Oracle9*i*AS TopLink

Tutorials

Release 2 (9.0.3)

August 2002

Part No. B10062-01

ORACLE

Oracle9*i*AS TopLink Tutorials, Release 2 (9.0.3)

Part No. B10062-01

Contents

2 Advanced Tutorial

Index

Send Us Your Comments

Oracle9*i*AS TopLink Tutorials, Release 2 (9.0.3)

Part No. B10062-01

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: iasdocs_us@oracle.com
- FAX: 650-506-7407 Attn: Oracle9*i* Application Server Documentation Manager
- Postal service:
  Oracle Corporation
  Oracle9*i* Application Server Documentation
  500 Oracle Parkway, M/S 2op3
  Redwood Shores, CA 94065
  USA

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

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

Preface

This document provides basic and advanced tutorials, illustrating how to use TopLink Mapping Workbench.

This preface contains the following topics:

Intended Audience

This document is intended for new Mapping Workbench users who want to quickly build a Mapping Workbench project and use many of TopLink's features.

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.

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

Structure

This document includes the following chapters:

Chapter 1, "Introductory Tutorial"

This chapter describes how to create a simple Mapping Workbench project from a database including: enabling Java classes, working with database tables, and implementing mappings.

Chapter 2, "Advanced Tutorial"

This chapter includes information on advanced TopLink functions such as: self relationships and advanced mapping types, inheritance, transformations, and generating code.

Related Documents

For more information, see these Oracle resources:

Oracle9*i*AS TopLink Getting Started

Provides installation procedures to install and configure TopLink. It also introduces the concepts with which you should be familiar to get the most out of TopLink.

Oracle9*i*AS TopLink Tutorial

Provides tutorials illustrating the use of TopLink. It is written for developers who are familiar with the object-oriented programming and Java development environments.

Oracle9*i*AS TopLink Foundation Library Guide

Introduces TopLink and the concepts and techniques required to build an effective TopLink application. It also gives a brief overview of relational databases and describes who TopLink accesses relational databases from the object-oriented Java domain.

Oracle9*i*AS TopLink Mapping Workbench Reference Guide

Includes the concepts required for using the TopLink Mapping Workbench, a stand-alone application that creates and manages your descriptors and mappings for a project. This document includes information on each Mapping Workbench function and option and is written for developers who are familiar with the object-oriented programming and Java development environments.

Oracle9*i*AS TopLink Container Managed Persistence for Application Servers

Provides information on TopLink container-managed persistence (CMP) support for application servers. Oracle provides an individual document for each application server specifically supported by TopLink CMP.

Oracle9*i*AS TopLink Troubleshooting

Contains general information about TopLink's error handling strategy, the types of errors that can occur, and Frequently Asked Questions (FAQs). It also discusses

troubleshooting procedures and provides a list of the exceptions that can occur, the most probable cause of the error condition, and the recommended action.

In North America, printed documentation is available for sale in the Oracle Store at

`http://oraclestore.oracle.com/`

Customers in Europe, the Middle East, and Africa (EMEA) can purchase documentation from

`http://www.oraclebookshop.com/`

Other customers can contact their Oracle representative to purchase printed documentation.

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/admin/account/membership.html`

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/index.htm`

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 for Microsoft Windows Operating Systems

Conventions in Text

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

Convention	Meaning	Example
Italics	Italic typeface indicates book titles or emphasis.	*Oracle9i Database Concepts* Ensure that the recovery catalog and target database do *not* reside on the same disk.
`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` 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`* `])`

Convention	Meaning	Example
{ }	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 ■ That you can repeat a portion of the code	CREATE TABLE ... AS *subquery*; SELECT *col1*, *col2*, ... , *coln* FROM employees;
. . .	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	
Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown.	acctbal NUMBER(11,2); acct CONSTANT NUMBER(4) := 3;
Italics	Italicized text indicates placeholders or variables for which you must supply particular values.	CONNECT SYSTEM/*system_password* DB_NAME = *database_name*
UPPERCASE	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase.	SELECT last_name, employee_id FROM employees; SELECT * FROM USER_TABLES; DROP TABLE hr.employees;
lowercase	Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files. **Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	SELECT last_name, employee_id FROM employees; sqlplus hr/hr CREATE USER mjones IDENTIFIED BY ty3MU9;

Conventions for Microsoft Windows Operating Systems

The following table describes conventions for Microsoft Windows operating systems and provides examples of their use.

Convention	Meaning	Example
Choose Start >	How to start a program.	To start the Oracle Database Configuration Assistant, choose Start > Programs > Oracle - *HOME_NAME* > Configuration and Migration Tools > Database Configuration Assistant.
File and directory names	File and directory names are not case sensitive. The following special characters are not allowed: left angle bracket (<), right angle bracket (>), colon (:), double quotation marks ("), slash (/), pipe (|), and dash (-). The special character backslash (\) is treated as an element separator, even when it appears in quotes. If the file name begins with \\, then Windows assumes it uses the Universal Naming Convention.	`c:\winnt"\"system32` is the same as `C:\WINNT\SYSTEM32`
`C:\>`	Represents the Windows command prompt of the current hard disk drive. The escape character in a command prompt is the caret (^). Your prompt reflects the subdirectory in which you are working. Referred to as the *command prompt* in this manual.	`C:\oracle\oradata>`

	The backslash (\) special character is sometimes required as an escape character for the double quotation mark (") special character at the Windows command prompt. Parentheses and the single quotation mark (') do not require an escape character. Refer to your Windows operating system documentation for more information on escape and special characters.	`C:\>exp scott/tiger TABLES=emp QUERY=\"WHERE job='SALESMAN' and sal<1600\"`
		`C:\>imp SYSTEM/`*`password`*
		`FROMUSER=scott TABLES=(emp, dept)`

HOME_NAME	Represents the Oracle home name. The home name can be up to 16 alphanumeric characters. The only special character allowed in the home name is the underscore.	`C:\> net start Oracle`*`HOME_NAME`*`TNSListener`

Convention	Meaning	Example
ORACLE_HOME and *ORACLE_BASE*	In releases prior to Oracle8*i* release 8.1.3, when you installed Oracle components, all subdirectories were located under a top level *ORACLE_HOME* directory that by default used one of the following names: ■ `C:\orant` for Windows NT ■ `C:\orawin95` for Windows 95 ■ `C:\orawin98` for Windows 98 This release complies with Optimal Flexible Architecture (OFA) guidelines. All subdirectories are not under a top level *ORACLE_HOME* directory. There is a top level directory called *ORACLE_BASE* that by default is `C:\oracle`. If you install Oracle9*i* release 1 (9.0.1) on a computer with no other Oracle software installed, then the default setting for the first Oracle home directory is `C:\oracle\ora90`. The Oracle home directory is located directly under *ORACLE_BASE*. All directory path examples in this guide follow OFA conventions. Refer to *Oracle9i Database Getting Starting for Windows* for additional information about OFA compliances and for information about installing Oracle products in non-OFA compliant directories.	Go to the *ORACLE_BASE*`\`*ORACLE_HOME*`\rdbms\admin` directory.

1

Introductory Tutorial

In the introductory tutorial, you will create mappings from a simple database application using TopLink Mapping Workbench. You will learn how to:

- Create a new project
- Enable and adding 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

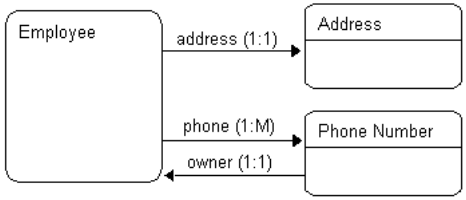
This tutorial project will manage the employee database at the ACME Leisure Company. The system tracks each employee's name, address, and phone number.

The system use these classes:

- `Employee` – Represents both full-time ACME employees and temporary contractors working on ACME projects. It includes the employee's personal information as well as references to his or her home address 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.

The following figure illustrates the object model for this system.

Figure 1–1 The Simple ACME Object Model



Creating the Database Schema

The ACME employee system stores the employee data in three database tables. To use this tutorial, create these tables in your database application.

> **Note:** The column types listed here are generic; the actual column types depend on the database used.

Table 1–1 The EMPLOYEE Table

Column name	Column type	Details
EMP_ID	NUMERIC(15)	Primary key
NAME	VARCHAR(40)	
ADDRESS_ID	NUMERIC(15)	

Table 1–2 The 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 1–3 The PHONENUMBER 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

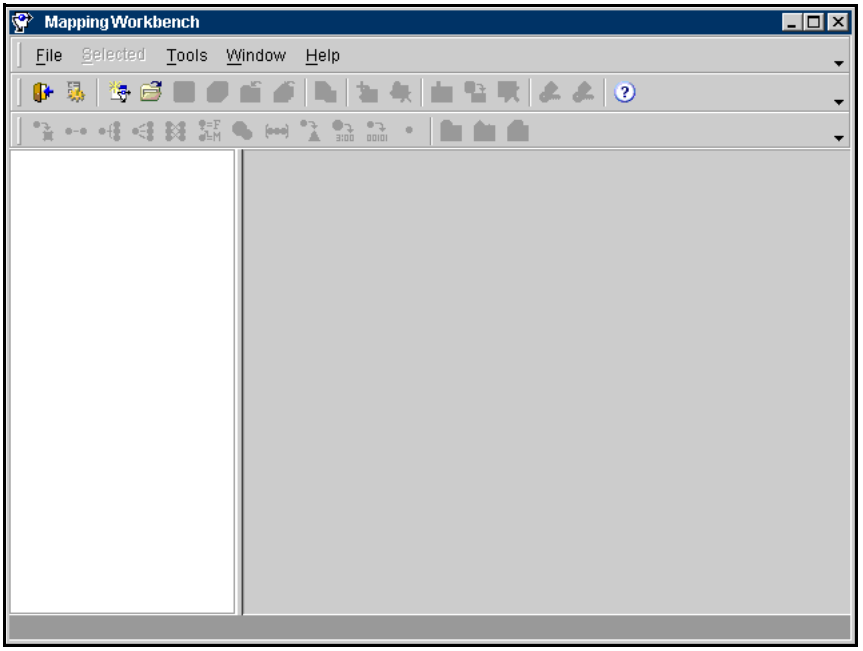
TopLink Mapping Workbench stores project information in the `.mwp` file and associated folders. You should always start a Mapping Workbench project in a new folder.

