Oracle® AppWizard for Microsoft Visual C++

User’s Guide

Release 8.1.6

January 2000

Part No. A73028-01

Oracle AppWizard for Visual C++ User's Guide, Release 8.1.6

Part No. A73028-01

Primary Authors: Larry Faulks, Bronya Feldmann, Mark Kennedy, Jeff Stein

Contributors: Riaz Ahmed, Sinclair Hsu, Steve Norall, Andrew Quan, Helen Slattery, Nicole Sullivan

Contents

Contact Us!

Oracle AppWizard for Microsoft Visual C++, Release 8.1.6

Part No. A73028-01

This document describes how to contact Oracle Corporation if you have issues with the documentation or software. It also provides a list of useful resources for Oracle partners and developers.

Read the section...	If you...
"How to Contact Oracle Technical Publications" on page viii	Have issues with Documentation
"How to Contact Oracle Support Services" on page ix	Have issues with Software
"Resources for Oracle Partners and Developers" on page xiv	Want to join an Oracle partner or application developer program

How to Contact Oracle Technical Publications

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. 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 about this guide?
- Do you have suggestions for improvement? Please indicate the chapter, section, and page number (if available).

You can send comments regarding documentation in the following ways:

- Electronic mail - ntdoc@us.oracle.com
- FAX - (650) 506-7370   Attn: Oracle Windows Platforms Server Documentation
- Postal service:
  Oracle Corporation
  Windows Platforms Server Documentation Manager
  500 Oracle Parkway, MS 1OP8,
  Redwood Shores, CA 94065
  USA

If you would like a reply, please provide your name, address, and telephone number.

How to Contact Oracle Support Services

Please copy this form and distribute within your organization as necessary.

Oracle Support Services can be reached at the following telephone numbers and Web sites. The hours of business are detailed in your support contract and the *Oracle Customer Support Guide* in your kit.

Oracle Support Services In...	**Call...**
United States of America	+ (650) 506-1500 for customers with support contracts. + (650) 506-5577 to obtain a support contract.
Europe	+44 1344 860 160 or the local support center in your country.
All other locations	The telephone number for your country listed at the following Web site: `http://www.oracle.com/support/contact_us/sup_hot_phone.html` Oracle Support Services telephone numbers are also listed in the *Oracle Customer Support Guide* in your kit.

Please complete the following checklist before you call. If you have this information ready, your call can be processed much quicker.

❏ Your CPU Support Identification Number (CSI Number) if applicable.

____

____

❏ The hardware name on which your application is running.

____

____

____

- ❏ The operating system name and release number on which your application is running.
  - To verify the operating system version on Windows NT, enter the following at the MS-DOS command prompt:

    ```
    C:\> WINMSD
    ```

    The *Windows NT Diagnostics* dialog box displays the operating system and Service Pack version.

- ❏ The release numbers of the Oracle Server and associated products involved in the current problem. For example, Oracle8*i* Enterprise Edition release 8.1.6.0.0 and Oracle Enterprise Manager release 2.1.0.0.0.
  - To verify the release number of the Oracle Server, connect to the database using a tool such as SQL*Plus. The release number is displayed. For example:

    ```
    Connected to:
    Oracle8i Enterprise Edition Release 8.1.6.0.0 - Production
    With the Partitioning and Java options
    PL/SQL Release 8.1.6.0.0 - Production
    ```

- ❏ The third-party software version you are using.
  - To verify an application version, from the application's Help menu, select About...

❑ The exact error codes and messages. Please write these down as they occur. They are critical in helping Oracle Support Services to quickly resolve your problem. Note whether there were no errors reported.

❑ A description of the issue, including:

- **What happened?** For example, the command used and its result.

- **When did it happen?** For example, during peak system load, or after a certain command, or after an operating system upgrade. In addition, what was happening when the problem occurred?

- **Where did it happen?** For example, on a particular system, or within a certain procedure or table.

- **What is the extent of the problem?** For example, production system unavailable, or moderate impact but increasing with time, or minimal impact and stable.
- Did the problem affect one user, several users, or all users?
- Has anything changed? For example, if this is an operation that used to work and now fails, what is different? Can you undo any recent changes, to verify whether they are relevant to the issue?

- **Can the problem be reproduced?** This is a critical question for support analysts. For example, did the problem recur on the same system, under the same circumstances? Can the problem be reproduced on another system? Additionally:
- Does installing a software component fail on all client machines, or just one?
- Do all clients fail to connect to the server, or just one?
- If you are able to restart the server or database, does restarting the database or rebooting the server or client machine (if applicable) make a difference?

❏ Keep copies of the Oracle alert log, any trace files, core dumps, and redo log files recorded at or near the time of the incident. Oracle Support Services may need these to further investigate your problem.

  To help analyze problems:

  - Archive or delete old alert logs. When the database is started without an alert log, a new one is created. In some cases, if you force the problem to recur with a new alert log, the timestamps for the recorded events may indicate which events are relevant.
  - Archive or delete old trace files. To check whether the file was modified, right-click and select Properties. The *Properties* dialog box displays the modification date.
  - Check the operating system error logs, especially the System log and Application log. These files are relevant to the Oracle Server. To view these files, from the Start menu, choose Programs > Administrative Tools > Event Viewer, and choose System or Application from the Log main menu.

Resources for Oracle Partners and Developers

This section provides information on partner programs and resources for Oracle database administrators and application developers.

Information Source	Description
Oracle Corporation Home Page `http://www.oracle.com`	This Web site is the starting point for general information on Oracle Corporation.
Alliance Online `http://alliance.oracle.com`	Oracle provides leading-edge technology, education, and technical support that enables you to effectively integrate Oracle into your business. By joining the Oracle Partner Program, you demonstrate to customers that you are committed to delivering innovative Oracle-based solutions and services. The greater your commitment to Oracle, the more we can help you grow your business. It's that simple. The value you derive is associated directly with your level of commitment.
Oracle Education `http://education.oracle.com/`	Customers come to Oracle Education with a variety of needs. You may require a complete curriculum based on your job role to enable you to implement new technology. Or you may seek an understanding of technology related to your key area of responsibility to help you meet technical challenges. You may be looking for self-paced training that can be used as an ongoing resource for reference and hands-on practice. Or, you may be interested in an overview of a new product upgrade. Whatever your training need, Oracle Education has the solution.
Oracle Technology Network `http://technet.oracle.com/`	The Oracle Technology Network is your definitive source for Oracle technical information for developing for the Internet platform. You will be part of an online community with access to free software, Oracle Technology Network-sponsored Internet developer conferences, and discussion groups on up-to-date Oracle technology. Membership is free.
Oracle Store `http://oraclestore.oracle.com/`	This is Oracle's online shopping center. Come to this site to find special deals on Oracle software, documentation, publications, computer-based training products, and much more.

Information Source	Description
Oracle Support Services' Support Web Center `http://www.oracle.com/support/`	Oracle Support Services offers a range of programs so you can select the support services you need and access them in the way you prefer: by telephone, electronically, or face to face. These award-winning programs help you maintain your investment in Oracle technology and expertise. Here are some of the resources available in the Support Web Center:
Oracle*MetaLink* `http://www.oracle.com/support/elec_sup/index.html`	Oracle*MetaLink* is Oracle Support Services' premier Web support service. It is available to Oracle*metals* customers (Gold, Silver, Bronze), 24 hours a day, seven days a week.
Oracle*Lifecycle* `http://www.oracle.com/support/sup_serv/lifecycle/index.html`	Oracle*Lifecycle* is designed to deliver customized, industry-focused, full life-cycle support solutions that enable industry leaders to use Oracle technology to make smart business decisions, achieve operational excellence, and succeed in their markets.
Expert*ONLINE* `http://www.oracle.com/support/sup_serv/online/index.html`	Oracle Support Services has launched a new line of services called Expert*ONLINE*. These services provide online database administration for companies looking to supplement their existing DBA staff or fill a DBA role. Services range from Expert*DETECT*, a monitoring, diagnostic, and recommendation service, to Expert*DBA*, a full online database administration service.
Virtual Support Analyst (VSA) `http://www.oracle.com/support/sup_serv/vsa_start.html`	VSA is Oracle's Internet e-mail service; it is available to U.S. customers with an Oracle*metals* support agreement. With VSA, you can initiate a request for assistance through e-mail, bypassing the queues you may encounter when using telephone support. VSA also enables you to access Oracle's bug database.
Customer Service `http://www.oracle.com/support/cus_serv/index.html`	This site provides resources to make your interactions with Oracle as easy as possible. Among the things you can do are: ■ Learn what is a CPU Support Identification (CSI) number ■ Update your technical contact information ■ Find out whom to contact for invoice and collection issues ■ Request product update shipments ■ Access a glossary of Oracle Support Services terms

Information Source	Description
U.S. Customer Visit Program `http://www.oracle.com/support/cus_serv/cus_visit.html`	This U.S.-based program has been established to help our customers understand and obtain maximum benefit from the support services they have purchased. The visit typically offers a customized orientation presentation, a comprehensive overview and demonstration of Oracle’s electronic services, and helpful tips on working more effectively with Oracle Support Services.
Support Web Center Library `http://www.oracle.com/support/library/index.html`	This site contains articles, guides, and other documentation to help you leverage the wealth of knowledge and reference material that Oracle Support Services produces.

Before You Begin

This guide helps you to get started with Oracle AppWizard for Microsoft Visual C++.

Specific topics discussed are:

- Prerequisites
- Intended Audience
- How This Guide Is Organized
- Documentation and Code Conventions Explained
- Documentation Library

Prerequisites

This guide assumes that you have an understanding of Microsoft Visual C++ version 5.0 or 6.0.

Intended Audience

This guide is necessary for anyone who wants to use Oracle AppWizard for Microsoft Visual C++.

How This Guide Is Organized

This guide is organized as follows:

Chapter 1, "Introduction"

Describes the main features of Oracle AppWizard for Microsoft Visual C++ and defines basic concepts for the selections on Oracle AppWizard for Microsoft Visual C++.

Chapter 2, "Creating a Starter Application"

A step-by-step description of how to use Oracle AppWizard for Microsoft Visual C++ to create an application skeleton.

Chapter 3, "Understanding Your Application’s Code"

Describes the files and code underlying the application skeleton created by Oracle AppWizard for Microsoft Visual C++, making customization easier.

Chapter 4, "Tutorial"

Takes you through all the steps of creating a starter application with Oracle AppWizard for Microsoft Visual C++, using a business case as an example. After completing the tutorial, readers will gain a much fuller understanding of the real-life applications to which they can put Oracle AppWizard for Microsoft Visual C++.

Documentation and Code Conventions Explained

The following conventions are used in this guide.

Convention	Example	Meaning
All uppercase plain	SQL> ALTER DATABASE	Indicates command names, file names, SQL reserved words, and keywords.
Italic	Italic used to indicate a variable or the title of a guide: *filename* *Oracle SQL Reference*	Indicates a value that you must provide. For example, if a command asks you to type *filename*, you enter the actual name of the file. Italic is also used for emphasis in the text and to indicate the titles of other guides.
square brackets [ ]	`X:\[PATHNAME]\ORACLE\HOME_NAME`	Encloses optional items. For example, when you create an Optimal Flexible Architecture (OFA)-compliant Oracle home directory, you can place an optional pathname before the `\ORACLE` pathname. Square brackets also indicate a function key, for example [Enter].
C:\>	C:\ORACLE>	Represents the Windows platforms command prompt of the current hard disk drive. Your prompt may differ and may, at times, reflect the subdirectory in which you are working. Referred to as the *MS-DOS command prompt* in this guide.
Backslash (\) before a directory name	`\BIN`	Indicates that the directory is a subdirectory of the root directory.

Convention	Example	Meaning
oracle_home and oracle_base	Go to the `ORACLE_BASE\ORACLE_HOME\BIN` directory.	In this Optimal Flexible Architecture (OFA)-compliant release, all subdirectories are no longer under a top level `oracle_home` directory. There is now a new top-level directory called `oracle_base` that by default is `C:\ORACLE`. The Oracle home directories are located directly under `oracle_base`. If you install Oracle8i release 8.1.6 on a computer where there is no other Oracle software on the computer, the default settings for the first Oracle home directory is `C:\ORACLE\ORA81`. If you run Oracle Universal Installer again and install release 8.2.x, the second Oracle home directory is called `\ORA82`. All directory path examples in this guide follow OFA conventions. For more information on OFA, see the *Oracle8i Administrator's Guide for Windows NT*.
oracle_home	OracleHOME_NAMETNSListener	Represents the Oracle home name. The home name can be up to sixteen alphanumeric characters. The only special character allowed in the home name is the underscore
HOME*ID*	HOME0, HOME1, HOME2	Represents a unique registry subkey for each Oracle home directory in which you install products. A new HOME*ID* is created and incremented each time you install products to a different Oracle home directory on one machine. Each HOME*ID* contains its own configuration parameter settings for installed Oracle products.

Convention	Example	Meaning
Symbols	period .	Symbols other than brackets and vertical bars must be entered in commands exactly as shown.
	comma ,	
	hyphen -	
	semicolon ;	
	colon :	
	equal sign =	
	backslash \	
	single quote ‘	
	double quote “	
	parentheses ()	

Documentation Library

This guide is part of a larger library of Oracle documentation. The Oracle documentation library consists of two types of documentation:

Documentation Type	Describes...
Operating system-specific	Installation, configuration, and use of Oracle products in a Windows NT or Windows 95/98 environment. Operating system-specific documents are occasionally referred to in the generic documentation set. These documents are easy to identify because they always mention their specific operating system in their title.
Generic	Oracle database, Oracle networking, and Application Programming Interface information that is uniform across all operating system platforms. The majority of documents in your documentation set belong to this category. While reading through the generic documentation set, you are occasionally asked to refer to your platform (or operating system) documentation for procedures specific to the Windows NT or Windows 95/98 operating systems. To easily identify where these generic documentation references are described in your operating system documentation, see the index of this guide for the following entry: *generic documentation references* All generic documentation references described in this guide appear under this index entry.

1 Introduction

This chapter describes the main features of Oracle AppWizard for Microsoft Visual C++ and defines terminology.

Specific topics discussed are:

- What is Oracle AppWizard for Microsoft Visual C++?
- Oracle Objects for OLE Overview
- Installing Oracle AppWizard for Microsoft Visual C++
- Basic Oracle AppWizard for Microsoft Visual C++ Concepts
- Microsoft Visual C++ 6.0 Limitations

What is Oracle AppWizard for Microsoft Visual C++?

Oracle AppWizard for Microsoft Visual C++ enables you to easily create a starter Oracle application. Oracle AppWizard for Microsoft Visual C++ enables your application to:

- Connect to an Oracle database
- Execute SQL statements and retrieve information from the Oracle database
- Navigate effortlessly through records
- Update, insert, and delete data from the Oracle database

The basic functionality that Oracle AppWizard for Microsoft Visual C++ designs into your application enables you to spend time doing what you do best: writing code for important custom features that make your application successful.

Oracle AppWizard for Microsoft Visual C++ uses standard Oracle Objects for OLE and Microsoft Foundation Class (MFC) Library classes to build an application with these advantages:

- Adds commented code
- Manipulates and retrieves information easily from the Oracle database
- Handles any errors returned by the database
- Uses the document/view architecture for MFC-compliant code
- Uses Hungarian notation
- Lets you optionally update the data through your new application

The Oracle AppWizard for Microsoft Visual C++ runs seamlessly within the Microsoft Visual C++ 5.0 or 6.0 development environment on Windows NT and Windows 95/98, and operates almost identically to the standard MFC AppWizard.

Oracle Objects for OLE Overview

Oracle Objects for OLE is an application programming interface for Windows that interacts with the Oracle database. The application you generate with the Oracle AppWizard for Microsoft Visual C++ uses the Oracle Objects for OLE C++ Class Library to access the Oracle database.

To learn more about Oracle Objects for OLE C++ Class Library:

1. Choose Start > Programs > Oracle **-** *HOME_NAME* > Application Development > Oracle Objects for OLE Class Library Help.

The online Help describes each class in the C++ Class Library.

Installing Oracle AppWizard for Microsoft Visual C++

Oracle AppWizard for Microsoft Visual C++ is installable through the following installation types:

For this Top-Level Component...	Oracle AppWizard for Microsoft Visual C++ Is Installable Through This Installation Type...
Oracle8*i* Enterprise Edition or Oracle8*i*	Custom
Oracle8*i* Client	Programmer, Application User, and Custom

IMPORTANT: You must install Microsoft Visual C++ 5.0 or 6.0 *before* you install Oracle AppWizard for Microsoft Visual C++.

See the Oracle8i Installation Guide for Windows NT or Oracle8i Client Installation Guide for Windows for instructions on installing Oracle AppWizard for Microsoft Visual C++.

Basic Oracle AppWizard for Microsoft Visual C++ Concepts

To use the Oracle AppWizard for Microsoft Visual C++ to successfully generate a starter application, you must understand how to select and build the type of application you want.

Displaying Records

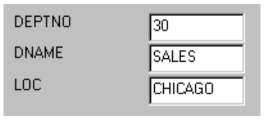
You can generate three types of applications with Oracle AppWizard for Microsoft Visual C++.

- Single-Record Display Form
- Multiple-Record Display Form
- Master-Detail Display Form

Each of these displays records differently.

Single-Record Display Form

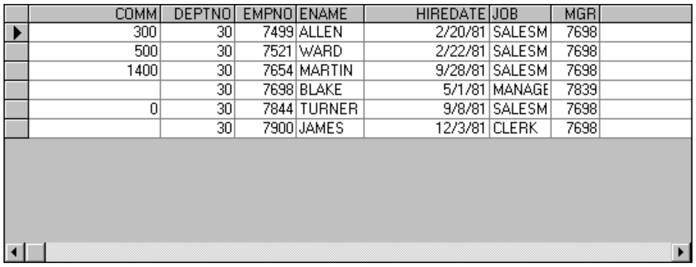
When you design your application to display one record at a time, your application uses edit controls to display records, as seen below:



DEPTNO	30
DNAME	SALES
LOC	CHICAGO

Multiple-Record Display Form

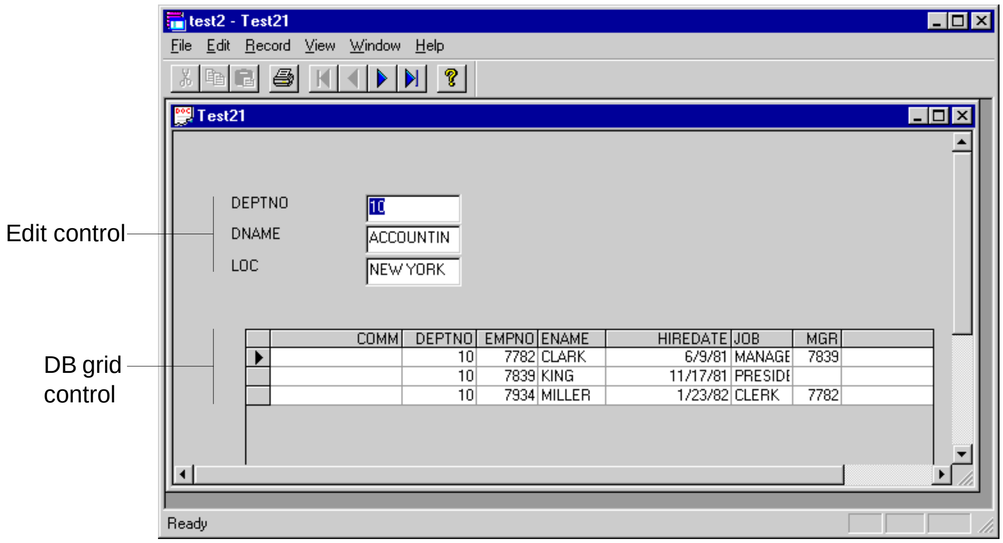
When you design your application to display more than one record at a time, your application uses a database grid control, as seen below:



COMM	DEPTNO	EMPNO	ENAME	HIREDATE	JOB	MGR
300	30	7499	ALLEN	2/20/81	SALESM	7698
500	30	7521	WARD	2/22/81	SALESM	7698
1400	30	7654	MARTIN	9/28/81	SALESM	7698
	30	7698	BLAKE	5/1/81	MANAGE	7839
0	30	7844	TURNER	9/8/81	SALESM	7698
	30	7900	JAMES	12/3/81	CLERK	7698

Master-Detail Display Form

When you design a master-detail form, the master records are displayed in edit controls and the detail records are displayed in a database grid control:



DEPTNO 10
DNAME ACCOUNTIN
LOC NEW YORK

	COMM	DEPTNO	EMPNO	ENAME	HIREDATE	JOB	MGR
▶		10	7782	CLARK	6/9/81	MANAGE	7839
		10	7839	KING	11/17/81	PRESIDE	
		10	7934	MILLER	1/23/82	CLERK	7782

Definitions

The following terms are discussed throughout this guide:

Term	Description
Join	A query that combines rows from two or more tables or views.
Single-record display form	Displays one record from a table at a time in a form.
Multiple-record display form	Displays information in multiple-record format from a table at one time in a form.
Master-detail display form	Displays information in single-record format from a master table and in multiple-record format from a detail table.
Single-document interface (SDI)	SDI applications allow only one open document frame to be open at once during a session.
Multiple-document interface (MDI)	MDI applications allow multiple document frames to be open at once during a session.
Dialog-based interface	Dialog-based applications use a dialog box interface exclusively.

