustrates creating the `EMPLOYEE` table.

```
import oracle.toplink.tools.schemaframework.*;
    import java.math.*;

    // Create table definition which supplies information about the table to be
created.
    TableDefinition employeeTable = new TableDefinition();
    employeeTable.setName("EMPLOYEE");
    employeeTable.addIdentityField("EMP_ID", BigDecimal.class, 15);
    employeeTable.addField("NAME", String.class, 40);
    employeeTable.addField("ADDRESS_ID", BigDecimal.class, 15);

    // Create the table in the database.
    SchemaManager schemaManager = new SchemaManager(session);
    schemaManager.replaceObject(employeeTable);

    // Create an empty table named SEQUENCE if it is not already there. This is
used to hold the sequence number information such as name and counter.
    schemaManager.createSequences();
```

Using Descriptors in an Application

After creating the descriptor files, you must write Java code to register the files with the OracleAS TopLink session. After registering the files, the application can read and write Java class instances from the database.

- To read instances from the database, use the database session object.
- To write instances to the database, use a unit of work object.

Transactions and Units of Work

A *transaction* is a set of database operations that can be either committed (accepted) or rolled back (undone). Transactions can be as simple as inserting an object into a database, but also allow complex operations to be committed or rolled back as a single unit. Unsuccessful transactions can be discarded, leaving the database in its original state.

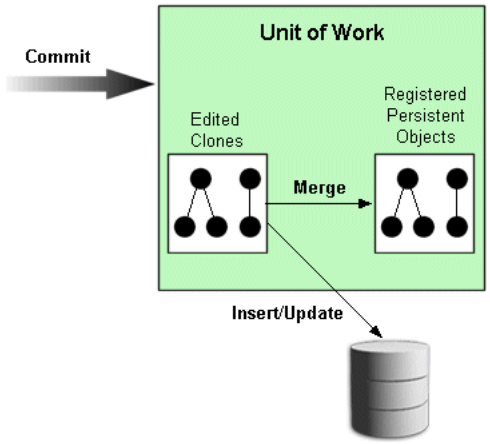
A *unit of work* is an object that simplifies the transaction process and stores transaction information for its registered persistent objects. The unit of work enhances database commit performance by updating only the changed portions of

an object. Units of work are the preferred method of writing to a database in OracleAS TopLink.

To use a unit of work, create an instance of `UnitOfWork` and register the desired persistent objects. The registering process returns clones that can be modified. After changes are made to the clones, use the `commit()` method to commit an entire transaction. The unit of work inserts new objects or updates changed objects in the database, as Figure B–29 illustrates.

If an error occurs when writing the objects to the database, a `DatabaseException` is thrown, and the unit of work is rolled back to its original state. If no database error occurs, the original objects are updated with the new values from the clones.

Figure B–29 Unit of Work Example



Reading and Writing Java Class Instances

Sessions can read instances from the database using the `readObject()` method. Database sessions can write instances to the database using the `writeObject()` method, but note that write is neither required nor used when employing a unit of work. An application typically uses the session to read the instances of a given class from the database and determines which of the instances require changes. The instances requiring changes are then registered with a unit of work. After the changes have been made, the unit of work is used to commit only the changed objects to the database.

This model provides the optimum performance for most applications. Read performance is optimized by using the session because the unit of work does not have to keep track of objects that do not change. Write performance is optimized because the unit of work keeps track of transaction information and writes only the changed portions of an instance to the database.

Using a Unit of Work to Write an Object

After the descriptors have been registered with the session, you are ready to read and write objects to the database. Objects are registered with a unit of work and then committed to the database.

The code fragment in the following example is a continuation of the fragment in Example B–3 and uses the session created there.

Example B–4 Unit of Work

The following code example illustrates using a unit of work to write an object.

```
//Create an Employee object for the company president, as well as the associated
personal information objects.
Employee president = new Employee();

Address presidentHome = new Address();
presidentHome.setStreet("601-1140 Meadowlands Dr.");
presidentHome.setCity("Ottawa");
presidentHome.setPostalCode("K2E 6J6");
presidentHome.setProvince("ON");
presidentHome.setCountry("Canada");

PhoneNumber homePhone = new PhoneNumber();
homePhone.setType("Home");
homePhone.setAreaCode("555");
homePhone.setNumber("555-1234");

PhoneNumber businessPhone = new PhoneNumber();
businessPhone.setType("Business");
businessPhone.setAreaCode("555");
businessPhone.setNumber("555-5678");

president.setName("John Smith");
president.setAddress(presidentHome);
president.addPhoneNumber(homeNumber);
president.addPhoneNumber(businessPhone);
```

```
//Register objects with a new unit of work. Registered objects will return a
clone which should be used to make changes to the object.
UnitOfWork unitOfWork;
unitOfWork = session.acquireUnitOfWork();
Employee tempPresident = (Employee)unitOfWork.registerObject(president);

//Register any other objects, or change registered objects.
tempPresident.setName("Johnny Smith");

//Commit the objects to the database.
unitOfWork.commit();
```

Using a Session to Read an Object

To change the information in the database, the application must create an `Expression` that contains information about the query to be made. The session then searches the database for an object that matches the query and returns the instance. The returned object is registered with the unit of work, and the application makes changes to the object. The application then commits the changes to the database using the `commit()` method.

Example B–5 Session

The following code example illustrates using a session to read an object.

```
//Import the Expression classes.
import oracle.toplink.expressions.*;

//Import the other classes. Create a session and login. Create a query
expression to find the database object.
ExpressionBuilder builder = new ExpressionBuilder();
Expression expression = builder.get("name").equal("John Smith");

//Read the object from the database using the query expression.
Employee president = (Employee) session.readObject(Employee.class, expression);

//Register the object with a new unit of work.
UnitOfWork unitOfWork = session.acquireUnitOfWork();
Employee tempPresident = (Employee)unitOfWork.registerObject(president);

//Make the change to the object.
tempPresident.setName("Johnny Smith");

//Commit the change to the database. Only the NAME field is actually updated.
```

```
unitOfWork.commit();
```

Conclusion

This introductory tutorial explained the basic steps required to create a Java project that accesses a relational database through OracleAS TopLink. The main concepts explained include:

- Creating Java classes which represent database tables
- Using OracleAS TopLink Mapping Workbench to create tables on the database
- Creating *descriptors* for those classes using OracleAS TopLink Mapping Workbench
- Registering the descriptors with the OracleAS TopLink session
- Logging in to the database and doing simple read and write operations

Advanced Tutorial

In this advanced tutorial, you will improve the ACME Employment Management System (built in the introductory tutorial) to manage additional information. You will update the introductory application with new project information and reuse existing components from previous applications.

You will also learn how to:

- Work with self-relationships
- Create the following advanced mapping types: object type mappings, aggregate object mappings, direct collection mappings, and many-to-many mappings
- Implement indirection and value holders
- Use inheritance
- Create transformations
- Work with the **Automap** tool
- Use multiple tables for one class
- Create and generate code

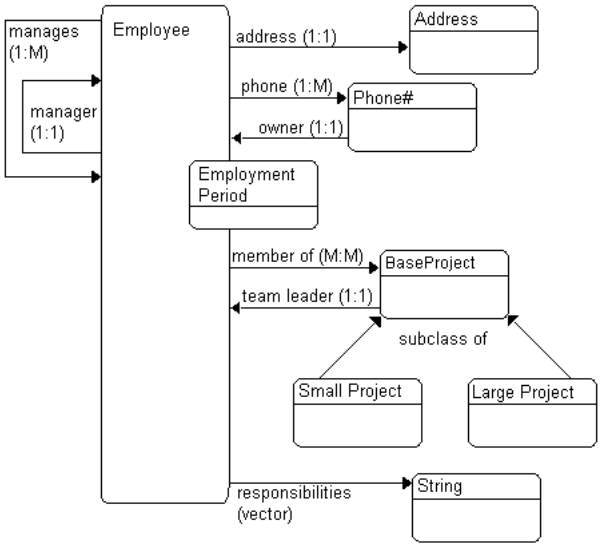
This advanced tutorial adds the ability to track employees' current projects, managers, and contract period. You will reuse components from the introductory tutorial.

In addition to the `Employee`, `Address`, and `PhoneNumber` classes from the introductory tutorial (see "Overview" on page B-2), the advanced tutorial uses these classes:

- `EmploymentPeriod` – Defines the contract term for contractors and the hire date for ACME employees. Each `Employee` class has an `EmploymentPeriod`.
- `ResponsibilityList` – Each `Employee` has a collection of text that describes the employee's job.
- `BaseProject` – Maintains information about a particular project and the people working on it. The `Project` class contains two subclasses: `LargeProject` and `SmallProject`. Each `Employee` can be involved in more than one project.
- `TeamLeader` – Each `Project` can have a team leader (the `Employee` responsible for the project.
- `Manager` – Each `Employee` may have a manager and a collection of managed employees.