To create a new project:

1. Start TopLink Mapping Workbench. From the Windows **Start** menu, select **Programs > Oracle9iAS TopLink > Mapping Workbench**.

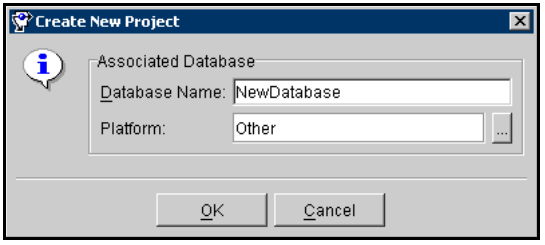
   The splash screen appears, followed by the TopLink Mapping Workbench screen.

Figure 1–2 Mapping Workbench



2. Click on the **New Project** button in the toolbar or select **File > New Project** from the menu. The Create a New Project window appears.

Figure 1–3 Create New Project

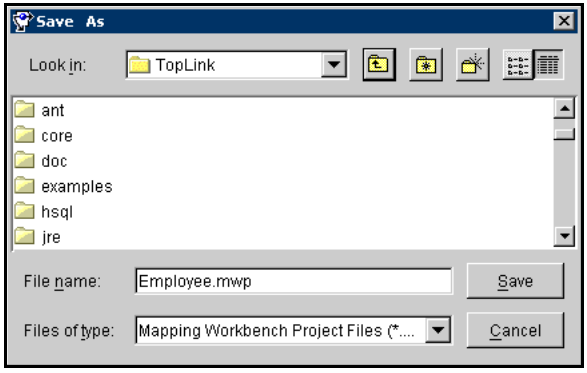


3. From the Create New Project window:
   - In the **Database Name** field, type `INTRO_TUTORIAL_DB`.

- In the **Platform** field, click on the browse button and select the appropriate database platform. Contact your database administrator if you need additional information.

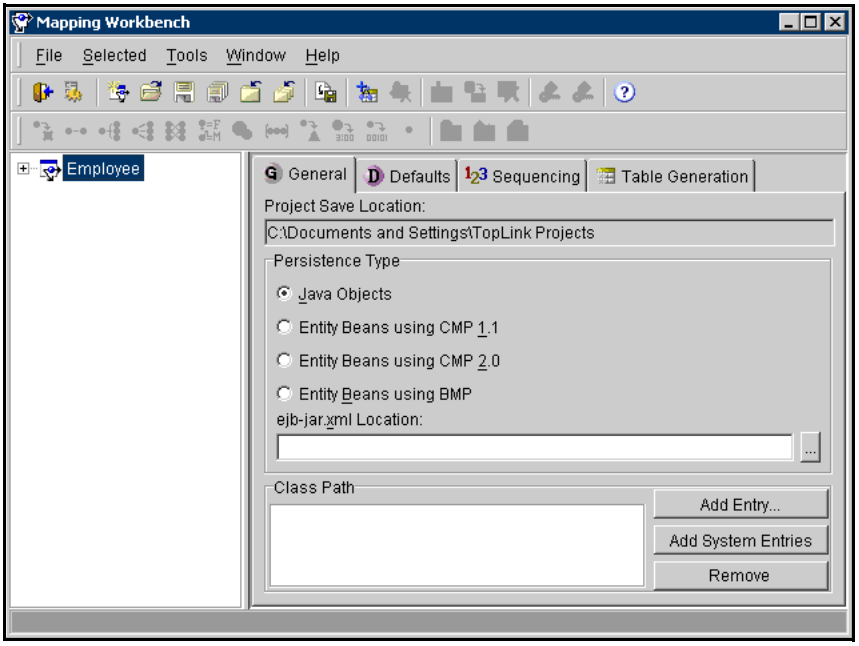
4. Click **OK**. The Save As screen appears.

Figure 1–4 Save As



5. Select the folder in which to save the **Employee** project.
6. In the **File Name:** field, type `Employee.mwp`.
7. Click **Save** to save your work and return to the TopLink Mapping Workbench.

Figure 1–5 Mapping Workbench



8. Click on **Save** in the toolbar or select **File > Save** to save the project.

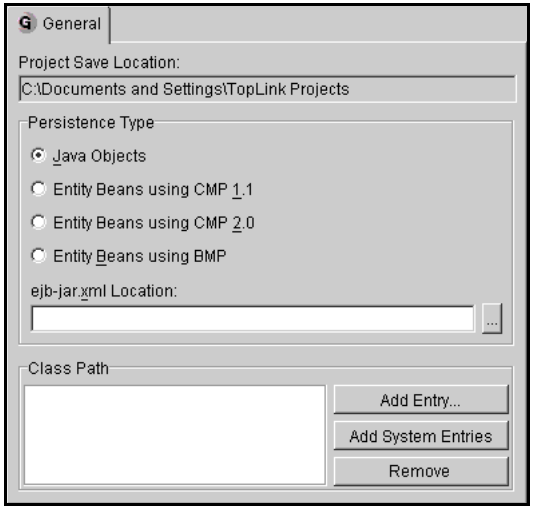
> **Note:** TopLink Mapping Workbench does not automatically save your work; remember to save periodically.

Setting the Project's Classpath

Each TopLink project uses a classpath – a set of directories, `.jar` files, and `.zip` files – when importing Java classes and defining object types.

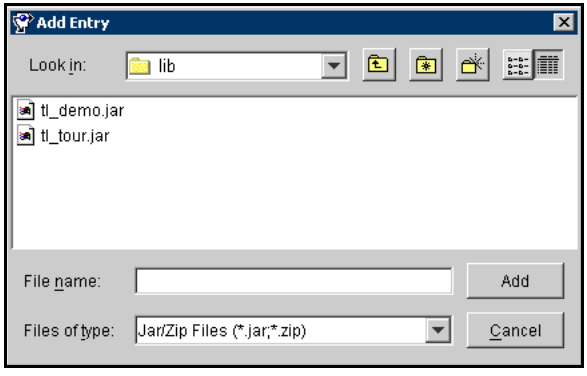
1. In the Mapping Workbench, highlight the `Employee` project in the **Project Tree** pane.
2. In the **Properties** pane on the right-hand side of the TopLink Mapping Workbench screen, click on the **General** tab.

Figure 1–6 General Tab



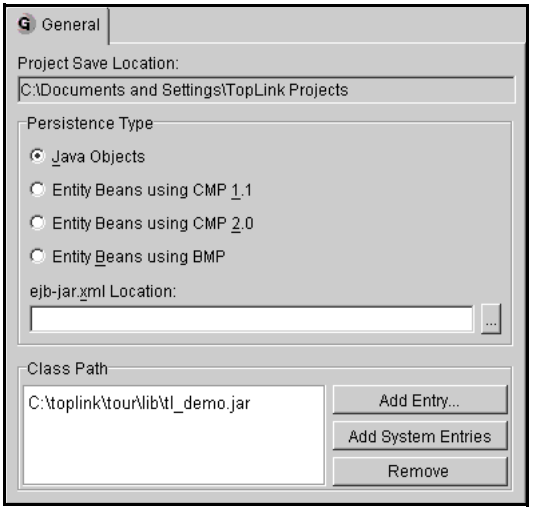
3. Click on **Add Entry**. The Add Entry screen appears.

Figure 1–7 Add Entry



4. Browse to the *<INSTALL_DIR>*`\tour\lib\tl_demo.jar` directory and click on **Add**.

Figure 1–8 General Tab with Classpath Information



5. Click on **Save** in the toolbar or select **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 `oracle.toplink.demos.employee.domain package`) for this tutorial, as described in "Generating the Class Definitions" on page 1-10.

The following table shows how the classes relate to the database tables.

Table 1–4 Employee Classes and Database Tables

Column	Class Attribute	Database Type	Java Type
EMPLOYEE	Employee		
EMP_ID	id	NUMERIC(15)	BigDecimal

Table 1–4 Employee Classes and Database Tables (Cont.)

Column	Class Attribute	Database Type	Java Type
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: It is good programming practice to supply each of the class members in TopLink enabled classes with accessor methods. For this tutorial, the `get` and `set` methods for each of the attributes are provided. The `Employee` class should have an `addPhoneNumber()` method to allow a new `PhoneNumber` to store a reference to its parent.