Microsoft Visual C++ 6.0 Limitations

This release works with Microsoft Visual C++ 5.0 and 6.0. However, when using version 6.0, selecting or deselecting the following options either has no effect or does not generate the code for that option:

For This Option...	If You Choose...	The Result Is...
What type of application would you like to create?	Choose "Document/View architecture support"	Default is selected. Deselect does not take effect because the generated MFC-based application must use this architecture.
What compound document support would you like to include?	If you choose "Both container and server" and "Active document container"	MSVC 5.0 container and server support only.
What do you want your toolbars to look?	If you choose "Internet Explorer ReBars"	Internet Explorer ReBars is not supported in this release.
What style of project would you like?	If you choose "Windows Explorer"	Internet Explorer ReBars is not supported in this release.

2

Creating a Starter Application

This chapter describes how to create a starter application with Oracle AppWizard for Microsoft Visual C++.

Specific topics discussed are:

Overview

After starting Oracle AppWizard for Microsoft Visual C++, you complete a series of steps in which you specify which of three application types you want to create:

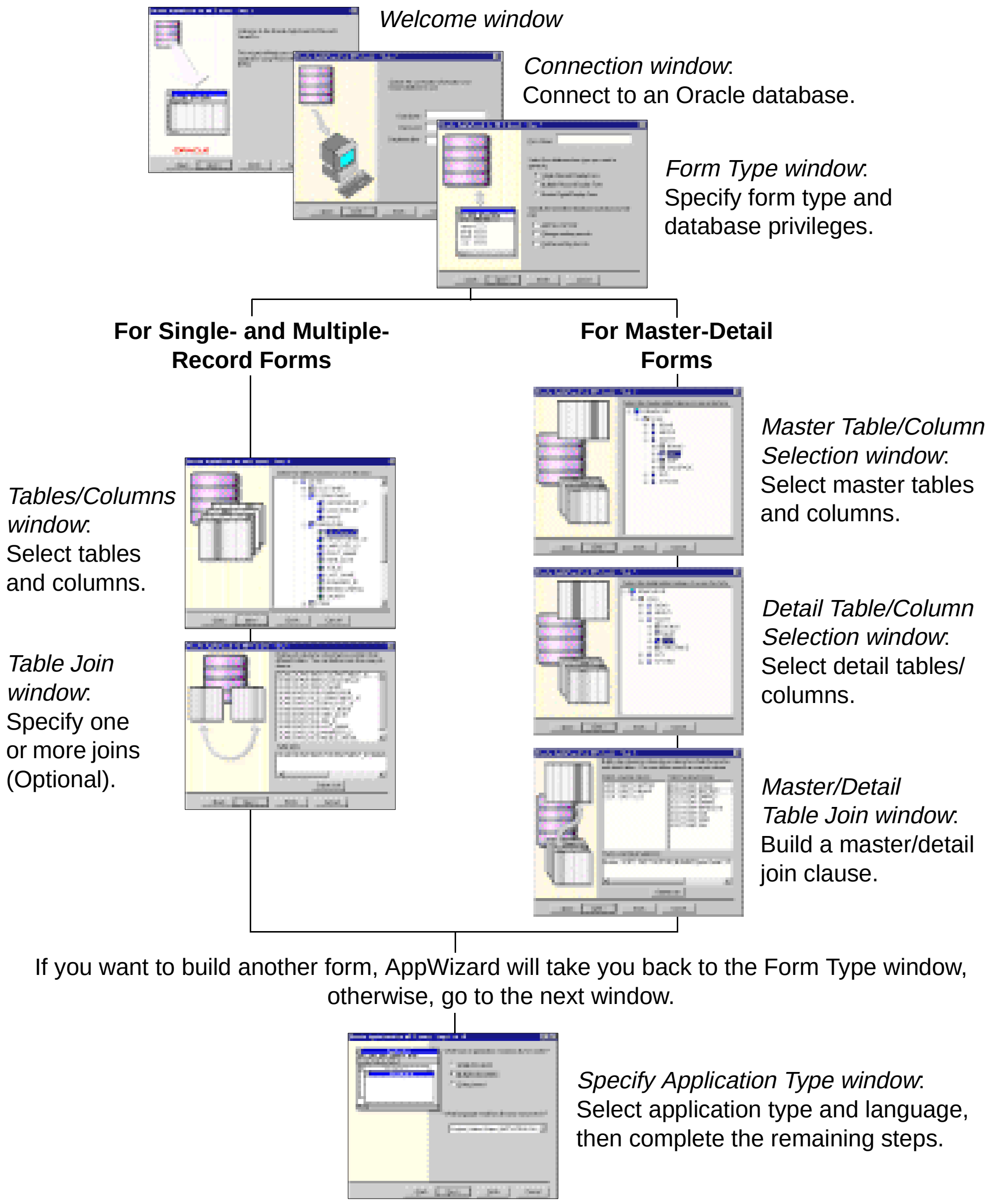
Application Type	Description
Single-Record Display Form	Enables your application to display one record from one or more tables at a time.
Multiple-Record Display Form	Enables your application to display more than one record from one or more tables at a time.
Master-Detail Display Form	Enables your application to display records from tables that have a master-detail relationship to each other.

The first three steps are the same whether you create a single-record, multiple-record, or a master-detail display.

When you finish the last step, Oracle AppWizard for Microsoft Visual C++ generates the application. Application files include the following:

- Source files
- Header files
- Resource files
- Standard MFC project files
- A ReadMe.file describing all the files comprising your application

You then build and run the executable, using Microsoft Visual C++.



Welcome window

Connection window: Connect to an Oracle database.

Form Type window: Specify form type and database privileges.

For Single- and Multiple-Record Forms

Tables/Columns window: Select tables and columns.

Table Join window: Specify one or more joins (Optional).

For Master-Detail Forms

Master Table/Column Selection window: Select master tables and columns.

Detail Table/Column Selection window: Select detail tables/ columns.

Master/Detail Table Join window: Build a master/detail join clause.

If you want to build another form, AppWizard will take you back to the Form Type window, otherwise, go to the next window.

Specify Application Type window: Select application type and language, then complete the remaining steps.

Starting Oracle AppWizard for Microsoft Visual C++

To start Oracle AppWizard for Microsoft Visual C++:

1. Start Microsoft Visual C++ 5.0 or 6.0.
2. Choose File > New.

   The *New* dialog box appears.
3. Click the Projects tab.
4. Select Oracle AppWizard for MFC (exe) from the list of project types.
5. Specify the project name and location.
6. Click OK.

   Oracle AppWizard for Microsoft Visual C++ starts.

Note: If Oracle AppWizard for Microsoft Visual C++ cannot be loaded successfully, ensure that you have installed Oracle Objects for OLE (OO4O) and that the following directory has been created:

`ORACLE_BASE\ORACLE_HOME\BIN`

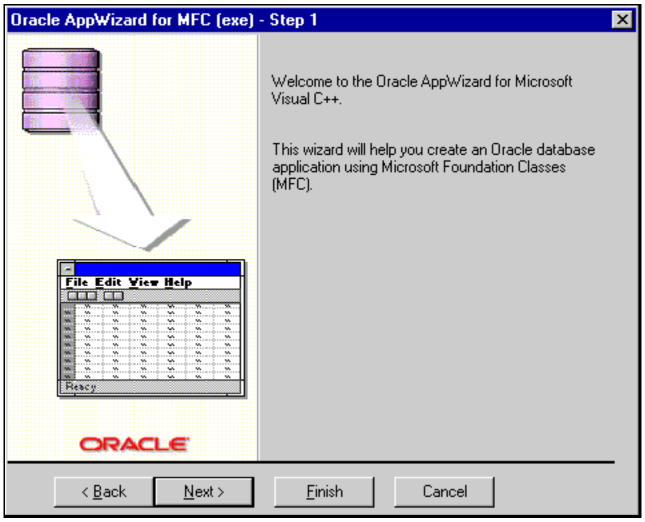
Creating a Single- or Multiple-Record Display Application

This section describes how to create a single- or multiple-record display application.

Welcome Window

The *Welcome* window appears when you start Oracle AppWizard for Microsoft Visual C++.

Click Next to continue.

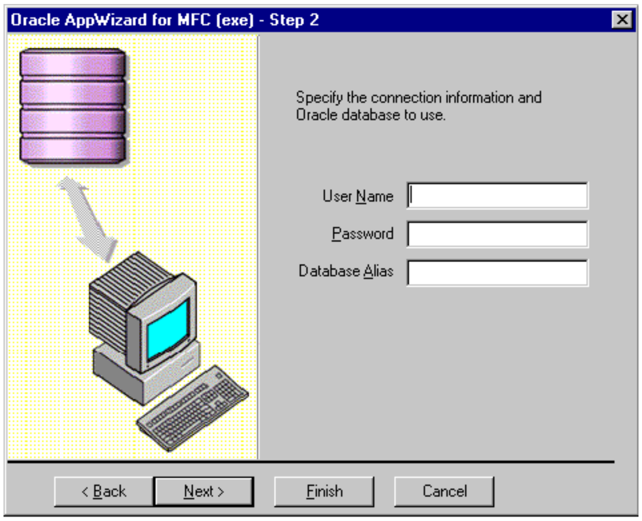


Welcome to the Oracle AppWizard for Microsoft Visual C++.

This wizard will help you create an Oracle database application using Microsoft Foundation Classes (MFC).

Connecting to an Oracle Database

In the *Connection* window, you connect to an Oracle database.



Specify the connection information and Oracle database to use.

User Name
Password
Database Alias

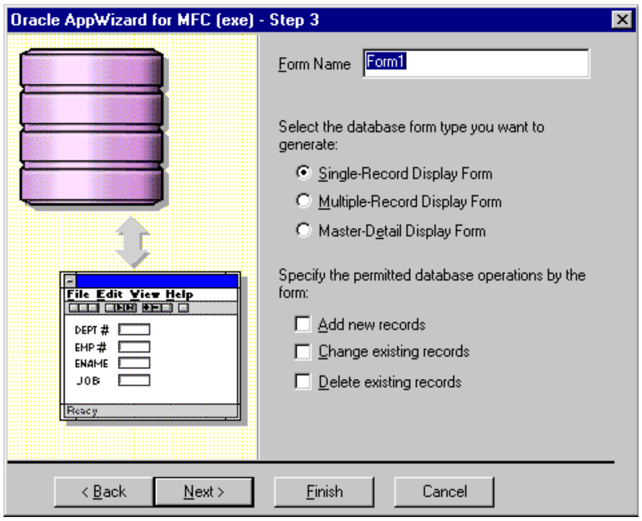
To connect to the Oracle database:

1. Enter your user name in the User Name text box.
2. Enter your password in the Password text box.
3. If connecting to a remote database, type the database alias in the Database Alias text box. If connecting to a local default database, leave the text box blank.

   For more information about database aliases, refer to the *Net8 Administrator's Guide*.
4. Click Next.

Specifying the Type of Form

Use the *Form Type* window to name your form, generate a specific type of database form, and to specify the database privileges for users.


Oracle AppWizard for MFC (exe) - Step 3

Form Name: Form1

Select the database form type you want to generate:

- (●) Single-Record Display Form
- ( ) Multiple-Record Display Form
- ( ) Master-Detail Display Form

Specify the permitted database operations by the form:

- [ ] Add new records
- [ ] Change existing records
- [ ] Delete existing records

To complete the *Form Type* window:

1. Enter the name of the form in the Form Name text box. Each form name must be unique and the first character of the form name must be a letter. The rest of the form name must be alphanumeric. For example:

   a1b3eux

2. Forms can be single-record, multiple-record, or master-detail display. Select the database form type you want to generate for the application:

If You Want to Create a...	Then Select...
Single-record display form that displays one record at a time on a form	Single-Record Display Form
Multiple-record display form that displays multiple records on a form	Multiple Record Display Form
Master-detail display form that displays infromation in master-detail format	Master Detail Display Form and see section "Creating a Master-Detail Display Application" on page 2-15

3. Select the database options permitted by the form:

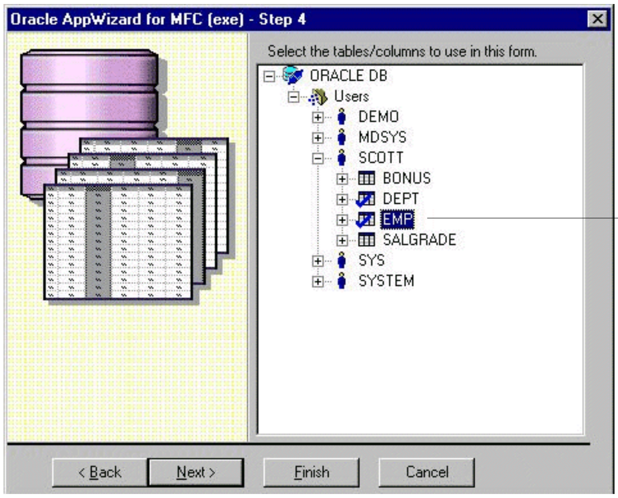
If You Want Users to Be Able to...	Then Select the...
Add records	Add new records checkbox
Change records	Change existing records checkbox
Delete records	Delete existing records checkbox

If you select any or all permitted database operations for your users, Oracle AppWizard generates code for this purpose. This enables methods that allow the users of your application to update data. If you do not select any permitted database operations, Oracle AppWizard does not generate code for this purpose and the form will be *read-only*.

4. Click Next.

Selecting Tables and Columns

In the *Tables/Columns Selection* window, you select columns from the tables you want the form to reference. The tables available to you appear in the list.


Oracle AppWizard for MFC (exe) - Step 4
Select the tables/columns to use in this form.
ORACLE DB
Users
DEMO
MDSYS
SCOTT
BONUS
DEPT
EMP
SALGRADE
SYS
SYSTEM
< Back
Next >
Finish
Cancel

Select the tables you want the form to contain

To complete the *Tables/Columns Selection* window:

1. Select one or more tables from the list to appear in your application. By default, all the columns of the table you select are also automatically selected unless you manually deselect them by clicking them.

   The Add, Change, and Delete options, specified in the *Form Type* window, work only with a single table. If you select multiple tables and have checked these options, the following error message appears:

   ```
   Add, Change, and Delete are supported in applications using a single table.
   You have selected multiple tables.
   ```

   ```
   Database Functionality defaults to Read Only.
   ```

   If necessary, expand the table you selected and select additional columns that should be displayed in the application.

2. Click Next.

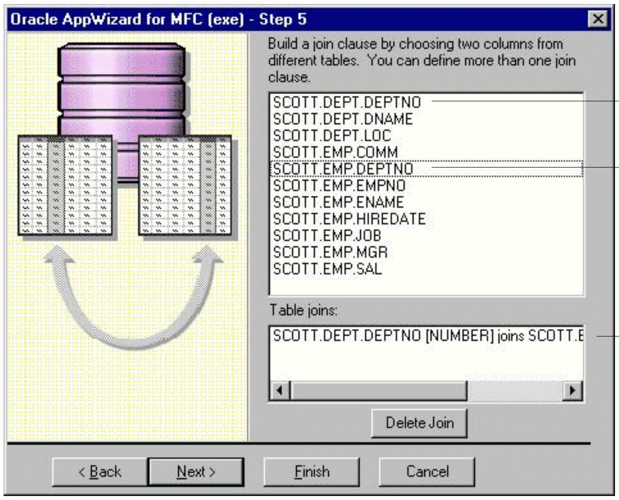
Specifying One or More Joins (Optional)

Oracle AppWizard for Microsoft Visual C++ automatically creates a simple join (also called an equi-join) between two tables, or views, based on a Primary Key and a Foreign Key.

Joins are used in the WHERE clause of a SELECT statement to avoid a Cartesian product, which would combine every row in one table with every row in another table. For example, a 90-row table combined with a 100-row table would produce a 9000-row result.

Note: If you only specify one table, Oracle AppWizard for Microsoft Visual C++ skips the table join step.

Additional Information: See the description of the SELECT command in Oracle8i SQL Reference for more information about various types of joins and Cartesian products.



Oracle AppWizard for MFC (exe) - Step 5

Build a join clause by choosing two columns from different tables. You can define more than one join clause.

```
SCOTT.DEPT.DEPTNO
SCOTT.DEPT.DNAME
SCOTT.DEPT.LOC
SCOTT.EMP.COMM
SCOTT.EMP.DEPTNO
SCOTT.EMP.EMPNO
SCOTT.EMP.ENAME
SCOTT.EMP.HIREDATE
SCOTT.EMP.JOB
SCOTT.EMP.MGR
SCOTT.EMP.SAL
```

Table joins:

```
SCOTT.DEPT.DEPTNO [NUMBER] joins SCOTT.E
```

In this example, the AppWizard joins the SCOTT.DEPT and SCOTT.EMP tables, based on the Primary Key and the Foreign Key.

Resulting join

Tip: When selecting tables or columns, use the Ctrl key and your mouse to select items that are not adjacent. Also, you can use the Ctrl key and your mouse to deselect an item.

The *Oracle AppWizard Table Join* window displays a join in the following format for single or multiple tables:

```
SCHEMA.TABLE_NAME.COLUMN_NAME [DATA_TYPE] joins
          SCHEMA.TABLE_NAME.COLUMN_NAME [DATA_TYPE]
```

If you do not want to use the suggested join created by Oracle AppWizard for Microsoft Visual C++ (shown in the previous illustration), you can delete it and create a new join. You can also do this by modifying the WHERE clause of a query statement in the source file generated by Oracle AppWizard for Microsoft Visual C++.

To accept the default join:

1. Click Next.

To delete the default join:

1. Highlight the default join displayed in the Table joins list.
2. Click the Delete Join button.

   Oracle AppWizard for Microsoft Visual C++ deletes the join.

To create a new a join:

1. Highlight two columns from different tables that you want to join by using your mouse and the Ctrl key.

   The join appears in the Table joins list with the following syntax:

   SCHEMA.*TABLE*.*COLUMN* [*DATA_TYPE*] joins *SCHEMA*.*TABLE*.*COLUMN* [*DATA_TYPE*]

   Note: You can specify more than one join.

2. Click Next.

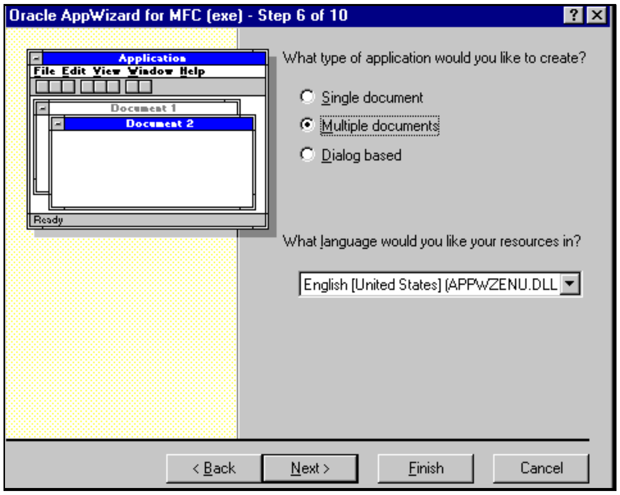
   Oracle AppWizard for Microsoft Visual C++ prompts you to build another form. If you decide to create a new form, you cannot go back and use Oracle AppWizard for Microsoft Visual C++ to modify the form you have already created.

3. If you want to build another form, click Yes and Oracle AppWizard returns you to the *Application Type* window.
4. If you do not want to build another form, click No.

Specifying the Application Type and User Language

When you are finished building forms and click Next, you can specify the application type and user language.

Note: If you are using Microsoft VC++ 6.0, there is an additional option for Document/View architecture support. See "Microsoft Visual C++ 6.0 Limitations" on page 1-6.


Oracle AppWizard for MFC (exe) - Step 6 of 10
Application
File Edit View Window Help
Document 1
Document 2
Ready
< Back
Next >
Finish
Cancel

What type of application would you like to create?

Single document [✓ Multiple documents] Dialog based

What language would you like your resources in?

English [United States] (APPWZENU.DLL

To complete the Application Type window:

1. Indicate the type of application you are creating:
   - Single document
   - Multiple documents
   - Dialog-based

Note: Dialog-based applications are not supported in this release.

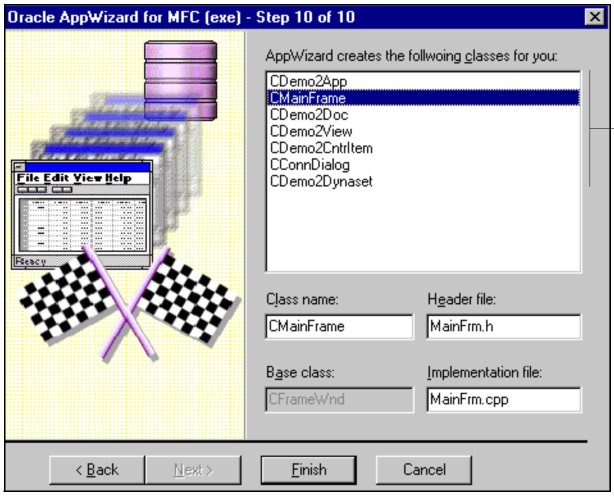
2. Select the language appropriate for your application or accept the default language in the list box.
3. Click Next.

Completing the Remaining MFC Windows

Complete the remaining standard MFC windows as appropriate until you reach the window illustrated below. For more information about MFC windows, refer to your MSVC++ documentation.