Figure B–1 illustrates the object model for the advanced tutorial.

Figure B–30 Advanced Tutorial Object Model



Creating the Database Schema

The advanced ACME employee system stores the employee data in the following database tables. To use this tutorial, create these tables in your database application. Table B–13 describes how each class relates to the database tables.

The column types listed here are generic; the actual column types depend on the database used.

Table B–5 EMPLOYEE Table

Column Name	Column Type	Details
EMP_ID	NUMERIC(15)	Primary key
F_NAME	VARCHAR(40)	
L_NAME	VARCHAR(40)	
ADDR_ID	NUMERIC(15)	
GENDER	CHAR(1)	
START_DATE	DATE	
END_DATE	DATE	

Table B–5 EMPLOYEE Table

Column Name	Column Type	Details
START_TIME	TIME	
END_TIME	TIME	
MANAGER_ID	NUMERIC(15)	
VERSION	NUMERIC(15)	

Table B–6 SALARY Table

Column Name	Column Type	Details
EMP_ID	NUMERIC(15)	Primary key
SALARY	NUMERIC(10)	

Table B–7 ADDRESS Table

Column Name	Column Type	Details
ADDRESS_ID	NUMERIC(15)	Primary key
COUNTRY	VARCHAR(80)	
STREET	VARCHAR(80)	
CITY	VARCHAR(80)	
PROVINCE	VARCHAR(80)	
P_CODE	VARCHAR(20)	

Table B–8 PHONE Table

Column Name	Column Type	Details
EMP_ID	NUMERIC(15)	Primary key
AREA_CODE	CHAR(3)	
P_NUMBER	CHAR(7)	
TYPE	VARCHAR(15)	Primary key

Table B–9 PROJECT Table

Column Name	Column Type	Details
PROJ_ID	NUMERIC(15)	Primary key
DESCRIP	VARCHAR(200)	
PROJ_NAME	VARCHAR(30)	
PROJ_TYPE	CHAR(1)	
LEADER_ID	NUMERIC(15)	
VERSION	NUMERIC(15)	

Table B–10 LPROJECT Table

Column Name	Column Type	Details
PROJ_ID	NUMERIC(15)	Primary key
BUDGET	NUMERIC(10,2)	
MILESTONE	TIMESTAMP	

Table B–11 RESPONS Table

Column name	Column type	Details
EMP_ID	NUMERIC(15)	Primary key
DESCRIP	VARCHAR(200)	

Table B–12 PROJ_EMP Table Between PROJECT and EMPLOYEE

Column Name	Column Type	Details
EMP_ID	NUMERIC(15)	Primary key
PROJ_ID	NUMERIC(15)	Primary key

Table B–13 Relationships Between Classes and Database Table

Column	Class Attribute	Database Type	Java Type
EMPLOYEE	`Employee`		
EMP_ID	`id`	NUMERIC(15)	BigDecimal
F_NAME	`firstName`	VARCHAR(40)	String
L_NAME	`lastName`	VARCHAR(40)	String
ADDR_ID	`address`	NUMERIC(15)	Address
not applicable	`phoneNumbers`	*not applicable*	Vector
GENDER	`gender`	CHAR(1)	String
START_TIME	`normalHours[0]`	TIME	Time
END_TIME	`normalHours[1]`	TIME	Time
MANAGER_ID	`manager`	NUMERIC(15)	Employee
not applicable	`managedEmployees`	*not applicable*	Vector
not applicable	`projects`	*not applicable*	Vector
see Employment Period	`period`	*not applicable*	EmploymentPeriod
SALARY	`Employee`		
EMP_ID	*not applicable*	NUMERIC(15)	*not applicable*
SALARY	`salary`	NUMERIC(10)	int
EMPLOYEE	`EmploymentPeriod`		
START_DATE	`startDate`	DATE	Date
END_DATE	`endDate`	DATE	Date
RESPONS	`Employee`		
EMP_ID	*not applicable*	NUMERIC(15)	*not applicable*
DESCRIP	`responsibilitiesList`	VARCHAR(200)	String
PROJECT	`LargeProject` and `SmallProject`		
PROJ_ID	`id`	NUMERIC(15)	BigDecimal
DESCRIP	`description`	VARCHAR(200)	String

Table B–13 Relationships Between Classes and Database Table (Cont.)

Column	Class Attribute	Database Type	Java Type
LEADER_ID	`teamLeader`	NUMERIC(15)	Employee
PROJ_NAME	`name`	VARCHAR(30)	String
PROJ_TYPE	*not applicable*	CHAR(1)	*not applicable*
VERSION	*not applicable*	NUMERIC(15)	*not applicable*
LPROJECT	`LargeProject`		
PROJ_ID	*not applicable*	NUMERIC(15)	*not applicable*
BUDGET	`budget`	NUMERIC(10,2)	double
MILESTONE	`milestoneVersion`	TIMESTAMP	TimeStamp
ADDRESS	`Address`		
ADDRESS_ID	`id`	NUMERIC(15)	BigDecimal
COUNTRY	`country`	VARCHAR(80)	String
STREET	`street`	VARCHAR(80)	String
CITY	`city`	VARCHAR(80)	String
PROVINCE	`province`	VARCHAR(80)	String
P_CODE	`postalCode`	VARCHAR(20)	String
PHONE	`PhoneNumber`		
AREA_CODE	`areaCode`	CHAR(3)	String
P_NUMBER	`number`	CHAR(7)	String
EMP_ID	`owner`	NUMERIC(15)	Employee
TYPE	`type`	VARCHAR(15)	String
PROJ_EMP	*Relation Table*		
PROJ_ID	*not applicable*	NUMERIC(15)	*not applicable*
EMP_ID	*not applicable*	NUMERIC(15)	*not applicable*

Creating a New Project

1. Create a new project for the Advanced Tutorial as "Creating a New Project" on page B-4 describes.
   - For the database name, use **ADVANCED_TUTORIAL_DB**.
   - For the project name, use **Advanced Tutorial**.
2. Set the project's class path to include the `examples.session.threetier.model` package. See "Setting the Project's Class Path" on page B-6.
3. Enable the following classes in the `examples.session.threetier.model` package, and generate a TopLink descriptor for each Java class as "Generating the Class Definitions" on page B-11 describes:
   - `Address`
   - `Employee`
   - `EmploymentPeriod`
   - `LargeProject`
   - `PhoneNumber`
   - `BaseProject`
   - `SmallProject`

   Table B–13 shows how the classes relate to the database tables.
4. Log into the database as "Logging Into the Database" on page B-13 describes to create or import the database information.

   Select one of the following methods to add database information:
   - Creating Tables Using the OracleAS TopLink Mapping Workbench
   - Importing Tables from the Database

   Refer to Table B–5 through Table B–12 for complete database information.

Mapping Classes to Tables

Map each Java class in the Advanced tutorial to a database table as "Mapping Classes to Tables" on page B-20 describes.

Map this class...	To this database table...
Address	ADDRESS
Employee	EMPLOYEE
LargeProject	LPROJECT
PhoneNumber	PHONENUMBER
BaseProject	PROJECT

Ensure that the primary keys are correctly indicated, as Table B–5 through Table B–12 specify.

> **Note:** A warning message appears indicating that you have not yet mapped the attributes. This is addressed later in the tutorial.

Using the Automap Tool

TopLink can automatically map class attributes to similarly named database tables. This **Automap** function creates mappings only for unmapped attributes—it does not change previously defined mappings.

You can automap classes for an entire project or for specific tables.

> **Note:** Although **Automap** correctly maps most one-to-one and direct-to-field mappings, examine each mapping for valid and correct information. You may need to add or change some mappings.

To Automap the Address descriptor:

1. Choose the `Address` class in the **Navigator** pane and click the **Descriptor Info** tab in the **Editor** pane.
2. In the **Associated Table** drop-down list, choose the ADDRESS table.

   You must associate the class with a table before using the **Automap** tool.

3. Right-click the `Address` class in the **Navigator** pane and choose **Automap** from the pop-up menu.

   You can also automap descriptors by choosing **Selected** > **Automap** from the menu.

The system automatically maps each attribute to the appropriate database table. Do not **Automap** any other classes. You will manually map these classes later in this tutorial.

Implementing Indirection

Indirection allows you to retrieve objects from the database as needed.

- With indirection turned *off*, when an object is retrieved from the database all the other objects that it references are also retrieved.
- With indirection turned *on*, each object is retrieved from the database only when asked for.

Using indirection can be a great performance benefit and we strongly recommended using it. See "Working with Indirection" on page 6-5 for more information.

Preparing Java Code for Indirection

To prepare your object model for indirection, you must alter the application slightly:

- Replace each relationship reference with a `ValueHolderInterface`. This interface is located in the `oracle.toplink.indirection` package and allows for indirection to be used.
- Instantiate all variables with indirection references to empty value holders. Normally, this is done in the constructor of the object.
- Modify the `get` methods for these variables to extract the value from the value holder.
- Modify the `set` methods for these variables to insert the value into the value holder.