Example 1–1 Accessor Method Example

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);
}
```

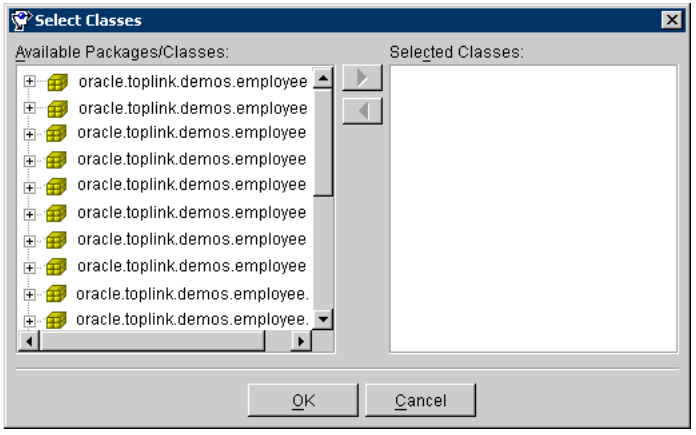
Generating the Class Definitions

You must generate a TopLink descriptor for each Java class in the project.

To create descriptors from the class definition file:

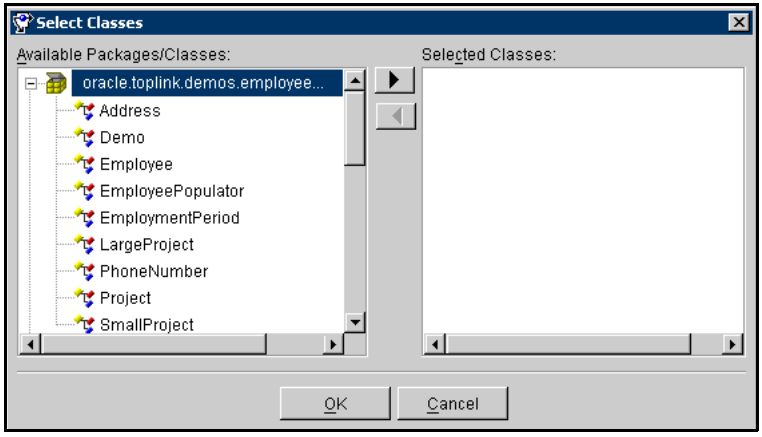
1. From the TopLink Mapping Workbench screen, click on the `Employee` project.
2. Click on the **Add/Update Class** button or select **Selected > Add/Update Classes** from the menu. The Select Classes screen appears.

Figure 1–9 Select Classes



3. Locate the `oracle.toplink.tutorials.intro` package. Click on the plus sign (⊞) to expand that package (or double-click on the name to expand the package).

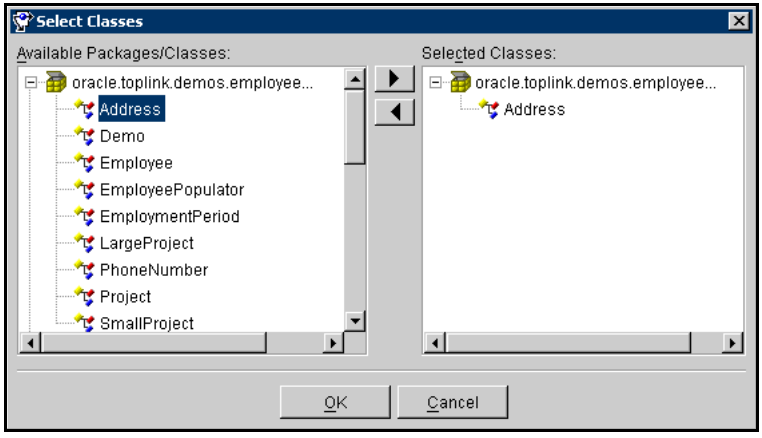
Figure 1–10 Demo Classes



4. Highlight the `Address` class, and click the ▶ button or double-click on the class.

   TopLink copies the `Address` class to the **Selected Classes** pane.

Figure 1–11 Selected Class



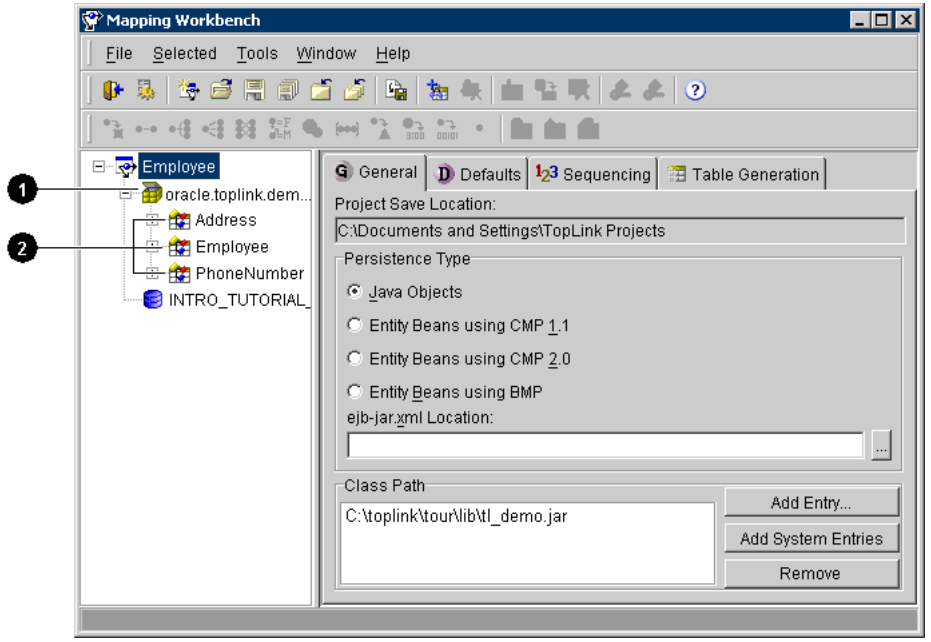
5. Repeat step Step 4 for `Employee` and `PhoneNumber` classes in that package. (Or, highlight both classes using **Shift+click** or **Ctrl+click** as necessary, and click the ▶ button once to import all of the remaining classes.)

6. Click **OK** to import the classes. TopLink creates a descriptor for each class and an unmapped mapping for each attribute.

   The descriptors and their attributes appear in the **Project Tree** pane on TopLink Mapping Workbench screen.

   Note: If an error occurs during this operation, check that the given classes are included in the CLASSPATH and that JDK has been installed correctly.

Figure 1–12 Mapping Workbench with Employee Project



User-interface elements called out in Figure 1–12:

1. Package
2. Class/descriptors

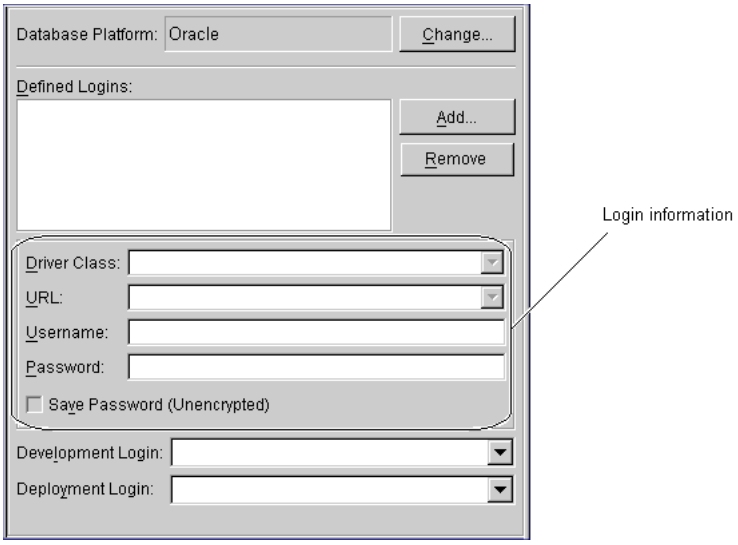
7. Save your changes. Click on the **Save Project** button or select **File > Save** from the menu.

Logging into the Database

You can enter database table information directly from 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.

1. Click on the **INTRO_TUTORIAL_DB** database object in the **Project Tree** pane. The Database Properties pane appears on the right-hand side of the TopLink Mapping Workbench screen.

Figure 1–13 Database Properties



2. Click on **Add** to create a new database login. Contact your database administrator for the necessary database login information.
3. In the toolbar, click the **Login to Database** button or select **Selected > Login** from the menu. The database icon changes to .

> **Note:** 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 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 *Oracle9iAS TopLink Troubleshooting Guide* for details.

Creating Tables

You can enter database table information (as specified in "Creating the Database Schema" on page 1-2) directly from TopLink Mapping Workbench or import the tables from the database.

- To create the tables from TopLink Mapping Workbench, continue with "Creating Tables Using the Mapping Workbench" on page 1-14.
- To create the tables by importing from the database, continue with "Importing Tables from the Database" on page 1-16.

Creating Tables Using the Mapping Workbench

Use this procedure to create the database tables from the 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 use the Mapping Workbench to create table definitions. Later you can create the actual tables on the database.

To create the tables:

1. Click on the database in the **Project Tree** pane and click on the **Add a New Table to the Database** button  or right-click on the database in the project tree pane, and choose **Add New Table.** The New Table dialog box appears.
2. Type `ADDRESS` for the table name, and click **OK**.

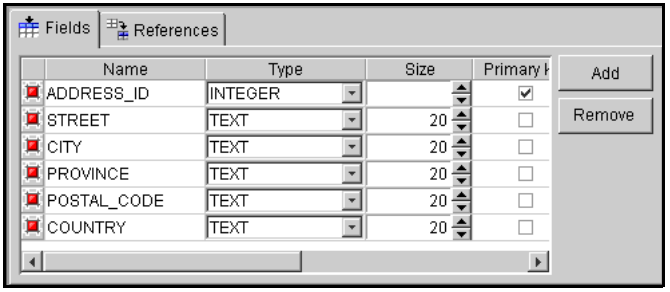
Note: Leave the **Catalog** and **Schema** fields blank.

---

3. Click on the **Fields** tab in the **Properties** pane.
4. Click on the **Add** button to add each database field for the ADDRESS table. Refer to the tables in "Creating the Database Schema" on page 1-2 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**).

Figure 1–14 Database Fields Tab



5. Save your changes. Click on the **Save Project** button or select **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 TopLink Mapping Workbench, you can automatically create the tables on the database.

To create tables on the database:

1. Right-click on one of the database tables in the tree pane, and select **Create on Database > All Tables** from the pop-up menu.

> **Note:** To use the **Create on Database** option you must be logged into the database.

If the Confirm Replace window 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)

or

- by someone using the same database name for a business project (in which case you should not replace it)

The system displays a message indicating that the three tables were created.

2. Click **OK** to close the dialog and return to the TopLink Mapping Workbench screen.
3. In the toolbar, click the **Logout of Database** button  or select **Selected > Log Out** from the menu.
4. Save your changes. Click on the **Save Project** button  or select **File > Save** from the menu.

Continue with "Mapping Classes and Tables in the Descriptor" on page 1-18.

Tip: TopLink Mapping Workbench can generate Data Definition Language (DDL) creation scripts that can be used to create tables on the desired database. See the *Oracle9iAS TopLink Mapping Workbench Reference Guide* for more information. If the table creation fails, there may 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. The DDL may not be compatible with some databases, so you may have to edit the generated DDL and execute the DDL manually.

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 Mapping Workbench.

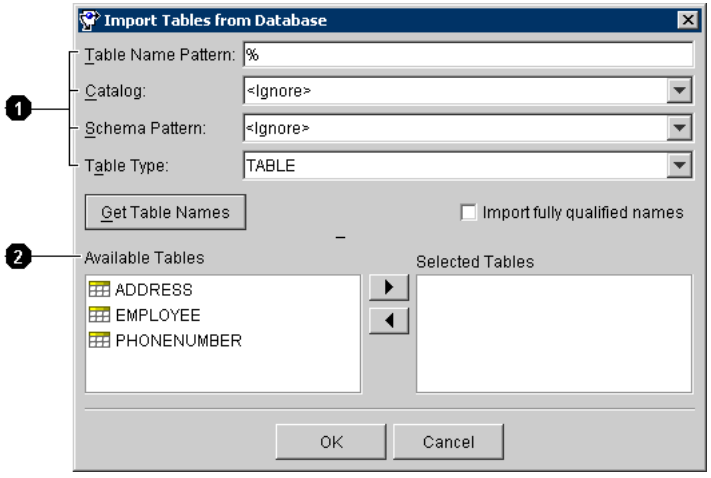
Caution: Do not use this procedure if you plan to create the tables directly from Mapping Workbench.

To import tables from the database:

1. Click the **Login to Database** button or select **Selected > Login** from the menu.
2. Click on the database in the **Project Tree** pane and click on the **Add/Update Existing Tables from Database** button or right-click on the database choose **Add/Update Existing Tables From Database** from the pop-up menu.

   The Import tables from database window appears.

Figure 1–15 Import Tables from Database



User-interface elements called out in Figure 1–15:

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 on the **Get Table Names** button to display all tables in the database.

> **Note:** You may use the table filters to specify which database tables are available for import. For this tutorial, leave the filters as their default values.

4. Click on 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 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, choose a table in the **Project Tree** pane and click on the **Fields** tab in the **Properties** pane.
8. In the toolbar, click the **Logout of Database** button  or select **Selected > Log Out** from the menu.
9. Save your changes. Click on the **Save Project** button  or select **File > Save** from the menu.

Continue with "Mapping Classes and Tables in the Descriptor" on page 1-18.

Mapping Classes and Tables in the Descriptor

When you create a new project and generate class definitions, 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 the mapping the descriptors, you can access the database from a Java application.

Mappings

The 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. TopLink Mapping Workbench stores the descriptors in XML files in the `Descriptor` directory. At run time, TopLink creates instances of the `Descriptor` class for each of the descriptor files and stores them in a database session.

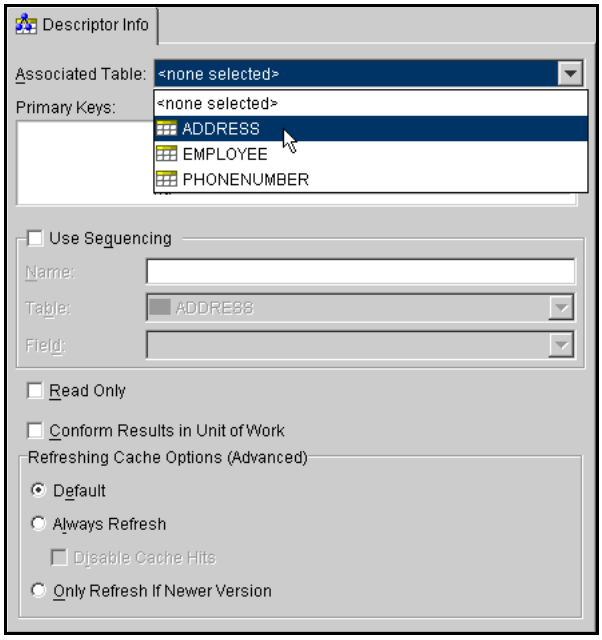
Mapping Classes to Tables

To map Java classes to a table:

1. Click on the `Address` descriptor from the **Project Tree** pane.

2. Click on the **Descriptor Info** tab of the **Properties** pane.
3. In the **Associated Table** drop-down, select the `Address` table.

Figure 1–16 Descriptor Info Tab



> **Note:** A warning message appears indicating that the primary key fields are unmapped. This will be 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 on the **Save Project** button or select **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

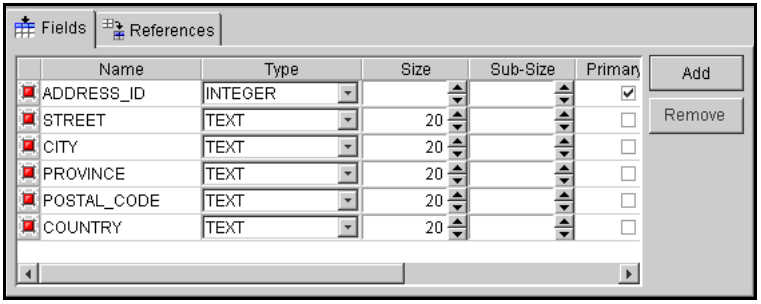
A table's primary key is the field (or fields) used to uniquely identify its records. The PHONE table has a compound primary key (EMP_ID and TYPE fields).

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. 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 is inserted into a database.

To specify the primary key:

1. In the **Project Tree** pane, click the ADDRESS database table.
2. On the **Field** tab of the **Properties** pane, make sure the ADDRESS_ID column is selected as a **Primary Key**.

Figure 1–17 Database Table Field Tab



3. Repeat steps 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 on the **Save Project** button or select **File > Save** from the menu.

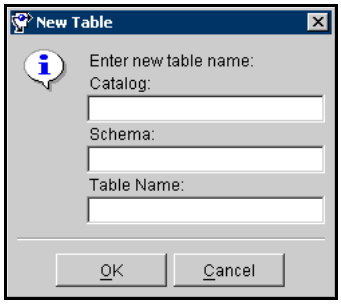
Setting the Sequence Table

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.

To create a sequence table:

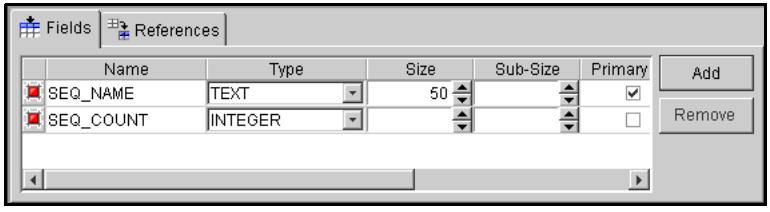
1. Click on the database in the **Project Tree** pane and log into the database by clicking the **Login** button or by right-clicking on the **Database** in the **Project Tree** and choosing **Login** from the pop-up menu.
2. Click on the **Database** in the **Project Tree** pane and click the **Add Table** button. The New Table dialog box appears.

Figure 1–18 New Table



3. Create a table named `SEQUENCE`.
4. Add the following fields to the table:
   - `SEQ_NAME (TEXT type)`
   - `SEQ_COUNT (INTEGER type)`

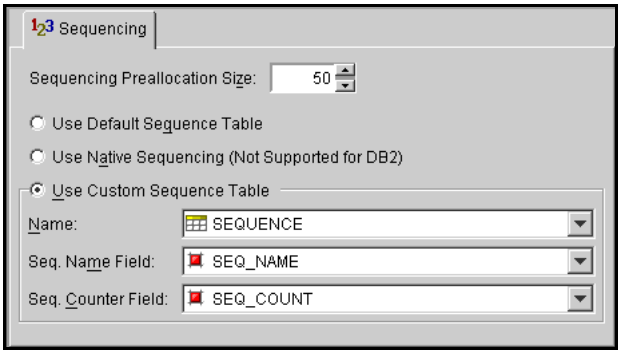
Figure 1–19 Database Table Fields Tab



5. Set the `SEQ_NAME` field as the primary key.
6. Log out of the database by right-clicking on the **Database** in the project tree and choosing **Log Out** from the pop-up menu.
7. Click on the **Project** in the **Project Tree** pane.

8. Click the project's **Sequencing** tab in the **Properties** pane.

Figure 1–20 Sequencing Tab



9. Select **Use Custom Sequence Table** and use the drop lists to choose the **Name**, **Seq. Name Field**, and **Seq. Counter Field**, as shown in Figure 1–20.
10. Save your changes. Click on the **Save Project** button or select **File > Save** from the menu.

Note: You will set the individual sequence names for each table later.

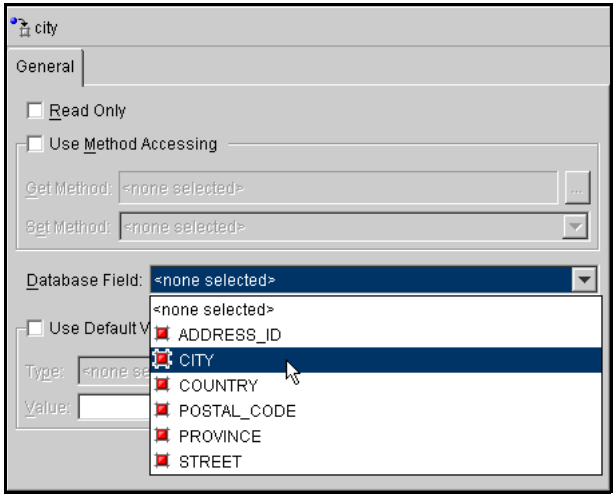
Implementing Direct-to-field Mappings

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 **Project Tree** pane.
2. Click on the `city` attribute.
3. Click the **Direct-to-field** mapping button in the mapping toolbar. The Direct-to-Field mapping tab appears in the **Properties** pane.
4. Use the **Database field** drop-down list to select the `CITY` field.

Figure 1–21 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
   - 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 on the **Save Project** button or select **File > Save** from the menu.

Setting the Sequence Name

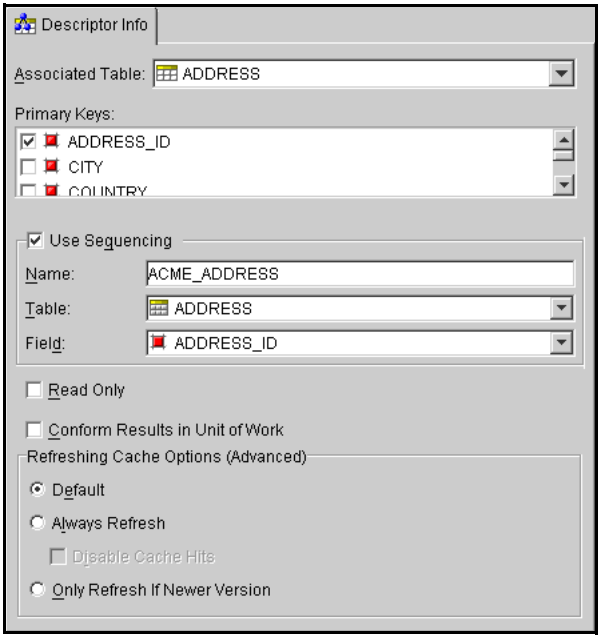
The `Address` and `Employee` classes use non-native sequencing for primary keys.

> **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 TopLink's SchemaManager. Refer to the *Oracle9iAS TopLink Mapping Workbench Reference Guide* for more information.

To set the sequencing for the Address and Employee classes:

1. Select the **Address** descriptor in the **Project Tree** pane.
2. Click the **Descriptor Info** in the **Properties** 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 1–22.

Figure 1–22 Descriptor Info Tab



5. Save your changes. Click on the **Save Project** button or select **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 on the **Save Project** button or select **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.

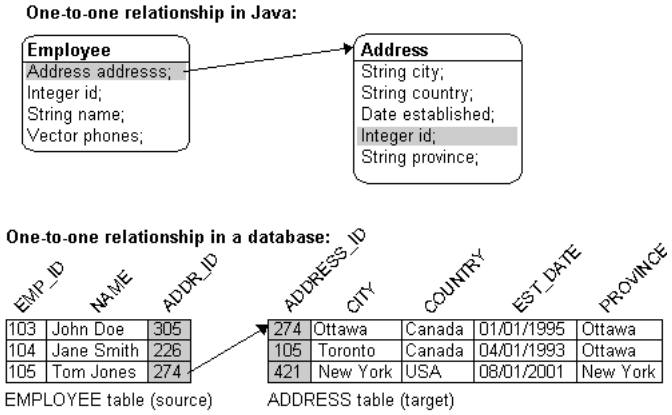
Creating One-to-one Mappings Between Objects