Viewing the Application Classes

In the *Class Information* window, Oracle AppWizard for Microsoft Visual C++ displays the classes it creates for your application, including their names, header files, base classes, and the implementation file:


Oracle AppWizard for MFC (exe) - Step 10 of 10

AppWizard creates the follwoing classes for you:

CDemo2App
CMainFrame
CDemo2Doc
CDemo2View
CDemo2CntrlItem
CConnDialog
CDemo2Dynaset

Class name: CMainFrame
Header file: MainFrm.h
Base class: CFrameWnd
Implementation file: MainFrm.cpp

What is shown below depends on the class selected here.

To complete the *Class Information* window:

1. Review the list for completeness and accuracy.
2. If you are not satisfied with the listed classes, click Back to go to the appropriate previous dialog box to make changes.
3. Depending on which class you have selected, you may change the class name, header file name, and the implementation file name. If you rename these files, the first character must be a letter. The rest of the name must be alphanumeric.
4. When you are satisfied with the result, click Finish.

   The *New Project Information* window appears.

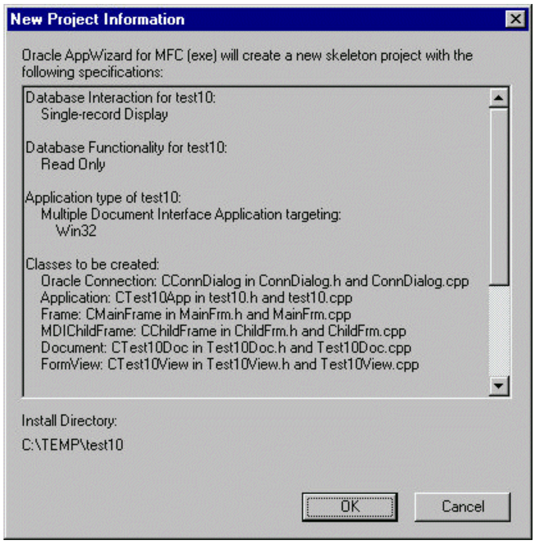
Viewing the Specifications for the New Application

The *New Project Information* window displays the specifications for the new skeleton application you are creating.

- If the specifications appear to be correct, click OK.

  Oracle AppWizard generates the files for your single-record or multiple-record application.

- If the specifications are not correct, click Cancel.



Oracle AppWizard for MFC (exe) will create a new skeleton project with the following specifications:

```
Database Interaction for test10:
   Single-record Display

Database Functionality for test10:
   Read Only

Application type of test10:
   Multiple Document Interface Application targeting:
      Win32

Classes to be created:
   Oracle Connection: CConnDialog in ConnDialog.h and ConnDialog.cpp
   Application: CTest10App in test10.h and test10.cpp
   Frame: CMainFrame in MainFrm.h and MainFrm.cpp
   MDIChildFrame: CChildFrame in ChildFrm.h and ChildFrm.cpp
   Document: CTest10Doc in Test10Doc.h and Test10Doc.cpp
   FormView: CTest10View in Test10View.h and Test10View.cpp
```

Install Directory:

C:\TEMP\test10

Creating a Master-Detail Display Application

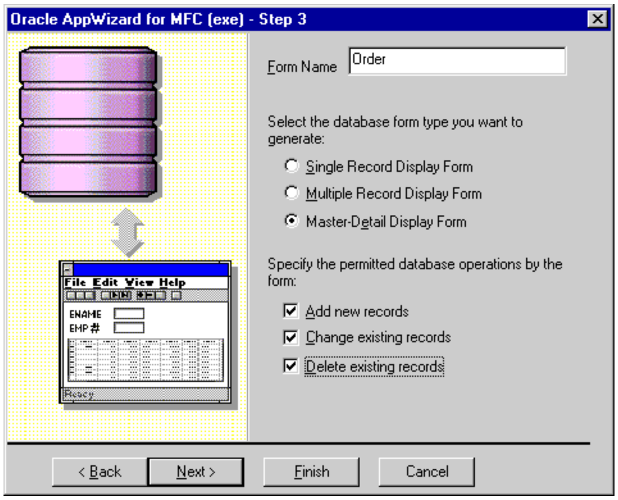
This section describes how to create an application that can display records from two or more tables that have a master-detail relationship.

The first three windows are completed in the same way as those described in "Creating a Single- or Multiple-Record Display Application" on page 2-5, except that, for a master-detail display, you will complete the *Application Type* window as follows:

- Select Master-Detail Display Form as the database form type
- Optionally, you can permit the user of your application to perform the database operations listed in the *Application Type* window by selecting any or all of the three check boxes.

Specifying the Type of Form

In the *Form Type* window, type the name you want to give the form, select the type of form, and the database privileges that your application requires.


Oracle AppWizard for MFC (exe) - Step 3

Form Name: Order

Select the database form type you want to generate:

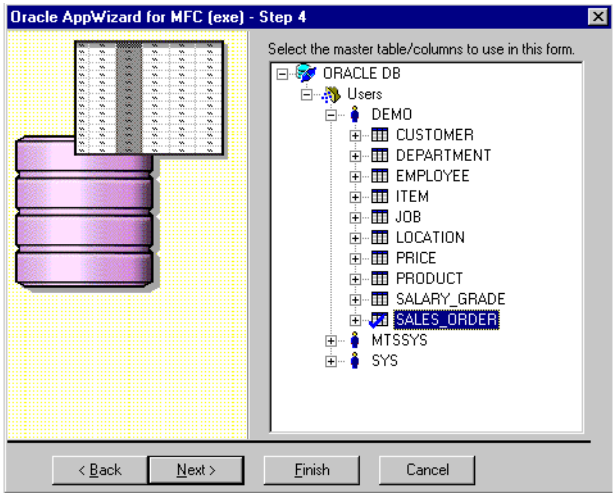
( ) Single Record Display Form
( ) Multiple Record Display Form
(●) Master-Detail Display Form

Specify the permitted database operations by the form:

☑ Add new records
☑ Change existing records
☑ Delete existing records

Selecting Master Tables and Columns

In the *Master Table/Column Selection* window, the available master tables appear in a list:



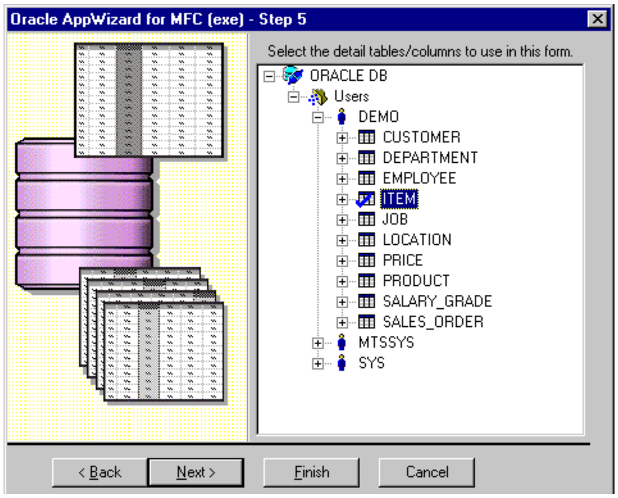
To complete the *Master Table/Column Selection* window:

1. Select the master table columns from the list. In the example above, the Sales Order table is the master table. All the columns from the Sales Order table have been selected.
2. If necessary, expand the table you selected by clicking the '+' to the left of the table name and change which columns are to be displayed in the application.
3. Click Next.

Note: The master table is created as a read-only table in your application.

Selecting Detail Tables and Columns

In the *Detail Table/Column Selection* window, the available detail tables appear in a list. Select the detail tables and the columns you want displayed in the application.



Select the detail tables/columns to use in this form.

```
ORACLE DB
  Users
    DEMO
      CUSTOMER
      DEPARTMENT
      EMPLOYEE
      ITEM
      JOB
      LOCATION
      PRICE
      PRODUCT
      SALARY_GRADE
      SALES_ORDER
    MTSSYS
    SYS
```

To complete the *Detail Table/Column Selection* window:

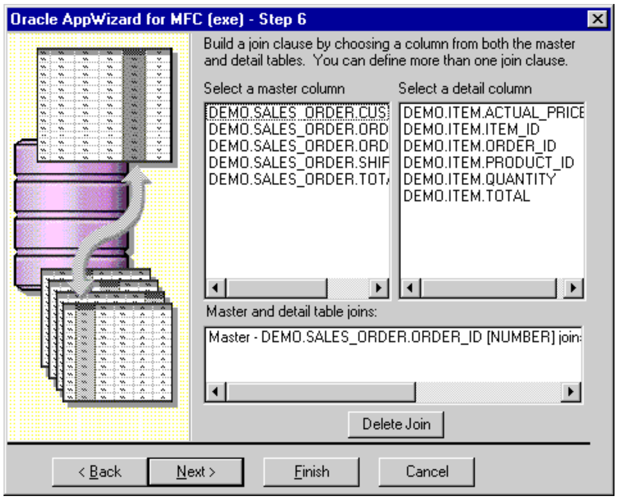
1. Select columns from one or more detail tables in the list. In the illustration above, the Item table is the detail table. The Iten table and all of its columns have been selected.
2. If necessary, expand the table you selected by clicking the '+' to the left of the table name, and change which columns are to be displayed in the application.
3. Click Next.

Building a Join Clause

After you select the tables between which to set up a master-detail relationship, use the *Master/Detail Table Join* window to specify how to join the columns from each of the tables selected in the last two windows.

Oracle AppWizard for Microsoft Visual C++ automatically creates a simple join (also called an equi-join) between two tables, or views, based on a Primary Key and a Foreign Key.

Such joins are used in the WHERE clause of a SELECT statement to avoid a Cartesian product, which combines every row in one table with every row in another table. For example, a 90-row table combined with a 100-row table would produce a 9000-row result.



Build a join clause by choosing a column from both the master and detail tables. You can define more than one join clause.

Select a master column:
- DEMO.SALES_ORDER.CUS
- DEMO.SALES_ORDER.ORD
- DEMO.SALES_ORDER.ORD
- DEMO.SALES_ORDER.SHIF
- DEMO.SALES_ORDER.TOT

Select a detail column:
- DEMO.ITEM.ACTUAL_PRICE
- DEMO.ITEM.ITEM_ID
- DEMO.ITEM.ORDER_ID
- DEMO.ITEM.PRODUCT_ID
- DEMO.ITEM.QUANTITY
- DEMO.ITEM.TOTAL

Master and detail table joins:
Master - DEMO.SALES_ORDER.ORDER_ID [NUMBER] join

If you do not want to use the default join created by Oracle AppWizard for Microsoft Visual C++ (shown in the previous illustration), you can delete it and create a new join. Alternatively, you can also do this by modifying the WHERE clause of a query statement in the source file generated by Oracle AppWizard.

If you want to create more than one join, choose one column from the master table and one column from the detail table to create each additional join clause.

To accept the default join in the Master/Detail Table Join window:

1. Click Next.

To delete the default join:

1. Highlight the default join displayed in the Master and Detail Table Joins list in the Master / Detail Table Join window.
2. Click the Delete Join button.

   Oracle AppWizard for Microsoft Visual C++ deletes the join.

To create a new join:

1. Select one column from the master column list.

2. Select one column from the detail column list.

   The join appears in the Master and Detail Table Joins list with the following syntax:

   ```
   Master - SCHEMA.TABLE_NAME.COLUMN_NAME [DATA_TYPE] joins Detail -
   SCHEMA.TABLE_NAME.COLUMN_NAME [DATA_TYPE]
   ```

3. If you want to create multiple joins, repeat Steps 1 and 2.
4. Click Next. Oracle AppWizard asks you whether or not you would like to build another form.
5. If you want to build another form, click Yes, and Oracle AppWizard for Microsoft Visual C++ returns you to the *Form Type* window.
6. If you do not want to build another form, click No.

Specifying the Application Type and User Language

When you are finished building forms and click Next, you are ready to specify the application type and the user language.

Note: If you are using Microsoft VC++ 6.0, there is an additional option for Document/View architecture support. See "Microsoft Visual C++ 6.0 Limitations" on page 1-6.

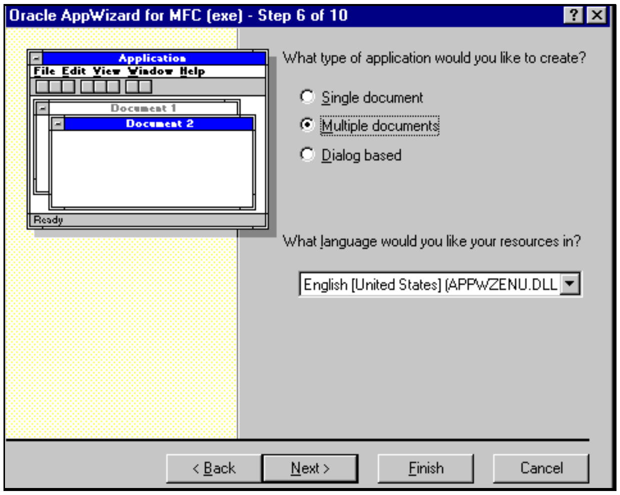
To complete the *Application Type* window:

1. Indicate the type of application you are creating by clicking the appropriate option:
   - Single document
   - Multiple documents
   - Dialog-based

   Note: Dialog-based applications are not supported in the current release.

2. Select the language appropriate for your application or accept the default language in the list box.

3. Click Next to proceed to the standard MFC windows.


Oracle AppWizard for MFC (exe) - Step 6 of 10
Application
File Edit View Window Help
Document 1
Document 2
Ready
< Back
Next >
Finish
Cancel

What type of application would you like to create?

( ) Single document
(●) Multiple documents
( ) Dialog based

What language would you like your resources in?

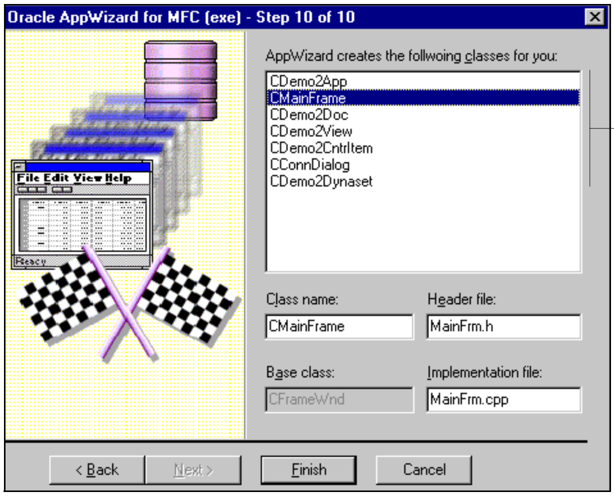
English [United States] (APPWZENU.DLL

Completing the Remaining MFC Windows

Complete the remaining standard MFC windows as appropriate until you reach the window illustrated below. For more information about MFC windows, refer to your MSVC++ documentation.

Viewing the Application Classes

In the Class Information window, Oracle AppWizard for Microsoft Visual C++ displays the classes it will create for your application, including their names, header files, base classes, and the implementation file.



AppWizard creates the follwoing classes for you:

CDemo2App
CMainFrame
CDemo2Doc
CDemo2View
CDemo2CntrItem
CConnDialog
CDemo2Dynaset

Class name:	Header file:
CMainFrame	MainFrm.h
Base class:	Implementation file:
CFrameWnd	MainFrm.cpp

What is shown below depends on the class selected here.

To complete the Class Information window:

1. Review the list for completeness and accuracy.
2. If you are not satisfied with the listed classes, click Back to go to the appropriate previous dialog box to make changes.
3. Depending on which class you have selected, you may change the class name, header file name, and implementation file name. If you rename these files, the first character must be a letter. The rest of the name must be alphanumeric.
4. When you are satisfied with the result, click Finish.

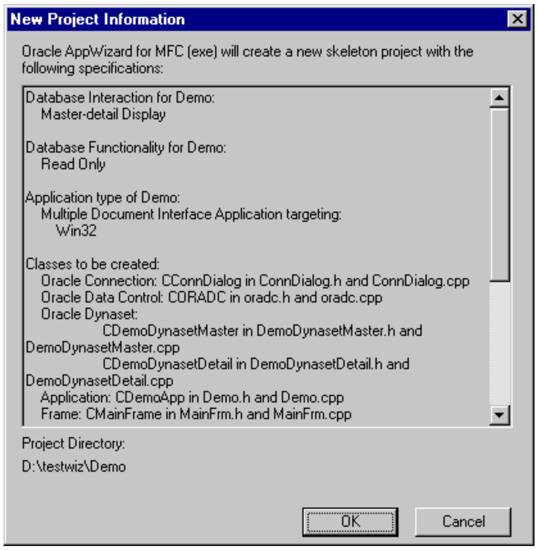
   The New Project Information window appears.

Viewing the Specifications for the New Application

The *New Project Information* window displays the specifications for the new skeleton application you are creating.

1. If the specifications are correct, click OK.

   Oracle AppWizard for Microsoft Visual C++ creates the files for your master-detail application.
2. If the specifications are not correct, click Cancel.



Oracle AppWizard for MFC (exe) will create a new skeleton project with the following specifications:

```
Database Interaction for Demo:
    Master-detail Display

Database Functionality for Demo:
    Read Only

Application type of Demo:
    Multiple Document Interface Application targeting:
        Win32

Classes to be created:
    Oracle Connection: CConnDialog in ConnDialog.h and ConnDialog.cpp
    Oracle Data Control: CORADC in oradc.h and oradc.cpp
    Oracle Dynaset:
                CDemoDynasetMaster in DemoDynasetMaster.h and
DemoDynasetMaster.cpp
                CDemoDynasetDetail in DemoDynasetDetail.h and
DemoDynasetDetail.cpp
    Application: CDemoApp in Demo.h and Demo.cpp
    Frame: CMainFrame in MainFrm.h and MainFrm.cpp
```

Project Directory:

D:\testwiz\Demo

Building the Executable

After Oracle AppWizard generates the application, you can build an executable.

Note: You may want to check the project settings to see if they are correct for your configuration. Choose Settings from the Project menu to view the project settings.

To build the executable:

1. Choose Build <*executable name*> from the Build menu.

   Oracle AppWizard compiles and links your project.

Running the Executable

You are now ready to execute your application.

Note: By default, the active project configuration is the debug version. To change the active project configuration, choose Set Active Configuration from the Build menu.

To run the executable:

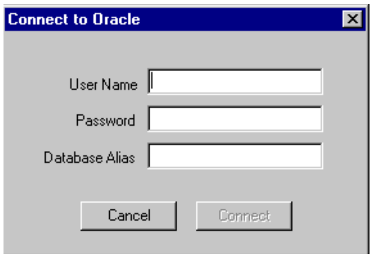
1. Choose Execute <*executable name*> from the Build menu.

   The *Connect to Oracle* dialog box appears.

2. Enter your User Name and Password.

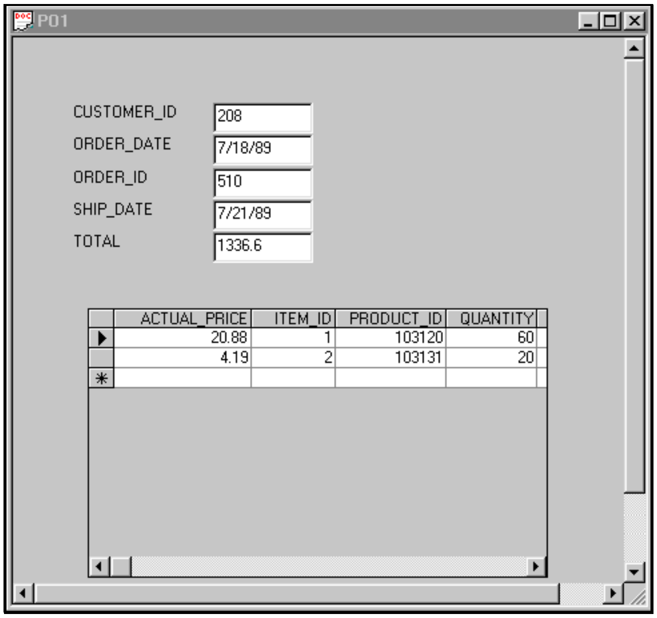
Note: The user name must be a database user who has privileges for accessing the tables in this application.

3. If connecting to a remote database, type its alias into the Database Alias text box.



Note: If you click Cancel, an empty form appears. This is in accordance with Microsoft ODBC application behavior. You can close this empty form.

After your application connects to the database successfully, the form you generated appears showing data retrieved from the database. The following illustration is a sample of what your form can look like, although the appearance of your actual form and data varies based on the database information you select.



CUSTOMER_ID	208
ORDER_DATE	7/18/89
ORDER_ID	510
SHIP_DATE	7/21/89
TOTAL	1336.6

	ACTUAL_PRICE	ITEM_ID	PRODUCT_ID	QUANTITY
▶	20.88	1	103120	60
	4.19	2	103131	20
*				

By design, no toolbar for inserting, updating, and deleting records is available. This is because the master table is read-only. If you have a single detail table, you can edit the fields by making a change and clicking on a different row to commit the change.

3
Understanding Your Application’s Code

This chapter describes the building blocks that Oracle AppWizard for Microsoft Visual C++ uses to create an application. Using this information, you can easily customize your application.