You can implement indirection using *direct access* or *method access*.

- For method access, TopLink requires additional `get` and `set` methods that provide access to the value holders.

- For direct access, TopLink can access the value holders directly—the additional `get` and `set` methods are not required.

If the instance variable returns a `Vector` instead of an object, then define the value holder in the constructor as follows:

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

In the following examples, the `Employee` class uses indirection with method access for its one-to-one mapping to `Address`. The class definition is modified so that the address attribute of `Employee` is a `ValueHolderInterface` instead of an `Address`. In both examples, the application uses the `getAddress()` and `setAddress()` methods to access the `Address` object.

Example B–6 Indirection Examples

The following example illustrates code *before* using indirection.

```
protected Address address;
    public Employee() {
        address = null;
    }
    public Address getAddress() {
        return address;
    }
    public void setAddress(Address address) {
        this.address = address;
    }
```

The following example illustrates the same code *after* using indirection.

```
protected ValueHolderInterface address;
    public Employee() {
        address = new ValueHolder();
    }
    public Address getAddress() {
        return (Address)address.getValue();
    }
    public void setAddress(Address address) {
        this.address.setValue(address);
    }
```

The indirection example could also use method access instead of direct access. This would be implemented by adding `getAddressValueHolder()` and `setAddressValueHolder()` methods.

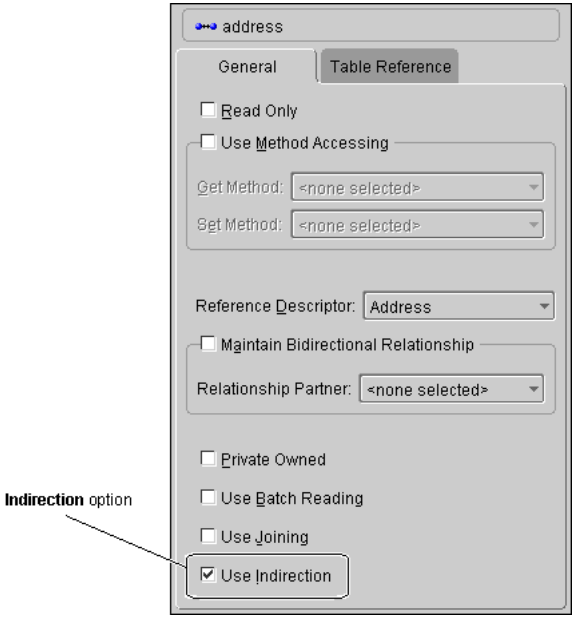
Implementing Indirection in the OracleAS TopLink Mapping Workbench

After modifying the code, update the OracleAS TopLink Mapping Workbench descriptors to use indirection.

To implement indirection in the Workbench:

1. Map the one-to-one and one-to-many mappings for each class as normal.
2. On the **General** tab for each mapping, select the **Use Indirection** option.

Figure B–31 General Tab of a Mapping



Implementing Indirection in the Tutorial

The following attributes in the Advanced tutorial sample code have been implemented using `ValueHolderInterfaces`:

Employee

```
address
manager
managedEmployees
projects
responsibilitiesList
phoneNumbers
```

PhoneNumber

```
owner
```

BaseProject

```
teamLeader
```

When you create mappings for these attributes, be sure to enable the **Use Indirection** option.

Implementing a One-to-One Self Relationship

Some object models require a class to reference another instance of the same class. In the advanced tutorial, the `Manager` attribute in the `Employee` class references another employee (see Figure B–1).

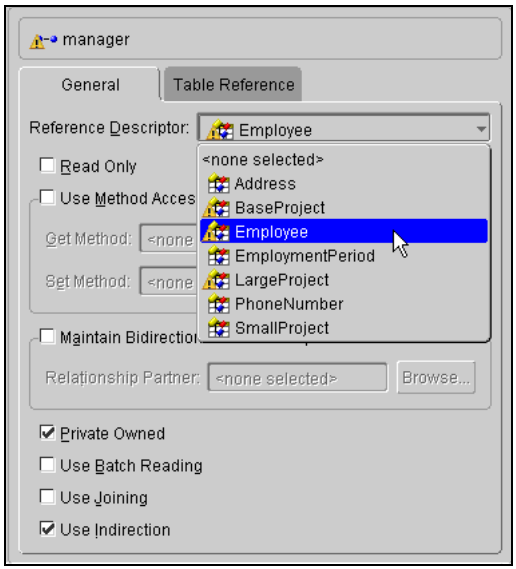
To map the manager attribute:

1. Select the Employee's `manager` attribute in the **Navigator** pane, then click the **One-to-One Mapping** button  on the mapping toolbar.

   The **Editor** pane displays the appropriate information for a one-to-one relationship to be specified.

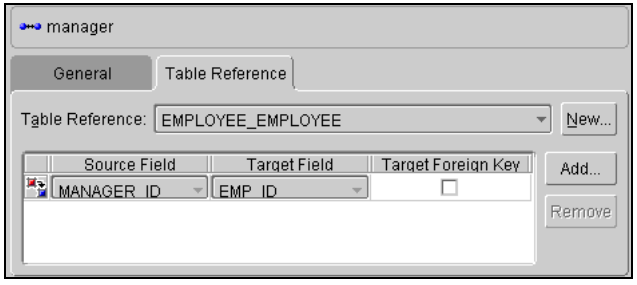
2. Use the **Reference Descriptor** drop-down list on the **General** tab to choose `Employee` as the reference descriptor.

Figure B–32 One-to-One Mapping General Tab



3. Select the **Use Indirection** option. See "Implementing Indirection in the Tutorial" on page B-50.
4. Click the **Table Reference** tab.

Figure B–33 One-to-One Mapping Table Reference Tab



5. Create a new table reference by clicking the **New** button.

6. In the New Reference dialog box, create a reference whose:
   - Name is EMPLOYEE_EMPLOYEE
   - Source table is EMPLOYEE
   - Target database is EMPLOYEE

   Note: If you leave the **Name** field blank, then TopLink automatically names the table as `<SourceTable>_<TargetTable>`.

7. Select `EMPLOYEE_EMPLOYEE` (created in step 5 from the **Table Reference** drop-down list.
8. Click the **Add** button to define the foreign key fields.
   - In the **Source Field** column, choose `MANAGER_ID` (foreign key) field.
   - In the **Target Field** column, choose `EMP_ID` (primary key) field.
   - Leave the **Target Foreign Key** option unchecked.

   Note: The mapping is **from** the `EMPLOYEE` table, `MANAGER_ID` field **to** the `EMP_ID` *field*.

9. Click **Save**  on the toolbar or choose **File** > **Save Project** to save the project.

Creating Other One-to-one Mappings

The Advanced tutorial also includes a one-to-one mapping for the following attributes:

- `address` attribute in the `Employee` descriptor
- `owner` attribute in the `PhoneNumber` descriptor
- `teamLeader` attribute in the `BaseProject` descriptor

Create these mappings as shown in "Creating One-to-One Mappings Between Objects" on page B-27. Refer to Table B–13 for the correct relationships. Enable indirection for each of these mappings, as "Implementing Indirection in the Tutorial" on page B-50 indicates.

Implementing a One-to-Many Self-Relationship

Some object models require a class to reference another instance of the same class. In the advanced tutorial, a manager can have a collection of managed employees (see Figure B–1).

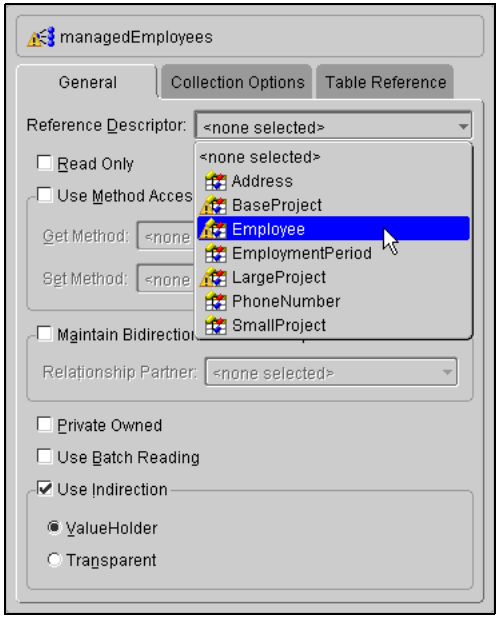
To map the managedEmployee attribute:

1. Select the Employee's `managedEmployees` attribute in the **Navigator** pane, then click the **One-to-Many Mapping** button on the mapping toolbar.

   The **Editor** pane displays the appropriate information for a one-to-many relationship to be specified.

2. Use the **Reference Descriptor** drop-down list to choose `Employee`.

Figure B–34 One-to-Many Mapping General Tab



3. Select the **Use Indirection** option, and choose **ValueHolder**. See "Implementing Indirection in the Tutorial" on page B-50.