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 name attribute as direct-to-field to the NAME field.
2. Map the id attribute as a direct-to-field mapping to the EMP_ID field (the primary key).

There is only one home address associated with each employee, so the address attribute requires a *one-to-one mapping* with the Address class. Figure 1–23 illustrates a sample one-to-one mapping.

Figure 1–23 One-to-one Mappings

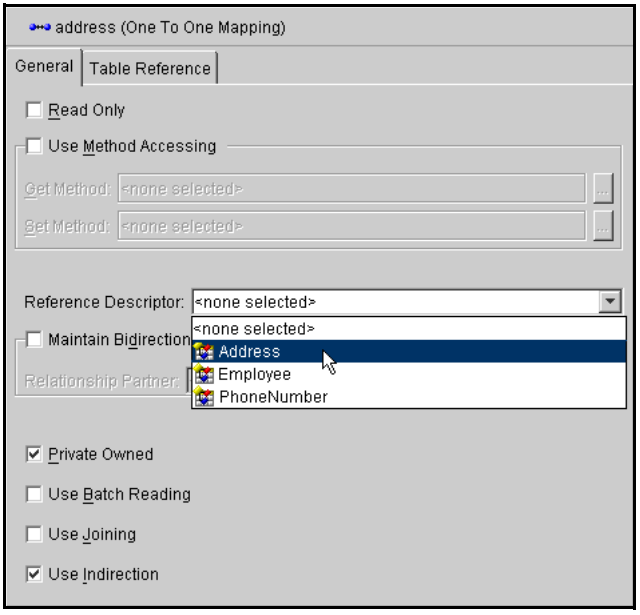


To create a one-to-one mapping

1. Click on the Employee's address attribute in the **Project Tree** pane, then click the **One-to-one Mapping** button in the mapping toolbar.

   The **Properties** pane displays the appropriate information for a one-to-one relationship to be specified.

Figure 1–24 One-to-One Mapping General Tab



2. Select the **Use Indirection** check box.
3. 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.
4. Use the **Reference Descriptor** drop-down list in the to select `Address` as the reference descriptor.

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 TopLink knows how to search for a referenced object, as illustrated in Figure 1–23.

1. Click the **Table Reference tab**.
2. Create a new table reference by clicking the **New** button.
3. In the New Reference Dialog, create a reference whose:
   - Name is `EMPLOYEE_ADDRESS`

- Source table is `EMPLOYEE`
- Target database is `ADDRESS`

Tip: If you leave the **Name** field blank, TopLink automatically builds the name as `<SourceTable>_<TargetTable>`.

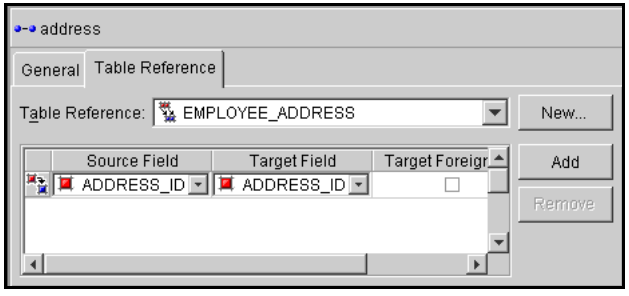
Select the **On Database** option if you want to create the reference on the database when you create the tables. TopLink doesn't require that you actually have the constraint on the database, but you may wish to do this for other reasons. Consult your database administrator for more information.

Note: The mapping is **from** the `EMPLOYEE` table, `ADDRESS_ID` *attribute* **to** the `ADDRESS` *table*.

---

4. Select `EMPLOYEE_ADDRESS` from the **Table Reference** drop-down list.
5. Click the **Add** button to define the foreign key fields.
   - In the **Source** column, choose `ADDRESS_ID` (foreign key).
   - In the **Target** column, choose `ADDRESS_ID` (primary key).
   - Leave the **Target Foreign Key** option unchecked.

Figure 1–25 One-to-One Mapping Table Reference Tab

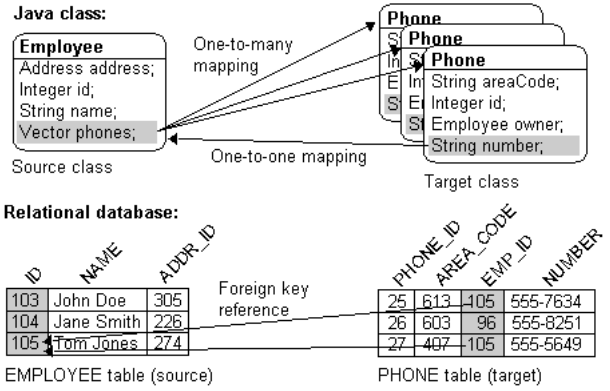


6. Save your changes. Click on the **Save Project** button or select **File > Save** from the menu.

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 1–26 One-to-many Mappings



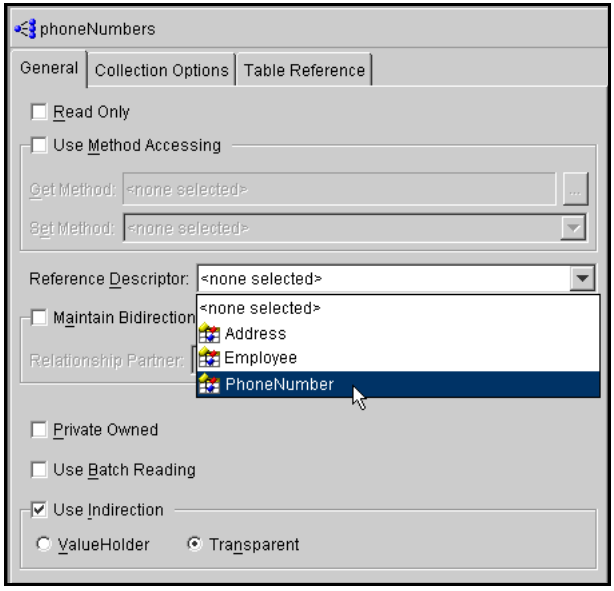
In this tutorial, the Employee project requires:

- 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:

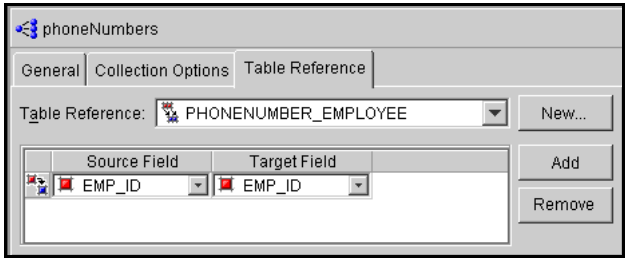
1. Expand the `Employee` class in the **Project tree** pane.
2. Select the `phoneNumbers` attribute.
3. Click the **One-to-many Mapping** button  in the mapping toolbar. The **Properties** pane changes to allow the appropriate information for a one-to-many relationship to be specified.
4. Use the **Reference Descriptor** drop-down list to choose `PhoneNumber`.

Figure 1–27 One-to-many Mapping General Tab



5. Click the **Table Reference** tab, and add a new reference by clicking on **New**.
   - Create a new reference named `PHONE_EMPLOYEE` with a source table of `PHONE` and target table of `EMPLOYEE` and click on **OK**.
   - In the **Table Reference** drop-down list, select 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 1–28 One-to-many Mapping Table Reference Tab



6. Save your changes. Click on the **Save Project** button  or select **File > Save** from the menu.

> **Note:** Leave the remaining attributes of the `Employee` descriptor as unmapped. They will be used 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 1-25).

> **Note:** You do not need to create a new table reference. You can select the same `PHONE_EMPLOYEE` reference you created when you mapped the one-to-many from `Employee` to `PhoneNumber`.

2. Select `EMPLOYEE` as the **Reference Descriptor**.
3. Map the remaining attributes in the `Phone` descriptor as direct-to-field mappings (refer to "Implementing Direct-to-field Mappings" on page 1-22).
4. Remove all unmapped attributes in the `Employee` descriptor. Right-click on the attribute and select **Remove** from the pop-up menu.

   You can also remove attributes by selecting **Selected >Remove** from the menu.

Setting up Database Sessions

A *database session* in TopLink for Java 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 — used for caching and maintaining 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.

A test class has been included with TopLink 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 code fragment in the following Example 1–2, "Logging in and Creating a Project Example Code" illustrates this approach.

Example 1–2 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:\\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 TopLink Mapping Workbench to create database tables. 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 examples illustrates how to create the `EMPLOYEE` table after having logged in to 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 1–3 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 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 either be 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.

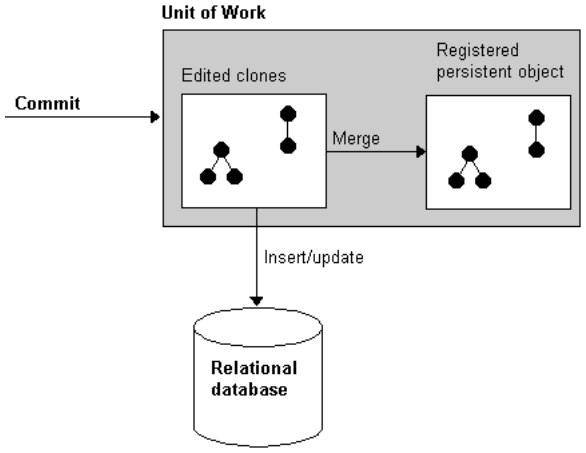
A *unit of work* is an object that simplifies the transaction process and stores transaction information for its registered persistent objects. The unit of work enhances database commit performance by updating only the changed portions of

an object. Units of work are the preferred method of writing to a database in 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 illustrated in Figure 1–29.

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 1–29 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 using 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 "Logging into a Database" on page 1-31, Example 1–3 and uses the session created there.