Specific topics discussed are:

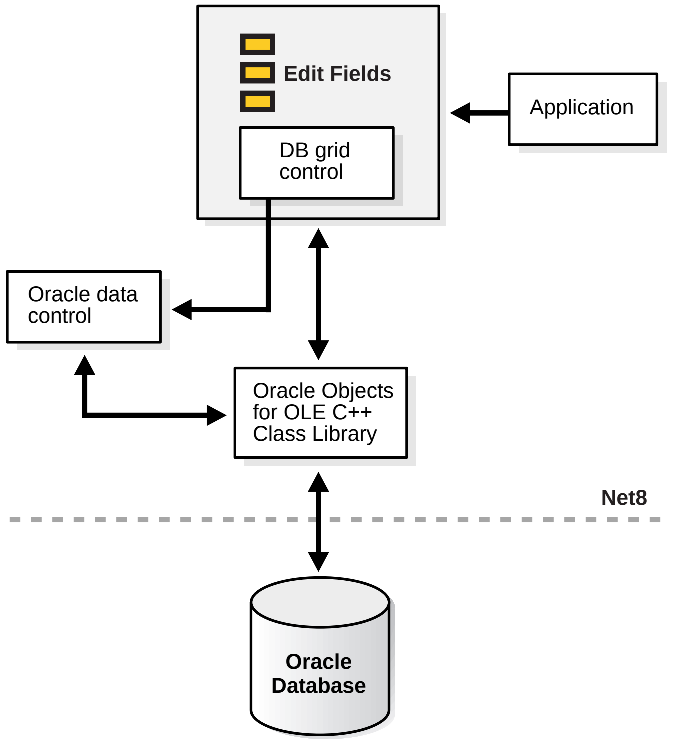
- Introduction
- Understanding the Generated Files
- Understanding the Code Within Generated Files

Introduction

Oracle AppWizard for Microsoft Visual C++ uses Microsoft Foundation Classes (MFC) to provide the application framework. Oracle Objects for OLE C++ Class Library is used to create a starter application. Oracle AppWizard generates a complete set of source and resource files your Visual C++ project that you can customize as required for your business needs.

Figure 3–1, "Oracle AppWizard Framework" shows how the framework created by Oracle AppWizard uses the Oracle Objects for OLE C++ to interact with the Oracle database.

Figure 3–1 Oracle AppWizard Framework



Understanding the Generated Files

After you complete all Oracle AppWizard for Microsoft Visual C++ steps and have chosen the options appropriate for the application you want to create, Oracle AppWizard generates a complete set of files:

- Source and header
- Precompiled header
- Resource
- Miscellaneous files

Source and Header Files

Oracle AppWizard for Microsoft Visual C++ creates the following source and header files. Your files contain the actual name of your project, instead of *PRJNAME*.

File	Description
PRJNAME.h, *PRJNAME*.cpp	Derives from and implements the application class C*PRJNAME*App. This class provides member functions for initializing and running the application.
PRJNAME.clw	Used by the ClassWizard to store information about the classes in your project.
ConnDialog.h, ConnDialog.cpp	Derives from and implements the dialog class CConnDialog. This class enables the application to request connection information from the user.
*PRJNAME*Doc.h, *PRJNAME*Doc.cpp	Derives from and implements the document class C*PRJNAME*Doc. This class also contains the application data, variables, and objects associated with the Oracle database. Data access occurs through Oracle Objects for OLE.
*PRJNAME*View.h, *PRJNAME*View.cpp	Derives from and implements the View class C*PRJNAME*View. This class displays document data graphically to the user and accepts and interprets user input as changes to the document.
*PRJNAME*Dynaset.h, *PRJNAME*Dynaset.cpp (for Single-Record and Multiple-Records Display Form)	Derives from and implements the Dynaset class *CPRJNAME*Dynaset. This class creates, manages, and accesses records in the database.

File	Description
*PRJNAME*DynasetMaster.h, *PRJNAME*DynasetMaster.cpp (for Master-Detail Display Form)	Derives from and implements the Dynaset class C*PRJNAME*DynasetMaster. This class creates, manages, and accesses records in the database.
*PRJNAME*DynasetDetail.h, *PRJNAME*DynasetDetail.cpp (for Master-Detail Display Form)	Derives from and implements the Dynaset class C*PRJNAME*DDynasetDetail. This class creates, manages, and accesses records in the database.
ChildFrm.h, ChildFrm.cpp (for Multiple-Document interface application)	Derives from and implements the Child Window class CChildFrame. This class is used for MDI document frames.
oradc.h, oradc.cpp (for Multiple-Records and Master-Detail Display form)	Derives from and implements the Window class CORADC. This class is the wrapper class for Oracle Data Control, an ActiveX control and is used to access data for an application using multiple-record display mode or master-detail display mode.
*PRJNAME*Util.h, *PRJNAME*Util.cpp	Declare and implement stand-alone functions used by other classes. ProcessOO4OError() and DDX_ FieldText() functions are defined in these files.
MainFrm.h, MainFrm.cpp	Derives from and implements the Frame class CMainFrame. This class provides the functionality of an overlapped single-document interface (SDI) or a pop-up window, along with members for managing the window.

Precompiled Header Files

Oracle AppWizard for Microsoft Visual C++ creates the following standard files.

StdAfx.h, StdAfx.cpp

These files are used to build a precompiled header file *PRJNAME*.PCH and a precompiled types file StdAfx.OBJ.

Resource Files

Oracle AppWizard creates the following standard header file and a main resource file:

Files	Description
Resource.h, *PRJNAME*.rc	These files contain the default menu definition, accelerator, and string tables for the generated application.
PRJNAME.rc2	This file is useful for including resources used by several different projects. Instead of having to create the same resources several times for different projects, you can put them in an RC2 file and include the RC2 file in the main RC file.
.\RES\*PRJNAME*.ico	This is the icon file for the application. This icon appears when the application is minimized and is also used in the About box.
.\RES\PRJNAMEDoc.ico	This is the icon file for the child window in the Multiple Document Interface application.
.\RES\TOOLBAR.BMP	This bitmap file is used to represent your program or control in a toolbar or palette.

Miscellaneous Files

Oracle AppWizard for Microsoft Visual C++ also creates the following file.

ReadMe.txt

This file contains information about the files that Oracle AppWizard for Microsoft Visual C++ creates for your application.

Understanding the Code Within Generated Files

This section is divided into the following subsections, which explain the function of the code generated by Oracle AppWizard for Microsoft Visual C++:

This Section...	Describes the...
Oracle Objects for OLE classes	Building blocks for all aspects of application functionality
What Happens When the Application Starts	Code at the heart of application initialization, execution of SQL statements, connection to the database, display of database table columns
Navigational Flow	Code at the heart of record navigation
Data Manipulation Flow	Code at the heart of record insertion, update, and deletion
Generated Code for a Multiple-Record Display Form	Code used to create an application whose forms display multiple-record at one time

Note: The generated code described in all but the last list item above is based on an application using single-record display mode exclusively.

Oracle Objects for OLE classes

Oracle AppWizard generates Oracle Object for OLE C++ code to provide connectivity and data access to the Oracle database. The following table illustrates the classes that Oracle AppWizard uses to communicate with the Oracle database, using the Oracle Objects for OLE C++ Class Library.

Class	Description
ODatabase	A database object representing an Oracle database
ODynaset	Creates, manages, and accesses records in the Oracle database
OField	A single column of data in an Oracle database record
OValue	Stores values of varying types in the Oracle database

The oracl.h header file in ORACLE_HOME\OO4O\CPP\INCLUDE contains the declarations for Oracle Objects for OLE classes.

What Happens When the Application Starts

This subsection describes the methods that control the following activities either during or after start-up as you work through the sequence of windows designed to build the starter application.

- Initialization of the Oracle Objects for OLE C++ Class Library in the generated application
- Connection to database
- Execution of SQL statements
- Table column displays

Initializing Oracle Object for OLE C++ Class Library

When the application starts, the OStartup() call in C*PRJNAME*App::InitInstance() initializes the Oracle Object for OLE C++ Class Library as described below:

Note: Text in **bold** type represents Oracle code.

```
BOOL CTestApp::InitInstance()
{
        ...
    // initializing Oracle Object for OLE C++ Class Library
    OStartup();
       ...
}
```

Connecting to Database

The application framework creates a document object (C*PRJNAME*Doc class) that stores the application data. During document initialization, it calls DbConnect() to initialize the database (m_database) associated with the document and connects to it, using the following declaration:

```
class CPRJNAMEDoc : public CDocument
{
 ...
      // Attributes
      public:
            ODatabase m_database;
            CPRJNAMEDynaset m_PRJNAMEDynaset;
      // Implementation
      public:
        virtual ~CTestDoc();
        bool DbConnect();
...
};
```

Constructor method of C*PRJNAME*Doc class:

```
CPRJNAMEDoc::CPRJNAMEDoc()
{
    while (!m_database.IsOpen())
          {
           if (DbConnect() == false)
            return;
        }

    }
```

To initialize and connect to the database, the application:

- Creates a dialog box (connDlg) to request database connection information
- Connects to the database
- If the connection fails, the application processes the error

The method C*PRJNAME*Doc::DbConnect() implements the database connection as shown below:

```
bool CPRJNAMEDoc::DbConnect()
{
       CConnDialog connDlg;
       oresult dbresult;
       int dlgResult;
// a dialog box to get connection information from users
     dlgResult = connDlg.DoModal();
     if (dlgResult == IDOK)
 {
// Connect to the database
     dbresult = m_database.Open(connDlg.GetDbAlias(),
     connDlg.GetUsername(), connDlg.GetPassword());
     if (dbresult == OFAILURE)
     {
// processing error message
      ProcessOO4OError(&m_database);
      }
      return(true);
}
else
     return(false);
}
```

Executing SQL Statements

If successfully connected, the framework creates a view (C*PRJNAME*View class) for the application, using the following declaration:

```
class CPRJNAMEView : public CFormView
{
     ...
     public:
        CTestDynaset *m_pDynaset;
        bool m_bEditingRecord;              //if the record must be edited
    // Operations
    protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        virtual void OnInitialUpdate(); // called first time after construct
    ...

// Implementation
public:
     virtual ~CPRJNAMEView();
protected:
...
 void PerformMove(int nCommand);

// Generated message map functions
protected:
    ...
    afx_msg void OnMoveNext();
    afx_msg void OnMovePrev();
    afx_msg void OnMoveFirst();
    afx_msg void OnMoveLast();
    afx_msg void OnUpdateMoveNext(CCmdUI* pCmdUI);
    afx_msg void OnUpdateMovePrev(CCmdUI* pCmdUI);
    afx_msg void OnUpdateMoveFirst(CCmdUI* pCmdUI);
    afx_msg void OnUpdateMoveLast(CCmdUI* pCmdUI);
    afx_msg void OnAddNewRecord();
    afx_msg void OnUpdateAddNewRecord(CCmdUI* pCmdUI);
    afx_msg void OnDeleteRecord();
    afx_msg void OnUpdateDeleteRecord(CCmdUI* pCmdUI);
    afx_msg void OnUpdateRecord();
    afx_msg void OnUpdateUpdateRecord(CCmdUI* pCmdUI);
    afx_msg void OnCancelRecord();
    afx_msg void OnUpdateCancelRecord(CCmdUI* pCmdUI);
    afx_msg void OnChangeEdit();
    ...
}
```

The constructor method for the C*PRJNAME*View class initializes its member variables as:

```
CPRJNAMEView::CPRJNAMEView()
: CFormView(CPRJNAMEView::IDD)
{
...
m_pDynaset = NULL;
m_bEditingRecord = false;
...
 }
```

Before the view appears, the application framework calls the C*PRJNAME*View::OnInitialUpdate() method to perform initialization, requiring the following document information:

- A view needs to display information from a table. To get information from the document for this purpose, the m_pDynaset member variable in the C*PRJNAME*View class points to the m_*PRJNAME*Dynaset member variable in the C*PRJNAME*Doc class.

- A dynaset needs to be associated with a set of records in a table. To open a dynaset in a database associated with the document, the application calls ODynaset::OpenQuery() method.

This implements the *CPRJNAME*View::OnInitialUpdate() method, illustrated below.

```
void CPRJNAMEView::OnInitialUpdate()
{
    ...
    m_pDynaset = &GetDocument()->m_PRJNAMEDynaset;
    m_pDynaset->OpenQuery(GetDocument()->m_database);
    ...
}
```

The C*PRJNAME*Dynaset class represents a set of records in a table. It creates, manages, and accesses records in the database, using the declaration shown below. In this declaration, data member variables have comments that indicate which real columns they represent.

```
class CTestDynaset : public ODynaset
{
...
public:
     // strings needed for creating the queries
     CString m_strSQLQuery;  // the query to be sent to the database
     CString m_strSQLSelect; // the select portion
     CString m_strSQLFilter; // the where portion
     CString m_strSQLSort;   // the order by portion
    // Field/Param Data
    OField m_Column1; //for COL1
    OField m_Column2; //for COL2
...
// Operations
public:
    void OpenQuery(ODatabase theDB);
    void CreateSQLSelect();
    void CreateSQLFilter();
    void AddFilter(CString strFilter);
    void CreateSQLSort();
    void ResetToDefaultFilter();
    void RefreshQuery();
...
}
```

To open a dynaset associated with the document, the method *CPRJNAME*Dynaset::OpenQuery() is called in *CPRJNAME*View::OnInitialUpdate(). The *CPRJNAME*Dynaset::OpenQuery() method takes the following actions:

- Creates a query statement to be executed
- Asks the database for a set of records and sets up a dynaset to access them
- Binds data controls with the OField member variables (m_Column1, m_Column2,...etc.) in the CPRJNAMEDynaset class

The *CPRJNAME*Dynaset::OpenQuery() method is implemented as illustrated below:

```
 // opens the query
 void CPRJNAMEDynaset::OpenQuery(ODatabase theDB)
{
     oresult dbresult;
 // m_strSQLQuery, m_strSQLSelect, m_strSQLFilter are CString objects.
 // These are member variables in CPRJNAMEDynaset class

 // create a query statement to be executed by the Open() call
 m_strSQLQuery = m_strSQLSelect;
 if (m_strSQLFilter)
   m_strSQLQuery += m_strSQLFilter;
 if (m_strSQLSort)
  m_strSQLQuery += m_strSQLSort;

// executing a SQL statement
dbresult = Open(theDB, m_strSQLQuery);

 // binding edit controls with columns in the table
  m_Column1 = GetField("COL1");
  m_Column2 = GetField("COL2");
     ...
}
```

Displaying Columns in a Table

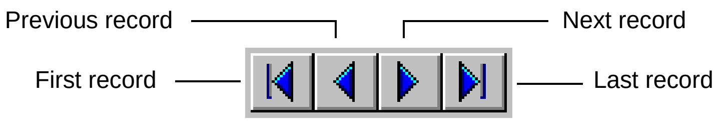
A view displays the stored data in the associated document. Data in both the document and the view must be consistent. Therefore, the OField member variables (the columns in a database table) of the *CPRJNAME*Dynaset class and the edit controls in the view must be validated and exchanged, using the *CPRJNAME*View::DoDataExchange() method, which is implemented as:

```
void CPRJNAMEView::DoDataExchange(CDataExchange* pDX)
{
    ...
     // performing data exchange
     DDX_FieldText(pDX, IDC_COL1,
    m_pDynaset->m_Column1, m_pDynaset);
     DDX_FieldText(pDX, IDC_COL2,
    m_pDynaset->m_Column2, m_pDynaset);
    ...
}
```

Navigational Flow

This section explains the code generated by Oracle AppWizard for Microsoft Visual C++ to allow navigation through a dynaset record set.

These four controls enable your application users to navigate through records:



These navigational controls work as shown in the following two tables. Event handlers handle the COMMAND message, and Message handlers handle UPDATE_COMMAND_UI messages.

Action	Button Control ID	Event Handler	Description
Move to first record	ID_RECORD_MOVEFIRST	OnMoveFirst	Move to first record in the dynaset's result set.
Move to previous record	ID_RECORD_MOVEPREV	OnMovePrev	Move to previous record in the dynaset's result set.
Move to next record	ID_RECORD_MOVENEXT	OnMoveNext	Move to next record in the dynaset's result set.
Move to last record	ID_RECORD_MOVELAST	OnMoveLast	Move to last record in the dynaset's result set.

Action	Button Control ID	Message Handler for Update Command	Description
Move to first record	ID_RECORD_ MOVEFIRST	OnUpdateMoveFirst	Update first record in the dynaset's result set.
Move to previous record	ID_RECORD_ MOVEPREV	OnUpdateMovePrev	Update previous record in the dynaset's result set.
Move to next record	ID_RECORD_ MOVENEXT	OnUpdateMoveNext	Update next record in the dynaset's result set.
Move to last record	ID_RECORD_ MOVELAST	OnUpdateMoveLast	Update last record in the dynaset's result set.

The navigational operations all use the CPRJNAMEView::PerformMove() method to perform the requested navigation. To explain the generated code for the navigational operations, we must explain the generated code for the CPRJNAMEView::PerformMove() method.

How Record Navigation Works

All navigational operations use the *CPRJNAME*View::PerformMove() method.

After a request for record navigation by the user and before any navigation occurs, the application verifies whether or not the current record must be updated or inserted into the database by calling *CPRJNAME*View::PerformMove(). The method performs the following operations:

- Checks if the record can be edited
- Performs data exchange between the view and the dynaset
- If the record has been modified, updates the record in the database
- Performs the requested navigation

*CPRJNAME*View::PerformMove() method is implemented as:

```
void CPRJNAMEView::PerformMove(int nCommand)
{
    // nCommand is the control ID of the navigational button and represents
    // the requested operation.
 oresult dbresult;
    // Verify whether or not the record can be edited.
    if (m_pDynaset->GetEditMode() != ODYNASET_EDIT_NEWRECORD)
        m_pDynaset->StartEdit();
    // exchange data between the the view and the dynaset
    if (!UpdateData())
        return;
    // if the record changes, update the record in the dynaset
    if (m_pDynaset->get_CurrentRowModified())
    {
    dbresult = m_pDynaset->Update();
    if (dbresult == OFAILURE)
    {

         ProcessOO4OError(m_pDynaset);
         return;
    }
}
else
    dbresult = m_pDynaset->CancelEdit();
switch(nCommand) // perform action, depends on the requeset navigation
{
      case ID_RECORD_MOVENEXT:
        m_pDynaset->MoveNext();
        break;
    case ID_RECORD_MOVEPREV:
        m_pDynaset->MovePrev();
        break;
    case ID_RECORD_MOVEFIRST:
        m_pDynaset->MoveFirst();
        break;
    case ID_RECORD_MOVELAST:
        m_pDynaset->MoveLast();
        break;
}
// exchange data between the view and the dynaset
UpdateData(FALSE);
m_bEditingRecord = false;
}
```

Navigating to the First Record in the Database

When the current record is not the first record and the user clicks the "First Record" button, corresponding to control ID = ID_RECORD_MOVEFIRST, users navigate to the first record.

The *CPRJNAME*View::OnUpdateMoveFirst() method enables / disables the "First Record" button and is implemented as:

```
Void CTestView::OnUpdateMoveFirst(CCmdUI* pCmdUI)
{
    pCmdUI->Enable((m_pDynaset->IsOpen() == TRUE
        && m_pDynaset->IsFirst() ==    FALSE) ? true : false);
}
```

When the button with control ID = ID_RECORD_MOVEFIRST is clicked, C*PRJNAME*View::OnMoveFirst() method is called to handle the “Move to First Record” event.

C*PRJNAME*View::OnMoveFirst() method is implemented as:

```
// move to the first record
void CPRJNAMEView::OnMoveFirst()
{
        PerformMove(ID_RECORD_MOVEFIRST);
}
```

Navigating to the Last Record

When the current record is not the last record and the user clicks the "Last Record" button, corresponding to control ID = ID_RECORD_MOVELAST, users navigate to the last record.

The C*PRJNAME*View::OnUpdateMoveLast() method enables/disables the “Move to Last Record” button as shown below:

```
void CTestView::OnUpdateMoveLast(CCmdUI* pCmdUI)
{
    pCmdUI->Enable((m_pDynaset->IsOpen() == TRUE
        && m_pDynaset->IsLast() == FALSE) ? true : false);
}
```

When the button with control ID = ID_RECORD_MOVELAST is clicked, C*PRJNAME*View::OnMoveLast() method is called to handle the “Move to Last Record” event.

This is the implementation of C*PRJNAME*View::OnMoveLast() method:

```
// move to the last record
void CPRJNAMEView::OnMoveLast()
{
        PerformMove(ID_RECORD_MOVELAST);
}
```

Navigating to the Previous Record

When the current record is not the first record and the user clicks the "Previous Record" button, corresponding to control ID = ID_RECORD_MOVEPREV, users navigate to the previous record.

The C*PRJNAME*View::OnUpdateMovePrev() method enables/disables the "Previous Record" button as shown below:

```
    void CTestView::OnUpdateMovePrev(CCmdUI* pCmdUI)
{
    CTestView::OnUpdateMoveFirst(pCmdUI);
}
```

Clicking the "Previous Record" button calls the C*PRJNAME*View::OnMovePrev() method to handle the "Move to Previous Record" event. It is implemented as shown below:

```
// move to the previous record
void CPRJNAMEView::OnMovePrev()
{
        PerformMove(ID_RECORD_MOVEPREV);
}
```

Navigating to the Next Record

When the current record is not the last record and the user clicks the "Next Record" button, corresponding to control ID = ID_RECORD_MOVENEXT, users navigate to the next record.