4. Click the **Table Reference** tab. Use the **Table Reference** drop-down list to choose the EMPLOYEE_EMPLOYEE table (previously created in "To map the manager attribute:" on page B-50).

- Click the **Add** button on the **Table Reference** tab to add a foreign key relationship.
- Set the **Source Field** (foreign key) to `MANAGER_ID`.
- Set the **Target Field** (primary key) to `EMP_ID`.

Figure B–35 One-to-Many Mapping Table Reference Tab

5. Click **Save** on the toolbar or choose **File** > **Save Project** to save the project.

Creating Other One-to-Many Mappings

The Advanced tutorial also includes a one-to-many mapping for the `phoneNumbers` attribute in the `Employee` descriptor. Create this mapping as shown in "Creating One-to-Many Mappings" on page B-30. Refer to Table B–13 for the correct relationship.

Enable indirection for this mapping, as indicated in "Implementing Indirection in the Tutorial" on page B-50.

Using Multiple Tables

In TopLink, it is possible to spread classes across two or more tables. In the advanced tutorial, the `Employee` class is stored in multiple tables: although most information is in the `EMPLOYEE` table, salary information is stored in the `SALARY` table.

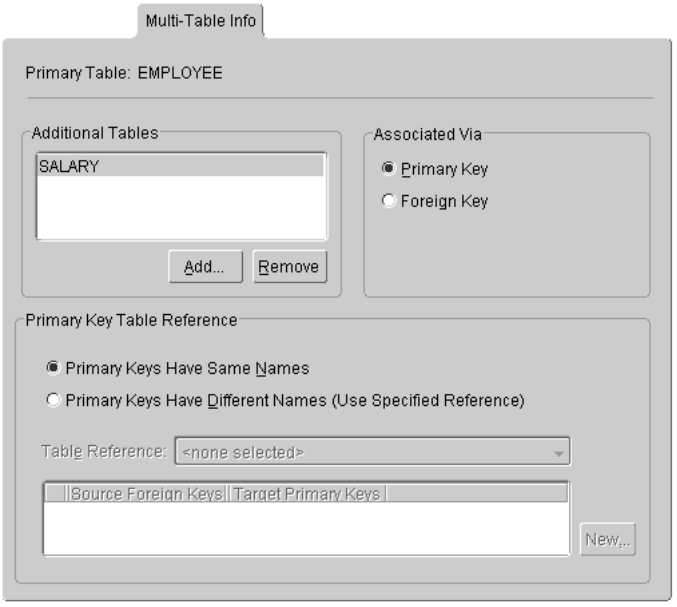
To map the Employee class to multiple tables:

1. Select the `Employee` descriptor in the **Navigator** pane.
2. Click the **Multi-Table Info** tab in the **Editor** pane.

If the **Multi-Table info** tab is not visible, right-click the `Employee` descriptor and choose **Set Advanced Properties** > **Multi-Table Info** from the pop-up menu.

3. In the **Additional Tables** pane, click **Add** and add the `SALARY` table.
4. In the **Associated Via** pane select **Primary Key** and in the **Primary Key Table Reference** pane, select **Primary Keys Have Same Names**.

Figure B–36 Multi-Table Info Tab



5. Click **Save** on the toolbar or choose **File** > **Save Project** to save the project.

Implementing Object Type Mapping

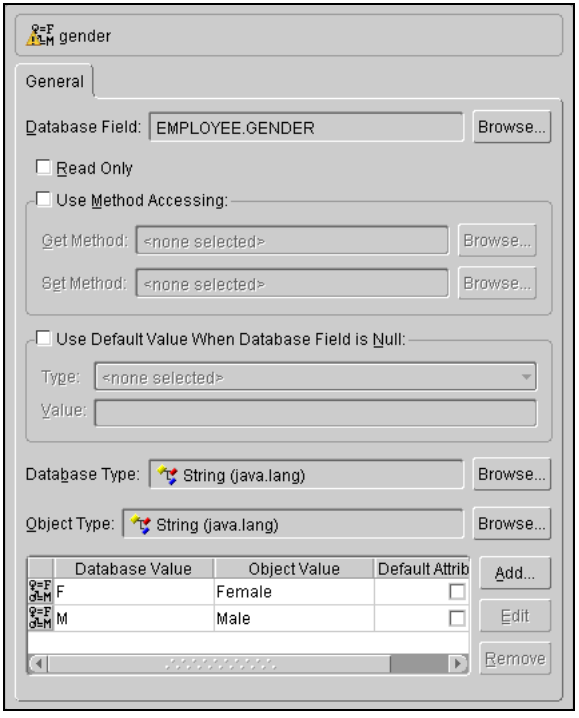
In TopLink, you can match a fixed number of database values to Java objects through *object type mappings*. In the advanced tutorial, each employee's gender is stored as a single letter in the database field (**M** or **F**), but the value is the full name (**Male** or **Female**).

To map the gender attribute:

1. Expand on the `Employee` descriptor in the **Navigator** pane.

2. Select the `gender` attribute and click the **Object-Type Mapping** button in the mapping toolbar.

Figure B–37 Object-Type Mapping Tab



3. In the **Database Field**, select the GENDER field from the EMPLOYEE table.
4. Select **String** as the **Database Type**, and **String** as the **Object Type**.
5. Click **Add** and create the following database mappings:

Database Value	Object Value
`F`	`Female`
`M`	`Male`

6. Click **Save** on the toolbar or choose **File** > **Save Project** to save the project.

Implementing an Aggregate Object

In TopLink, two objects are related by aggregation if there is a one-to-one relationship between the objects and all the attributes of the second object can be retrieved from the same table(s) as the owning object. In the advanced tutorial, the `EmploymentPeriod` is an aggregate descriptor, and the `period` attribute is an aggregate object.

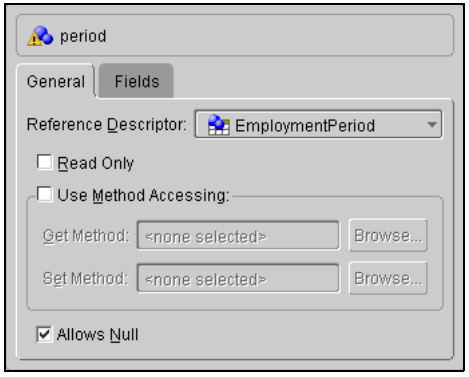
To map an aggregate object:

1. Select the `EmploymentPeriod` descriptor in the **Navigator** pane.
2. Click the **Aggregate Descriptor** button on the mapping toolbar. The descriptor's icon in the **Navigator** pane changes to an aggregate descriptor .
3. Map the `startDate` and `EndDate` attributes of the `EmploymentPeriod` as direct-to-field mappings.

> **Note:** The **Database Field** fields are disabled because the aggregate descriptor is not associated with a database table.

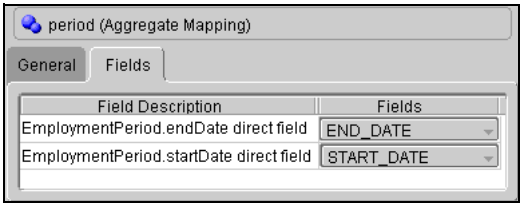
4. Expand the Employee descriptor in the **Navigator** pane.
5. Select the `period` attribute of the `Employee` descriptor.
6. Click the **Aggregate Mapping** button on the mapping toolbar.

Figure B–38 Aggregate Mapping General Tab



7. Use the **Reference Descriptor** drop-down list to choose the `EmploymentPeriod` aggregate descriptor.
8. Click the **Fields** tab.

Figure B–39 Aggregate Mapping Fields Tab



9. Use the **Fields** drop-down list to map each field as follows:
   - `endDate` – END_DATE
   - `startDate` – START_DATE
10. Click **Save** on the toolbar or choose **File** > **Save Project** to save the project.

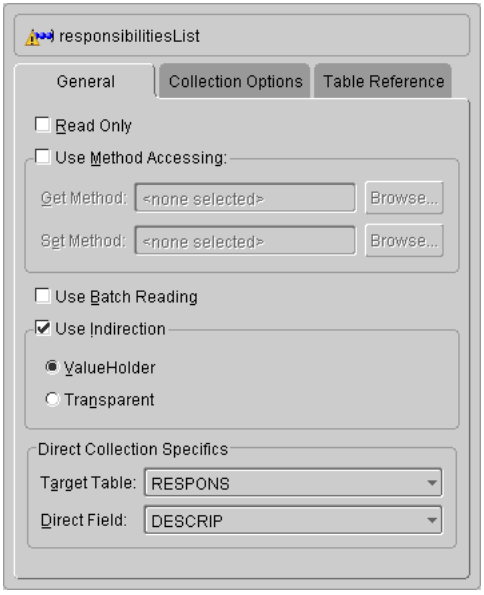
Implementing a Direct Collection Mapping

Direct collection mappings store collections of Java objects that are not TopLink-enabled. In the advanced tutorial, the `responsibilitiesList` attribute is a direct collection.