Example 1–4 Unit of Work Example

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 change to the database using the `commit()` method.

Example 1–5 Session Example

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 tutorial explained the basic steps required to create a Java project that accesses a relational database through TopLink. The main concepts explained include:

- creating Java classes which represent database tables
- using TopLink Mapping Workbench to create tables on the database
- creating *descriptors* for those classes using TopLink Mapping Workbench
- registering the descriptors with the TopLink session
- logging in to the database and doing simple read and write operations

2

Advanced Tutorial

In this advanced tutorial, we 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
- Implementing 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 will add the ability to track employees' current projects, managers, and contract period. You will reuse components from the introductory tutorial.

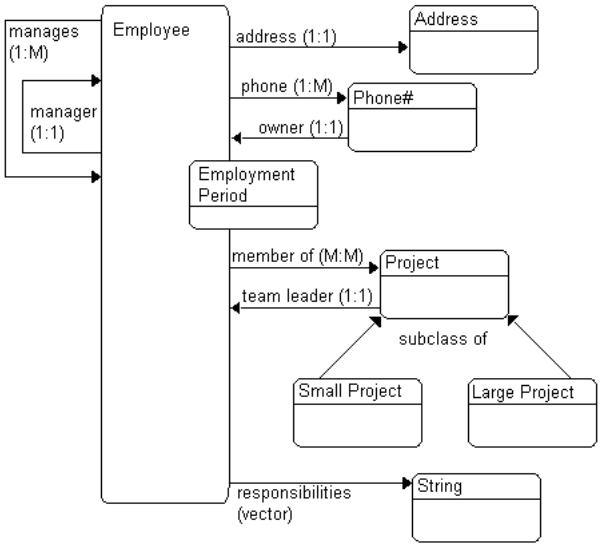
In addition to the `Employee`, `Address`, and `PhoneNumber` classes from the introductory tutorial (see "Overview" on page 1-1), the advanced tutorial uses these classes:

- EmploymentPeriod – Defines the contract term for contractors and the hire date for ACME employees. Each `Employee` class has an `EmploymentPeriod`.

- Responsibility List – Each `Employee` has a collection of text that describes the employee's job.
- `Project` – 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.
- Team Leader – 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 2–1 illustrates the object model for the advanced tutorial.

Figure 2–1 The 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 2–9 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 2–1 The 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	
START_TIME	TIME	
END_TIME	TIME	
MANAGER_ID	NUMERIC(15)	
VERSION	NUMERIC(15)	

Table 2–2 The SALARY Table

Column name	Column type	Details
EMP_ID	NUMERIC(15)	Primary key
SALARY	NUMERIC(10)	

Table 2–3 The 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 2–4 The 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 2–5 The 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 2–6 The LPROJECT Table

Column name	Column type	Details
PROJ_ID	NUMERIC(15)	Primary key
BUDGET	NUMERIC(10,2)	
MILESTONE	TIMESTAMP	

Table 2–7 The RESPONS Table

Column name	Column type	Details
EMP_ID	NUMERIC(15)	Primary key
DESCRIP	VARCHAR(200)	

Table 2–8 The 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 2–9 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*

Table 2–9 Relationships Between Classes and Database Table (Cont.)

Column	Class Attribute	Database Type	Java Type
DESCRIP	`responsibilitiesList`	VARCHAR(200)	String
PROJECT	`LargeProject` and `SmallProject`		
PROJ_ID	`id`	NUMERIC(15)	BigDecimal
DESCRIP	`description`	VARCHAR(200)	String
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 described in "Creating a New Project" on page 1-3.
   - 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 `oracle.toplink.tutorials.advanced` package. See "Setting the Project's Classpath" on page 1-6.
3. Enable the following classes in the `oracle.toplink.tutorials.employee` package and generate a TopLink descriptor for each Java class as described in "Generating the Class Definitions" on page 1-10:
   - `Address`
   - `Employee`
   - `EmploymentPeriod`
   - `LargeProject`
   - `PhoneNumber`
   - `Project`
   - `SmallProject`

   Table 2–9 shows how the classes relate to the database tables.
4. Log into the database as described in "Logging into the Database" on page 1-13 to create or import the database information.

   Select one of the following methods to add database information:
   - Creating Tables Using the Mapping Workbench
   - Importing Tables from the Database

   Refer to Table 2–1 through Table 2–8 for complete database information.

Mapping Classes to Tables

Map the each Java class in the Advanced tutorial to a database table as described in "Mapping Classes to Tables" on page 1-18.

Map this class...	To this database table...
Address	ADDRESS
Employee	EMPLOYEE
LargeProject	LPROJECT
PhoneNumber	PHONENUMBER
Project	PROJECT

Ensure that the primary keys are correctly indicated, as specified in Table 2–1 through Table 2–8.

> **Note:** A warning message appears indicating that you have not yet mapped the attributes. This will be addressed later in the tutorial

Using the Automap Tool

TopLink can automatically map class attributes to similarly named database. This **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:** Although **Automap** correctly maps most one-to-one and direct-to-field mappings, you should 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 **Project Tree** pane and click on the **Descriptor Info** tab in the **Properties** pane.
2. In the **Associated Table** drop-down list, select the ADDRESS table.

You must associate the class with a table before using the **Automap** tool.

3. Right-click on the `Address` class in the **Project Tree** pane and select **Automap** from the pop-up menu.

   You can also automap descriptors by selecting **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 *off*, when an object is retrieved from the database all of 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 is strongly recommended. See the *Oracle9iAS TopLink Mapping Workbench Reference Guide* 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.

Indirection can be implemented 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 the value holder should be defined 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 2–1 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.

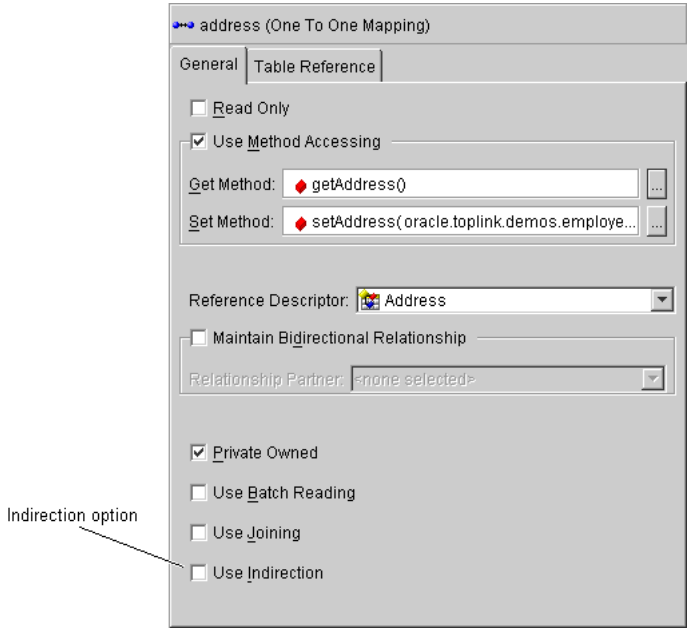
Implementing Indirection in the Mapping Workbench

After modifying the code, update the 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 2–2 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
```