The C*PRJNAME*View::OnUpdateMoveNext() method enables / disables the “Next Record” button as shown below:

```
void CTestView::OnUpdateMoveNext(CCmdUI* pCmdUI)
{
        CTestView::OnUpdateMoveLast(pCmdUI);
}
```

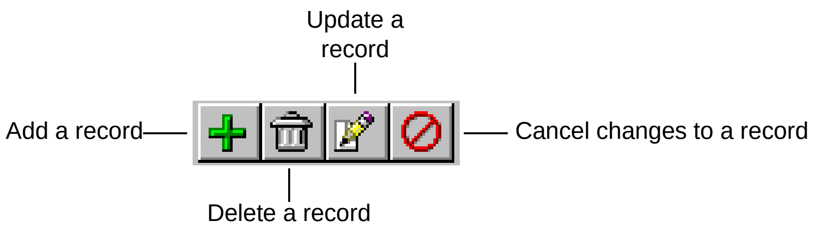
Clicking the "Next Record" button calls the C*PRJNAME*View::OnMoveNext() method to handle the "Move to Next Record" event. It is implemented as shown below:

```
// move to the next record
void CPRJNAMEView::OnMoveNext()
{
        PerformMove(ID_RECORD_MOVENEXT);
}
```

Data Manipulation Flow

This section explains the code that Oracle AppWizard for Microsoft Visual C++ generates to manipulation data in a table.

The following four controls enable application users to add, delete, and update records, or to cancel record updates before being sent to the database:



The controls illustrated above correspond to the ODynaset class methods shown in the table below. Event handlers handle the COMMAND message, and Message handlers handle UPDATE_COMMAND_UI messages.

Action	Button Control ID	Event Handler	Description
Add a new record	ID_RECORD_ADDNEW	OnAddNewRecord	Add a new record to the dynaset's result set
Delete a record	ID_RECORD_DELETE	OnUpdateRecord	Delete a record from the dynaset's result set
Update a record	ID_RECORD_UPDATE	OnDeleteRecord	Commit the changes for the current record
Cancel changes to a record	ID_RECORD_CANCEL	OnCancelRecord	Cancels the changes to the current record

Action	Button Control ID	Message Handler
Add new record	ID_RECORD_ADDNEW	OnUpdateAddNewRecord
Delete a record	ID_RECORD_DELETE	OnUpdateUpdateRecord
Update a record	ID_RECORD_UPDATE	OnUpdateDeleteRecord
Cancel changes to a record	ID_RECORD_CANCEL	OnUpdateCancelRecord

Adding a New Record

When the current record is not being updated and the user clicks the "Add New Record" button, corresponding to control ID = ID_RECORD_ADDNEW, users add a new record to the database.

The C*PRJNAME*View::OnUpdateAddNewRecord() method enables/disables the "Add New Record" button as shown below:

```
void CTestView::OnUpdateAddNewRecord(CCmdUI* pCmdUI)
{
       CmdUI->Enable(m_pDynaset->IsOpen() == TRUE && !m_bEditingRecord);
}
```

Clicking the "Add New Record" button, calls the C*PRJNAME*View::OnAddNewRecord() method to handle the “Add Record” event as shown below:

```
// Allows the user to add a new record to the table
void CPRJNAMEView::OnAddNewRecord()
{
    if (m_pDynaset->AddNewRecord() == OSUCCESS)
    {
        UpdateData(FALSE);
        m_bEditingRecord = true;
    }
    else
    {
        ProcessOO4OError(m_pDynaset);
    }
}
```

Updating a Record

When the current record is not being updated and the user clicks the "Update Record" button, corresponding to control ID = ID_RECORD_UPDATE, users update a record in the database.

The C*PRJNAME*View::OnUpdateUpdateRecord() method enables / disables the "Update Record" button as shown below:

```
void CTestView::OnUpdateUpdateRecord(CCmdUI* pCmdUI)
{
        pCmdUI->Enable(m_pDynaset->IsOpen() == TRUE && m_bEditingRecord);
}
```

Clicking the "Update Record" button calls the C*PRJNAME*View::OnUpdateRecord() method to handle the "Update Record" event as shown below:

```
// Updates the changes the user has made to the current record of the table
void CCPRJNAMEView::OnUpdateRecord()
{
    oresult dbresult;
    if (m_pDynaset->GetEditMode() != ODYNASET_EDIT_NEWRECORD)
         m_pDynaset->StartEdit();
    if (!UpdateData())
         return;
    if (m_pDynaset->get_CurrentRowModified())
    {
    dbresult = m_pDynaset->Update();
    if (dbresult == OFAILURE)
        {
         ProcessOO4OError(m_pDynaset);
         return;
            }
  }
      else
      dbresult = m_pDynaset->CancelEdit();
      UpdateData(FALSE);
      m_bEditingRecord = false;
}
```

Deleting a Record

When the current record is not being updated and the user clicks the "Delete Record" button, corresponding to control ID = ID_RECORD_DELETE, users delete a record in the database.

The C*PRJNAME*View::OnUpdateDeleteRecord() method enables/disables the "Delete Record" button as shown below:

```
void CTestView::OnUpdateDeleteRecord(CCmdUI* pCmdUI)
{
        pCmdUI->Enable(m_pDynaset->IsOpen() == TRUE && !m_bEditingRecord);
}
```

Clicking the "Delete Record" button calls the C*PRJNAME*View::OnDeleteRecord() method to handle the "Delete Record" event as shown below:

```
// Deletes the current record from the table
void CCPRJNAMEView::OnDeleteRecord()
{
    bool bWasLast = m_pDynaset->IsLast() == TRUE ? true : false;
    if (m_pDynaset->DeleteRecord() == OSUCCESS)
    {
         if (!bWasLast)
            m_pDynaset->MoveNext();
        else
            m_pDynaset->MovePrev();
        UpdateData(FALSE);
        m_bEditingRecord = false;
    }
    else
    {
        ProcessOO4OError(m_pDynaset);
    }
}
```

Cancelling Changes to a Record

When the current record is being updated and the user clicks the “Cancel record changes” button, corresponding to control ID = ID_RECORD_CANCEL, users cancel changes they have entered, but have not yet committed, to a record in the database.

The C*PRJNAME*View::OnUpdateCancelRecord() method enables/disables the “Cancel Changes” button as shown below:

```
void CTestView::OnUpdateCancelRecord(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_pDynaset->IsOpen() == TRUE && m_bEditingRecord);
}
```

Clicking the “Cancel Changes” button, calls the C*PRJNAME*View::OnCancelRecord() method to handle the “Cancel the change” event as shown below:

```
// Cancels the changes to the current record.
void CCPRJNAMEView::OnCancelRecord()
{
    if (m_pDynaset->GetEditMode() != ODYNASET_EDIT_NOEDIT)
    if (m_pDynaset->CancelEdit() == OFAILURE)
        ProcessOO4OError(m_pDynaset);
    UpdateData(FALSE);
    m_bEditingRecord = false;
}
```

Generated Code for a Multiple-Record Display Form

This subsection explains the generated code for applications using multiple-record display forms. The previous discussion in this section dealt with generated code for single-record display forms.

Note: Text in **bold** type represents Oracle code.

Oracle AppWizard for Microsoft Visual C++ uses Oracle Data Control class CORADC to display and access records in this mode. The C*PRJNAME*View class is defined as follows:

```
class CPRJNAMEView : public CFormView
{
    ...
    public:
        CTestDynaset *m_pDynaset;
        CORADC m_dataControl // Used by Oracle control to display
                                    and access data
    ...
}
```

Data exchange takes place, using the Oracle Data Control Object (m_dataControl) in the C*PRJNAME*View::DoDataExchange() as follows:

```
void CPRJNAMEView::DoDataExchange(CDataExchange* pDX)
{
    // performing data exchange
    // IDC_ORADC is the control ID for the Oracle Data Control grid
    DDX_Control(pDX, IDC_ORADC, m_dataControl);
}
```

4 Tutorial

This chapter guides you through the creation and customization of an application generated by Oracle AppWizard for Microsoft Visual C++. All of the customization lessons presented in this tutorial are based on information described in Chapter 3, "Understanding Your Application's Code" This tutorial introduces each lesson with an explanation of what you will learn upon completion.

By working through all the lessons of this tutorial, you will learn how to create and customize an Oracle AppWizard for Microsoft Visual C++ starter application.

Specific topics discussed are:

Introduction

A store owner must keep accurate records of all customers and their purchases. A database application is perfect for performing this job.

In this tutorial, you develop a custom purchase order application for Nicole's Sporting Goods called "Order." This business must track information about sales orders, the items sold, and to which of the store's customers. You create and examine the structure of the database tables, then create and customize the application to manage that information. The application you develop allows Nicole and her employees to create, update, and view purchase orders for her sporting goods store.

Each main section of the tutorial corresponds to a version number for Order, with each successive lesson building on what was learned in the last one. When you finish working through the final lesson, you will have programmed a completely customized purchase order system for Nichole's Sporting Goods.

You can access the files for the tutorial in the following directory:

ORACLE_BASE\ORACLE_HOME\APPWIZARD\VC++\TUTORIAL\ORDER[1-5]

These files show you what you should have built by the end of each lesson. The Order1 file corresponds to Lesson 1, Order2 corresponds to Lesson 2, and so on. Therefore, to see the set of completed files for Lesson 1, access:

ORACLE_BASE\ORACLE_HOME\APPWIZARD\VC++\TUTORIAL\ORDER1

Before You Start

Before you can create any application, you must set up your database tables, so that the application has information to process. Because Nicole needs to keep track of information about sales orders, the items sold, the store's customers, and the products sold, you must structure the tables to reflect those needs.

To create this purchase order system, you must set up the following database tables to interact with each other:

- Sales_Order
- Item
- Customer
- Product

If you need to create the user DEMO and the above tables, search for DEMO.SQL in the *ORACLE_BASE*\ORACLE_HOME\\RDBMS\\ADMIN directory and execute it within Server Manager or SQL*Plus. The DEMO.SQL file creates user DEMO and calls the SQL script BDEMOBLD.SQL to create the above tables for you.

The following illustrates the data models for the tables you will construct:

Sales_Order table:

```
Name                            Null?    Type
------------------------------- -------- ----
ORDER_ID                        NOT NULL NUMBER(4)
ORDER_DATE                               DATE
CUSTOMER_ID                              NUMBER(6)
SHIP_DATE                                DATE
TOTAL                                    NUMBER(8,2)
```

Item table:

```
Name                            Null?    Type
------------------------------- -------- ----
ORDER_ID                        NOT NULL NUMBER(4)
ITEM_ID                         NOT NULL NUMBER(4)
PRODUCT_ID                               NUMBER(6)
ACTUAL_PRICE                             NUMBER(8,2)
QUANTITY                                 NUMBER(8)
TOTAL                                    NUMBER(8,2)
```

Customer table:

```
Name                            Null?    Type
------------------------------- -------- ----
CUSTOMER_ID                     NOT NULL NUMBER(6)
NAME                                     VARCHAR2(45)
ADDRESS                                  VARCHAR2(40)
CITY                                     VARCHAR2(30)
STATE                                    VARCHAR2(2)
ZIP_CODE                                 VARCHAR2(9)
AREA_CODE                                NUMBER(3)
PHONE_NUMBER                             NUMBER(7)
SALESPERSON_ID                           NUMBER(4)
CREDIT_LIMIT                             NUMBER(9,2)
COMMENTS                                 LONG
```

Product table:

```
Name                              Null?    Type
------------------------------- -------- ----
PRODUCT_ID                        NOT NULL NUMBER(6)
DESCRIPTION                                VARCHAR2(30)
```

Lesson 1: Creating the Starter Application

In Lesson 1, you form the basis of the purchase order application for Nicole's Sporting Goods. In this lesson, you learn how to quickly complete a starter application as explained in Chapter 2, "Creating a Starter Application" The application you create using Oracle AppWizard for Microsoft Visual C++ is then ready for customization.

When you are finished with this section, you will understand how to use Oracle AppWizard for Microsoft Visual C++ to create a starter application.

Lesson 1 consists of the following parts:

- Part 1: Working with Oracle AppWizard for Microsoft Visual C++
- Part 2: Exploring Generated Classes and Files
- Part 3: Viewing the ReadMe.txt for the Generated Project
- Part 4: Building and Running the Application

Part 1: Working with Oracle AppWizard for Microsoft Visual C++

Starting the Oracle AppWizard for Microsoft Visual C++

1. Start Microsoft Visual C++.
2. Choose New from the File menu.

   The *New* dialog box appears.
3. Click the Projects tab.
4. Select Oracle AppWizard for MFC (exe) from the list of project types.
5. Enter the path from which you want to locate the application.
6. Enter "Order" in the Project Name box. This is the name of the application that you will create.
7. Make sure that "Create New Workspace" is selected.

8. Click OK.

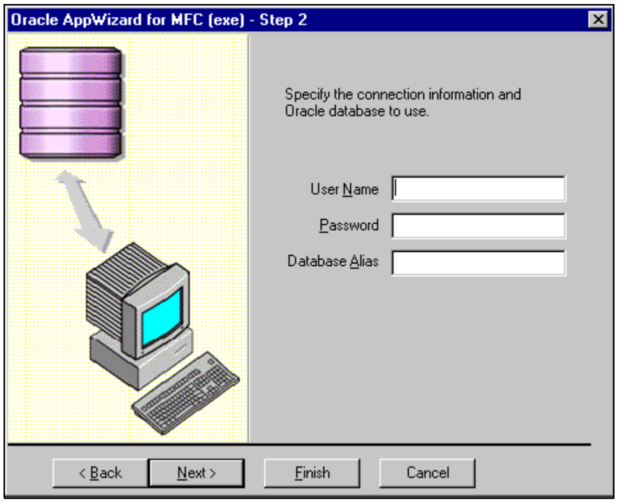
   Oracle AppWizard for Microsoft Visual C++ starts.

   The *Welcome* window appears.

9. Click Next to continue.

Connecting to the Oracle Database

1. Enter DEMO in the User Name text field.
2. Enter DEMO in the Password text field.
3. If connecting to a remote database, type the database alias in the Database Alias box. Otherwise, leave this field empty.
4. When you are done, click Next.



Oracle AppWizard for MFC (exe) - Step 2

Specify the connection information and Oracle database to use.

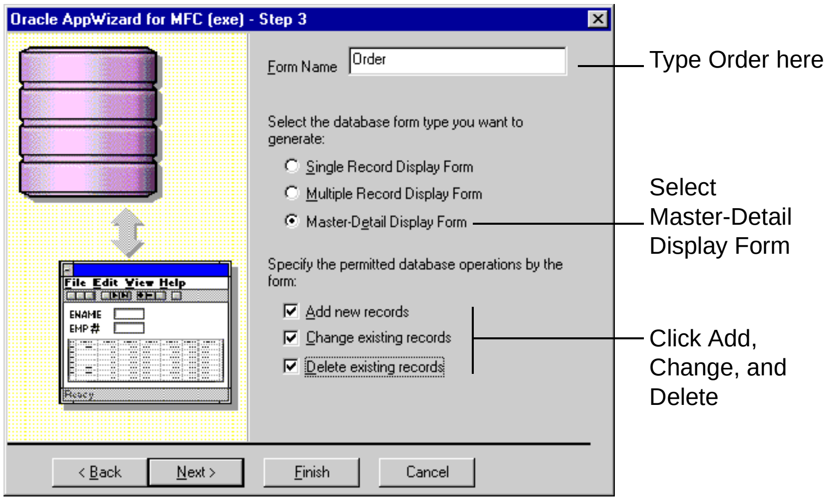
User Name

Password

Database Alias

Naming and Specifying the Type of Form

1. Enter the name of the form: Order.
2. Select Master-Detail Display Form.
3. Click the Add, Change, and Delete checkboxes.
4. Click Next.



Oracle AppWizard for MFC (exe) - Step 3

Form Name: Order

Select the database form type you want to generate:

- ( ) Single Record Display Form
- ( ) Multiple Record Display Form
- (●) Master-Detail Display Form

Specify the permitted database operations by the form:

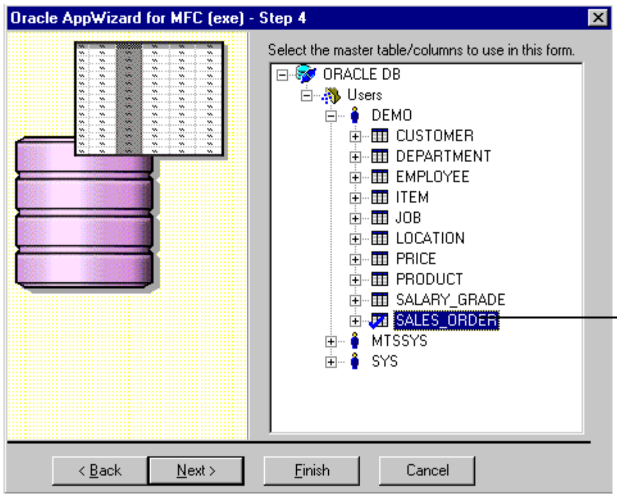
- [✓] Add new records
- [✓] Change existing records
- [✓] Delete existing records

Selecting Master Tables and Columns

1. Select the SALES_ORDER table under the DEMO user in the list. This becomes the master table.

   By default, all columns under SALES_ORDER are selected.

2. Accept the default and click Next.



Oracle AppWizard for MFC (exe) - Step 4

Select the master table/columns to use in this form.

```
ORACLE DB
  Users
    DEMO
      CUSTOMER
      DEPARTMENT
      EMPLOYEE
      ITEM
      JOB
      LOCATION
      PRICE
      PRODUCT
      SALARY_GRADE
      SALES_ORDER
    MTSSYS
    SYS
```

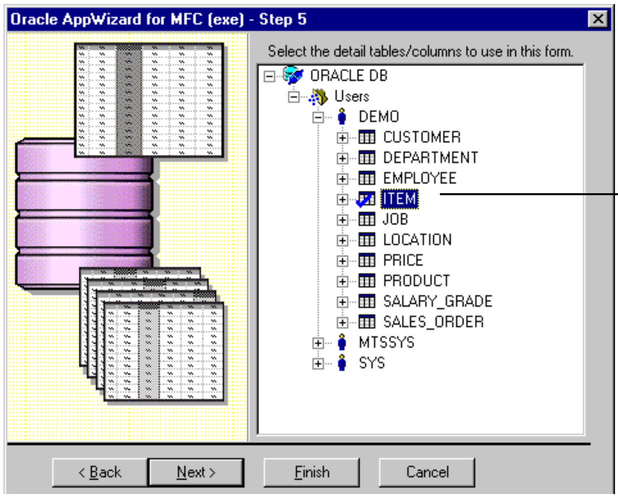
Select the SALES_ORDER table. It will be the master table.

Selecting Detail Tables and Columns

1. Select the ITEM table under the DEMO user in the list. This will be the detail table.

   By default, all columns under ITEM are selected.

2. Accept the default and click Next.


Oracle AppWizard for MFC (exe) - Step 5

Select the detail tables/columns to use in this form.

```
ORACLE DB
  Users
    DEMO
      CUSTOMER
      DEPARTMENT
      EMPLOYEE
      ITEM
      JOB
      LOCATION
      PRICE
      PRODUCT
      SALARY_GRADE
      SALES_ORDER
    MTSSYS
    SYS
```

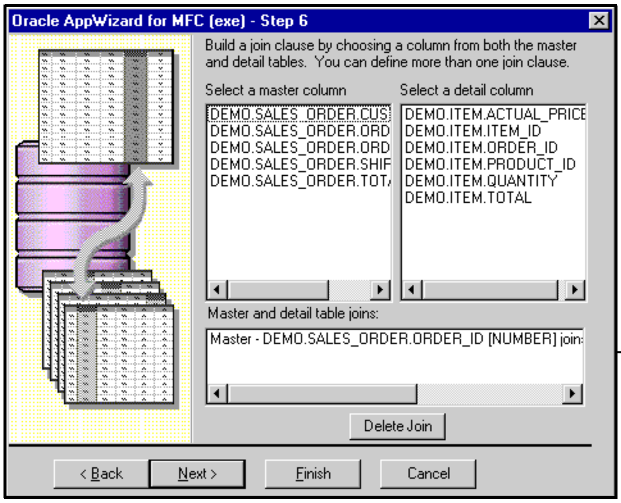
Select the ITEM table.
It will be the detail table.

Building Join Clauses Between Tables

The next window displays the columns from the master and the detail tables you have selected and the default join clause that Oracle AppWizard has created. Oracle AppWizard creates the default join it bases on the primary and foreign keys of the master and detail tables.

In our example, Oracle AppWizard has created a default join, based on the primary and foreign keys:

```
Sales.Order.Order_ID joined with Item.Order_ID
```



Build a join clause by choosing a column from both the master and detail tables. You can define more than one join clause.