To map a direct collection:

1. Expand the `Employee` descriptor in the **Navigator** pane.
2. Click the `responsibilitiesList` attribute of the `Employee` descriptor.
3. Click the **Direct Collect** button on the mapping toolbar.
4. To specify where to place the strings in the direct collection, use the **Target Table** and **Direct Field** drop-down lists to select the DESCRIP field on the RESPONS databases table.
5. Select the **Use Indirection** option and choose **ValueHolder**. See "Implementing Indirection in the Tutorial" on page B-50.

Figure B–40 Direct Collection Mapping General Tab



6. Click the **Table Reference** tab, and add a new table reference by clicking the **New** button.
   - Create a reference named `RESPONS_EMPLOYEE`, with a source table of `RESPONS` and target table of `EMPLOYEE` and click **OK**.
   - In the **Table Reference** drop-down list, choose the `RESPONS_EMPLOYEE`.
   - Click the **Add** button on the **Table Reference** tab to add a foreign key relationship.
   - Set the **Source Field** (foreign key) to `EMP_ID` (from the RESPONS table).
   - Set the **Target Field** (primary key) to `EMP_ID` (from the EMPLOYEE table).
7. Click **Save** on the toolbar or choose **File** > **Save Project** to save the project.

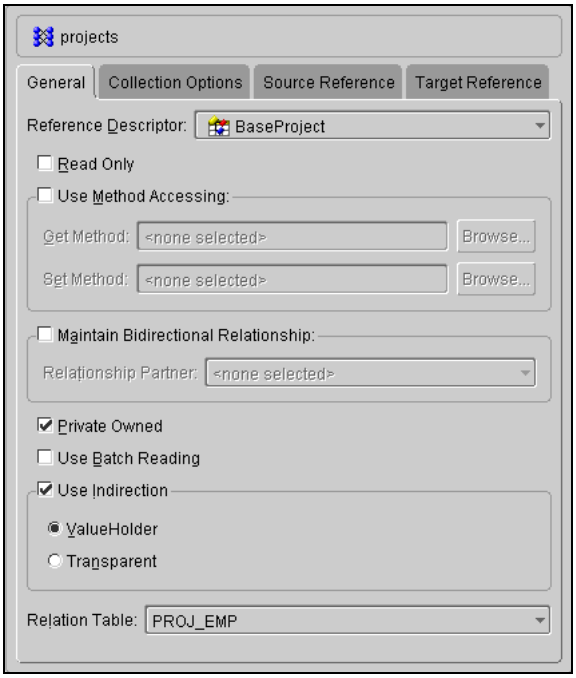
Implementing a Many-to-Many Mapping

Many-to-many mappings represent relationships between a collection of source objects and a collection of target objects. In the advanced tutorial, the projects attribute uses a many-to-many-mapping (for example, many employees can have many projects).

To map a many-to-many mapping:

1. Expand the `Employee` descriptor in the **Navigator** pane.
2. Click the `projects` attribute of the `Employee` descriptor.
3. Click the **Many-to-many Mapping** button on the mapping toolbar.
4. Use the **Reference Descriptor** drop-down list to choose the Base`Project` descriptor.
5. Use the **Relation Table** drop-down list to choose the `PROJ_EMP` table (the class to map to).
6. Ensure that the **Use Indirection** field is selected. See "Implementing Indirection in the Tutorial" on page B-50.

Figure B–41 Many-to-Many Mapping General Tab



7. Click the **Source Reference** tab, and add a new reference by clicking the **New** button.

- Create a new reference named `PROJ_EMP_EMPLOYEE`, with a source table of `PROJE_EMP` and target table of `EMPLOYEE`, and click **OK**.
- In the **Table Reference** drop-down list, choose the `PROJ_EMP_EMPLOYEE` reference.
- Click the **Add** button on the **Source Reference** tab to add a foreign key relationship.
- Set the **Foreign Key** field to `EMP_ID` (from the PROJ_EMP table).
- Set the **Primary Key** field to `EMP_ID` (from the EMPLOYEE table).

8. Click the **Target Reference** tab, and add a new reference by clicking the **New** button.
   - In the **Table Reference** drop-down list, choose the `PROJ_EMP_PROJECT` reference.
   - Click the **Add** button on the **Source Reference** tab to add a foreign key relationship.
   - Set the **Source Field** field to `PROJ_ID` (from the PROJ_EMP table).
   - Set the **Target Field** field to `PROJ_ID` (from the PROJECT table).
9. Click **Save**  on the toolbar or choose **File** > **Save Project** to save the project.

Implementing Inheritance

Inheritance describes how a child class inherits the characteristics of its parent class. TopLink uses multiple implementations to support database inheritance.

In the advanced tutorial, the `LargeProject` and `SmallProject` classes inherit the characteristics of the `BaseProject` class. Use the following procedures to set the `BaseProject` descriptor to implement inheritance, then enable the two subclasses.

To implement inheritance in the BaseProject descriptor:

1. Click the `BaseProject` descriptor in the **Navigator** pane.
2. Click the **Inheritance** tab in the **Editor** pane.

   If the Inheritance tab is not visible, right-click the Project descriptor and choose **Advanced Properties** > **Inheritance** from the pop-up menu or **Selected** > **Advanced Properties** > **Inheritance** from the menu.
3. Ensure that the **Is Root Descriptor** field is selected.

4. Select the **Use Class Indicator Field** option, and use the drop-down list to choose `PROJ_TYPE`.
5. Select the **Use Class Indicator Dictionary** field and choose a **String** type as the **Indicator Type**.

To implement inheritance in each subclass:

1. Click the `SmallProject` descriptor in the **Navigator** pane.
2. Click the **Inheritance** tab in the **Editor** pane.

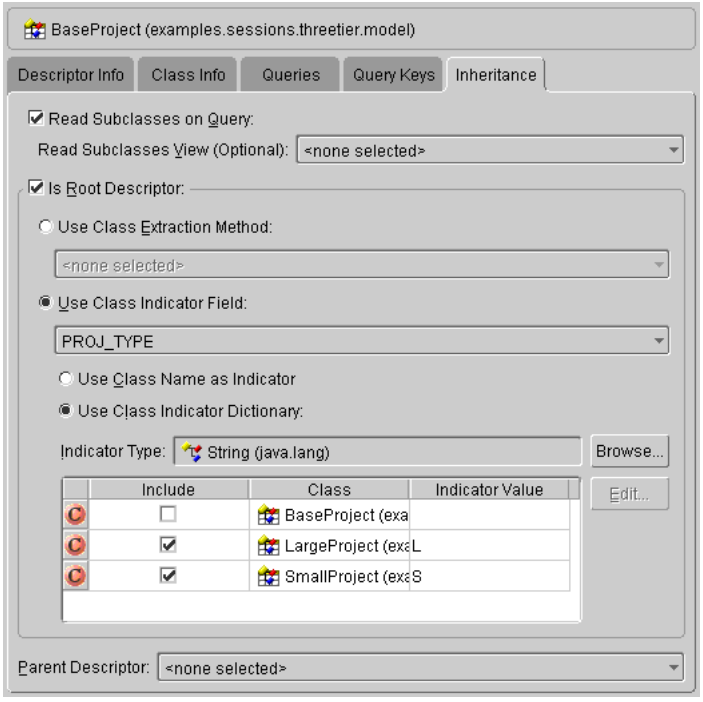
   If the Inheritance tab is not visible, right-click the `SmallProject` descriptor and choose **Advanced Properties** > **Inheritance** from the pop-up menu or **Selected** > **Advanced Properties** > **Inheritance** from the menu.
3. In the **Parent Descriptor** drop-down list, choose the `BaseProject` class.
4. Click **Save**  on the toolbar or choose **File** > **Save Project** to save the project.

Repeat this procedure for the `LargeProject` descriptor.

To complete the inheritance:

1. Click the `BaseProject` descriptor in the **Navigator** pane.
2. Click the **Inheritance** tab in the **Editor** pane.
3. Select the **Read Subclass on Query** option.
4. For each inherited descriptor:
   - Select the **Include** column.
   - Click **Edit**.
   - Enter an **Indicator Value** (**S** for `SmallProject` and **L** for `LargeProject`) for each child class.

Figure B–42 Inheritance Tab



Note: To have the root class store instances of itself in the table, it must include a value-subclass pair for itself. This is not required for the advanced tutorial because `Project` is an abstract class.