Project

```
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 2–1).

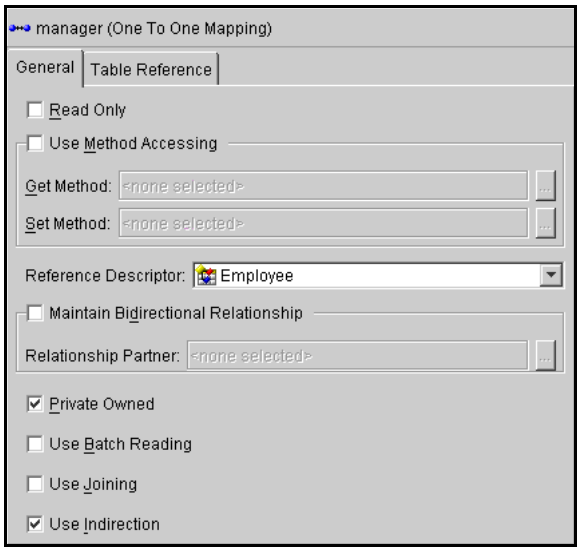
To map the manager attribute:

1. Click on the Employee's `manager` attribute in the **Project Tree** pane, then click the **One-to-one mapping** button  in the mapping toolbar.

   The **Properties** pane displays the appropriate information for a one-to-one relationship to be specified.

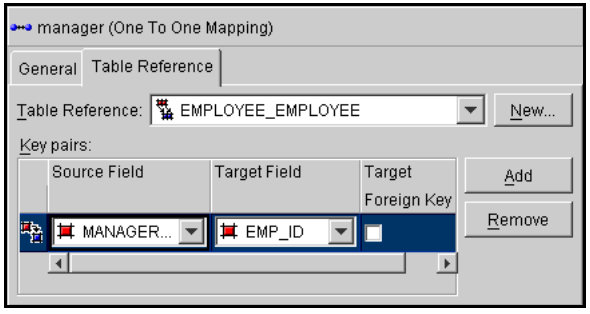
2. Use the **Reference Descriptor** drop-down list in the to select `Employee` as the reference descriptor.

Figure 2–3 One-to-one Mapping General Tab



3. Select the **Use Indirection** option. See "Implementing Indirection in the Tutorial" on page 2-11.
4. Click the **Table Reference** tab.

Figure 2–4 One-to-one Mapping Table Reference Tab



5. Create a new table reference by clicking the **New** button.
6. In the New Reference Dialog, create a reference whose:
   - Name is EMPLOYEE_EMPLOYEE

- Source table is EMPLOYEE
- Target database is EMPLOYEE

Note: If you leave the **Name** field blank, TopLink automatically builds the name 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 column**, choose `MANAGER_ID` (foreign key) field.
   - In the **Target** 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 on **Save**  in the toolbar or select **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 `Project` descriptor

Create these mappings as shown in "Creating One-to-one Mappings Between Objects" on page 1-25. Refer to Table 2–9 for the correct relationships. Enable indirection for each of these mappings, as indicated in "Implementing Indirection in the Tutorial" on page 2-11.

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 2–1).

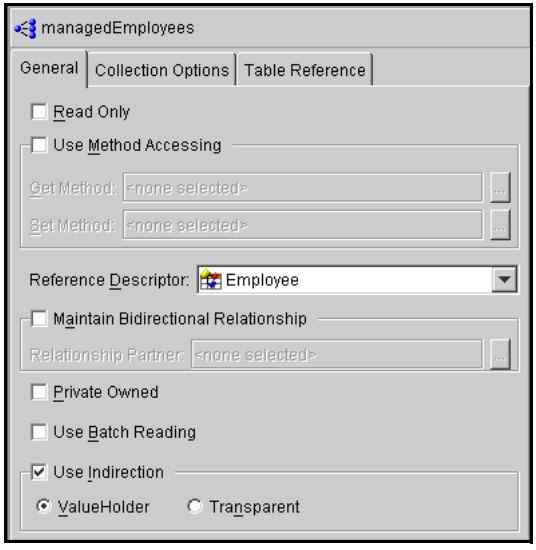
To map the managedEmployee attribute:

1. Click on the Employee's `managedEmployees` attribute in the **Project Tree** pane, then click the **One-to-many mapping** button in the mapping toolbar.

   The **Properties** 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 2–5 One-to-many Mapping General Tab

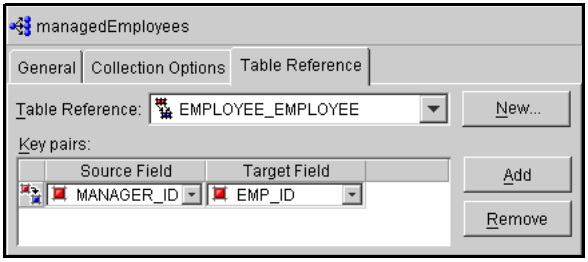


3. Select the **Use Indirection** option, and choose **ValueHolder**. See "Implementing Indirection in the Tutorial" on page 2-11.

4. Click the **Table Reference** tab. Use the **Table Reference** drop-down list to select the EMPLOYEE_EMPLOYEE table (previously created in "To map the manager attribute:" on page 2-12).

- Click the **Add** button on the **Table Reference** tab to add a foreign key relationship.
- Set the **Source** (foreign key) field to `MANAGER_ID`.
- Set the **Target** (primary key) field to `EMP_ID`.

Figure 2–6 One-to-many Mapping Table Reference Tab



5. Click on **Save** in the toolbar or select **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 1-28. Refer to Table 2–9 for the correct relationship.

Enable indirection for this mapping, as indicated in "Implementing Indirection in the Tutorial" on page 2-11.

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.

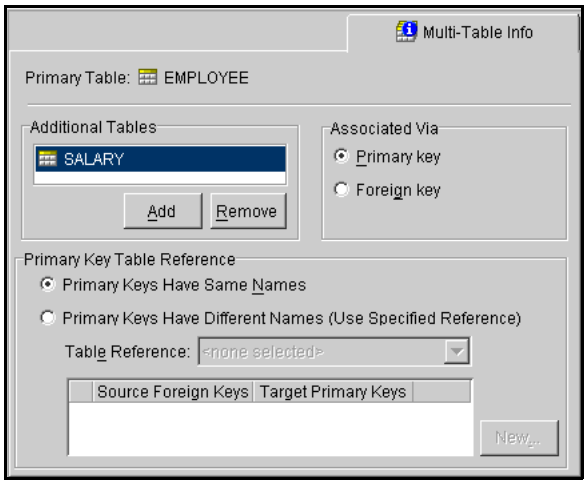
To map the Employee class to multiple tables:

1. Click on the `Employee` descriptor in the **Project Tree** pane.
2. Click on the **Multi-table Info** tab in the **Properties** pane.

If the **Multi-table info** tab is not visible, right-click on the Employee descriptor and select **Set Advanced Properties > Multi-table Info** from the pop-up menu.

3. In the **Additional Tables** pane, click on **Add** and add the `SALARY` table.

Figure 2–7 Multi-table Info Tab



4. Click on **Save** in the toolbar or select **File > Save Project** to save the project.

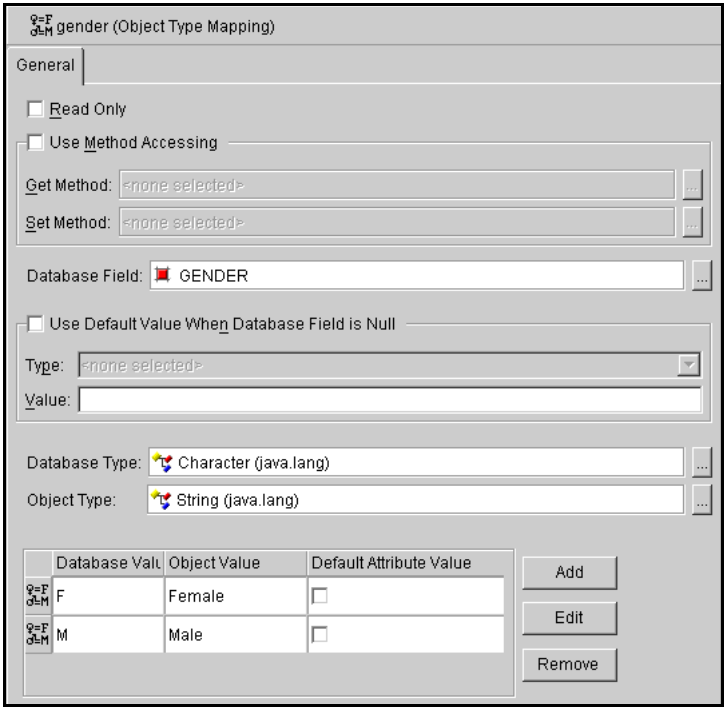
Implementing Object Type Mapping

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 (i.e., **M** or **F**), but the value is the full name (i.e., **Male** or **Female**).

To map the gender attribute:

1. Expand on the `Employee` descriptor in the **Project Tree** pane.
2. Select the `gender` attribute and click on the **Object-type Mapping** button in the mapping toolbar.

Figure 2–8 Object-type Mapping Tab



3. In the **Database Field**, select the GENDER field from the EMPLOYEE table.
4. Select **Character** as the **Database Type** and **String** as the **Object Type**.
5. Click on **Add** and create the following database mappings:

Database Value	Object Value
F	Female
M	Male

6. Click on **Save** in the toolbar or select **File > Save Project** to save the project.

Implementing an Aggregate Object

In TopLink, two objects are related by aggregation if there is 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.

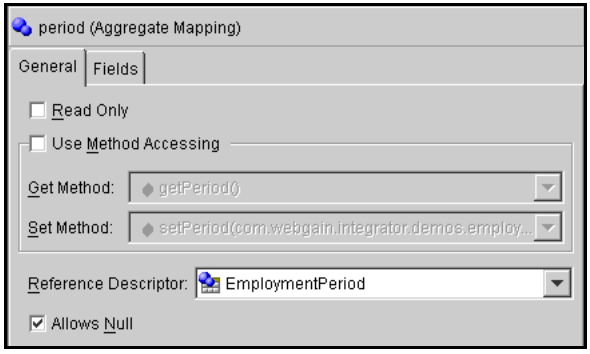
To map an aggregate object:

1. Click on the `EmploymentPeriod` descriptor in the **Project Tree** pane.
2. Click on the **Aggregate Descriptor** button in the mapping toolbar. The descriptor's icon in the **Project Tree** 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.

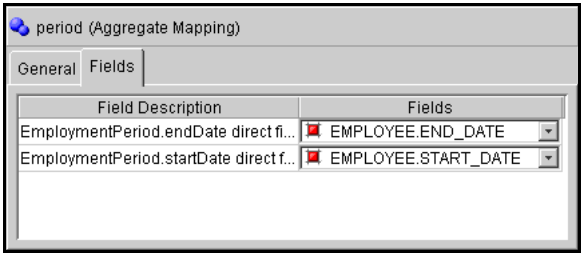
4. Expand the Employee descriptor in the **Project Tree** pane.
5. Select the `period` attribute of the `Employee` descriptor.
6. Click on the **Aggregate Mapping** button in the mapping toolbar.

Figure 2–9 Aggregate Mapping General Tab



7. Use the **Reference Descriptor** drop-down list to select the `EmploymentPeriod` aggregate descriptor.
8. Click on the **Fields** tab.