Select a master column

DEMO.SALES_ORDER.CUS
DEMO.SALES_ORDER.ORD
DEMO.SALES_ORDER.ORD
DEMO.SALES_ORDER.SHIF
DEMO.SALES_ORDER.TOT

Select a detail column

DEMO.ITEM.ACTUAL_PRICE
DEMO.ITEM.ITEM_ID
DEMO.ITEM.ORDER_ID
DEMO.ITEM.PRODUCT_ID
DEMO.ITEM.QUANTITY
DEMO.ITEM.TOTAL

Master and detail table joins:

Master - DEMO.SALES_ORDER.ORDER_ID [NUMBER] join:

The default join

3. Click Next to use the default join.

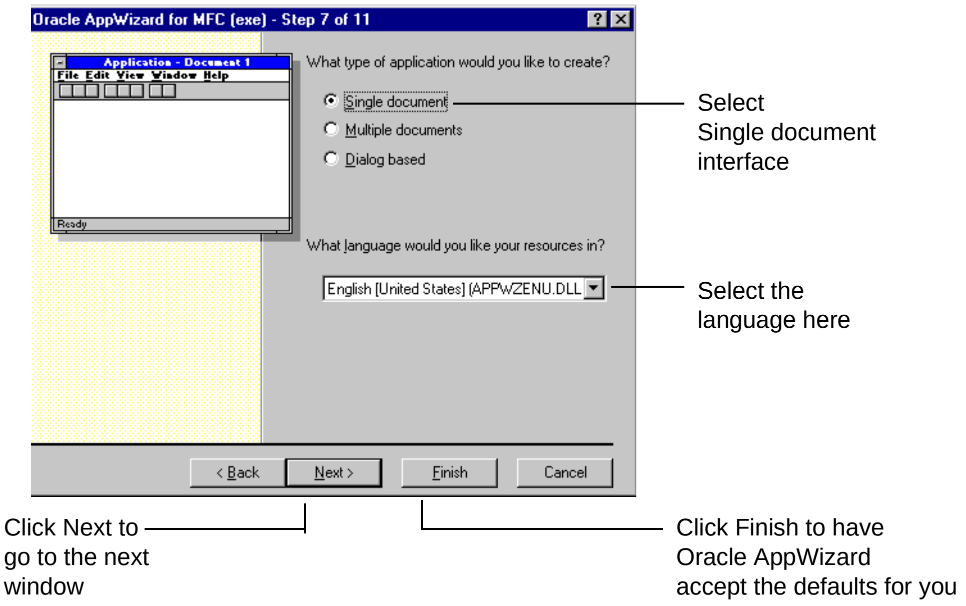
   Oracle AppWizard prompts you:

   Would you like to create another form?

4. Click No.

Specifying the Application Type and User Language

1. Select Single document.
2. Accept the default for Document/View Architecture Support.
3. Accept the current default language. In this tutorial, we use English [United States].
4. Click Next.


Oracle AppWizard for MFC (exe) - Step 7 of 11
Application - Document 1
File Edit View Window Help
Ready
< Back
Next >
Finish
Cancel

What type of application would you like to create?

[✓ Single document] Multiple documents Dialog based

What language would you like your resources in?

English [United States] (APPWZENU.DLL

Select Single document interface

Select the language here

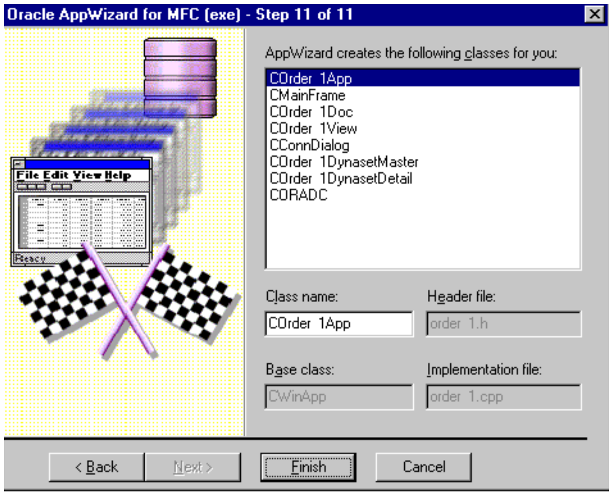
Click Next to go to the next window

Click Finish to have Oracle AppWizard accept the defaults for you

Completing the Remaining Steps

1. Accept the default options for the remaining steps by clicking Next on each window. You can also click Finish and Oracle AppWizard for Microsoft Visual C++ will accept the defaults for you.

   In the last window, the Next button is disabled, as shown in the following diagram.



AppWizard creates the following classes for you:

COrder 1App
CMainFrame
COrder 1Doc
COrder 1View
CConnDialog
COrder 1DynasetMaster
COrder 1DynasetDetail
CORADC

Class name: COrder 1App
Header file: order 1.h
Base class: CWinApp
Implementation file: order 1.cpp

Next is disabled

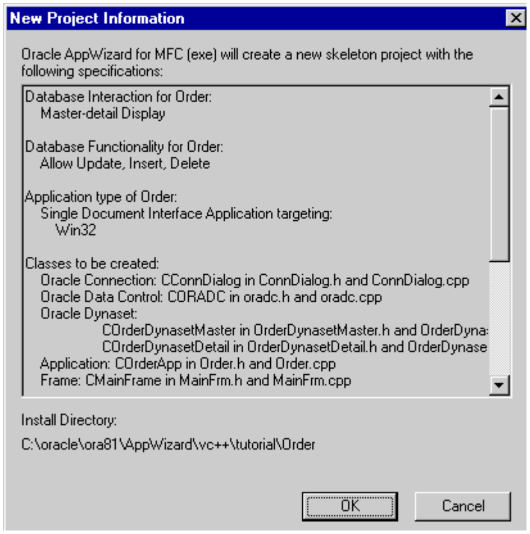
Click Finish to allow Oracle AppWizard to create the starter application

In this window, Oracle AppWizard for Microsoft Visual C++ displays the classes it creates for your application.

2. Click Finish to allow Oracle AppWizard for Microsoft Visual C++ to create the starter application.

   The *New Project Information* window appears. This window shows you the specifications that Oracle AppWizard used to create the Order application.

3. Examine the information in the *New Project Information* window for accuracy.



Oracle AppWizard for MFC (exe) will create a new skeleton project with the following specifications:

```
Database Interaction for Order:
    Master-detail Display

Database Functionality for Order:
    Allow Update, Insert, Delete

Application type of Order:
    Single Document Interface Application targeting:
        Win32

Classes to be created:
    Oracle Connection: CConnDialog in ConnDialog.h and ConnDialog.cpp
    Oracle Data Control: CORADC in oradc.h and oradc.cpp
    Oracle Dynaset:
                COrderDynasetMaster in OrderDynasetMaster.h and OrderDyna:
                COrderDynasetDetail in OrderDynasetDetail.h and OrderDynase
    Application: COrderApp in Order.h and Order.cpp
    Frame: CMainFrame in MainFrm.h and MainFrm.cpp
```

Install Directory:

C:\oracle\ora81\AppWizard\vc++\tutorial\Order

4. When done, click OK.

Oracle AppWizard for Microsoft Visual C++ creates the source, header, and resource files for your Order application and takes you automatically to the Microsoft Developer Studio with Workspace “Order” open.

Part 2: Exploring Generated Classes and Files

Now that you have created the Order application using Oracle AppWizard for Microsoft Visual C++, you can customize it to better suit your needs. To customize this application you need to know about the classes and files created for this purpose, as shown in the following table:

Files	Class	Description	Derived From...
AboutDlg.cpp AboutDlg.h	CAboutDlg	dialog class for *About Order* dialog box	CDialog
ConnDialog.cpp ConnDialog.h	CConnDialog	dialog class for *Connect to Oracle* dialog box	CDialog
MainFrame.cpp MainFrame.h	CMainFrame	frame window class	CFrameWnd
ORADC.cpp ORADC.h	CORADC	data control class	CWnd
OrderApp.cpp OrderApp.h	COrderApp	application class	CWinApp
OrderDynasetDetail.cpp OrderDynasetDetail.h	COrderDynasetDetail	dynaset class for Sales Order table	ODynaset
OrderDynasetMaster.cpp OrderDynasetMaster.h	COrderDynasetMaster	dynaset class for Item table	ODynaset
OrderView.cpp OrderView.h	COrderView	form view class	CFormView
OrderDoc.cpp OrderDoc.h	COrderDoc	document class	CDocument
OrderUtil.cpp OrderUtil.h	Not applicable	stand alone functions: Process OO4OError(), DDX_Field Text(), ...	

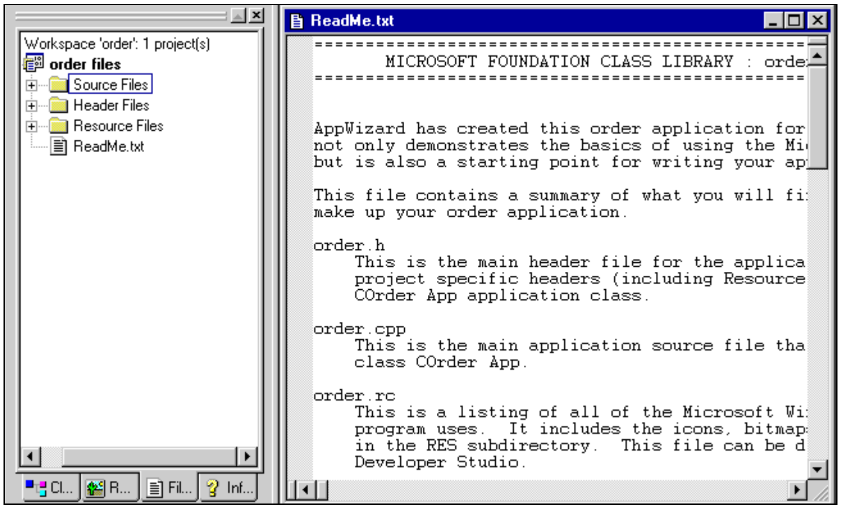
Part 3: Viewing the ReadMe.txt for the Generated Project

Oracle AppWizard for Microsoft Visual C++ creates a ReadMe.txt file for the generated Order project. This ReadMe.txt file describes the source, header, and resource files that Oracle AppWizard has created.

To view the contents of the ReadMe.txt file:

1. Select the FileView tab.
2. Click the name of your project, Order, to open it in the Workspace of the dialog box.
3. Click the ReadMe.txt file to open it.

The following diagram shows what the ReadMe.txt file for Order should look like:



```
================================================
       MICROSOFT FOUNDATION CLASS LIBRARY : orde
================================================


AppWizard has created this order application for
not only demonstrates the basics of using the Mi
but is also a starting point for writing your ap

This file contains a summary of what you will fi
make up your order application.

order.h
    This is the main header file for the applica
    project specific headers (including Resource
    COrder App application class.

order.cpp
    This is the main application source file tha
    class COrder App.

order.rc
    This is a listing of all of the Microsoft Wi
    program uses.  It includes the icons, bitmap
    in the RES subdirectory.  This file can be d
    Developer Studio.
```

Part 4: Building and Running the Application

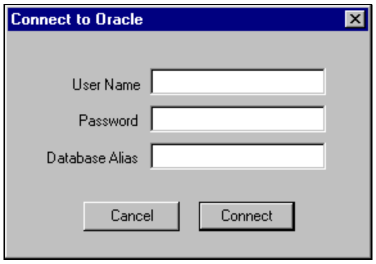
In this section you use Microsoft Developer Studio to build and run the executable files created by Oracle AppWizard for Microsoft Visual C++.

Note: By default, the active project configuration is the debug version. To change the active project configuration, choose Set Active Configuration from the Build menu.

---

1. Choose Build Order.exe from the Build menu to build the application.
2. Choose Execute Order.exe from the Build menu to run the application.

   The *Connect to Oracle* dialog box appears.



3. Enter DEMO in the User Name text field and DEMO in the Password text field. If connecting to a remote database, type the database alias in the Database Alias text field.

4. Click Connect.

   The Order application appears.

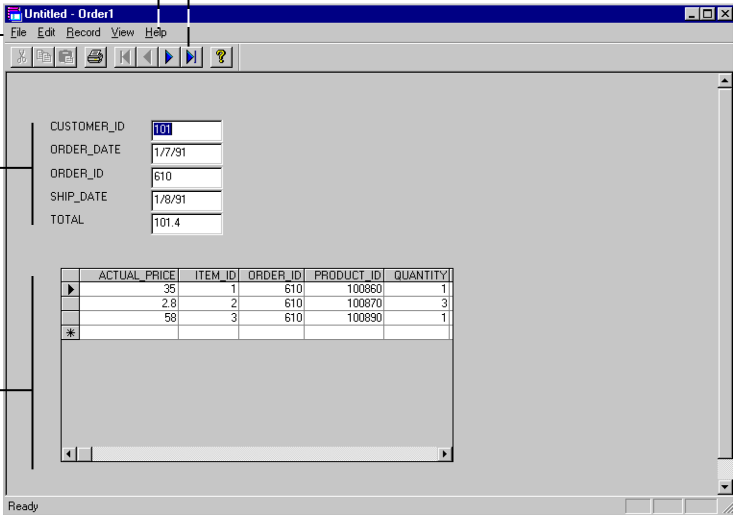
Help menu

Arrow controls for navigating through records

Exit through File menu

Master table controls: Shows data from Sales_Order table.

Detail table control: Shows data from Item table.



CUSTOMER_ID	101
ORDER_DATE	1/7/91
ORDER_ID	610
SHIP_DATE	1/8/91
TOTAL	101.4

ACTUAL_PRICE	ITEM_ID	ORDER_ID	PRODUCT_ID	QUANTITY
35	1	610	100860	1
2.8	2	610	100870	3
58	3	610	100890	1

Notice and try using the following features:

- Arrow controls

  These controls on the toolbar let you easily navigate through the records.
- Record menu

  Another method for navigating through the records.
- View menu

  Shows or hides the toolbar or status bar.
- Help > About Order

  This displays the *About Order* window. (Clicking OK closes the window.)
- File > Exit

  This exits the application.

Lesson 2: Adding Customer Information to a Purchase Order

In this first customization task, you learn how to add specific, detailed information about customers that should appear on any purchase order for the Order application. To accomplish this task, complete the following parts of this lesson:

- Part 1: Creating a Dynaset Class for the Customer Table
- Part 2: Adding Customer Information to a Purchase Order
- Part 3: Displaying Customer Information for a Purchase Order

To see the set of completed files for Lesson 2, access:

ORACLE_BASE\ORACLE_HOME\APPWIZARD\VC++\TUTORIAL\ORDER2.

Note: Some parts of this lesson require you to add or modify code in the application. The code that must be added or modified appears in **bold** type.

Part 1: Creating a Dynaset Class for the Customer Table

To display customer information on your purchase order form for an active order, you must first create an ODynaset class to represent the CUSTOMER table. To do this, create a new generic class called COrderCustomerDynaset, which is derived from the ODynaset class.

The ODynaset class is a class in the Oracle Objects for OLE C++ class library. It creates, manages, and accesses data records from the database.

To create a dynaset for the CUSTOMER table:

1. Right-click Order class and select New Class in the Class view of Microsoft Developer Studio.

   The *New Class* dialog box appears.

2. Select Generic Class from the Class Type list.

3. Enter the following in the Name text box under Class Information:

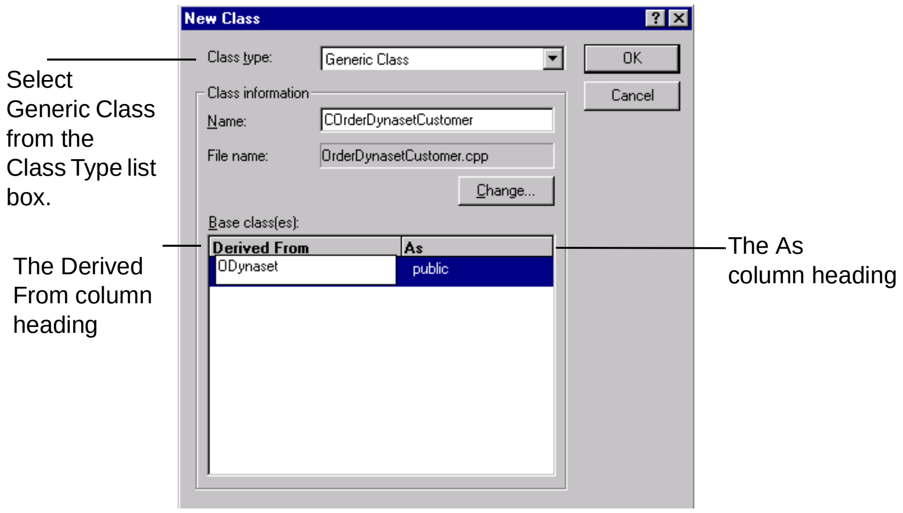
   ```
   COrderDynasetCustomer
   ```

4. Enter the following under the column heading Derived From in the Base Class(es) list:

   ```
   ODynaset
   ```

5. Accept the default entry, "public" under the As column heading.

   This is how the *New Class* dialog box appears before you click OK.


Select Generic Class from the Class Type list box.
The Derived From column heading
The As column heading
New Class
OK
Cancel
Change...

Class type: Generic Class

Class information

Name: COrderDynasetCustomer

File name: OrderDynasetCustomer.cpp

Base class(es):

Derived From	As
ODynaset	public

6. Click OK.

   Microsoft Developer Studio displays the following message:

The New Class Wizard could not find the appropriate header file(s) to include for the base class(es) ODynaset...

7. Click OK.

   This message occurs because the definition of the ODynaset class is not included in the generated file. Therefore, you must manually include the header file that defines the ODynaset class, ORACL.H, in the OrderDynaSetCustomer.h file. You can locate the header file in:

   ```
   ORACLE_BASE\ORACLE_HOME\0040\cpp\include
   ```

   Oracle AppWizard for Microsoft Visual C++ generates both an OrderDynasetCustomer.cpp file and OrderDynasetCustomer.h file for the COrderDynasetCustomer class with the following contents:

   The OrderDynasetCustomer.cpp file contains:

   ```
   //OrderDynasetCustomer.cpp: implmentation of the COrderDynasetCustomer
   class.
   //
   //////////////////////////////////////////////////////////
   #include "stdafcx.h"
   #include "Order.h"
   #include "OrderDynasetCustomer.h"
   COrderDynasetCustomer:COrderDynasetCustomer()
   {

   }
   COrderDynasetCustomer:~COrderDynasetCustomer()
   {

   }
   ```

   The OrderDynasetCustomer.h file contains:

   ```
   //OrderDynasetCustomer.h: interface for the COrderDynasetCustomer Class//
   //////////////////////////////////////////////////////////
   class COrderDynasetCustomer : public ODynaset
   {
   public:
   COrderDynasetCustomer();
   };
   ```

Adding Member Variables to the COrderDynasetCustomer Class

COrderDynasetCustomer class is derived from the ODynaset class. This class contains member variables that store information about a SQL statement to query the database. This class also stores information to represent columns in the USER.DEMO.CUSTOMER table. These are shown below:

Member Variables	Type	Type Description
m_strSQLQuery	CString	Query to be sent to the database
m_strSQLSelect	CString	Select clause of the SQL statement
m_strSQLFilter	CString	Where clause of the SQL statement
m_strSQLSort	CString	Order clause of the SQL statement
m_Column1	OField	CUSTOMER_ID column of CUSTOMER table
m_NAME	OField	NAME column of CUSTOMER table
m_ADDRESS	OField	ADDRESS column of CUSTOMER table
m_CITY	OField	CITY column of CUSTOMER table
m_STATE	OField	STATE column of CUSTOMER table
m_ZIP_CODE	OField	ZIP column of CUSTOMER table
m_AREA_CODE;	OField	AREA_CODE column of CUSTOMER table
m_PHONE_NUMBER	OField	TELEPHONE_NUMBER column of CUSTOMER table
m_SALESPERSON_ID	OField	SALESPERSON_ID column of CUSTOMER table
m_CREDIT_LIMIT	OField	CREDIT_LIMIT column of CUSTOMER table
m_COMMENTS	OField	COMMENTS column of CUSTOMER table

To add member variables for the COrderDynasetCustomer class:

1. Enter the following code in the COrderDynasetCustomer.h file:

```
class COrderDynasetCustomer : public ODynaset
{
...
public:
    COrderDynasetCustomer();
    virtual ~COrderDynasetCustomer();
public:
// strings needed for creating the queries
   CString m_strSQLQuery;  // the query to be sent to the database
   CString m_strSQLSelect; // the select portion
   CString m_strSQLFilter; // the where portion
   CString m_strSQLSort;   // the order by portion

// Field/Param Data
       OField m_CUSTOMER_ID;
       OField m_NAME;
       OField m_ADDRESS;
       OField m_CITY;
       OField m_STATE;
       OField m_ZIP_CODE;
       OField m_AREA_CODE;
       OField m_PHONE_NUMBER;
       OField m_SALESPERSON_ID;
       OField m_CREDIT_LIMIT;
       OField m_COMMENTS;
...
};
```

Adding Member Functions and Implementation Details to the COrderDynasetCustomer Class

Adding the member functions shown below to the COrderDynaset customer class enables you to create a query statement, have the database process it, then retrieve and store information.

Member Functions	Descriptions
OpenQuery	Starts a query and retrieves the information from the database.
CreateSQLSelect	Creates the default query (SELECT) statement.
AddFilter	Adds a condition to the query statement.
ResetToDefaultFilter	Resets the condition clause for the query.
RefreshQuery	Refreshes the query to be executed.