Implementing a Transformation Mapping

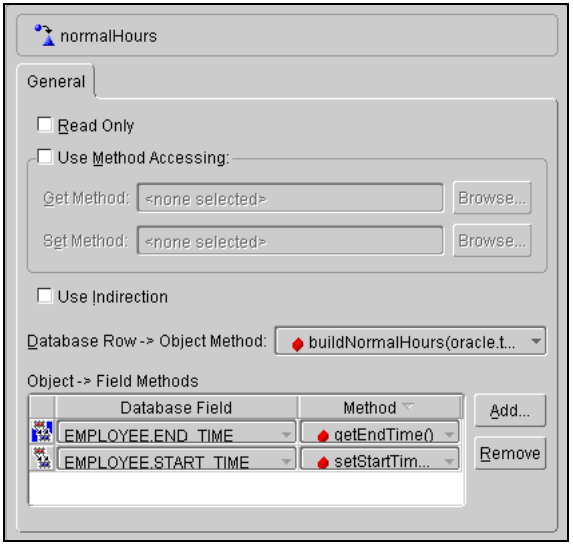
Use transformation mappings for specialized translations between how a value is represented in Java and in the database. The method takes a database row as an argument and is called whenever an object is written to the database. The method returns the value from the object that should be written to the field.

In the advanced tutorial, the transformation method corresponds to a single argument method on the `Employee` class and extracts the values from the fields, placing them into the `NormalHours` array.

To map the normalHours attribute:

1. Expand the `Employee` descriptor in the **Navigator** pane.
2. Click the `normalHours` attribute of the `Employee` descriptor.
3. Click the **Transformation Mapping** button on the mapping toolbar.
4. Use the **Database Row -> Object Method** drop-down list to select the **bulidNormalHours** method.
5. Click the **Add** button to create the following **Object->Field Methods**:
   - **Database Field**: START_TIME, **Method**: `getStartTime`
   - **Database Field**: END_TIME, **Method**: `getEndTime`

Figure B–43 Transformation Mapping Tab



6. Click **Save** on the toolbar or choose **File** > **Save Project** to save the project.

Mapping the Remaining Attributes

The remaining attributes for each descriptor are simple direct-to-field mappings. Use Table B–13 to map each attribute.

Completing the Tutorials

Generating Code

To use your project with the Foundation Library, you must either generate deployment XML files or export the project to Java source code.

In this tutorial, we will create a deployment XML file that can be read in at runtime. The code generator creates a single subclass that can be used instead of reading directly from the files.

To export to Java source code:

1. Right-click the project in the **Navigator** pane and choose **Export Project to Java Source** from the pop-up menu. The Choose an Export File dialog box appears.

   You can also export the project by choosing **File** > **Export to Java Source** or **Selected** > **Export to Java Source** from the menu.

2. Select a directory location and file name (`.java`) and click **OK**.

Congratulations! You have completed the Advanced Tutorial and are now familiar with TopLink's advanced topics and functions.

Completed versions of this tutorial (for various architectures and application servers) are included with the complete installation of TopLink in the following directory:

<ORACLE_HOME>\toplink\examples

C

Troubleshooting

This section contains the following information on troubleshooting the Oracle Application Server TopLink Mapping Workbench and Oracle Application Server TopLink Sessions Editor.

- Error Messages
- Class Path Issues
- Database Connections
- Troubleshooting Descriptors

In addition to the information in this chapter, refer the online help for additional information.

Error Messages

The following errors messages (listed in alphabetical order) may appear in the status bar of the OracleAS TopLink Mapping Workbench and OracleAS TopLink Sessions Editor. Each message includes a description of the action that caused the error and the recommended resolution. Use the Project Status report (see "Generating the Project Status Report" on page 2-5) to display all error messages in a project.

1-1 mappings for EJB 2.0 CMP descriptors must use ValueHolder indirection.

Cause: You created a mapping without indirection.

Action: When creating a one-to-one mapping for EJB descriptors in projects with 2.0 CMP persistence, you must select the **ValueHolder** indirection option on the mapping's **General** tab.

1-M and M-M mappings for EJB 2.0 CMP descriptors must use transparent indirection.

Cause: You created a mapping without indirection.

Action: When creating a one-to-many or a many-to-many mapping for projects with 2.0 CMP persistence, you must select the **Transparent** indirection option on the mapping's **General** tab.

A field/method pair is incomplete in the [mapping name] mapping.

Cause: You created a transformation mapping, but did not specify a complete field/pair method

Action: Complete the **Object** > **Field Method** for each database field on the mapping's **General** tab.

A key pair has not been completely specified for a reference.

Cause: You created a table reference without a key pair.

Action: You must specify a foreign key reference for the database table. Use the database table's **Reference** tab to add a key pair.

A locking policy is specified, but the lock field is not specified.

Cause: You selected a locking policy, but did not specify a field for optimistic locking.

Action: Specify the locking **Field** on the **Locking** tab.

Aggregate fields are not specified.

Cause: You created an aggregate mapping without specifying specific fields.

Action: For aggregate mappings, each **Field Description** on the **Fields** tab must contain a unique **Field**.

Aggregate mapping fields must be unique.

Cause: You created an aggregate mapping without specifying unique fields.

Action: For aggregate mappings, each **Field Description** on the **Fields** tab must contain a unique **Field**.

An aggregate shared by multiple source descriptors cannot have one-to-many or many-to-many mappings.

Cause: You attempted to create multiple one-to-many and many-to-many, or one-to-one mappings in which the target is the aggregate.

Action: Aggregate descriptors that are shared by multiple source descriptors cannot have mappings that contain a target object that references the descriptor.

An ejb-jar.xml file needs to be specified.

Cause: For a 2.0 CMP persistence project, you must specify an `ejb-jar.xml` file.

Action: Specify the location of the `ejb-jar.xml` file on the project's **General Properties** tab.

Attribute is typed as a ValueHolderInterface, but the mapping does not use Value Holder Indirection.

Cause: You did not specify indirection or transparent indirection for the mapping.

Action: If the class attribute is of type `ValueHolderInterface`, you must use value holder indirection for the mapping.

Attribute is typed as a ValueHolderInterface, but the mapping does not use Indirection.

Cause: For one-to-one and transformation mappings, you did not use indirection.

Action: If the class attribute is of type `ValueHolderInterface`, you must use indirection for the mapping.

Cannot Use Joining because the source and target (reference) descriptors are the same type.

Cause: You selected the **Use Joining** option on a one-to-one mapping in which the source and reference descriptors are the same.

Action: Either unselect the **Use Joining** option or select a difference **Reference Descriptor** on the **One-to-One Mapping General** tab.

Classes cannot reference an aggregate target with one-to-one, one-to-many, or many-to-many mappings.

Cause: You tried to select an aggregate descriptor as a reference.

Action: You cannot select an aggregate descriptor as the **Reference Descriptor** for a one-to-one, one-to-many, or many-to-many mapping.

EJB Class information is not compatible with project persistence type.

Cause: You entered EJB incorrect EJB information.

Action: Ensure that the information you entered on the EJB descriptor's **EJB Info** tab is incorrect, based on the project's persistence type (as specified on the project's **General Properties** tab).

If the mapping uses indirection, then the attribute must be a ValueHolderInterface.

Cause: Persistent classes that use indirection must replace the relationship attributes with a value holder (an instance of a class that implements the `ValueHolderInterface`, such as `ValueHolder`).

Action: On the attribute's **General** mapping tab, select the **Use Indirection** option and specify **ValueHolder** type.

[descriptor name] is not an implementor of the [descriptor name] interface, so it cannot have an indicator value.

Cause: You included a descriptor on the Variable One-to-One Class Indicator Info tab that is an implementor.

Action: Unselect the descriptor on the Variable One-to-One Class Indicator Info tab or add the descriptor to the Implementor tab.

Mapping references write lock field, but is not read-only.

Cause: You specified a locking policy for a descriptor, but one of the attribute mappings is not read-only.

Action: Select the **Read Only** option on the mapping's **General** tab.

Mapping uses Indirection, but its associated attribute is not a ValueHolderInterface.

Cause: You attempted to use Indirection for a one-to-one mapping.

Action: If you select the **Use Indirection** option for a one-to-one mapping with transparent indirection, the associated class attribute must be **ValueHolderInterface**.

Mapping uses Value Holder Indirection but its associated attribute is not a ValueHolderInterface.

Cause: You selected indirection without a **ValueHolderInterface**.

Action: If you select the **Use Indirection** (**ValueHolder**) option for a one-to-many, many-to-many, or direct collection mapping, the associated class attribute must be **ValueHolderInterface**.

Mappings for EJB 2.0 CMP descriptors that use Value Holder Indirection must not use method accessing.

Cause: You cannot use method accessing on mappings for EJB 2.0 CMP descriptors that use **ValueHolder Indirection**.

Action: Because EJB attributes are code-generated, reference mappings *should not* be set to use method access. The attributes are code-generated to be of type **ValueHolder** but the abstract methods are defined to return the local interface type of the related bean.

Method accessors have not been selected.

Cause: You selected to **Use Method Accessing** for a mapping but did not select a method.

Action: You must select a **Get** and **Set** method on the mapping's **General** tab.

More than one writable many-to-many mapping cannot use the same relation table.

Cause: A project cannot have more than one writable many-to-many mapping using the same relation table.