Figure 2–10 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 on **Save** in the toolbar or select **File > Save Project** to save the project.

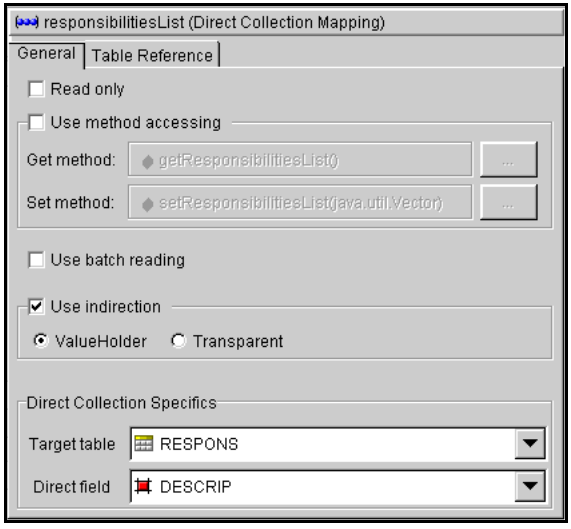
Implementing a Direct Collection Mapping

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 on the `Employee` descriptor in the **Project Tree** pane.
2. Click on the `responsibilitiesList` attribute of the `Employee` descriptor.
3. Click on the **Direct Collect** button in 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 specify 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 2-11.

Figure 2–11 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, select the `RESPONS_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` (from the RESPONS table).
   - Set the **Target** (primary key) field to `EMP_ID` (from the EMPLOYEE table).
7. Click on **Save** in the toolbar or select **File > Save Project** to save the project.

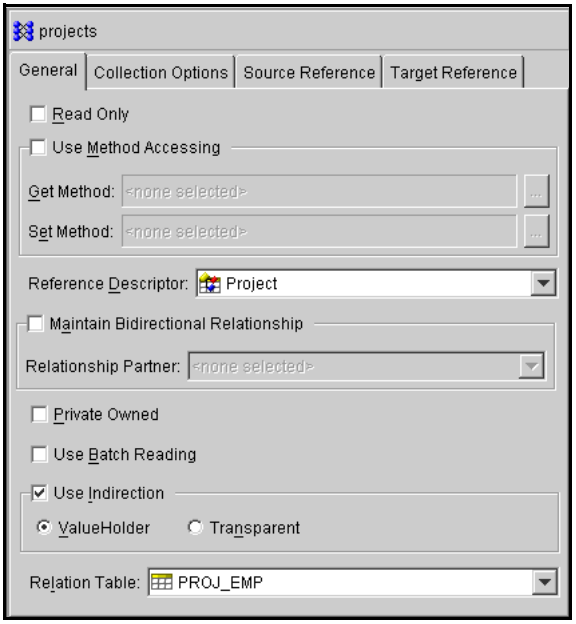
Implementing a Many-to-many Mapping

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 **Project Tree** pane.
2. Click on the `projects` attribute of the `Employee` descriptor.
3. Click on the **Many-to-many Mapping** button in the mapping toolbar.
4. Use the **Reference Descriptor** drop-down list to select the `Project` descriptor.
5. Use the **Relation Table** drop-down list to select 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 2-11.

Figure 2–12 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, select 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, select 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 on **Save**  in the toolbar or select **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 `Project` class. Use the following procedures to set the `Project` descriptor to implement inheritance, then enable the two subclasses.

To implement inheritance in the Project descriptor:

1. Click on the `Project` descriptor in the **Project Tree** pane.
2. Click on the **Inheritance** tab in the Properties pane.

   If the Inheritance tab is not visible, right-click on the Project descriptor and select **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 select `PROJ_TYPE`.

5. Select the **Use Class Indicator Dictionary** field and use the **Indicator Type** drop-down list to select a **String** type.

To implement inheritance in each subclass:

1. Click on the `SmallProject` descriptor in the **Project Tree** pane.
2. Click on the **Inheritance** tab in the Properties pane.

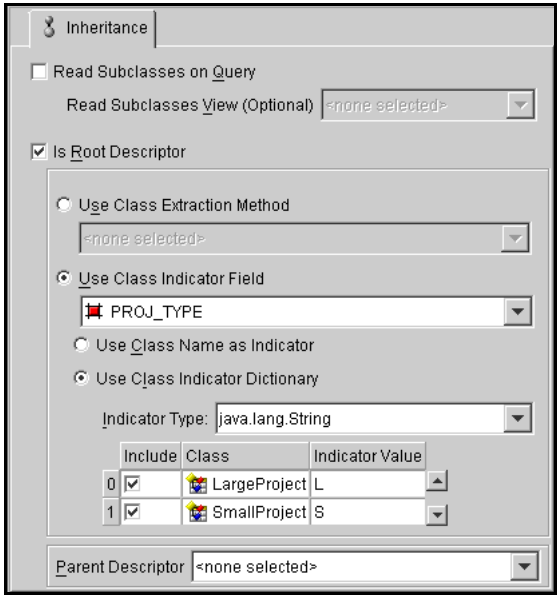
   If the Inheritance tab is not visible, right-click on the Project descriptor and select **Advanced Properties > Inheritance** from the pop-up menu or **Selected > Advanced Properties > Inheritance** from the menu.
3. In the **Parent Descriptor** drop-down list, select the `Project` class.
4. Click on **Save**  in the toolbar or select **File > Save Project** to save the project.

Repeat this procedure for the `LargeProject` descriptor.

To complete the inheritance:

1. Click on the `Project` descriptor in the **Project Tree** pane.
2. Click on the **Inheritance** tab in the Properties pane.
3. For each descriptor:
   - Select the **Include** column.
   - Enter an **Indicator Value** (**S** for `SmallProject` and **L** for `LargeProject`).

Figure 2–13 Inheritance Tab



Note: To have the root class store instances of itself in the table, it must include a value-subclass pair for itself. For the advanced tutorial this is not required 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 are called whenever an object is written to the database. The method returns the value from the object that should be written to the field.

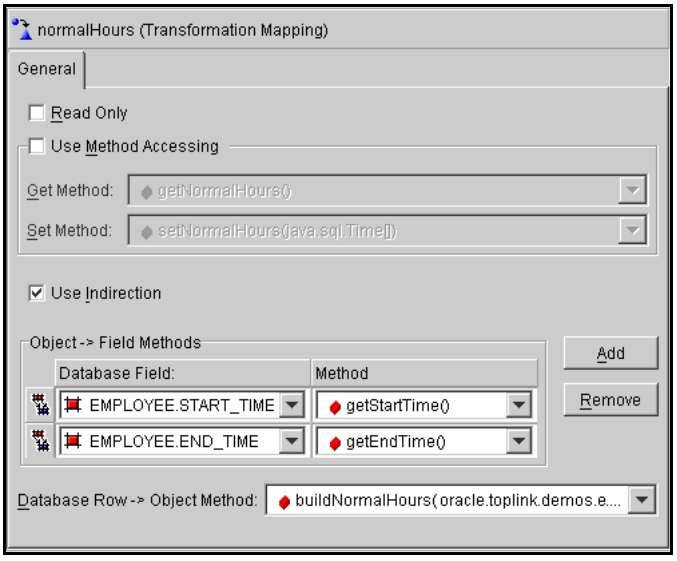
In the advanced tutorial, the transformation method corresponds to a single argument method on the `Employee` class and extracts the values from the fields and places them into the `NormalHours` array.

To map the normalHours attribute:

1. Expand on the `Employee` descriptor in the **Project Tree** pane.
2. Click on the `normalHours` attribute of the `Employee` descriptor.

3. Click on the **Transformation Mapping** button in the mapping toolbar.
4. Click on the **Add** button to create the following **Object->Field Methods**:
   - **Database Field**: START_TIME, **Method**: `getStartTime`
   - **Database Field**: END_TIME, **Method**: `getEndTime`
5. Use the **Database Row -> Object Method** drop-down list to select the **bulidNormalHours** method.

Figure 2–14 Transformation Mapping Tab



6. Click on **Save** in the toolbar or select **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 2–9 to map each attribute.

Generating Code

To use your project with the Foundation Library, you must either generate deployment XML 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 on the project in the **Project Tree** pane and select **Export Project to Java Source** from the pop-up menu. The Choose an Export File window appears.

   You can also export the project by selecting **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 the TopLink's advanced topics and functions.

A completed version of this tutorial is included with a standard installation of TopLink in the following directory:

`<INSTALL_DIR>\workbench\demos\employee\EmployeeDemo.mwp`

Index

A

B

C

D

E

F

G

I

J

K

L

M

N

O

P

R

S

T

U

V