To declare member functions for the COrderDynasetCustomer class:

1. Add the following code to the OrderDynasetCustomer.h file:

```
class COrderDynasetCustomer : public ODynaset
{
...

    public:
            COrderDynasetCustomer();
            virtual ~COrderDynasetCustomer();


   // Operations
   public:
   void OpenQuery(ODatabase theDB);
   void CreateSQLSelect();
   void AddFilter(CString strFilter);
   void ResetToDefaultFilter();
   void RefreshQuery();
    ...
 };
```

2. Add the following code to the constructor method of COrderDynasetCustomer Class in the OrderDynasetCustomer.cpp file to initialize the value in the member variables:

```
COrderDynasetCustomer::COrderDynasetCustomer()
{
   m_strSQLQuery.Empty();
   m_strSQLSelect.Empty();
   m_strSQLFilter.Empty();
   m_strSQLSort.Empty();

   // Create the default select clause of the
      statement m_strSQLSelect

     CreateSQLSelect();
}
```

3. Add the following code for the COrderDynasetCustomer class::*CreateSQLSelect()* method at the end of the OrderDynasetCustomer.cpp file to create the default SQL statement:

```
// Creates the default select
void COrderDynasetCustomer::CreateSQLSelect()
   {
        m_strSQLSelect = "select CUSTOMER_ID, NAME,
            ADDRESS, CITY, STATE, \
            ZIP_CODE, AREA_CODE, PHONE_NUMBER, \
            SALESPERSON_ID, CREDIT_LIMIT, COMMENTS \
            from DEMO.CUSTOMER";
    }
```

4. Add the following code for the COrderDynasetCustomer class::*OpenQuery()* method in the OrderDynasetCustomer.cpp file to:
   - Create a query statement
   - Process the query
   - Retrieve information from the database

```
// opens the query
void COrderDynasetCustomer::OpenQuery(ODatabase theDB)
{
    oresult dbresult;
    // create a query statement
        m_strSQLQuery = m_strSQLSelect;
        if (m_strSQLFilter)
              m_strSQLQuery += m_strSQLFilter;
        if (m_strSQLSort)
              m_strSQLQuery += m_strSQLSort;

    // query the database
        dbresult = Open(theDB, m_strSQLQuery);

    // retreive/store information from the database
        m_CUSTOMER_ID = GetField("CUSTOMER_ID");
        m_NAME = GetField("NAME");
        m_ADDRESS = GetField("ADDRESS");
        m_CITY = GetField("CITY");
        m_STATE = GetField("STATE");
        m_ZIP_CODE = GetField("ZIP_CODE");
        m_AREA_CODE = GetField("AREA_CODE");
        m_PHONE_NUMBER = GetField("PHONE_NUMBER");
        m_SALESPERSON_ID = GetField("SALESPERSON_ID");
        m_CREDIT_LIMIT = GetField("CREDIT_LIMIT");
        m_COMMENTS = GetField("COMMENTS");

    // display the first record from the Customer table
        dbresult = MoveFirst();
}
```

5. Add the following code for the COrderDynasetCustomer class::*AddFilter()* method in the OrderDynasetCustomer.cpp file to add a conditional clause for the query:

```
// adds a condition to the where clause
   void COrderDynasetCustomer::AddFilter(CString strFilter)
   {
      m_strSQLFilter += m_strSQLFilter.IsEmpty() ? "
          WHERE " : " AND ";
      m_strSQLFilter += strFilter;
   }
```

6. Add the following code for the COrderDynasetCustomer class::*ResetToDefaultFilter()* method in the OrderDynasetCustomer.cpp file to reset the conditional clause for the query:

```
// resets the filter to the default value
void COrderDynasetCustomer::ResetToDefaultFilter()
{
     m_strSQLFilter.Empty();
}
```

7. Add the following code for COrderDynasetCustomer class::*RefreshQuery()* method in the OrderDynasetCustomer.cpp file to refresh the query to be executed:

```
void COrderDynasetCustomer::RefreshQuery()
{
   oresult dbresult;

   // create a query statement
      m_strSQLQuery = m_strSQLSelect;
      if (m_strSQLFilter)
          m_strSQLQuery += m_strSQLFilter;
      if (m_strSQLSort)
          m_strSQLQuery += m_strSQLSort;

   // set the query statement to be used
      dbresult = SetSQL(m_strSQLQuery);
      dbresult = Refresh();
}
```

Part 2: Adding Customer Information to a Purchase Order

To display customer information for a purchase order, add the customer dynaset to the COrderView class and COrderDoc class.

To add customer information to a purchase order:

1. Add a member variable called m_pDynasetCustomer to the COrderView class in the COrderView.h header file. This member variable is a pointer to the type COrderCustomerDynaset class.

   ```
   class COrderView : public CFormView
   {
       ...
       COrderDynasetMaster *m_pDynasetMaster;
       COrderDynasetDetail *m_pDynasetDetail;
       COrderDynasetCustomer *m_pDynasetCustomer;
       ...
   }
   ```

2. Initialize the variable, m_pDynasetCustomer, to NULL in the constructor method of COrderView class in the the COrderView.cpp file.

   ```
   COrderView::COrderView()
       : CFormView(COrderView::IDD)
   {
       ...
       m_pDynasetMaster = NULL;
       m_pDynasetDetail = NULL;
       m_pDynasetCustomer = NULL;
       ...
   }
   ```

3. Add a member variable called m_OrderDynasetCustomer to the COrderDoc class in the COrderCustomerDynaset.cpp file. This is an object of COrderCustomerDynaset class.

   ```
   class COrderDoc : public CDocument
   {
       ...
       COrderDynasetMaster m_OrderDynasetMaster;
       COrderDynasetDetail m_OrderDynasetDetail;
       COrderDynasetCustomer m_OrderDynasetCustomer;
       ...
   }
   ```

4. Include the OrderDynasetCustomer.h header file in Order.cpp, OrderDoc.h, and OrderView.h to define the COrderDynasetCustomer class.

Part 3: Displaying Customer Information for a Purchase Order

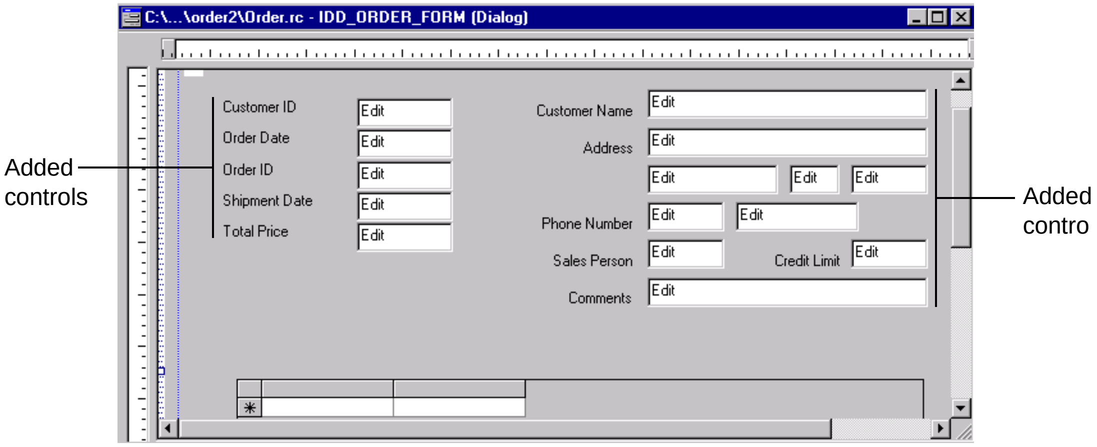
This section demonstrates how to display customer information for each purchase order.

1. Add 10 edit controls to display customer information in the IDD_ORDER_FORM dialog box with the Control IDs as shown in the following table:

Table 4–1 Control IDs with Member Variables

Control ID	Member Variables for COrderDynasetCustomer Class
IDC_CUST_NAME	m_NAME
IDC_CUST_ADDRESS	m_ADDRESS
IDC_CUST_CITY	m_CITY
IDC_CUST_STATE	m_STATE
IDC_CUST_ZIP_CODE	m_ZIP_CODE
IDC_CUST_AREA_CODE	m_AREA_CODE
IDC_CUST_PHONE_NUMBER	m_PHONE_NUMBER
IDC_CUST_SALESPERSON_ID	m_SALESPERSON_ID
IDC_CUST_CREDIT_LIMIT	m_CREDIT_LIMIT
IDC_CUST_COMMENTS	m_COMMENTS

After you add these edit controls, the IDD_ORDER_FORM dialog box appears similar to the one shown below. This illustration serves only as a sample. Yours may look slightly different.



To bind these edit controls with the OField members in the COrderDynasetCustomer class, call the *DDX_FieldText()* method for each customer edit control in the COrderView class::*DoDataExchange()* method, as described in the next step.

The OField members represent the columns in the Customer table. Bind these with a corresponding OField member variable of the COrderDynasetCustomer class shown in Table 4–1.

2. Enter the following code in COrderView:DoData Exchange in the OrderView.cpp file to bind the customer edit controls with the OField members in the COrderDyaset Customer class:

```
void COrderView::DoDataExchange(CDataExchange* pDX)
{
    ...
    CFormView::DoDataExchange(pDX);
    ...
    // for customer dynaset
    DDX_FieldText(pDX, IDC_CUST_NAME,
       m_pDynasetCustomer->m_NAME, m_pDynasetCustomer);
    DDX_FieldText(pDX, IDC_CUST_ADDRESS,
       m_pDynasetCustomer->m_ADDRESS, m_pDynasetCustomer);
    DDX_FieldText(pDX, IDC_CUST_CITY, m_pDynasetCustomer->m_CITY,
       m_pDynasetCustomer);
    DDX_FieldText(pDX, IDC_CUST_STATE,
       m_pDynasetCustomer->m_STATE, m_pDynasetCustomer);
    DDX_FieldText(pDX, IDC_CUST_ZIP_CODE,
       m_pDynasetCustomer->m_ZIP_CODE, m_pDynasetCustomer);
    DDX_FieldText(pDX, IDC_CUST_AREA_CODE,
        m_pDynasetCustomer->m_AREA_CODE, m_pDynasetCustomer);
    DDX_FieldText(pDX, IDC_CUST_PHONE_NUMBER,
        m_pDynasetCustomer->m_PHONE_NUMBER, m_pDynasetCustomer);
    DDX_FieldText(pDX, IDC_CUST_CREDIT_LIMIT,
        m_pDynasetCustomer->m_CREDIT_LIMIT, m_pDynasetCustomer);
    DDX_FieldText(pDX, IDC_CUST_SALESPERSON_ID,
        m_pDynasetCustomer->m_SALESPERSON_ID, m_pDynasetCustomer);
    DDX_FieldText(pDX, IDC_CUST_COMMENTS,
        m_pDynasetCustomer->m_COMMENTS, m_pDynasetCustomer);
    ...
}
```

3. Add the following code to the COrderView::*OnInitialUpdate()* method in OrderView.cpp to initialize the customer dynaset when the view is attached to the document:

```
void COrderView::OnInitialUpdate()
{
    CString strJoin;
    ...
    m_pDynasetMaster = &GetDocument()->m_OrderDynasetMaster;
    m_pDynasetDetail = &GetDocument()->m_OrderDynasetDetail;
    m_pDynasetCustomer = &GetDocument()->m_OrderDynasetCustomer;

    m_pDynasetMaster->OpenQuery(GetDocument()->m_database);
}
```

4. Add the following code to the COrderView::*OnInitialUpdate()* method in OrderView.cpp to perform a join operation between the CUSTOMER_ID column of the SALES_ORDER table and the CUSTOMER_ID column of the ITEM table to retrieve customer information for a purchase order:

```
    ...
    m_pDynasetMaster->OpenQuery(GetDocument()->m_database);

    // create a join based on CUSTOMER_ID
    // between the Customer table and
    // the Sales.Order table.
    m_pDynasetCustomer->ResetToDefaultFilter();
    strJoin = "CUSTOMER_ID = " +
        (CString)m_pDynasetMaster->m_Column1;
    m_pDynasetCustomer->AddFilter(strJoin);
    // create, process the query
    m_pDynasetCustomer->OpenQuery(GetDocument()->m_database);
    ...
CFormView::OnInitialUpdate();
    m_dataControl.SetRecordset((LPDISPATCH)(m_pDynasetDetail->Internal()));
    ...
}
```

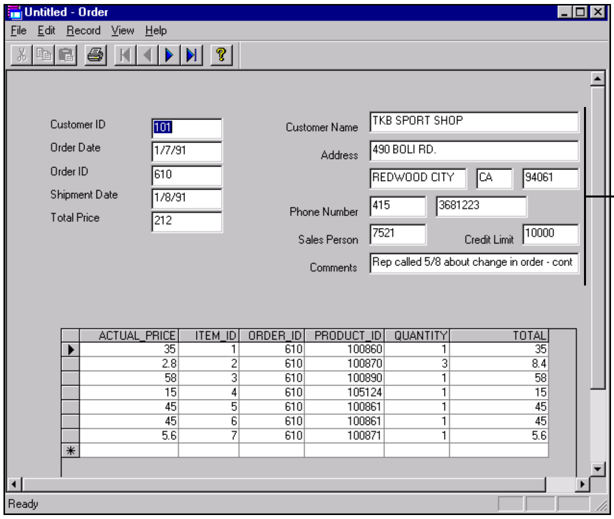
5. Enter the following code to the *PerformMove()* method of the COrderView class in OrderView.cpp to update the customer dynaset after a move operation has been performed:

```
void COrderView::PerformMove(int nCommand)
{
    ...
    // Update Customer information
    m_pDynasetCustomer->ResetToDefaultFilter();
    strJoin = "CUSTOMER_ID = " +
        (CString)m_pDynasetMaster->m_Column1;
    m_pDynasetCustomer->AddFilter(strJoin);
    m_pDynasetCustomer->RefreshQuery();

    UpdateData(FALSE);
}
```

6. Build and run the application.

This is how the application should look when you are done:



Customer ID	101
Order Date	1/7/91
Order ID	610
Shipment Date	1/8/91
Total Price	212
Customer Name	TKB SPORT SHOP
Address	490 BOLI RD.
	REDWOOD CITY · CA · 94061
Phone Number	415 · 3681223
Sales Person	7521
Credit Limit	10000
Comments	Rep called 5/8 about change in order - cont

	ACTUAL_PRICE	ITEM_ID	ORDER_ID	PRODUCT_ID	QUANTITY	TOTAL
▶	35	1	610	100860	1	35
	2.8	2	610	100870	3	8.4
	58	3	610	100890	1	58
	15	4	610	105124	1	15
	45	5	610	100861	1	45
	45	6	610	100861	1	45
	5.6	7	610	100871	1	5.6
*						

The customer information is now displayed

Lesson 3: Enabling Users to Add Products to a Purchase Order

Nicole’s employees must add products to a purchase order. This lesson demonstrates how to customize your application to:

- Display available products from the PRODUCT table
- Add product items to a purchase order
- Update the changes in the ITEM table and the detail table control when product items have been added to a purchase order
- Update the changes in the SALES_ORDER table and the master table controls when product items have been added to a purchase order

Lesson 3 contains the following parts:

- Part 1: Displaying a List of Items from the PRODUCT Table
- Part 2: Adding the Selected Products to the Purchase Order

To see the set of completed files for Lesson 3, access:

ORACLE_HOME\APPWIZARD\VC++\TUTORIAL\ORDER3.

Note: Some parts of this lesson require you to add or modify code in the application. The code that must be added or modified appears in **bold** type.

Part 1: Displaying a List of Items from the PRODUCT Table

Nicole's Sporting Goods carries a wide variety of products. Many of them have similar names and descriptions. To insure data integrity, display a list of the product items in the purchase order application window. This enables Nicole’s employees to select the items that customers have purchased from the list.

In this part, you will add a list box that displays the available products in the *IDD_ORDER_FORM* dialog box.

To add a list box:

1. Add a list box with a control ID = IDC_PRODUCTLIST that allows multiple selections. This enables users to select multiple purchased items, which can then be added to a purchase order, all at one time.
2. Create a new member variable, m_prodList, for the list box in COrderView class. The variable m_prodList is a control handler of type CListBox. It is used

to store the list of product items from the PRODUCT table. Bind the IDC_ PRODUCTLIST list box control with this new member variable.

3. Add the following code to COrderView::*OnInitialUpdate()* method in the OrderView.cpp file to display a list of available products from the PRODUCT table in the list box:

```
void COrderView::OnInitialUpdate()
{
    ...
    CFormView::OnInitialUpdate()
        ...

        // Put list of sales items into the listbox
        ODynaset oProdList;
        oProdList.Open(GetDocument()->m_database,
             "SELECT PRODUCT_ID,DESCRIPTION
             FROM DEMO.PRODUCT ORDER BY PRODUCT_ID");
        int index = 0;
        while (!oProdList.IsEOF())
    {
           m_prodList.InsertString(index,
               (CString)oProdList.GetField(0) + " - " +
                oProdList.GetField(1));

        m_prodList.SetItemData(index,
               (long)oProdList.GetField(0));
        oProdList.MoveNext();
        index++;
    }
}
```

Part 2: Adding the Selected Products to the Purchase Order

When Nicole's employees add a purchased item from the product list box to a purchase order, a new record for the purchased item is inserted into the ITEM table. Nicole wants the TOTAL column in the SALES_ORDER table to be re-calculated to reflect the amount added from the purchased item to the purchase order. Content in the detail table control and the master table control should also be updated accordingly.

1. To have the TOTAL column in the SALES_ORDER table updated when a purchased item is added to a purchase order, the SALES_ORDER table should be updated. To allow the updating of the SALES_ORDER table, remove the third parameter, ODYNASET_READONLY, from the Open() function call in the COrderDynasetMaster::*OpenQuery()* method:

Change:

```
dbresult = Open(theDB, m_strSQLQuery, ODYNASET_READONLY);
```

To:

```
dbresult = Open(theDB, m_strSQLQuery);
```

2. In the IDD_ORDER_FORM dialog, create a button that handles the event of adding a purchase item. The button requires the following properties and events:

Button Control ID	Button Caption	Event	Event Handler Function
IDC_ADDITEMS	Add to Order	BN_Clicked	OnAdditems

3. Add the following code to the COrderView::*OnAddItems()* method in the OrderView.cpp file to process the updates in the SALES _ORDER table, the ITEM table, the master table controls, and the detail table control when purchase items are added to a purchase order:

```
void COrderView::OnAdditems()
{
    int *pAddIndices;
    int nSelCount;
    oresult r;
    nSelCount = m_prodList.GetSelCount();
    pAddIndices = new int[nSelCount];
    m_prodList.GetSelItems(nSelCount, pAddIndices);
    for (int i = 0; i < nSelCount; i++)
    {
        char szProdID[8];
        char szItemID[8];
        itoa(m_prodList.GetItemData(pAddIndices[i]), szProdID,
             10);
        itoa(m_pDynasetDetail->GetRecordCount() + 1, szItemID,
             10);
        (CString strInsert = "INSERT INTO DEMO.ITEM (ORDER_ID,
             PRODUCT_ID, ITEM_ID, QUANTITY,
             ACTUAL_PRICE, TOTAL) SELECT " +
               CString)m_pDynasetMaster->m_Column3 + ", "
             + szProdID +", " + szItemID +
               ", 1, LIST_PRICE, LIST_PRICE FROM
             DEMO.PRICE WHERE PRODUCT_ID = " +
             szProdID + " AND ((SYSDATE BETWEEN
                START_DATE AND END_DATE) OR " +
              "(SYSDATE > START_DATE AND END_DATE IS NULL))";
          r = GetDocument()->m_database.ExecuteSQL(strInsert);
          if (r == OFAILURE)
          ProcessOO4OError(&GetDocument()->m_database);
          m_pDynasetDetail->Refresh();
}
TotalOrder();
m_prodList.SelItemRange(FALSE, 0, m_prodList.GetCount());
delete pAddIndices;
}
```

The function TotalOrder() is being called in the COrderView::*OnAdditems()* method. It is a member function for COrderView class. It re-calculates the total amount for a purchase order and updates the TOTAL column in the SALES_ ORDER table.

4. Enter the following code in the COrderView class of the OrderView.h file to declare the TotalOrder() in COrderView class:

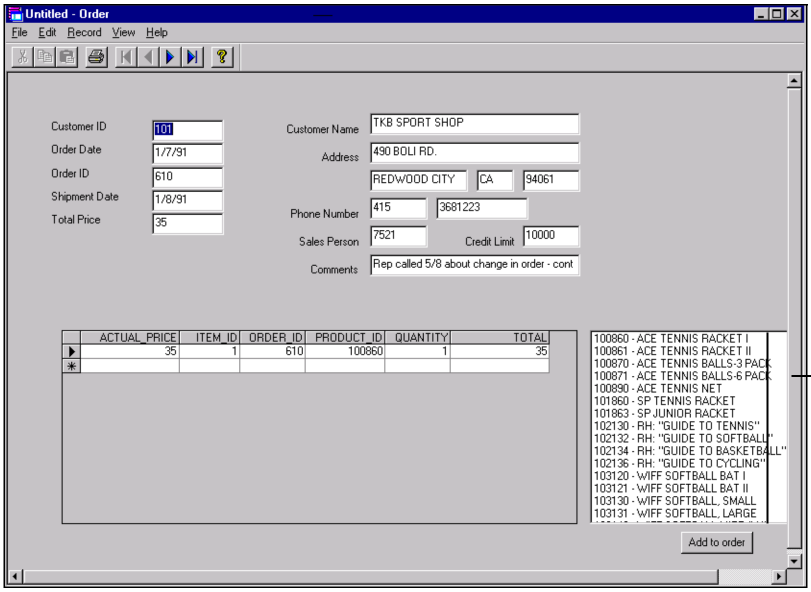
```
public:
    void TotalOrder();
```

5. Add the following code for the COrderView::*TotalOrder()* method in the OrderView.cpp file:

```
void COrderView::TotalOrder()
{
    ODynaset oTotal;
    oTotal.Open(GetDocument()->m_database, "SELECT SUM(TOTAL) FROM
        DEMO.ITEM WHERE ORDER_ID = " +
        (CString)m_pDynasetMaster->m_Column3);
    m_pDynasetMaster->StartEdit();
    m_pDynasetMaster->
        m_Column5.SetValue((double)oTotal.GetField(0));
    m_pDynasetMaster->Update();
    UpdateData(FALSE);
}
```

6. Build and run the application.

The application will look similar to the following screen:



Customer ID	101
Order Date	1/7/91
Order ID	610
Shipment Date	1/8/91
Total Price	35
Customer Name	TKB SPORT SHOP
Address	490 BOLI RD.
	REDWOOD CITY / CA / 94061
Phone Number	415 / 3681223
Sales Person	7521
Credit Limit	10000
Comments	Rep called 5/8 about change in order - cont

	ACTUAL_PRICE	ITEM_ID	ORDER_ID	PRODUCT_ID	QUANTITY	TOTAL
▶	35	1	610	100860	1	35
✳						

100860 - ACE TENNIS RACKET I
100861 - ACE TENNIS RACKET II
100870 - ACE TENNIS BALLS-3 PACK
100871 - ACE TENNIS BALLS-6 PACK
100890 - ACE TENNIS NET
101860 - SP TENNIS RACKET
101863 - SP JUNIOR RACKET
102130 - RH: "GUIDE TO TENNIS"
102132 - RH: "GUIDE TO SOFTBALL"
102134 - RH: "GUIDE TO BASKETBALL"
102136 - RH: "GUIDE TO CYCLING"
103120 - WIFF SOFTBALL BAT I
103121 - WIFF SOFTBALL BAT II
103130 - WIFF SOFTBALL, SMALL
103131 - WIFF SOFTBALL, LARGE

The product information is now displayed

Lesson 4: Enabling Users to Update a Purchase Order

This section allows you to make changes in the detail table control for the Order application. When the value in the QUANTITY column or the ACTUAL_PRICE column in the detail table changes, the application should update the appropriate changes to the ITEM table, the SALES_ORDER table, the content in the master detail controls, and the detail table control accordingly.