Action: You must either make sure only one of the mappings is writable or choose a different table for each many-to-many mapping.

Multi-table reference should be defined from the base table [table name] to the derived table.

Cause: This descriptor has **Inheritance** and **Multi-Table** advanced properties defined on it.

Action: The multi-table relationship that is defined between the base class's table and this derived class's table must be defined from base to derived.

NCharacter, NString, and NClob database types are currently only supported on the Oracle9 platform.

Cause: You attempted to map a database type that is not supported by your database.

Action: The database type for a type conversion mapping or direct-to-field mapping can be NCharacter, NString, or NCLOB only if you are using an Oracle9 database.

No attribute is associated with the mapping named [mapping name].

Cause: The specified mapping does not have an associated class attribute.

Action: Either refresh the class or remove the mapping.

No calss indicator value should be defined for the abstract class [class name].

Cause:

Action:

No class in inheritance tree is marked as root.

Cause: The inheritance hierarchy must contain one root descriptor.

Action: Select the **Is Root Descriptor** option on the **Inheritance** tab.

No class indicator field is selected for this root class.

Cause: You selected the **Use Class Indicator Dictionary** option for the root descriptor in the inheritance hierarchy, but did not specify an indicator value for the root and its children.

Action: Use the **Indicator Type** drop-down list on the **Inheritance** tab for the root class.

No class indicator field should be defined for the abstract class [class].

Cause: Abstract classes should not be included or contain an **Indicator Value** on a descriptor's Inheritance tab.

Action: Either remove the **Include** option for the class on the **Inheritance** tab, or remove the **abstract** modifier option on the descriptor's **Class Info – Class** tab.

No collection type is selected.

Cause: For collection mappings, you must select a collection type.

Action: Select a collection type on the mapping's **Collection** tab.

No database field is selected.

Cause: You created a direct-to-field or type conversion mapping without selecting a database field.

Action: For attributes with direct-to-field mappings, you must specify a **Database Field** on the mapping's **General** tab. For attributes with type conversion mappings, you must specify a **Database Field** on the mapping's **General** tab.

No database type is selected.

Cause: For type conversion and object type mappings, you must select a database type.

Action: Select a **Database Type** on the mapping's **General** properties tab.

No direct field is specified.

Cause: For direct collection mappings, you must specify the direct collection information.

Action: Select a **Target Table** and **Direct Field** that the direct collection specifies.

No field/method pairs defined for the [mapping name] mapping.

Cause: You created a transformation mapping, but did not specify an **Object --> Field Method**.

Action: You must specify at least one field/method pair, unless the mapping is **Read Only**.

No indicator field is selected.

Cause: You created a variable one-to-one mapping, but did not specify a database field in which to store indicator values.

Action: Select the **Class Indicator Field** on the **Class Indicator Info** tab.

No indicator values are specified.

Cause: You created a variable one-to-one mapping, but did not specify indicator values for each object type.

Action: Select the **Indicator Type** on the **Class Indicator Info** tab.

No method specified for transforming a database row into this attribute.

Cause: You created a transformation mapping, but did not specify the transformation method.

Action: Select a **Database Row -> Object Method** on the mapping's **General** tab.

If you are creating a write-only mapping, you can ignore this warning.

No null value type has been selected.

Cause: You selected to **Use Default Value When Database Field is Null** for a mapping, but did not specify the value.

Action: Specify a default **Type** and/or **Value** on the mapping's **General** tab.

This message may also appear after using the Package Rename tool when upgrading an older OracleAS TopLink Mapping Workbench project.

No object type is selected.

Cause: You created an object type mapping, but did not select the type.

Action: Select the **Object Type** and **Database Type** on the mapping's **General** tab.

No object-type mappings have been specified.

Cause: You created an object type mapping, but did not create n object-to-database mapping.

Action: Specify at least one mapping (**Database Value** and **Object Value**) on the mapping's **General** tab.

No primary key(s) specified in [table name] table.

Cause: You must specify a primary key for each database table. When importing tables from a database into the OracleAS TopLink Mapping Workbench, the primary key information will be retained only if the JDBC driver supports the `getPrimaryKeys()` method.

Action: Ensure that a primary key is specified for each descriptor on the **Descriptor Info** tab.

Not all query key associations have foreign key fields specified.

Cause: You created a query key association without a foreign key.

Action: You must specify a foreign key field for each query key association on the **Query Key Associations** tab for variable one-to-one mapping.

No query key associations have been defined.

Cause: You created a variable one-to-one mapping, but did not define a key pair.

Action: Create or select a key pair on the mapping's **Query Key Association** tab.

No reference descriptor is selected.

Cause: You created a mapping, but did not specify the reference descriptor

Action: You must select a **Reference Descriptor** for each relationship mapping on the mapping's **General** tab.

No relation table is selected.

Cause: You created a many-to-many mapping, but did not specify a relation table. The relation table represents the relationship between the primary keys of the source table and target table.

Action: Select or create a **Relation Table** on the mapping's **General** tab.

No sequence field name is selected.

Cause: You selected **Use Sequencing** on a descriptor's **Descriptor Info** tab but did not specify the sequence information.

Action: Specify a **Name**, **Table**, and **Field**.

No sequence name is selected.

Cause: You selected **Use Sequencing** on a descriptor's **Descriptor Info** tab but did not specify the sequence information.

Action: Specify a **Name**, **Table**, and **Field**.

No source reference is selected.

Cause: You created a many-to-many mapping, but did not select (or create) a source table reference on the mapping's **Source Reference** tab.

Action: The source table reference must contain a **Source** field (from the mapping's relation table) and a **Target** field (from one of the descriptor's associated tables).

No table reference is selected.

Cause: You create a relationship mapping, but did not specify a reference table.

Action: Select (or create) a table reference for each relationship mapping on the mapping's **Table Reference** tab.

No target reference is selected.

Cause: You created a many-to-many mapping, but did not select (or crate) a target table reference on the mappings **Target Reference** tab.

Action: The source table reference must contain a **Source** field (from the mapping's relation table) and a **Target** field (from one of the descriptor's associated tables).

Object values do not match the specified attribute or object type.

Cause: For object-type mappings, you entered an **Object Value** that does not match the **Object Type**, as specified on the mapping's **General** tab. For example, you specified an **integer** type, but entered a value of **A**.

Action: Ensure that the **Object Value** matches the **Object Type**.

One or more field types have not been specified.

Cause: You created a field with a missing or invalid **Type**.

Action: You must specify the **Type** for each database field on the **Field** tab.

One of the descriptors in this package is incomplete.

Cause: One (or more) of the descriptors in a package contains errors. The OracleAS TopLink Mapping Workbench places an error icon ⚠ beside the incorrect package.

Action: Expand the package to display its descriptors.

One of the packages is incomplete.

Cause: One (or more) of the packages in your project contains errors. The OracleAS TopLink Mapping Workbench places an error icon ⚠ beside the incorrect project.

Action: Expand the project to display its packages.

One of the tables is incomplete.

Cause: One (or more) of the database tables in your project contains errors. The OracleAS TopLink Mapping Workbench places an error icon ⚠ beside the incorrect table.

Action: Expand the database object to display its tables.

Primary keys do not match across associated tables and no reference(s) specified in multi-table policy info.

Cause: You attempted to associate multiple tables via primary key.

Action: Primary key field names must match across associated tables, or references must be defined from the primary table to each secondary table.

Primary keys(s) do not match parent's primary key(s).

Cause: In an inheritance hierarchy, the child's primary key(s) must match the root's primary key(s).

Action: Ensure that each child's **Primary Key** on the **Descriptor Info** tab matches the parent's primary key.

[descriptor name] references [reference descriptor name], which is not active.

Cause: You tried to select an inactive descriptor as a Reference Descriptor on the mapping's **General** tab.

Action: Either select a new **Reference Descriptor**, or make the descriptor active.

Root class does not include an indicator mapping for this descriptor.

Cause: The root class in the inheritance hierarchy is set to use the class indicator dictionary. The dictionary does not contain an indicator value for this child class.

Action: Select an **Indicator Type** on the **Inheritance** tab of the root class that includes the child types.

Some mappings are incomplete.

Cause: One (or more) of the attributes of a descriptor contains mapping errors. The OracleAS TopLink Mapping Workbench places an error icon ⚠ beside the incorrect attribute(s).

Action: Expand the descriptor to display its mappings.

The Collection class is a Map, but the key method is not selected.

Cause: You created a direct collection mapping that uses a map class, but did not specify the key method.

Action: Select the **Key Method** on the mappings **Collection Options** tab.

The inheritance hierarchy originating in this descriptor cannot contain both aggregate and nonaggregate child descriptors.

Cause: Aggregate and class descriptors cannot be in the same inheritance hierarchy.

Action: Ensure that the inheritance hierarchy contains either aggregate ***or*** nonaggregate children – but not both.