This section consists of the following parts:

- Part 1: Allowing the Detail Table Control to Handle Events
- Part 2: Adding Implementation Details to the Event Handler Function

To see the set of completed files for Lesson 4, access:

ORACLE_HOME\APPWIZARD\VC++\TUTORIAL\ORDER4

Note: Some parts of this lesson require you to add or modify code in the application. The code that must be added or modified appears in **bold** type.

Part 1: Allowing the Detail Table Control to Handle Events

1. Modify the detail table control (ID = IDC_DATAGRID) in the IDD_ORDER_FORM Dialog that handles event types with specified event handler functions. These functions and handlers are listed in the following table.

Event Type	Event Handler
AfterColUpdate	OnAfterColUpdateDatagrid
AfterUpdate	OnAfterUpdateDatagrid
AfterDelete	OnAfterDeleteDatagrid

2. Create a member variable, m_datagrid, for the detail table control in COrderView class. The variable, m_datagrid, is a control handler of type CMsDgridCtrl. It represents the detail table and allows for update or retrieval of data.

Part 2: Adding Implementation Details to the Event Handler Function

When the Actual Price or the Quantity sold for a purchased item has been changed, the total price for this item should change. To have the application do this, additions must be made to the code.

1. Add the following code to the COrderView::*OnAfterColUpdateDatagrid()* method in the OrderView.cpp file to update the TOTAL column for the detail table control when the ACTUAL_PRICE column or the QUANTITY column in the detail table control changes:

```
void COrderView::OnAfterColUpdateDatagrid(short ColIndex)

// recalcuates the TOTAL column in the detail table when the
value
// in the ACTUAL_PRICE or QUANTITY column changes

{
   if (ColIndex == 4 || ColIndex == 0)
   {
      CString strText = m_datagrid.GetText();
      CString strTotal;
      char *stopString;
      char szTotal[15];
      long quantity;
      double price;

      // QUANTITY column is updated
      if (ColIndex == 4)
      {
        quantity = strtol(strText, &stopString, 10);
        m_datagrid.SetCol(0);
        strText = m_datagrid.GetText();
        price = strtod(strText, &stopString);
        m_datagrid.SetCol(4);
      }
      else // PRICE column is updated
      {
        price = strtod(strText, &stopString);
        m_datagrid.SetCol(4);
        strText = m_datagrid.GetText();
        quantity = strtol(strText, &stopString, 10);
        m_datagrid.SetCol(0);
      }
```

```
        sprintf(szTotal, "%f", quantity * price);
        strTotal = szTotal;
        m_datagrid.SetCol(5);
        m_datagrid.SetText((LPCTSTR)strTotal);
    }
}
```

2. Add the following code to the COrderView::*OnAfterUpdateDatagrid()* method in the OrderView.cpp file to update the TOTAL column for the master table controls when the ACTUAL_PRICE column or the QUANTITY column in the detail table change:

```
void COrderView::OnAfterUpdateDatagrid()
{
    // Calculates the total amount for a purchase order
    TotalOrder();
}
```

3. Add the following code to the COrderView::*OnAfterDeleteDatagrid()* method in the OrderView.cpp file to update the TOTAL column for the master table controls when the a record has been deleted from the detail table control:

```
void COrderView::OnAfterDeleteDatagrid()
{
    // Calculates the total amount for a purchase order
    TotalOrder();
}
```

Lesson 5: Enabling Users to Add, Commit, or Cancel a New Purchase Order

This lesson describes how to update the application to enable Nicole's employees to create a new purchase order for a customer. When you have completed this last customization, Nicole's employees can also commit or cancel the creation of a new purchase order.

This section consists of the following parts:

To see the set of completed files for Lesson 5, access:

ORACLE_HOME\APPWIZARD\VC++\TUTORIAL\ORDER5

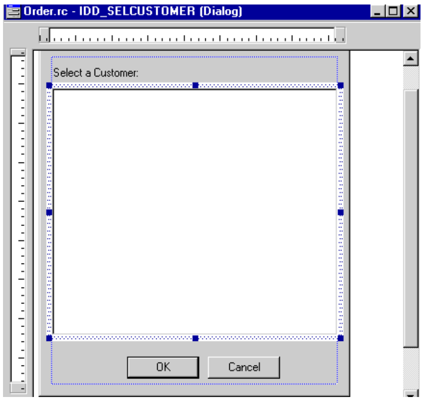
Note: Some parts of this lesson require you to add or modify code in the application. The code that must be added or modified appears in **bold** type.

Part 1: Creating a Customer List Dialog Box

Many of Nicole's customers have similar names. To ensure that the each customer gets the correct merchandise, the Order application will have a dialog box with a list of customers. Nicole's employees can then pick the appropriate name from the list when they create a new purchase order.

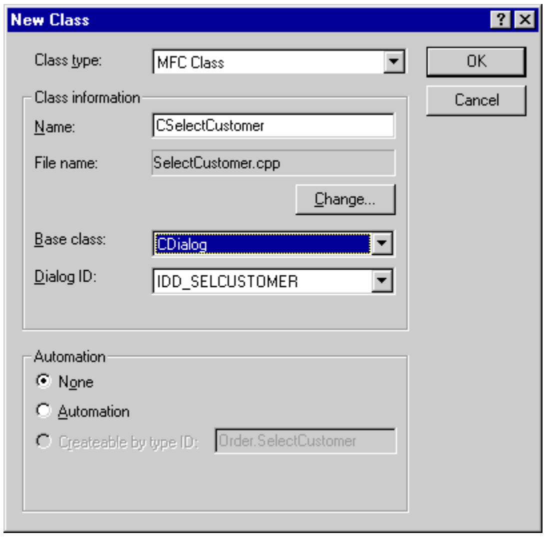
1. Create a dialog box with control ID = IDD_SELCUSTOMER. Customize the dialog box so that it contains an OK Button, Cancel Button, and List Box with control ID = IDC_CUSTLIST.

   The customer list dialog box should look similar to the one shown below. The list box displays a list of customers from the CUSTOMER table.



Part 2: Creating a New Class to Handle Events for the Customer Dialog Box

1. Create a new MFC class, CSelectCustomer, that is derived from CDialog. This class handles events occurring in the customer dialog box.



Class type: MFC Class

Class information

Name: CSelectCustomer

File name: SelectCustomer.cpp

Base class: CDialog

Dialog ID: IDD_SELCUSTOMER

Automation

[✓ None]

Automation

Createable by type ID: Order.SelectCustomer

2. Allow the CSelectCustomer class to handle the events shown below:

Object ID	Event	Event Handler Function
CSelectCustomer	WM_INITDIALOG	OnInitDialog
IDOK	BN_CLICKED	OnOK

3. Create a member variable, m_CustomerList, for the customer list box, IDC_CUSTLIST, in the CSelectCustomer class. The variable, m_CustomerList, is a control handler of type CListBox. This enables users to select a customer from the customer list box for a purchase order.

4. To implement the events for the Customer dialog box, add the following member declarations to the CSelectCustomer class in SelectCustomer.h file:

```
public:
long GetSelection();            // obtain the selected customer from the
                                  // customer list box.
int DoModal(ODatabase *db);  // involve the customer dialog box
long m_nSelection;             // nth location of the selected customers
                                 // in the customer list.
ODatabase * m_pDatabase;       // the database
```

5. Include the header file oracl.h in the SelectCustomer.h file.

6. Add the following code to the CSelectCustomer::*OnInitDialog()* method in the CSelectCustomer.cpp file to display a list of customers in the customer list box:

```
BOOL CSelectCustomer::OnInitDialog()
{
    ...
    CDialog::OnInitDialog();
    ODynaset oCustList;
    int index = 0;

    oCustList.Open(*m_pDatabase, "SELECT CUSTOMER_ID, NAME FROM
        DEMO.CUSTOMER");
    while (!oCustList.IsEOF())
    {
        m_CustomerList.InsertString(index,
              (CString)oCustList.GetField(0) +
              " - " + oCustList.GetField(1));
        m_CustomerList.SetItemData(index,
              (long)oCustList.GetField(0));
        oCustList.MoveNext();
        index++;
    }
...
}
```

7. Add the following code for the CSelectCustomer::*DoModal()* method in the SelectCustomer.cpp file to invoke the customer dialog box:

```
int CSelectCustomer::DoModal(ODatabase *db)
{
    m_pDatabase = db;
    return CDialog::DoModal();
}
```

8. Add the following code for the CSelectCustomer::*GetSelection()* method in the SelectCustomer.cpp file to obtain the selected customer for the new purchase order:

```
long CSelectCustomer::GetSelection()
{
    return(m_nSelection);
}
```

9. Add the following code for the OnOK() method of the CSelectCustomer class to get the selected customer from the customer list box for a new purchase order when the OK button in the customer dialog box is clicked:

```
void CSelectCustomer::OnOK()
{
    // TODO: Add extra validation here
    int nItem = -1;

    nItem = m_CustomerList.GetCurSel();
    if (nItem >=0)
    {
        // location of selected customer in the customer list box
        m_nSelection = m_CustomerList.GetItemData(nItem);
        CDialog::OnOK();
    }
    else
        AfxMessageBox(_T("Please select a customer.\n"),
            MB_OK, 0);
}
```

Part 3: Creating “New Order”, Commit Order”, and “Cancel Order” Buttons

To enable Nicole’s employees to easily add, commit, or cancel a new purchase order, create buttons in the IDD_ORDER_FORM dialog that enable them to perform these operations.

1. Include the header file, SelectCustomer.h, in the OrderView.h.file.

   This header file defines the CSelectCustomer class that is used when getting customer information for a new purchase order.

2. Create the buttons with the following properties shown in the Button Control ID and Event Handler Functions in the IDD_ORDER_FORM dialog.

Button Control ID	Button Caption	Event	Event Handler Function
IDC_NEWORDER	New Order	BN_CLICKED	OnNeworder
IDC_COMMITORDER	Commit Order	BN_CLICKED	OnCommitorder
IDC_CANCELORDER	Cancel Order	BN_CLICKED	OnCancelorder

These event handler functions are added to the COrderView class when the buttons specify the event handle.

Part 4: Enabling Users to Add a New Purchase Order

When a new purchase order is created, a new record is inserted into the SALES_ ORDER table and the ITEM table. Content in the master table controls and detail table control is also updated to reflect the new purchase order.

1. Add the following code to the COrderView::*OnNeworder()* method in the OrderView.cpp file to add a new purchase order to the purchase order system:

```
#include "SelectCustomer.h" // define CSelectCustomer class
                            // to get customer information for
                      // a new purchase order
...
void COrderView::OnNeworder()
{
   CWnd *tempWindow;
   char szCUSTOMER_ID[8];
   OField oORDER_ID;
   CSelectCustomer selectCustomer;
   oresult dbresult;
```

```
    if (selectCustomer.DoModal(&GetDocument()->m_database) ==
        IDCANCEL)
        return;

    if (GetDocument()->m_database.GetSession().
        BeginTransaction() == OSUCCESS &&
        m_pDynasetMaster->AddNewRecord() == OSUCCESS)
{
            UpdateData(FALSE);
            tempWindow = GetDlgItem(IDC_Column1);
            itoa(selectCustomer.GetSelection(),
                 szCUSTOMER_ID, 10);
            tempWindow->SetWindowText(szCUSTOMER_ID);
            if (m_pDynasetMaster->GetEditMode() ==
                 ODYNASET_EDIT_NEWRECORD)
            {
                    ODynaset oSequenceDynaset;

                    dbresult = oSequenceDynaset.Open(
                         GetDocument()->m_database,
                         "SELECT MAX(ORDER_ID) + 1 FROM
                         DEMO.SALES_ORDER");
                    oORDER_ID = oSequenceDynaset.GetField(0);
                    dbresult = oSequenceDynaset.Close();
                    tempWindow = GetDlgItem(IDC_Column3);
                    tempWindow->SetWindowText
                      ((CString)oORDER_ID);
}
dbresult = UpdateData();
if (m_pDynasetMaster->Update() == OFAILURE)
{
                    ProcessOO4OError(m_pDynasetMaster);
                    return;
        }
        m_pDynasetMaster->ResetToDefaultFilter();
        m_pDynasetMaster->AddFilter("ORDER_ID = " +
        (CString)oORDER_ID);
        m_pDynasetMaster->RefreshQuery();
        m_pDynasetCustomer->ResetToDefaultFilter();
        m_pDynasetCustomer->AddFilter("CUSTOMER_ID = " +
        (CString)m_pDynasetMaster->m_Column1);
        m_pDynasetCustomer->RefreshQuery();
        m_pDynasetDetail->ResetToDefaultFilter();
        m_pDynasetDetail->AddFilter("ORDER_ID = " +
        (CString)m_pDynasetMaster->m_Column3);
```

```
            m_pDynasetDetail->RefreshQuery();
            UpdateData(FALSE);
        }
        CButton *tempButton = (CButton *)GetDlgItem(IDC_NEWORDER);
        tempButton->EnableWindow(FALSE);
        tempButton = (CButton *)GetDlgItem(IDC_COMMITORDER);
        tempButton->EnableWindow(TRUE);
        tempButton = (CButton *)GetDlgItem(IDC_CANCELORDER);
        tempButton->EnableWindow(TRUE);
    }
```

Part 5: Enabling Users to Commit a New Purchase Order

When a new purchase order is created, users must be able to commit or cancel the change made to the purchase order system.

1. Add the following code to the COrderView::*OnCommitorder()* method in the OrderView.cpp file to commit a new purchase order:

```
void COrderView::OnCommitorder()
{
    // TODO: Add your control notification handler code here
    CString strORDER_DATE;
    CString strSHIP_DATE;
    CWnd *tempWindow;

    m_pDynasetMaster->StartEdit();
    tempWindow = GetDlgItem(IDC_Column2);
    tempWindow->GetWindowText(strORDER_DATE);
    tempWindow = GetDlgItem(IDC_Column4);
    tempWindow->GetWindowText(strSHIP_DATE);
    m_pDynasetMaster->m_Column2.SetValue(LPCTSTR(strORDER_DATE));
    m_pDynasetMaster->m_Column4.SetValue(LPCTSTR(strSHIP_DATE));
    m_pDynasetMaster->Update();

    OSession oSess = GetDocument()->m_database.GetSession();
    if (oSess.Commit() == OFAILURE)
        ProcessOO4OError(&oSess);
    m_pDynasetMaster->ResetToDefaultFilter();
    m_pDynasetMaster->RefreshQuery();
    UpdateData(FALSE);
    OnMoveFirst();
    CButton *tempButton = (CButton *)GetDlgItem(IDC_NEWORDER);
    tempButton->EnableWindow(TRUE);
    tempButton = (CButton *)GetDlgItem(IDC_COMMITORDER);
    tempButton->EnableWindow(FALSE);
    tempButton = (CButton *)GetDlgItem(IDC_CANCELORDER);
    tempButton->EnableWindow(FALSE);
}
```

Part 6: Enabling Users to Cancel a New Purchase Order

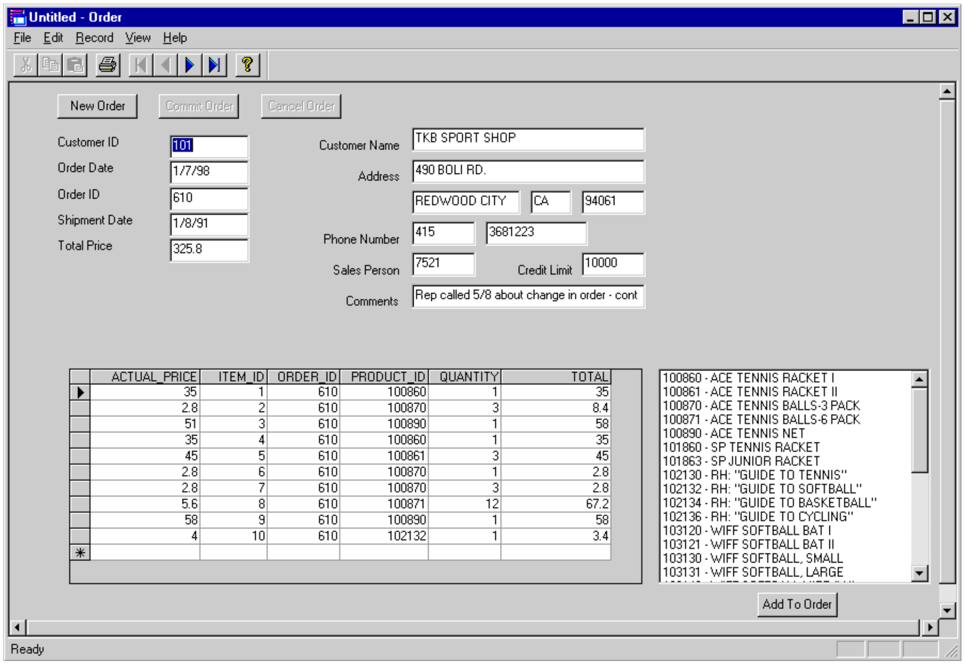
When a new purchase order is created, users must be able to commit or cancel the change made to the purchase order system.

1. Add the following code to the COrderView::*OnCancelorder()* method in the OrderView.cpp file to enable cancelling a new purchase order:

```
void COrderView::OnCancelorder()
{
    // TODO: Add your control notification handler code here
    GetDocument()->m_database.GetSession().Rollback();
    m_pDynasetMaster->ResetToDefaultFilter();
    m_pDynasetMaster->RefreshQuery();
    UpdateData(FALSE);
    OnMoveFirst();
    CButton *tempButton = (CButton *)GetDlgItem(IDC_NEWORDER);
    tempButton->EnableWindow(TRUE);
    tempButton = (CButton *)GetDlgItem(IDC_COMMITORDER);
    tempButton->EnableWindow(FALSE);
    tempButton = (CButton *)GetDlgItem(IDC_CANCELORDER);
    tempButton->EnableWindow(FALSE);
}
```

2. Build and run the application.

   It should look similar to the following screen:



Customer ID	101
Order Date	1/7/98
Order ID	610
Shipment Date	1/8/91
Total Price	325.8
Customer Name	TKB SPORT SHOP
Address	490 BOLI RD.
	REDWOOD CITY · CA · 94061
Phone Number	415 · 3681223
Sales Person	7521
Credit Limit	10000
Comments	Rep called 5/8 about change in order - cont

ACTUAL_PRICE	ITEM_ID	ORDER_ID	PRODUCT_ID	QUANTITY	TOTAL
35	1	610	100860	1	35
2.8	2	610	100870	3	8.4
51	3	610	100890	1	58
35	4	610	100860	1	35
45	5	610	100861	3	45
2.8	6	610	100870	1	2.8
2.8	7	610	100870	3	2.8
5.6	8	610	100871	12	67.2
58	9	610	100890	1	58
4	10	610	102132	1	3.4

100860 - ACE TENNIS RACKET I
100861 - ACE TENNIS RACKET II
100870 - ACE TENNIS BALLS-3 PACK
100871 - ACE TENNIS BALLS-6 PACK
100890 - ACE TENNIS NET
101860 - SP TENNIS RACKET
101863 - SP JUNIOR RACKET
102130 - RH: "GUIDE TO TENNIS"
102132 - RH: "GUIDE TO SOFTBALL"
102134 - RH: "GUIDE TO BASKETBALL"
102136 - RH: "GUIDE TO CYCLING"
103120 - WIFF SOFTBALL BAT I
103121 - WIFF SOFTBALL BAT II
103130 - WIFF SOFTBALL, SMALL
103131 - WIFF SOFTBALL, LARGE

Your application is now complete.

Index