The following mappings "Use Joining" and are mapped to the same reference descriptor: [mapping name]

Cause: You have multiple one-to-one mappings to the same descriptor, that use joining.

Action: Unselect the **Use Joining** option on the **One-to-One Mapping General** tab for all but one mapping.

The [access method type] method for this mapping's method accessing field is no longer visible to this descriptor.

Cause: You changed the class hierarchy within the project, causing the method access type (`get` or `set`) to no longer be visible to the class.

Action: Ensure that the selected method is visible to the class.

The method specified for the copy policy on this descriptor is no longer a visible member of this class.

Cause: You changed the class hierarchy within the project, causing the copy policy to no longer be visible to the class.

Action: Ensure that the selected copy policy is visible to the class.

The database type for the mapping does not match the database type of the database field.

Cause: You selected a **Database Type** for an Object Type mapping that differs from the type in the database.

Action: Select the **Database Type** on the mapping's **General** tab that matches the actual type in the database.

The database is incomplete.

Cause: The databases in a project contains errors. The OracleAS TopLink Mapping Workbench places an error icon  beside the database.

Action: Expand the database to display its tables.

The event policy's [method type] method is no longer a visible member of this descriptor's associated class.

Cause: You changed the class hierarchy within the project, causing the method to no longer be visible to the class.

Action: Ensure that the selected method is visible to the class.

The expression [line number] on query [query name] is invalid.

Cause: One of the arguments in the query expression is missing or invalid.

Action: Edit the query and ensure that all query keys and parameters have been specified.

The following fields have multiple writable mappings: [field name].

Cause: Multiple mappings cannot write to the save database field.

Action: Each database field must have a single, writable mapping.

The following Query Keys do not have associated database fields:

Cause: The database field(s) for the query key(s) listed have been removed from the associated table.

Action: If you want to use these query key(s), then you must specify a database field(s) for them.

The following primary key fields have no writable mappings: [field name].

Cause: Each primary key field must have a writable mapping.

Action: Ensure that the primary key field mappings *are not* read only.

The following primary key fields are unmapped: [field name].

Cause: Each primary key field must have a writable mapping.

Action: Ensure that the primary key(s) are mapped.

The method specified for the copy policy on this descriptor is no longer a visible member of this class.

Cause: You changed the class hierarchy within the project, causing the copy policy to no longer be visible to the class.

Action: Ensure that the copy policy is visible to the class.

The method specified for the inheritance policy's class extraction method on this descriptor is no longer a visible member of this class.

Cause: You changed the class hierarchy within the project, causing the inheritance policy to no longer be visible to the class.

Action: Ensure that the inheritance policy is visible to the class.

The multi-table reference should not be defined on the database.

Cause: When using multi-tables with differently named primary keys, you must set a reference from the TOP table to the BOTTOM table. This reference must not be an actual constraint on the database.

Action: Select the table in which this is defined, and deselect the **On Database** option.

The parent and children of an aggregate descriptor must also be aggregates.

Cause: You created an inheritance hierarchy for an aggregate descriptor.

Action: If an aggregate descriptor is in an inheritance policy hierarchy, then *all* descriptors in the hierarchy must be aggregates.

The reference [table reference] does not have any field associations.

Cause: You selected a table reference for a mapping, but did not add a key pair.

Action: You must specify source and target key pairs for the reference.

The reference must have at least one field association.

Cause: You selected a table reference for a relationship mapping, but did not define a source and target field key pair.

Action: For variable one-to-one mappings, you must define a query key pair (in the source descriptor's tables) to use for the common query key.

The selected parent descriptor for this descriptor's inheritance policy does not have an associated inheritance policy.

Cause: You selected a **Parent Descriptor** for a descriptor's inheritance policy that does not have an inheritance policy.

Action: **Parent Descriptors** must have a valid inheritance policy.

The source reference does not contain any field associations.

Cause: You created a many-to-many mapping, but did not define a source and target reference for the *source* reference.

Action: You must define a table reference and the appropriate key pairs for each source reference.

The source reference must be specified.

Cause: You created a many-to-many mapping, but did not define a source table reference.

Action: Select or create a table reference on the mapping's **Source** tab.

The target reference does not contain any field associations.

Cause: You created a many-to-many mapping, but did not define a source and target reference for the *target* reference.

Action: You must define a table reference and the appropriate key pairs for each target reference.

The target reference must be specified.

Cause: You created a many-to-many mapping, but did not define a target table reference.

Action: On the mapping's **Target** tab, define a target table reference.

There is no database associated with the query key [query key name].

Cause: You did not associated the specified query key with a database table.

Action: You must select a database **Name** and **Table** on the **Query Keys** tab.

This class is a subclass of a final class.

Cause: If you select the **Final** option on the descriptor's **Class Info**, **Class** tab for a class, then the class cannot contain subclasses.

Action:

This root class has no class indicator mappings for its hierarchy.

Cause: You created an inheritance policy with the **Use Class Indicator Dictionary** option but did not specify the indicator values for all subclasses.

Action: Specify the indicator values for all subclasses on the descriptor's **Inheritance** tab.

Note: OracleAS TopLink displays a list of each subclass and indicator value if you have identified the subclasses' parent descriptor.

"Use factory" is specified for the Instantiation policy, but all required information is not specified.

Cause: You selected the **Use Factory** option on the descriptor's **Instantiation Policy** tab, but did not specify the **Factory Class**, **Factory Method**, or **Instantiation Method** fields.

Action: Complete the **Factory Class**, **Factory Method**, or **Instantiation Method** fields on the descriptor's **Instantiation** tab.

"Use method" is selected for the Instantiation policy, but no method is selected.

Cause: You selected the **Use Method** option on the descriptor's **Instantiation Policy** tab, but did not specify the field.

Action: Select the **Method** on the descriptor's **Instantiation** tab.

Writable mappings defined for the class indicator field [field name].

Cause: The class indicator field should not contain any writable mappings.

Action: Select a **Use Class Indicator Field** on the descriptor's **Inheritance** tab that does not contain any writable mappings.

Class Path Issues

The OracleAS TopLink Mapping Workbench does not display the class(es) to import.

Cause: Your classes are not available for import on the Select Classes dialog box.

Action: Ensure that the class is in your project's class path (on the project's **General** properties tab).

Ensure that the class is in the `.zip` or `.jar` file. You cannot import compressed classes.

The OracleAS TopLink Mapping Workbench generates an exception error when importing classes.

Cause: OracleAS TopLink's class import utility did not start correctly. One of the classes includes a static initialization method, which may cause the import utility to fail.

Action: Ensure that your project's class path points to the *root* folder of your package hierarchy. For example, to import the `com.company.class` package in the `C:\classes\com\company` directory, your project class path should be `C:\classes\`.

The OracleAS TopLink Mapping Workbench fails to import the class, but does not generate an exception error.

Action: Ensure that you have properly indicated the directories that contain the domain class(es) to map on the project's **General** tab.

The class path containing your JDBC drivers should still be on your system CLASSPATH. The OracleAS TopLink Mapping Workbench class path is for domain classes only.

Database Connections

If the OracleAS TopLink Mapping Workbench encounters problems communicating or logging into the database, you should:

- Ensure that the driver class, login name, password, and JDBC database URL are correct.
- Verify that your system class path includes all files (for example, native `.dll` files) required by the driver.

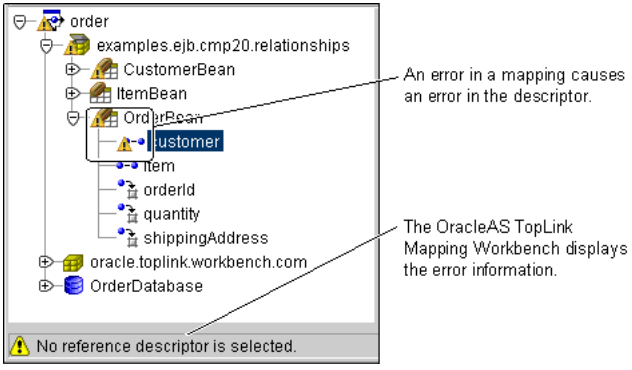
- Verify that your OracleAS TopLink class path includes all classes and files (for example, `.zip` or `.jar`) required by the driver.
- Consult with your database administrator and confirm that the:
  - Database server is setup correctly
  - Database permissions are set correctly
  - Database has enough available connections

Troubleshooting Descriptors

OracleAS TopLink checks each descriptor and mapping to ensure that you have properly defined the required settings.

If a descriptor contains a deficient mapping or property, then the OracleAS TopLink Mapping Workbench displays a yellow caution icon to the left of its icon. If you select the error, then the OracleAS TopLink Mapping Workbench displays the complete error message in the status bar.

Figure C–1 Sample Deficient Mapping



Refer to the specific error message (see "Error Messages" on page C-2) and Chapter 4, "Understanding Descriptors" for more information on working with descriptors.

The Project Status report (see "Generating the Project Status Report" on page 2-5) for information on generating an error report.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z